Watson Assistant Service
Deep Dive Hands on Technical Lab

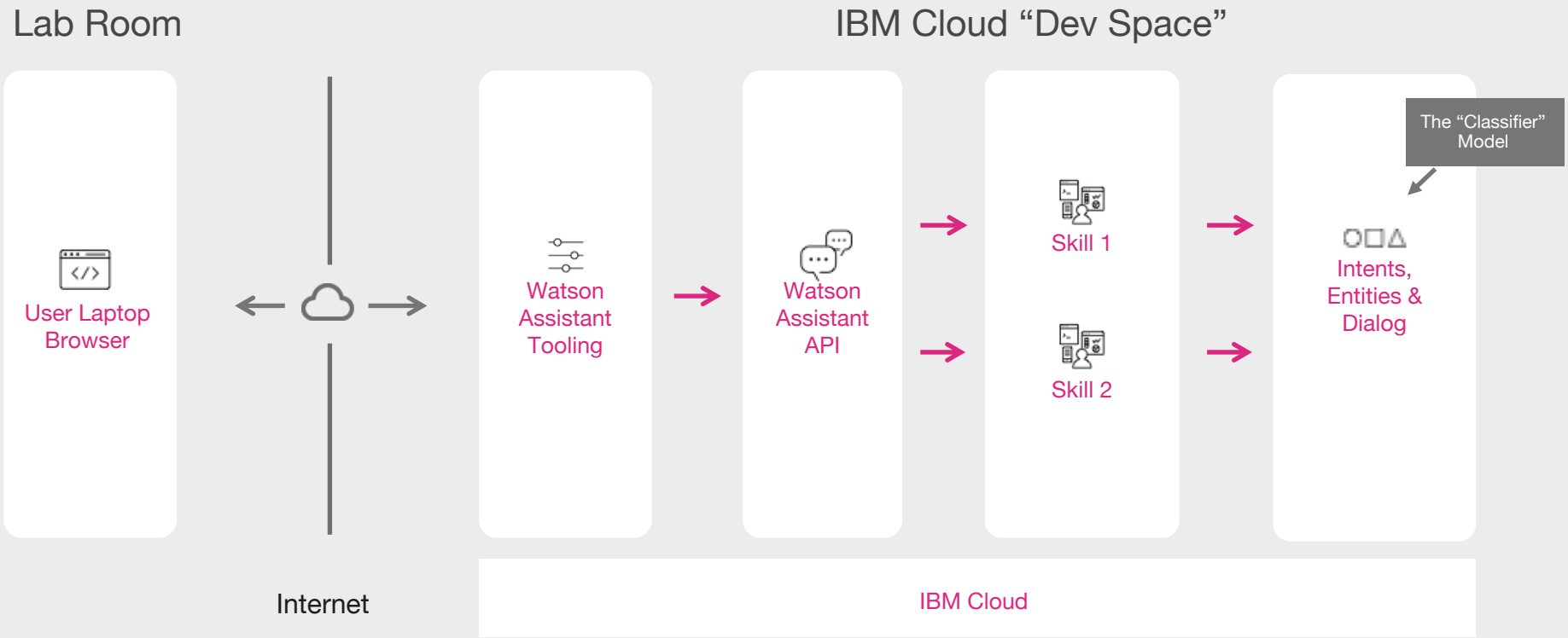
AT&T
Learn - Day 1
November 12th, 2018

Watson

IBM

Watson Assistant
Fundamental Concepts

IBM Cloud Spaces and Watson Assistant Skills



Intents

The action a user wants to take (verb).

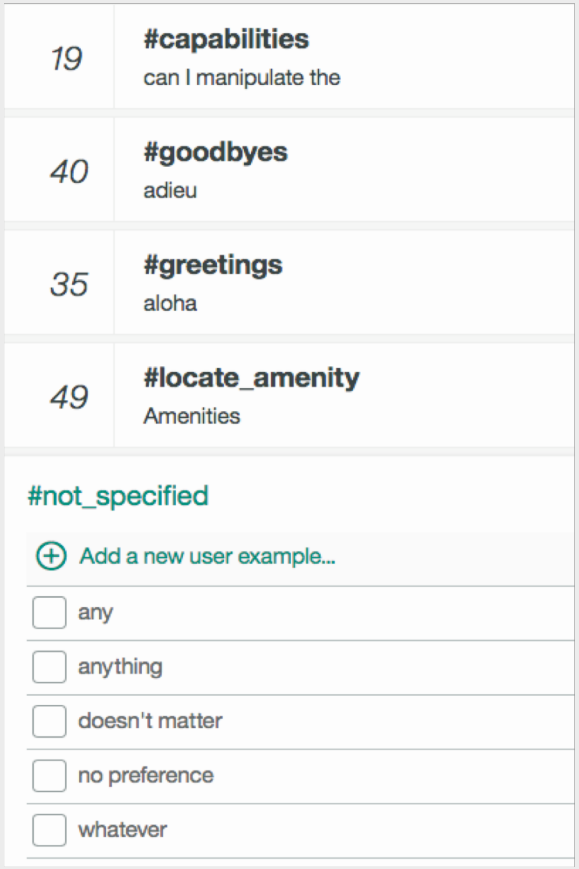
A label for a group of examples of things that a user might say to communicate a specific goal or idea.

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.



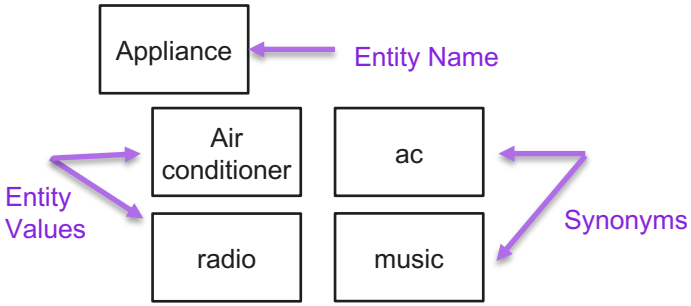
Entities

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.

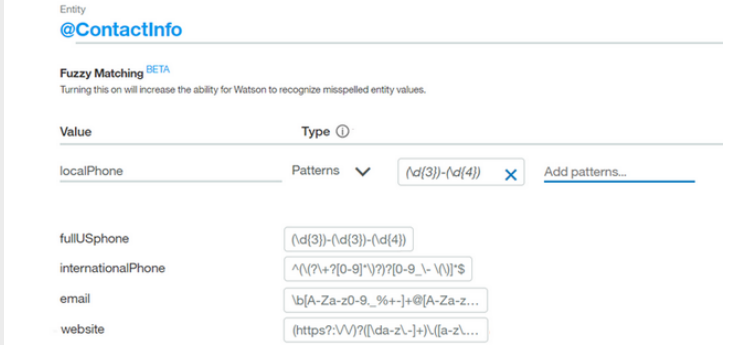
Patterns may not contain:

- Positive repetitions (e.g., x*+)
- Backreferences (e.g., \g1)
- Conditional branches (e.g., (?(cond)true))

Email example, e.g name@ibm.com
\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b

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

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



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 – Currenty unit (Euro, USD, etc)

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

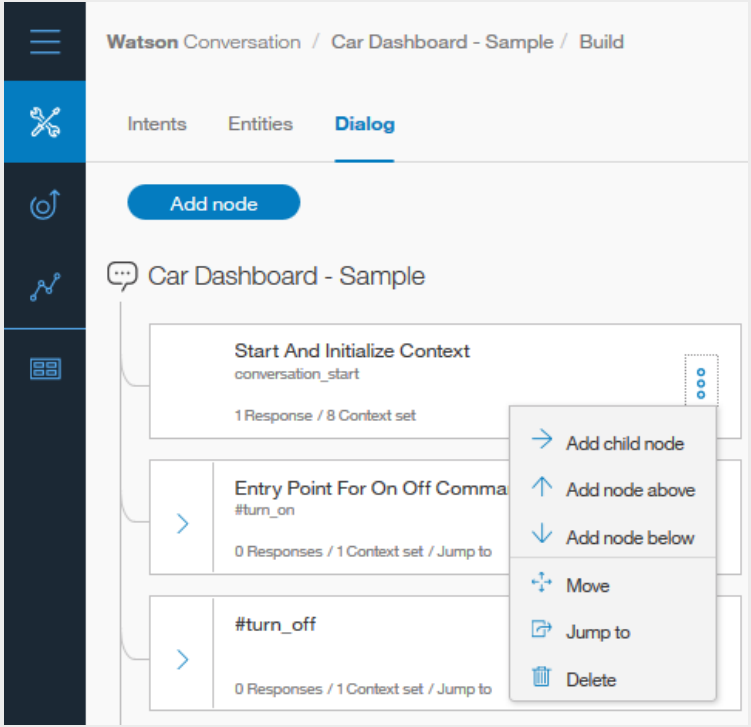
@sys-number.literal – String text representation

@sys-number.numeric_value – Number representation

Dialog

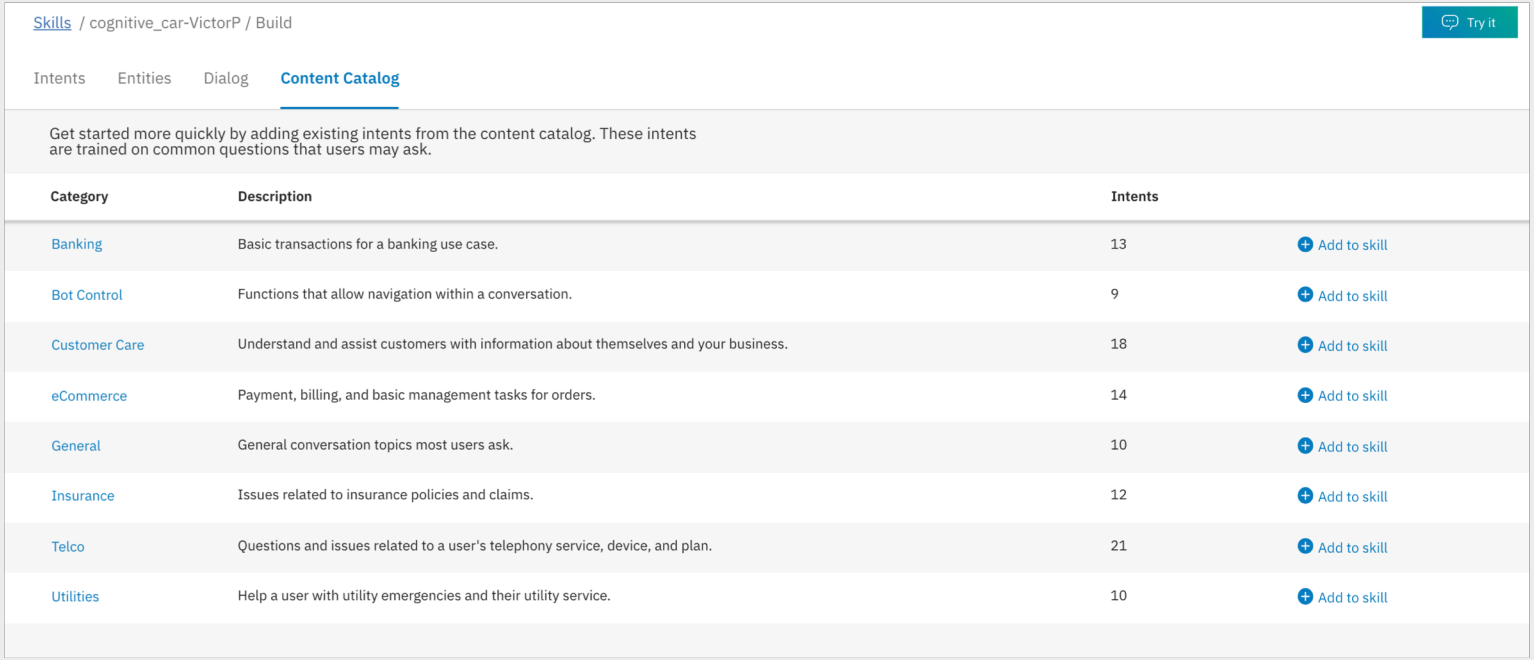
Dialog uses the intents and entities that are identified in the user's input, plus context from the application, to interact with the user and ultimately provide a response

- 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



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.



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

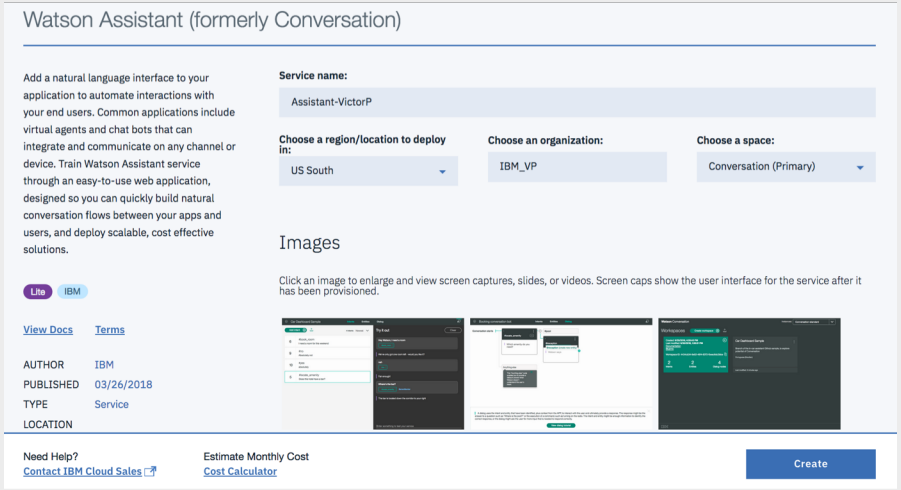
An Assistant service instance is created on the IBM Bluemix console. The procedure is identical to that of creating other Watson services.

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.

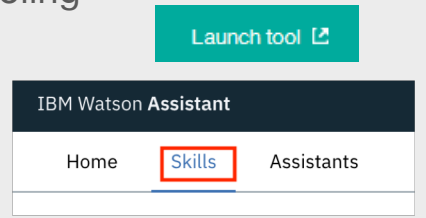


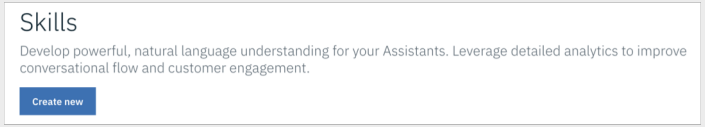
Create a Skill

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
2. Select the Skills tab
3. Click **Create new**.
4. Name it: cognitive_car - *FirstNameFirstLetterOfLastName, such as* cognitive_car-*VictorP*, click **Create**.





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

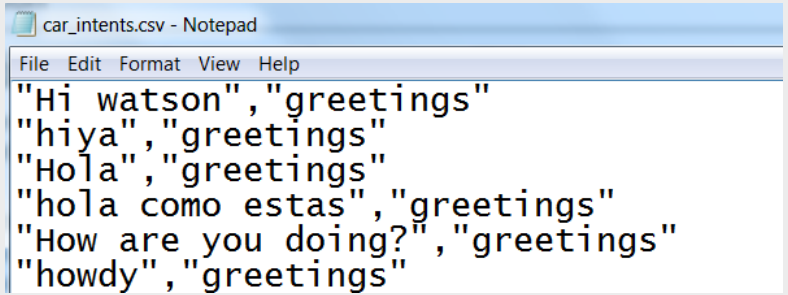
Intent	Utterance	Entity
		What objects could be used as entities in the following utterances?
turn_down	AC low	
turn_off	Turn off the air conditioning	
turn_down	lower ac	
turn_down	Reduce the fan speed	
turn_down	decrease the volume	
turn_down	Decrease volume	
turn_off	shut of the headlights	
turn_down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

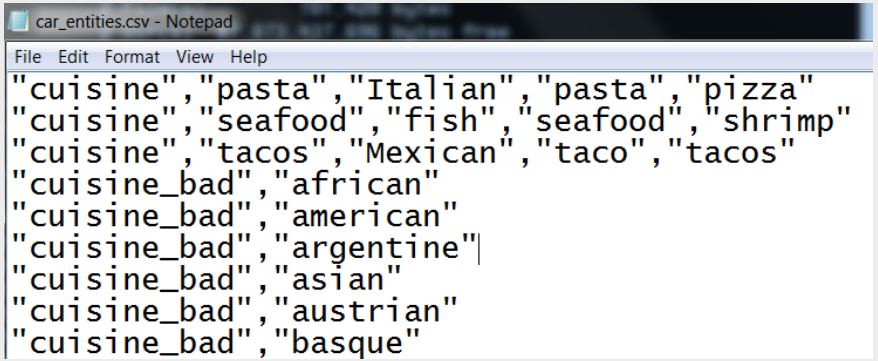


Creating an Entity Import File

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			

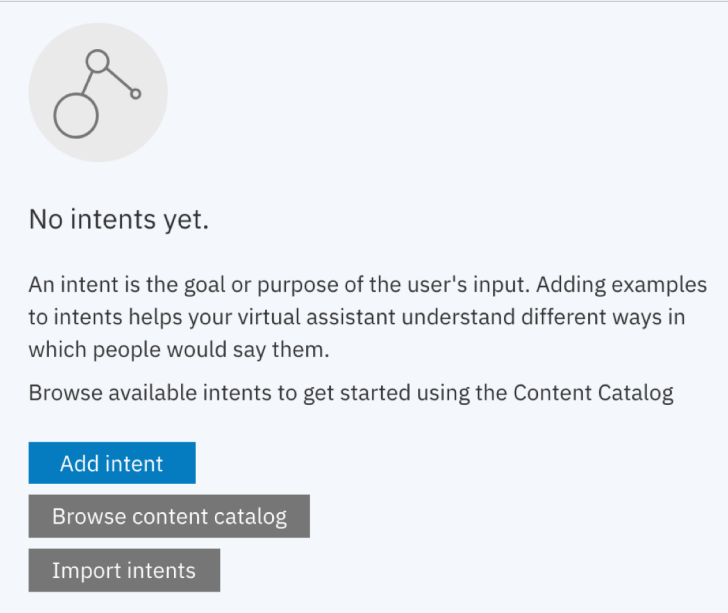


Import Intents in Watson Assistant Tooling

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**



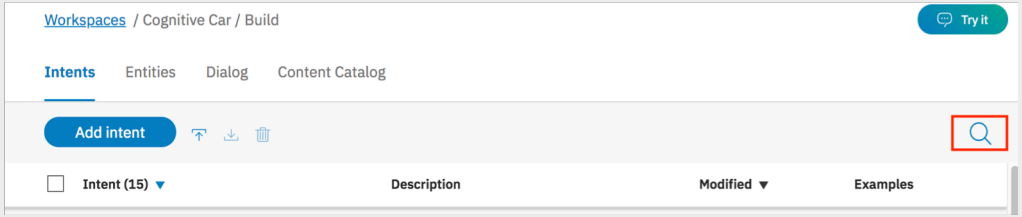
Search

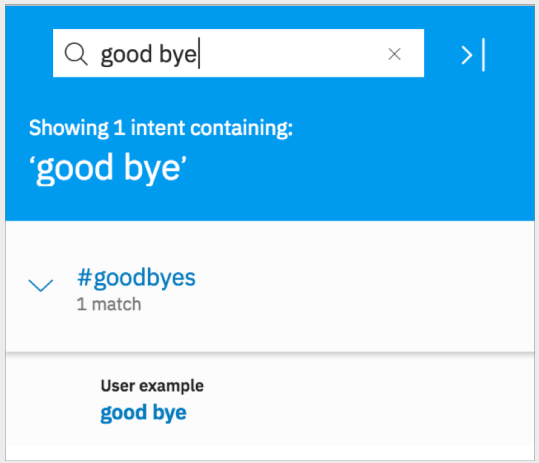
The Assistant allows to quickly find information within the Intent, Entities and Dialog pages.

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.

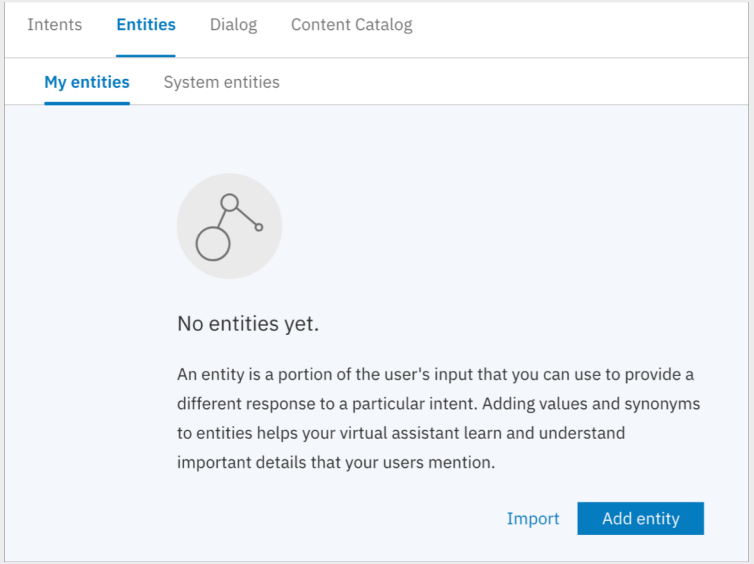


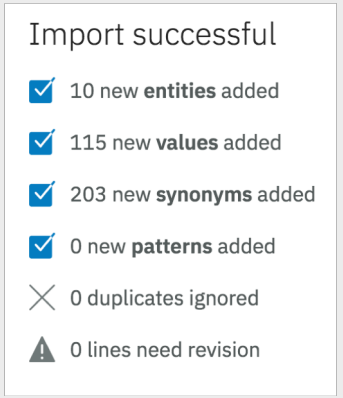


Note: Search will not be available while the system is indexing the imported data.

Import Entities in Watson Assistant Tooling

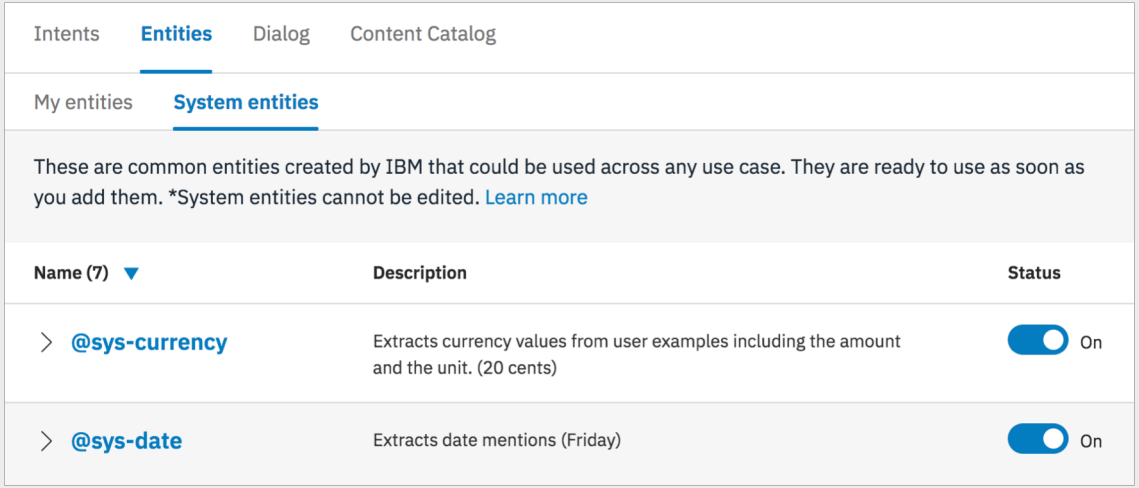
1. Switch to the **Entities** page.
2. On the Entities page - Click **Import**.
3. Click **Choose a File**.
4. Navigate to your lab directory and select your entities.csv or the lab provided car_entities.csv Navigate to your lab directory and select car_entities.csv.
5. Click **Import**.
6. Should get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





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



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

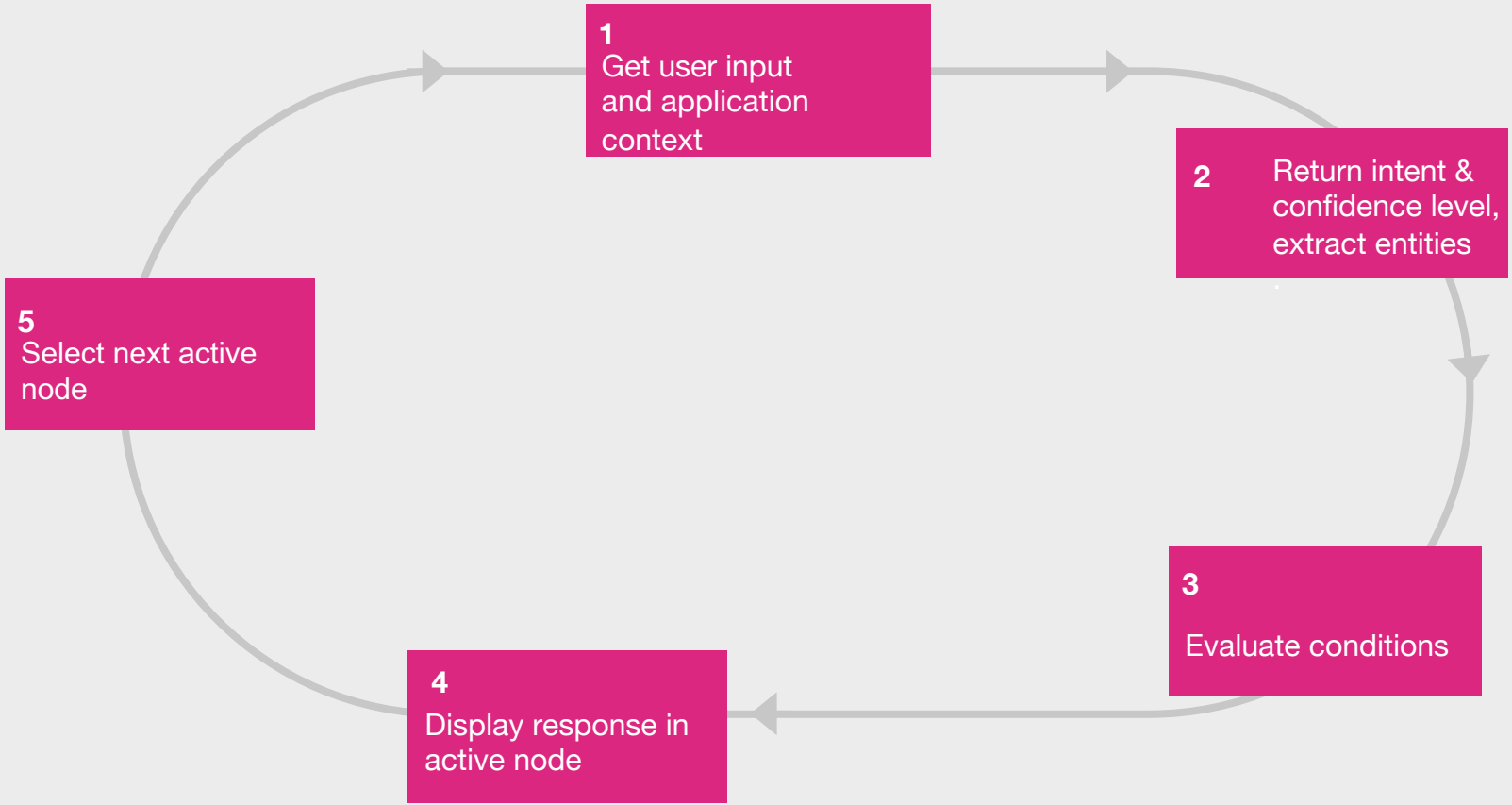
Good dialog design should reflect the solution's positioning to create a coherent and compelling solution that behaves as envisaged by you and the client.

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!

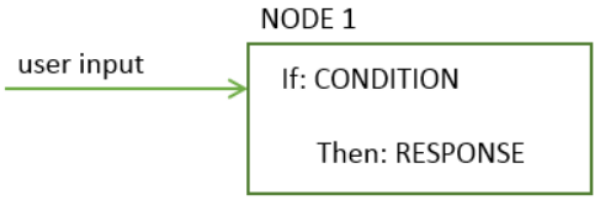
To Reflect the positioning, the dialog design must create behaviors that:

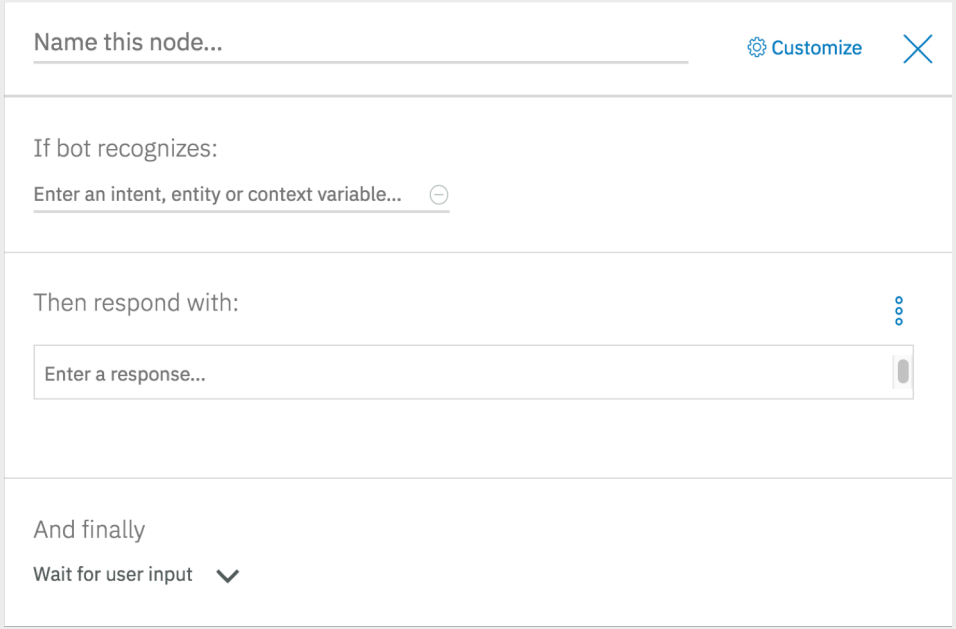
- Help achieve the defined **purpose**
- Consistently express the identified **viewpoint**
- Reflect the specified **proactivity**
- Reflect the **tone** and **personality**

Dialog – Assistant turn



Dialog Node


NODE 1
user input
If: CONDITION
Then: RESPONSE


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 sytnax #{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

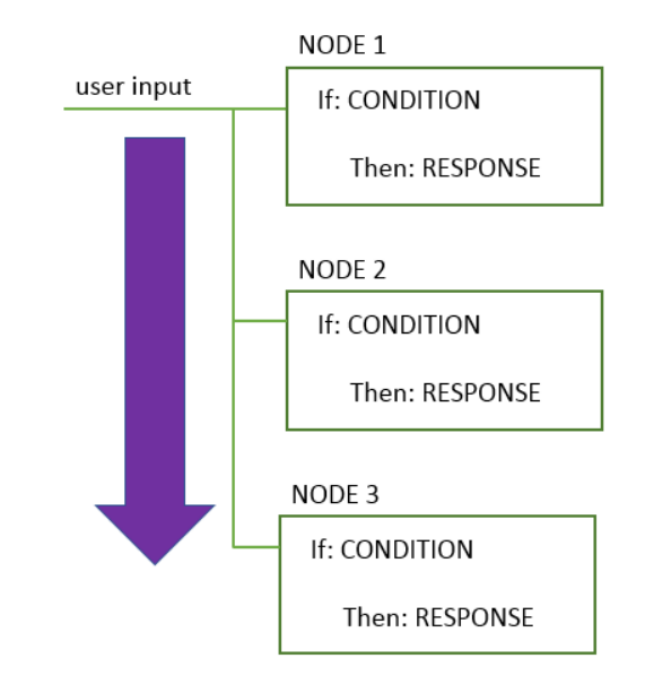
The dialog response defines how to reply to the user.

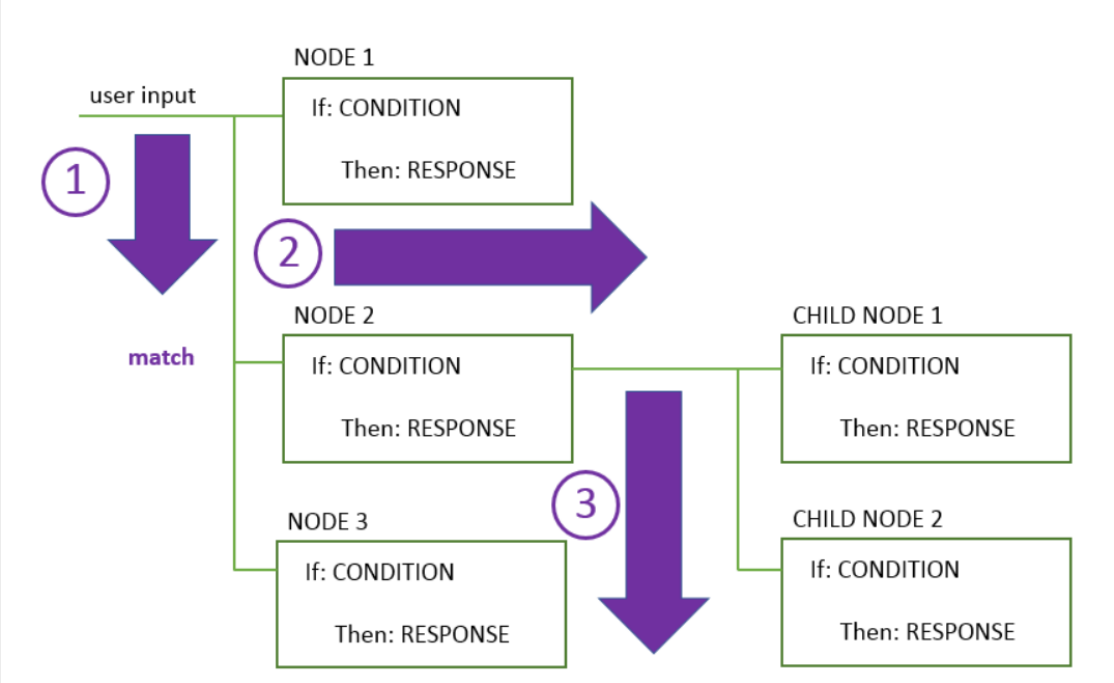
- **Simple text response -** text that the bot displays to the user
- **Multiple conditioned responses** - different responses, each one triggered by a different condition
- **Complex response** – use the JSON editor to specify the response in the "output":{} property

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





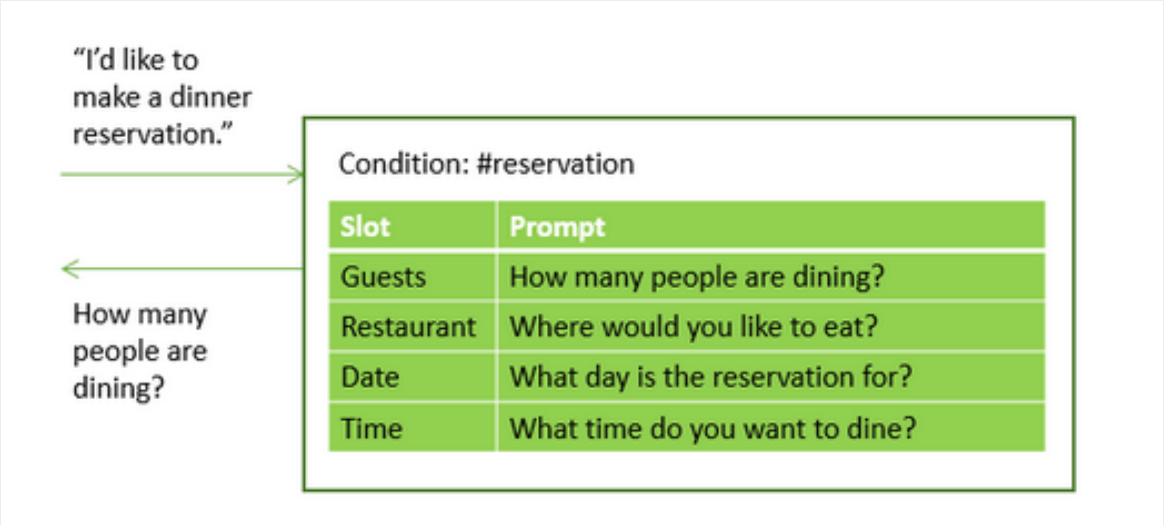
Context Variables

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.



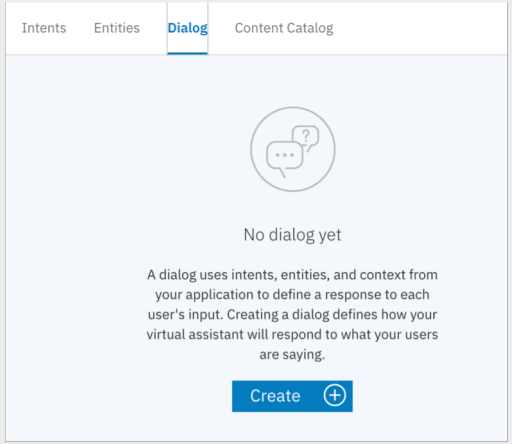
Dialog
Lab

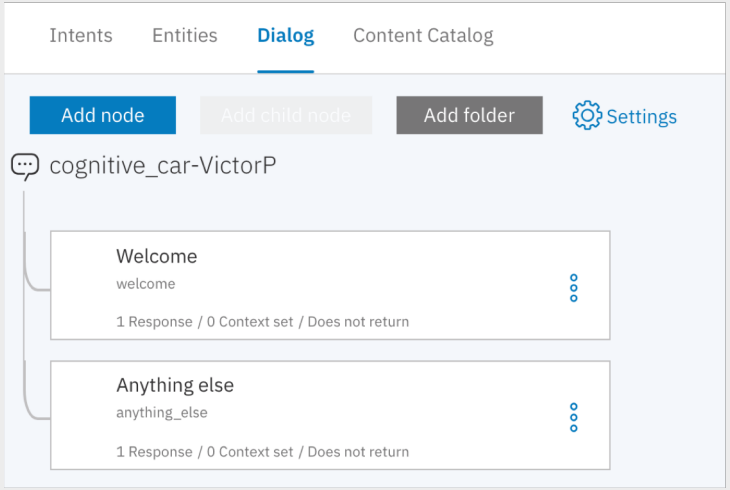
Create Dialog

1. Switch to the Dialog page.
2. Click Create.

Two nodes will automatically be created for you

- Welcome
- Anything else

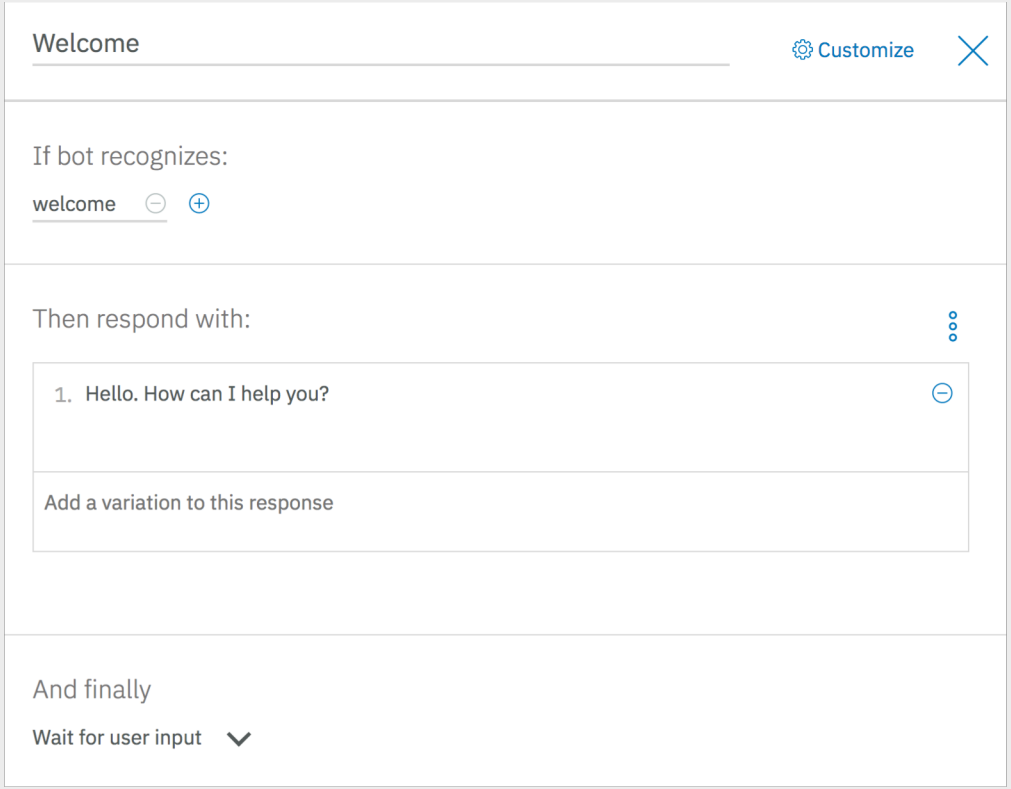




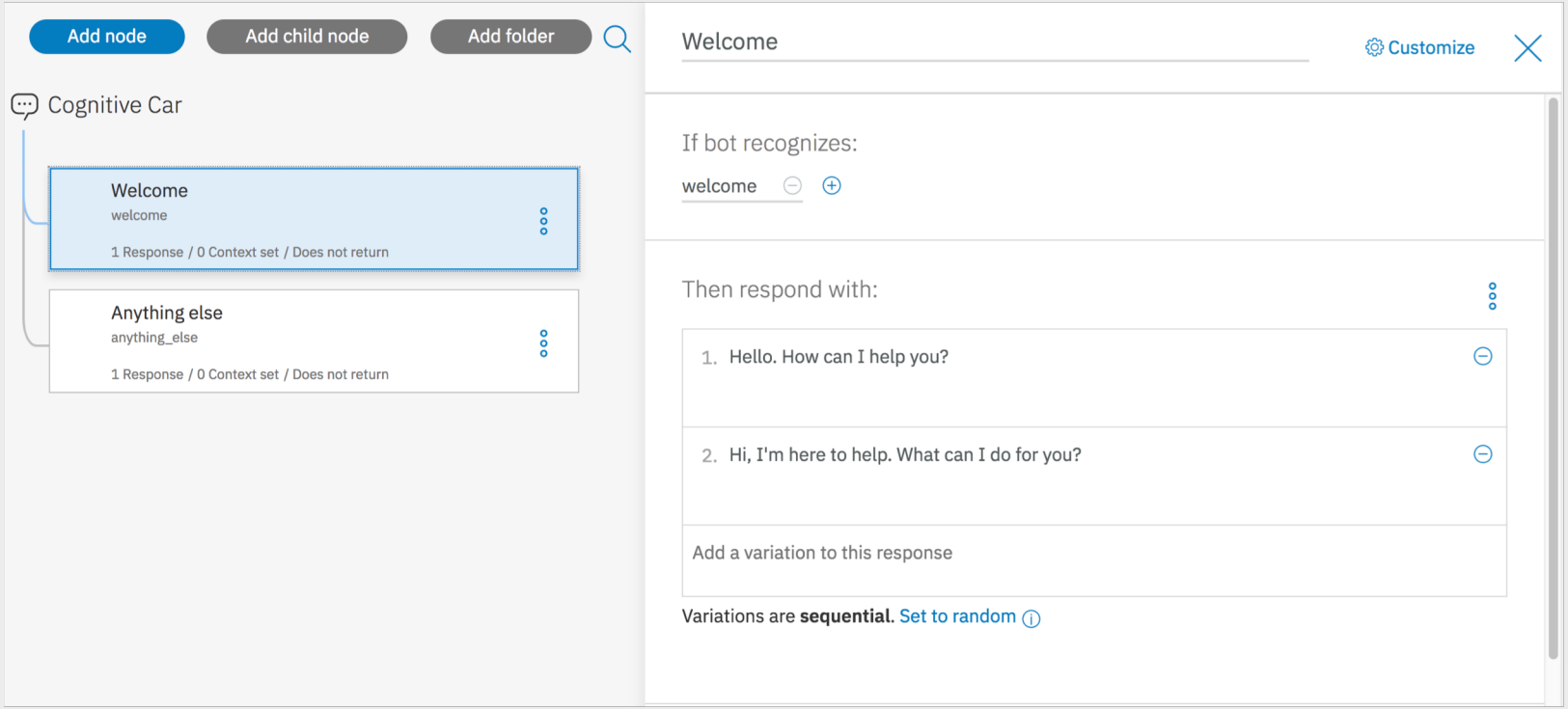
Welcome Node

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



Results – Welcome Node

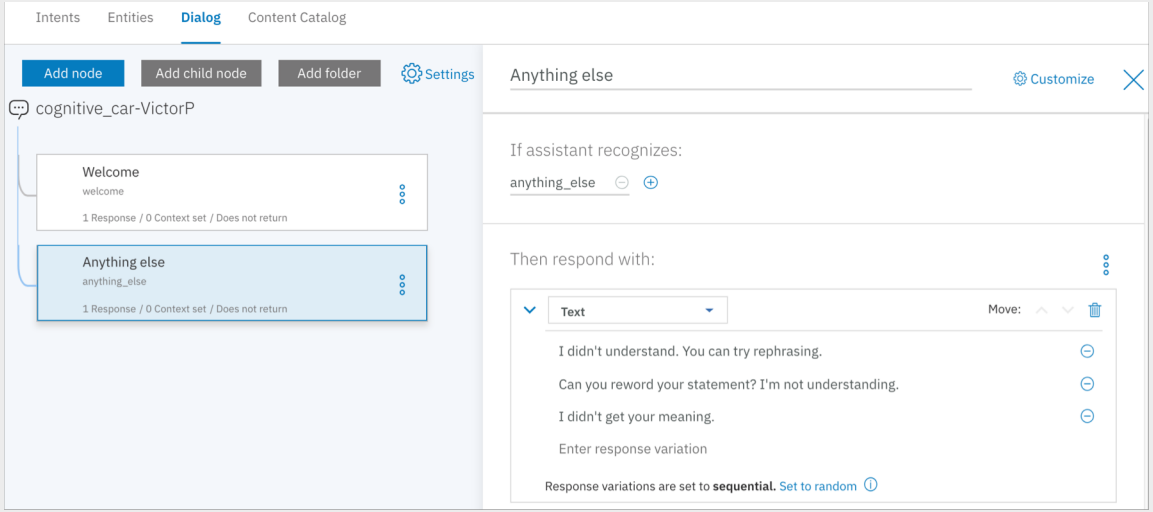


Anything_Else Node

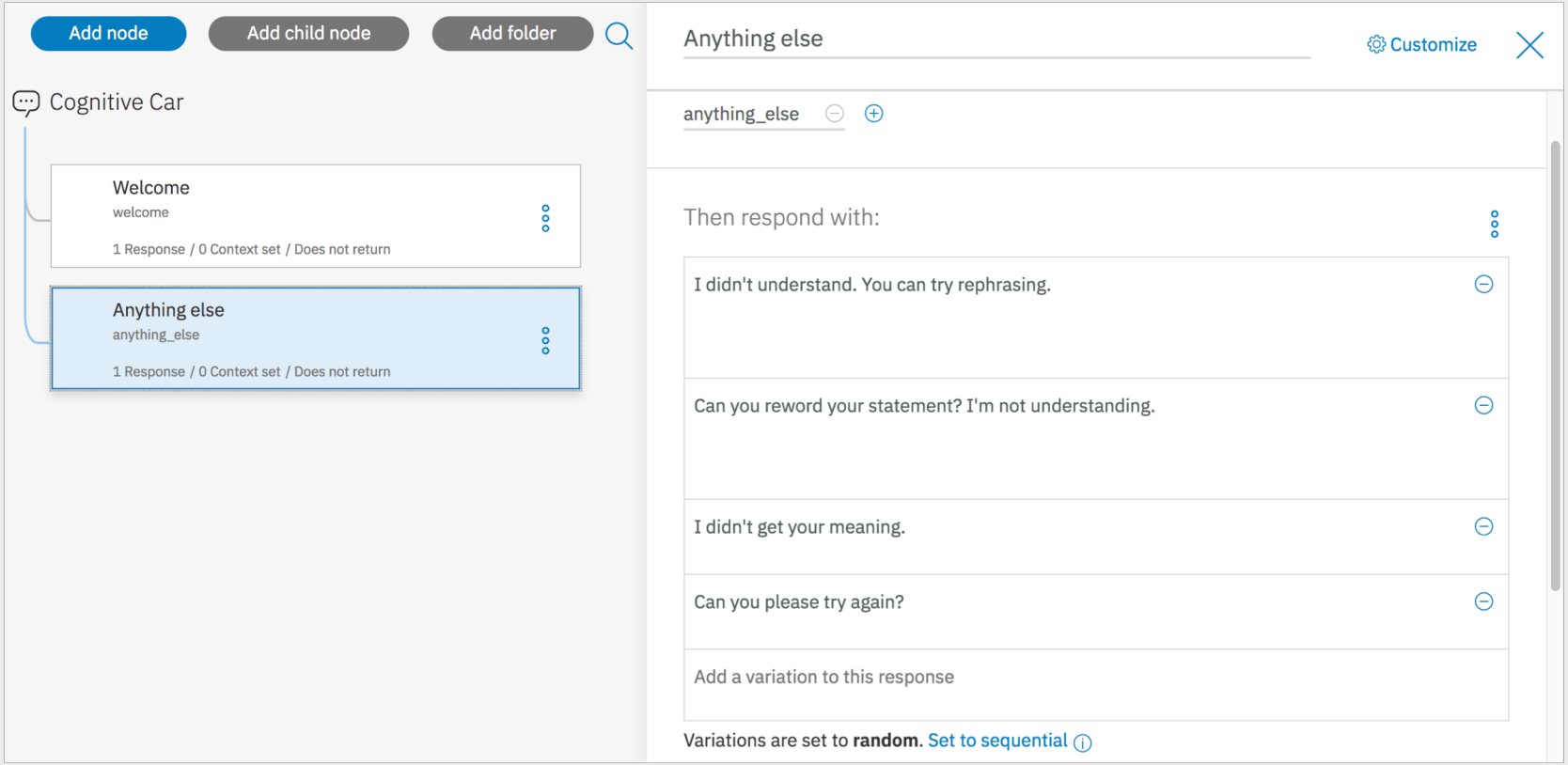
The anything_else node gets hit if the system has low confidence in the intent, or if no other dialog conditions matched.

1. Add a low-confidence message to your Anything else node

2. Set your response variations to random.



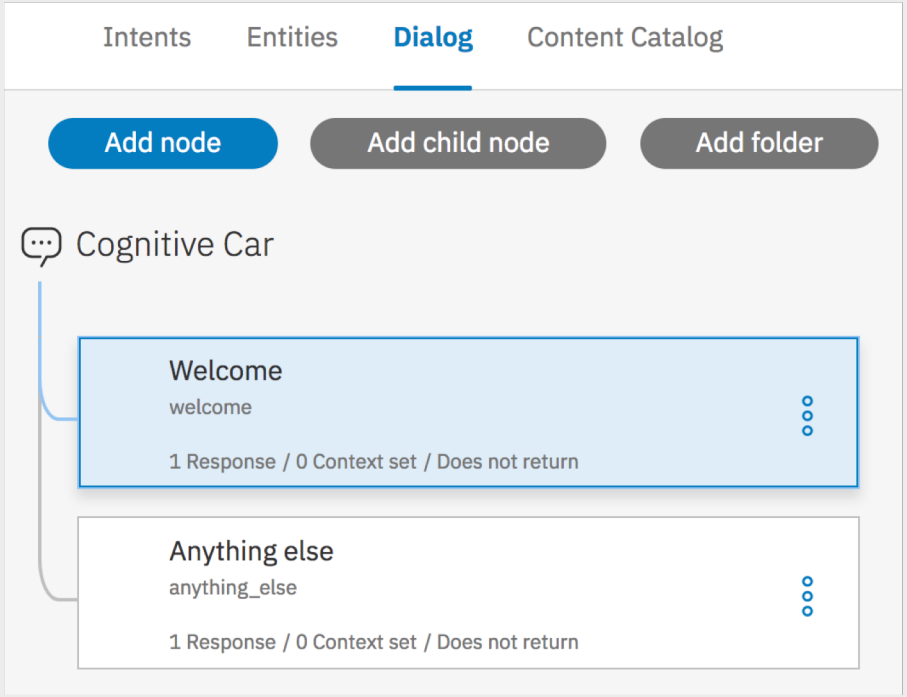
Results – Anything_Else Node



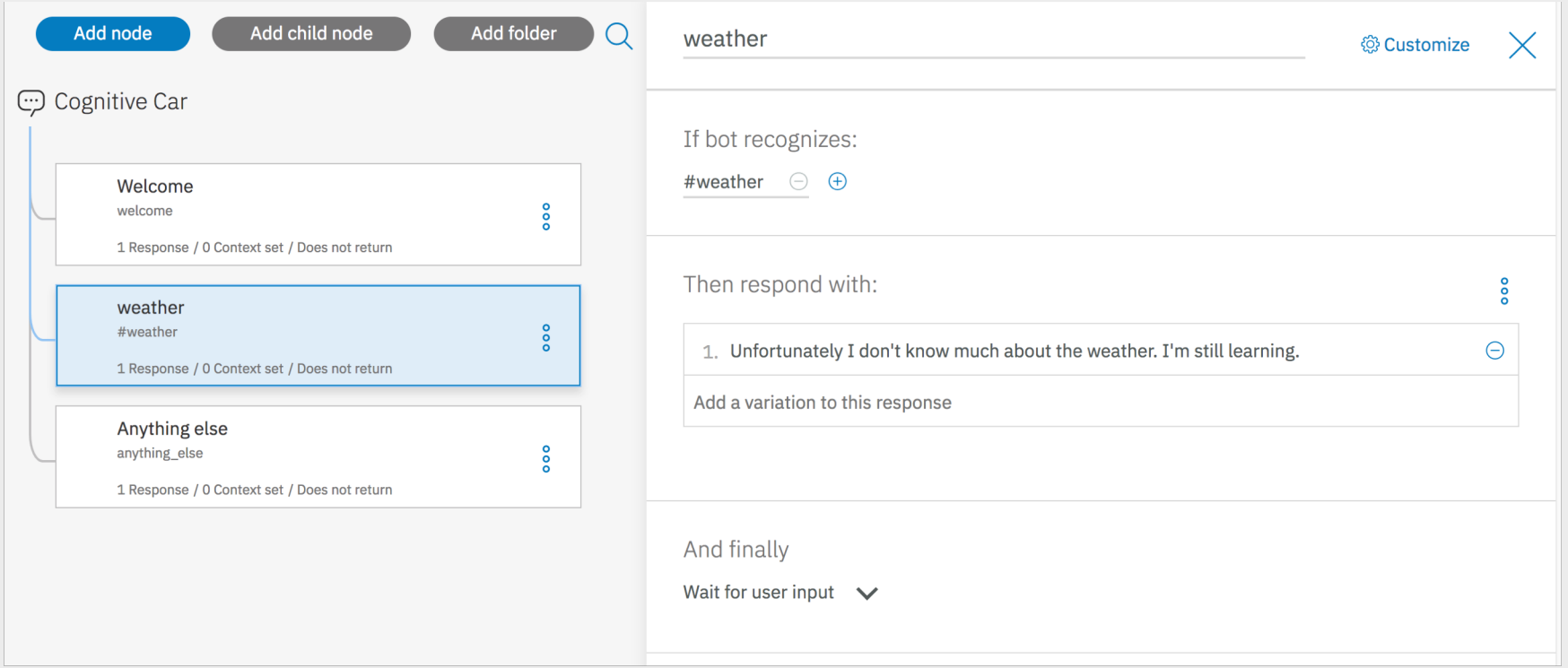
Create Weather Node

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



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 *not* 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 *not* exactly match the query term.
<=, >=, >, <	Comparison operators. Prefix the query term with these operators to match based on arithmetic comparison.

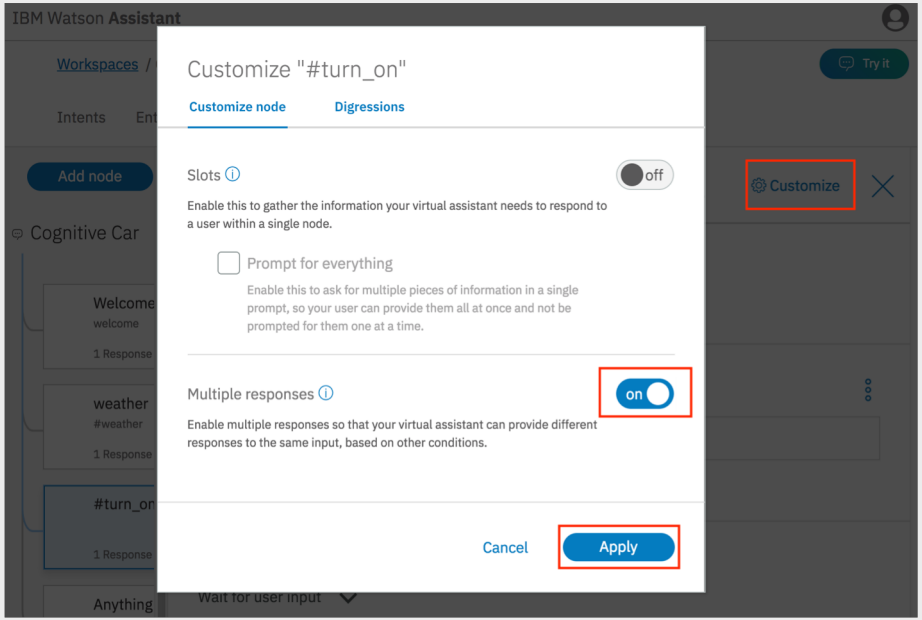
`\`	Escape operator. Use in queries that include control characters that would otherwise be parsed as operators. For example, `\!hello` would match the text `!hello`.
`" "`	Literal phrase. Use to encloses a query term that contains spaces or other special characters. No special characters within the 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.	`response_timestamp>=2017-07-01,response_timestamp<2017-08-01`
The timestamp of the response is earlier than `2016-11-01T04:00:00.000Z`.	`response_timestamp<2016-11-01T04:00:00.000Z`
The user input text contains the word "order" or a grammatical variant (for example, `orders` or `ordering`.	`request.input.text:order`
An intent name in the response exactly matches `place_order`.	`response.intents:intent::place_order`
An entity name in the response exactly matches `beverage`.	`response.entities:entity::beverage`
The user input text does not contain the word "order" or a grammatical variant.	`request.input.text:!order`

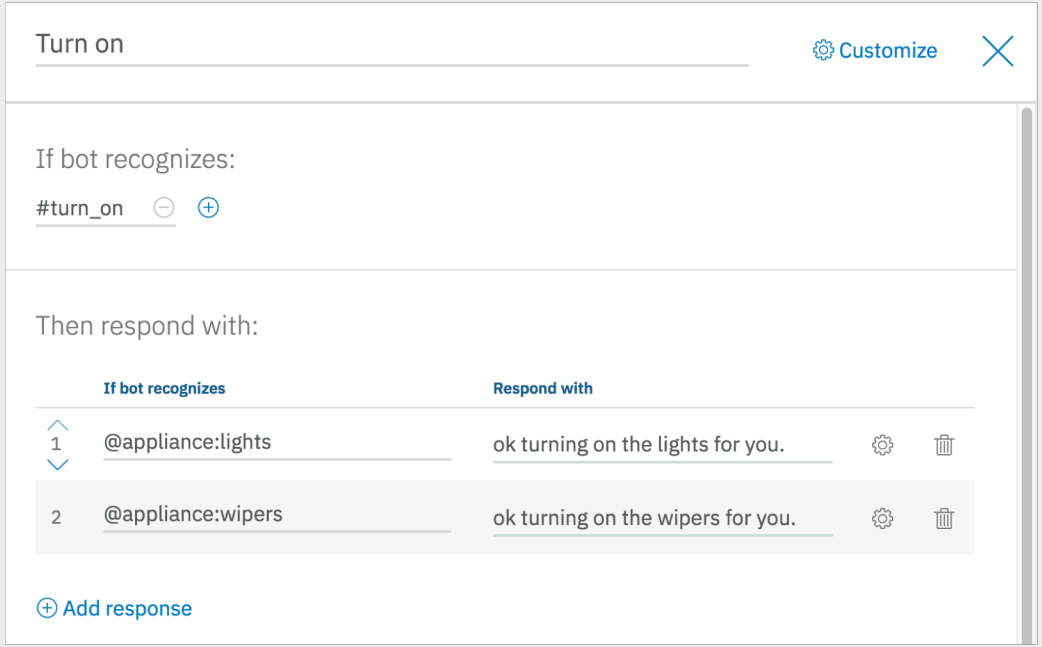
Create a Multi-conditional Response Node

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



Create a Multi-conditional Response Node Continued

1. Add conditional responses
   - lights @appliance:lights
   - wipers: @appliance:wipers

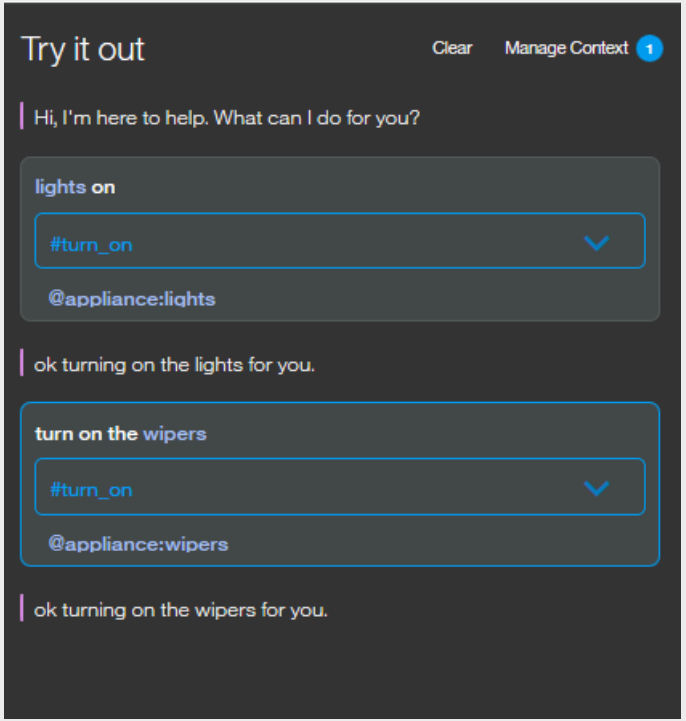


Test it Out!

Open your Try it out panel, and click clear.

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

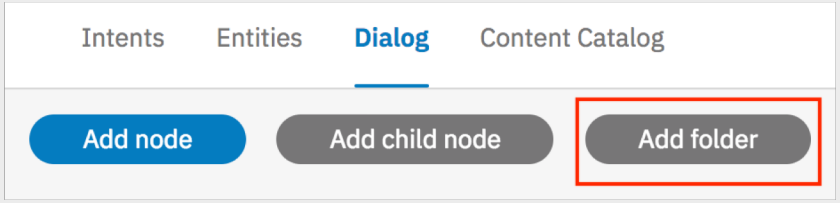
– I.e. "Turn on the Wipers"

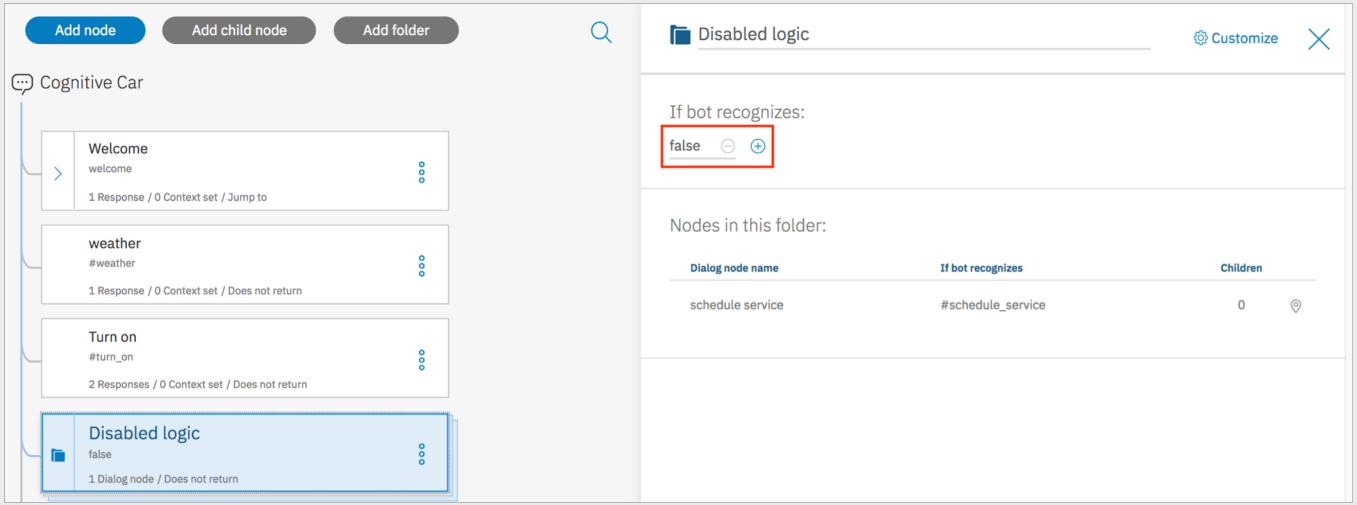


Folders

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.





Slots

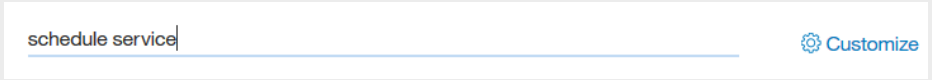
In this lab, we'll use slots to collect multiple pieces of information as needed to complete a complex task for a user, such as scheduling service for your cognitive car.

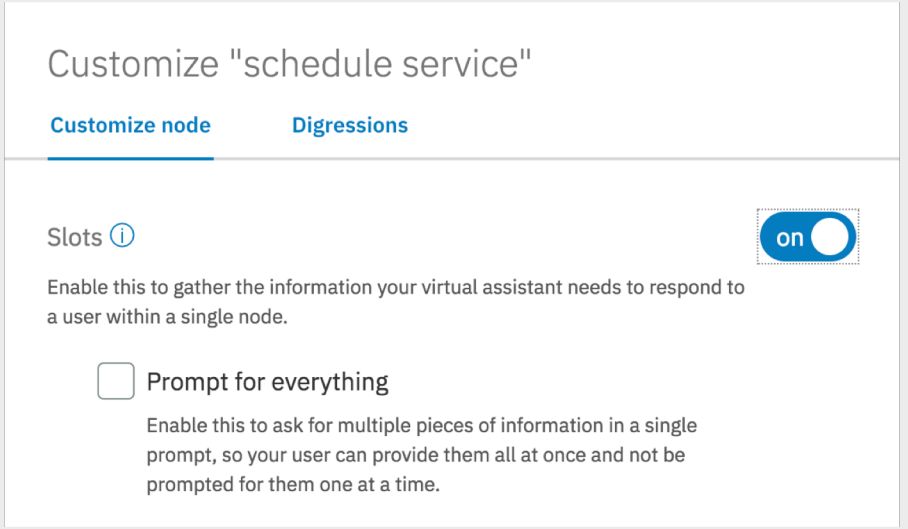
Condition #schedule_service

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





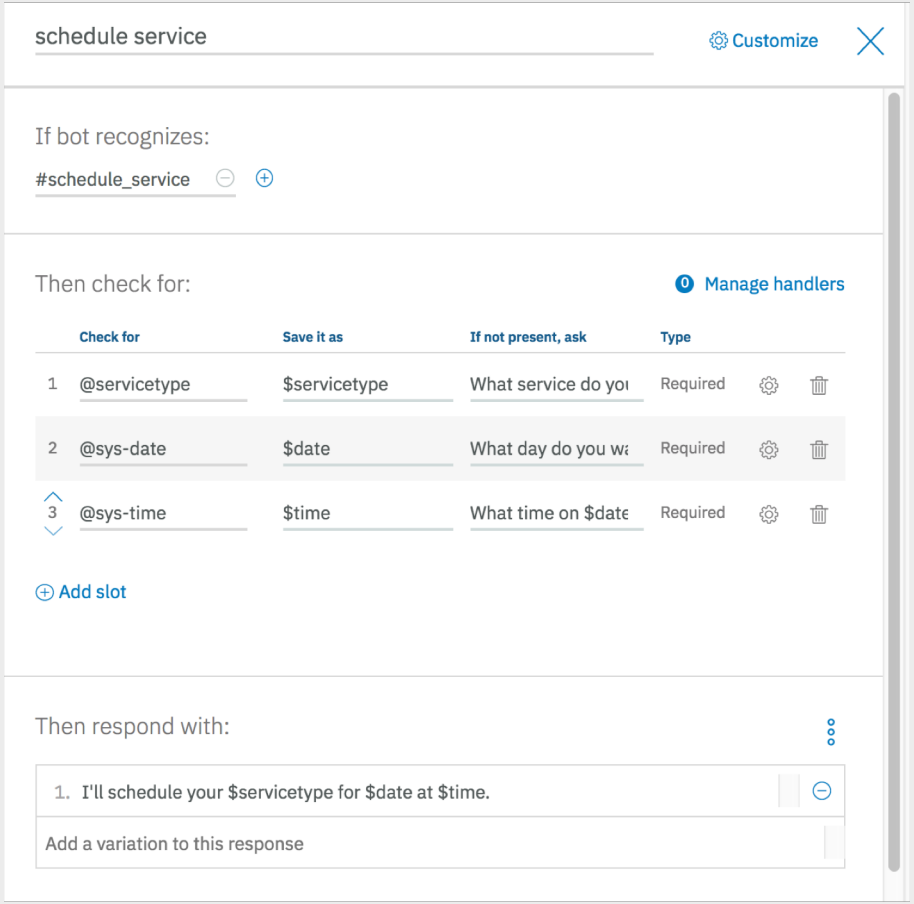
Configure the Slots

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

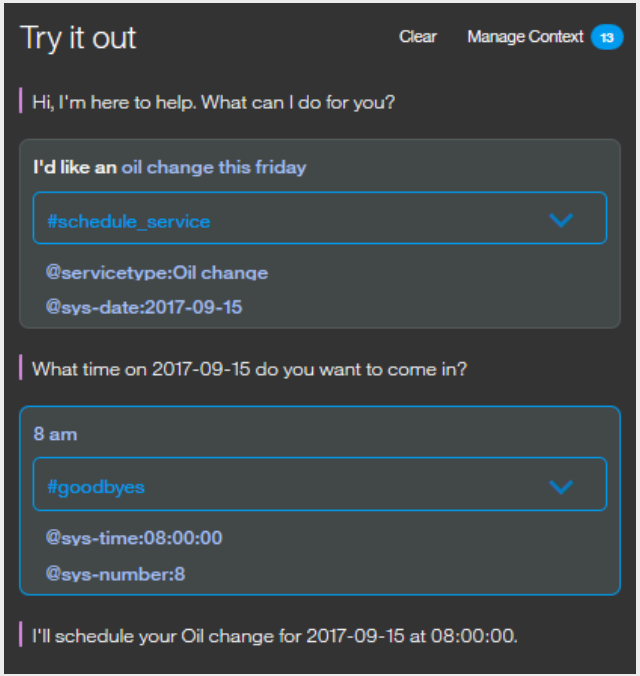
4. Respond with: I'll schedule your $servicetype for $date at $time.

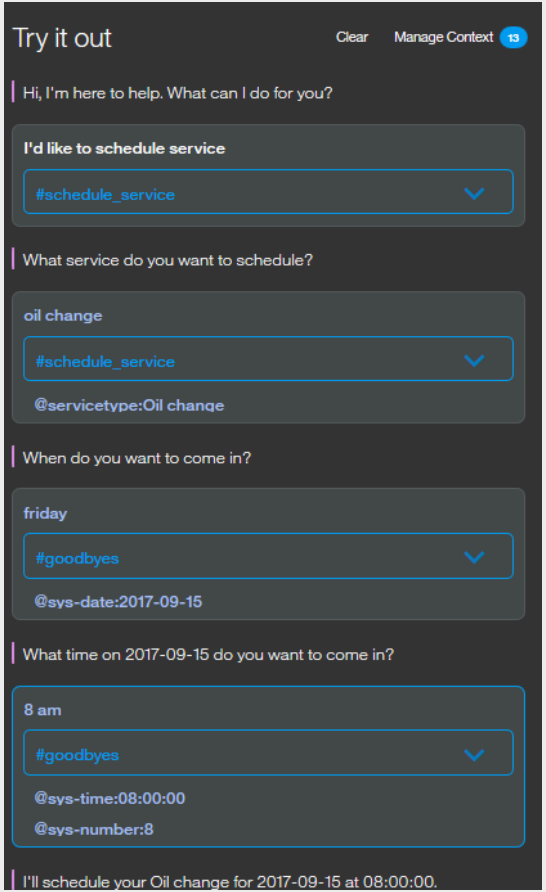
Results – Schedule Service slot



Test it Out!

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





Configure Slot Handlers

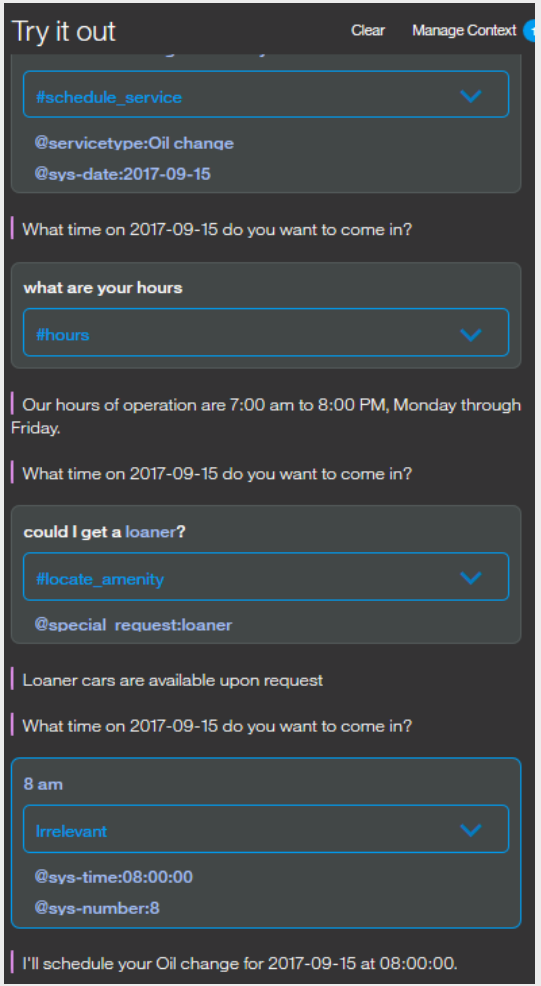
1. Edit the Schedule Service node
2. Click on Manage Handlers
3. Add handlers to respond to questions about loaner cars and hours

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.



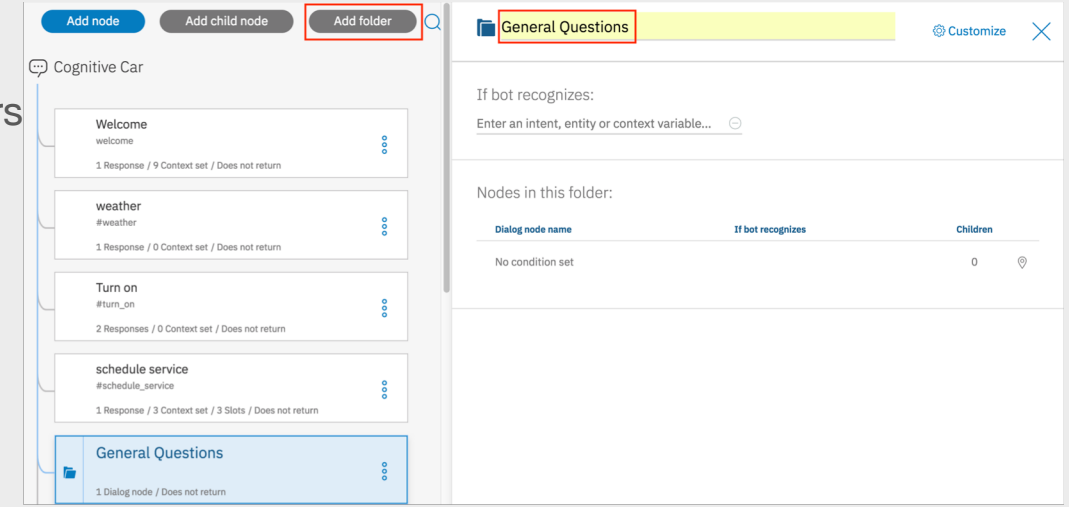
Digression

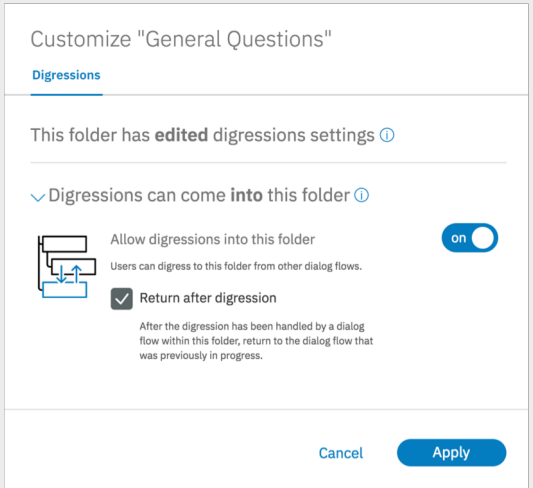
Digression is an alternative to Slot Handlers that can be reused in multiple places. Digressions allows to answer a common question outside the “schedule service” process and then return the user to where they left off in the process.

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.



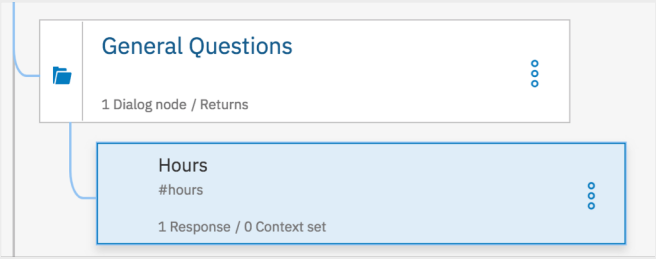


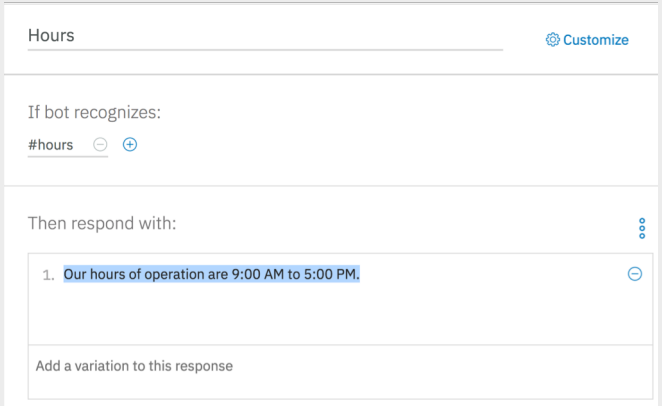
Digression Continued

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.

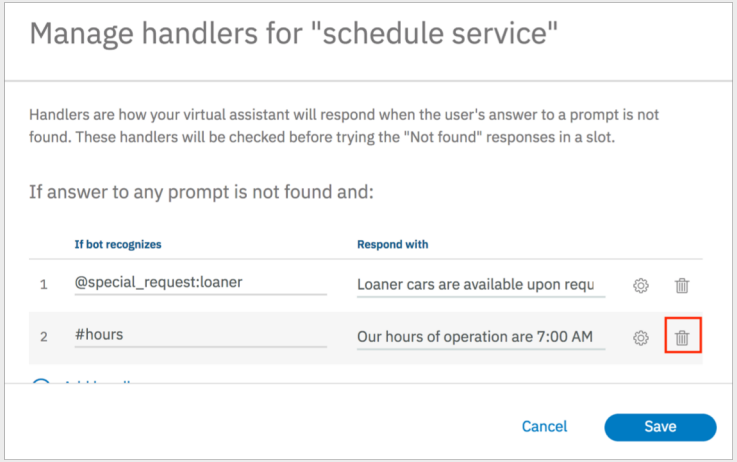




Digression Continued

Remove the Slot Handler for “#hours”

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

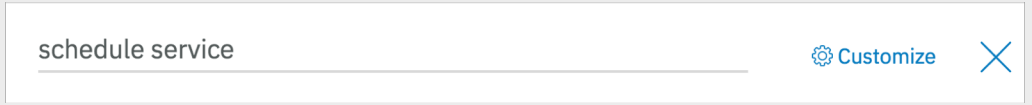


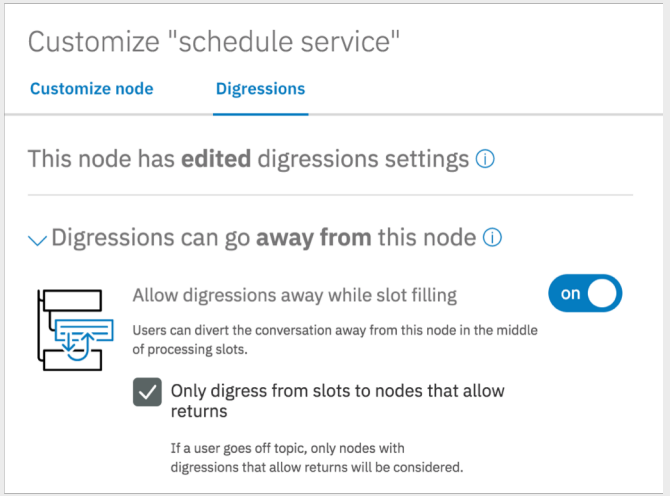
Digression Continued

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

1. Select the Schedule Service node and click Customize.

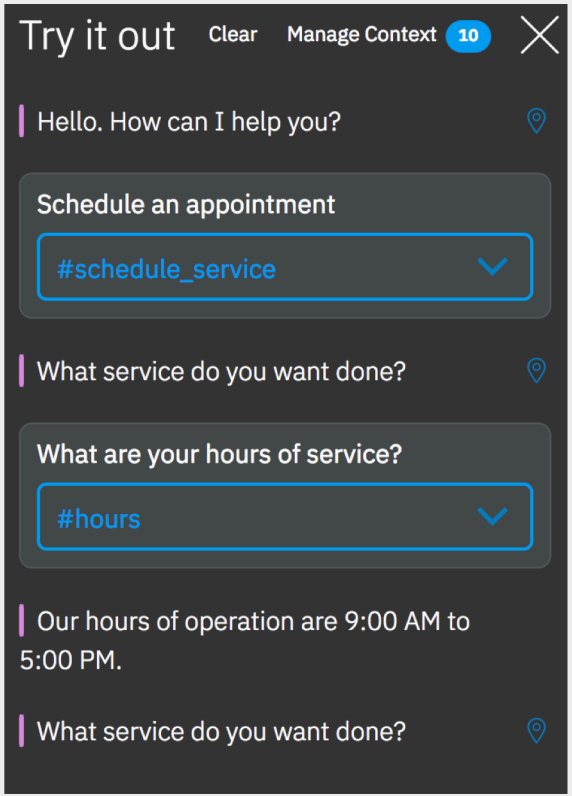
2. Select the Digression tab and turn on “Digressions can go **away from** this node” as well as select the “Only digress from slots to nodes that allow returns” option.





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.



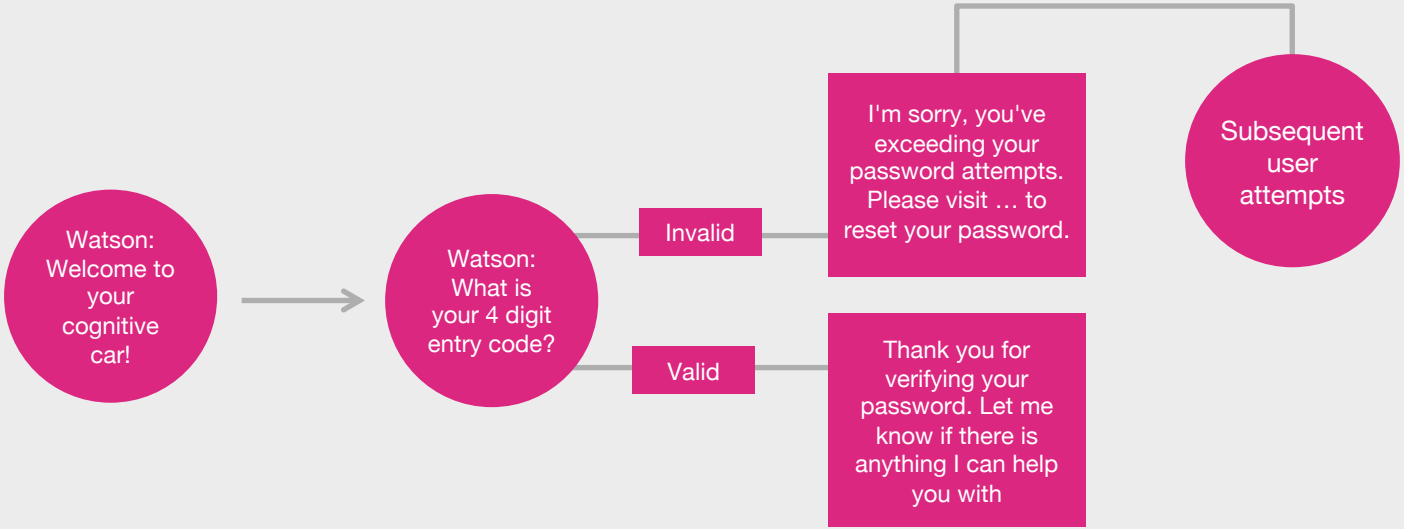
Now you're ready… Designing your first Multi-turn Interaction

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

```
{
  "context": {
    "AConoff": "off",
    "passkey": 1234,
    "lightonoff": "off",
    "musiconoff": "off",
    "appl_action": "",
    "heateronoff": "off",
    "volumeonoff": "off",
    "wipersonoff": "off",
    "default_counter": 0
  },
  "output": {
    "text": {
      "values": [
        "Welcome to your cognitive car!  anything I can assist you with?",
        "Hello!  How can I help you with your cognitive car?"
      ],
      "selection_policy": "random"
    }
```

Add Context to the Welcome Node (JSON)

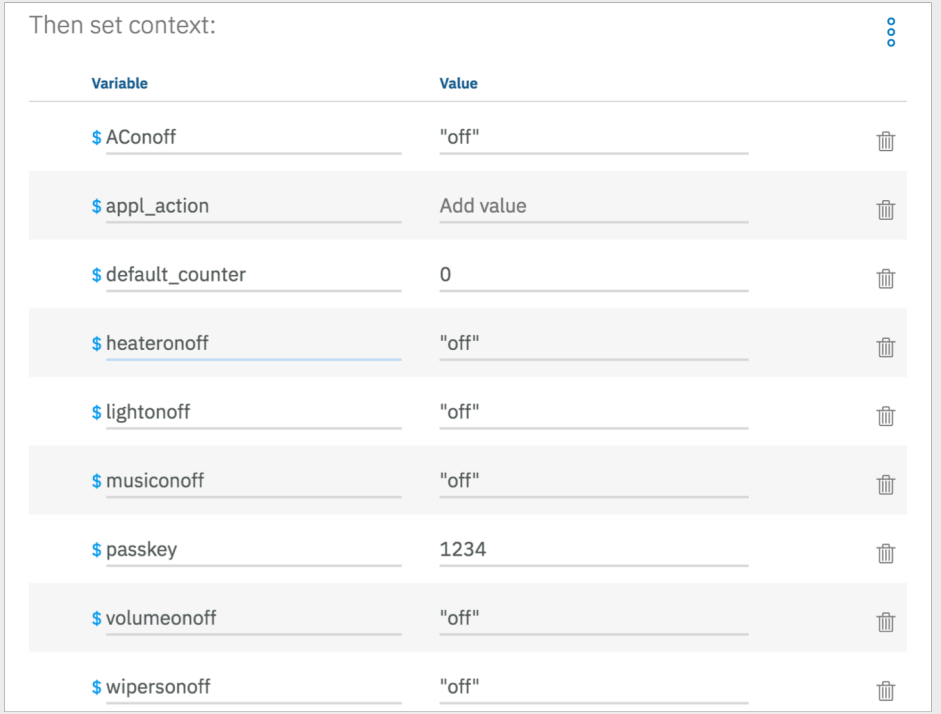
1. Use your favorite text editor to copy the contents of context.txt from your lab files.
2. Select the welcome node.
3. Under Responses, select  to open the JSON editor
4. Under the first { line, paste the snippet at the right to add context.

```
{
  "context": {
    "AConoff": "off",
    "passkey": 1234,
    "lightonoff": "off",
    "musiconoff": "off",
    "appl_action": "",
    "heateronoff": "off",
    "volumeonoff": "off",
    "wipersonoff": "off",
    "default_counter": 0
  },
  "output": {
    "text": {
      "values": [
        "Hello. How can I help you?",
        "Hi, I'm here to help.  What can I do for you?"
```

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.



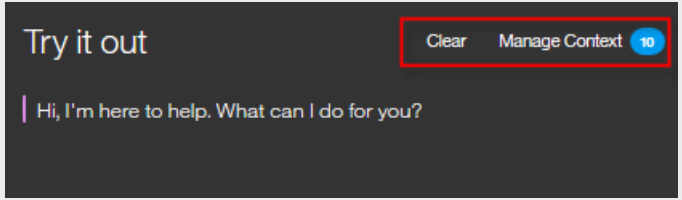
Results – Adding Context

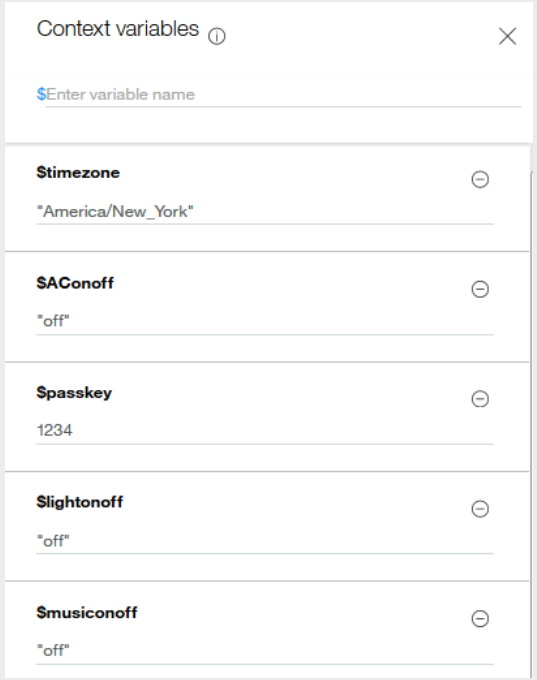
After context is set in a node, it can be referenced in conditions and responses with $variable-name

Try it

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

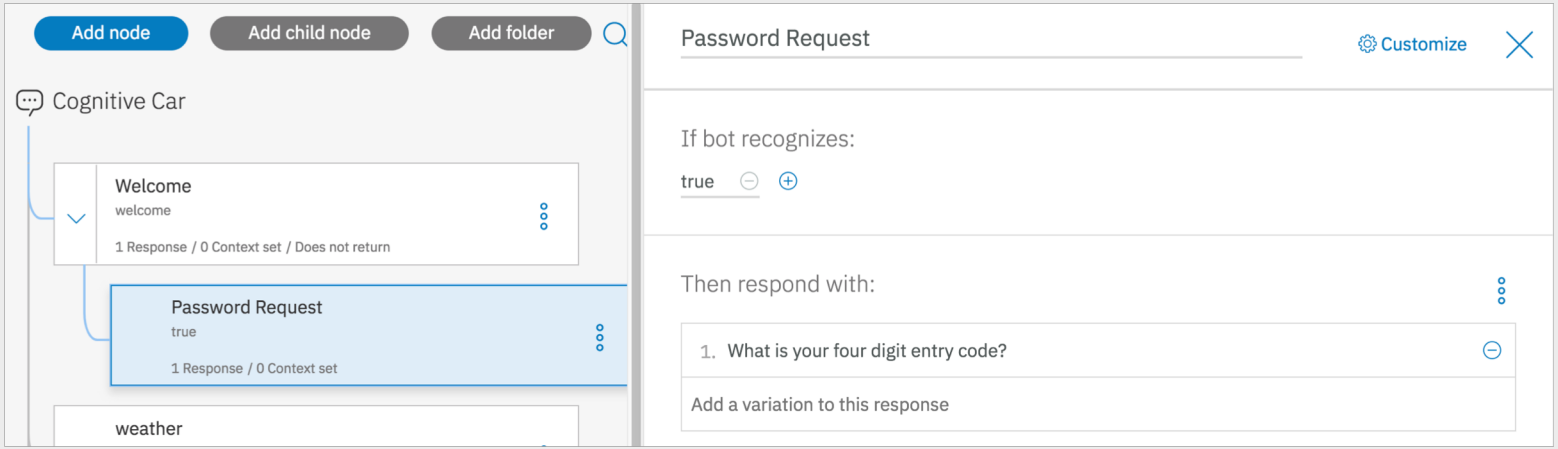
You should see all of your context variables and their current value.





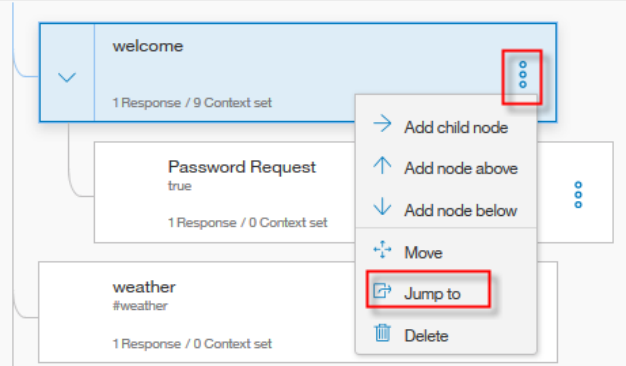
Password Request Node

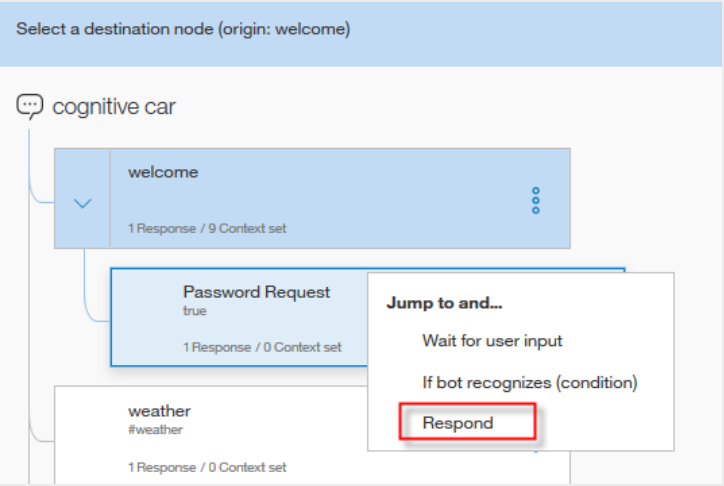
1. Add a child node from welcome node for Password Request.
2. Name the node Password Request
3. Condition on “true” (create new condition).
4. Response = What is your four digit entry code?



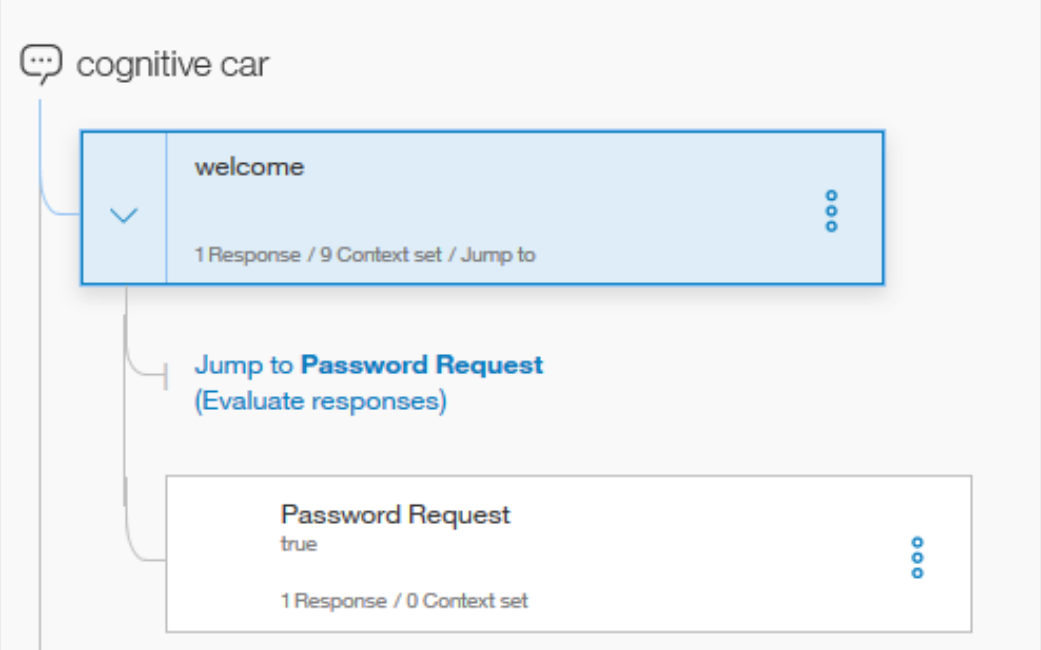
Create Jump to from Welcome to Password Request

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



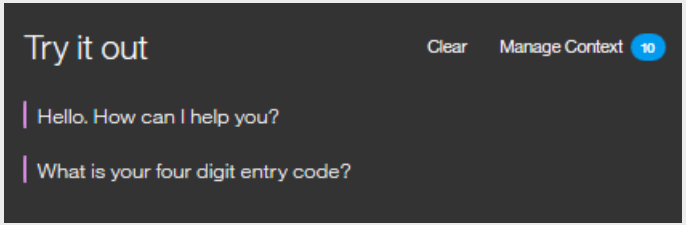


Results - Password Request Node



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

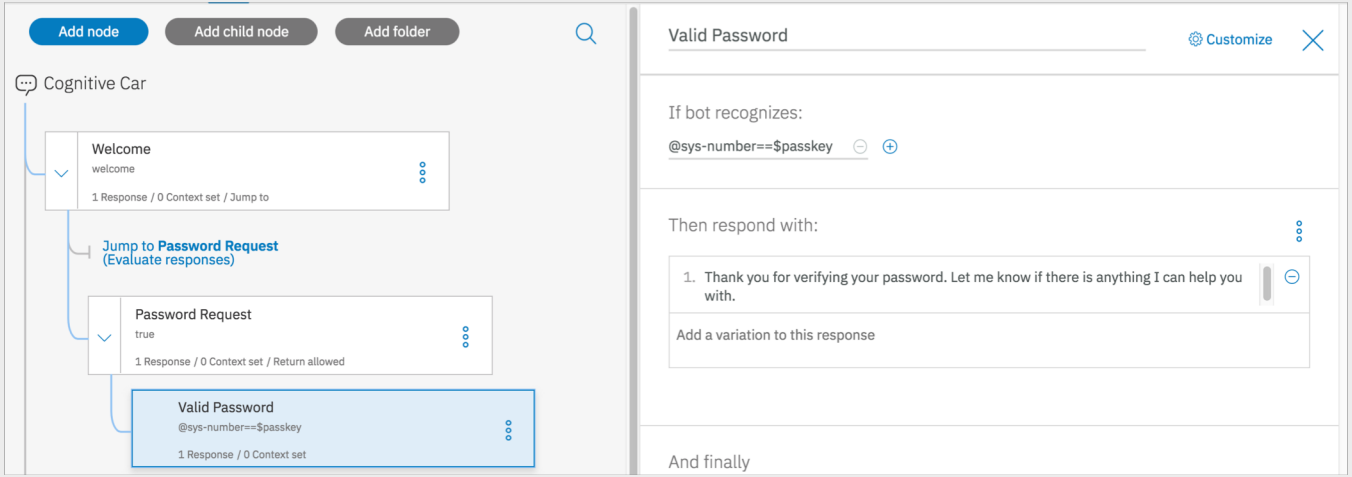
- Create a child of the password request node called Valid Password
- If no child nodes are evaluated as true, dialog jumps back to welcome
- Condition with @sys-number == $passkey
- Evaluates if the entity number was present and matches to context passkey present\
- Create response for valid password

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.

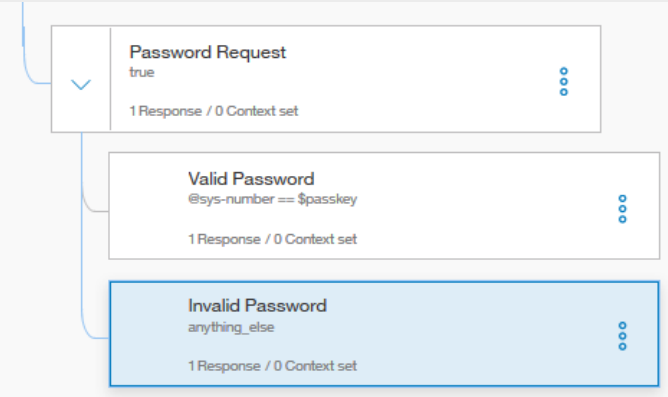
Create Valid Password Node

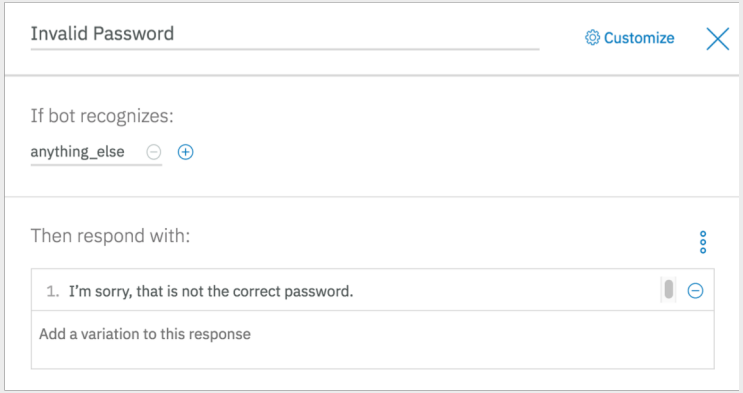
1. Add a child node of the password request node
2. Name the node Valid Password
3. Condition with @sys-number == $passkey
4. Response = Thank you for verifying your password. Let me know if there is anything I can help you with.



Create Invalid Password Node

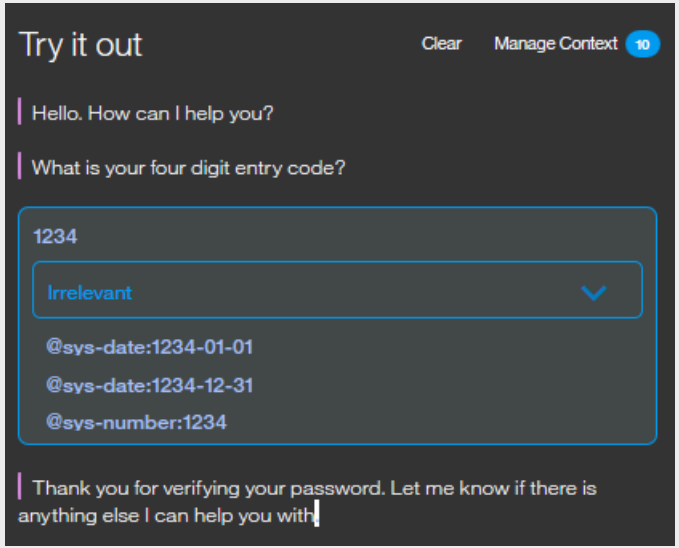
1. Add a second child node of the password request node
2. Name the node Invalid Password.
3. Condition with anything_else.
4. Response = I'm sorry, that is not the correct password.

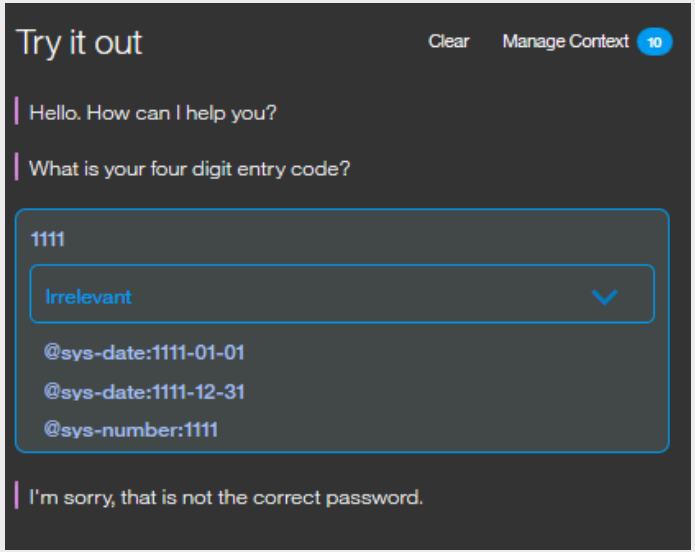




Test Time!

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

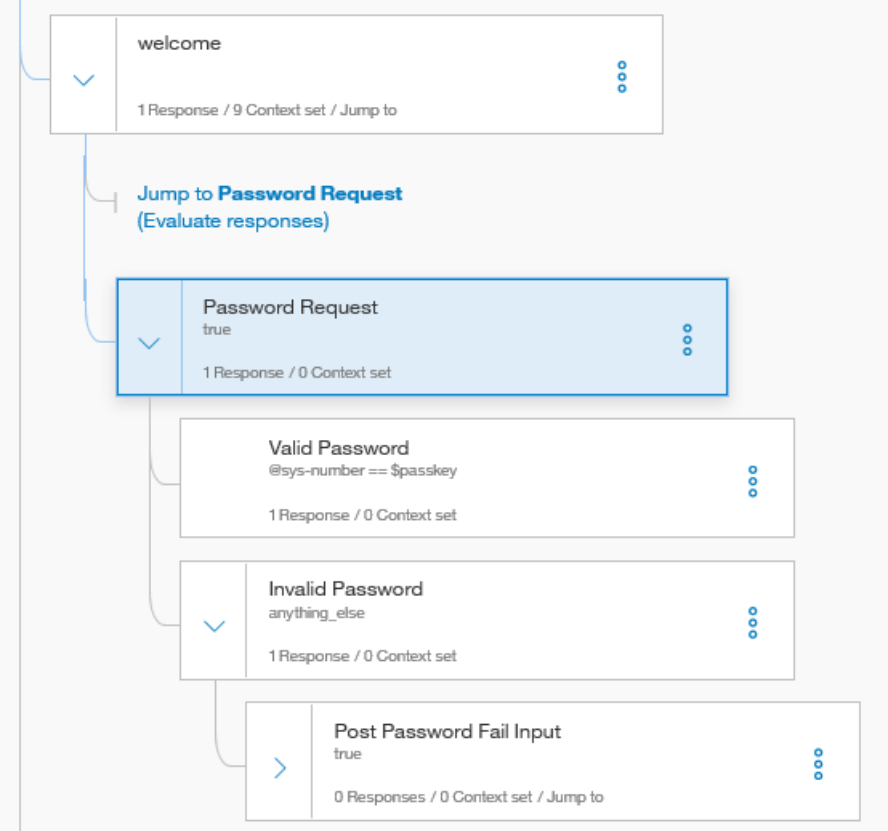
You can add more complex loops utilizing attempt counters, and multiple sequential follow-up questions based on the branching evaluations of inputs.

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.

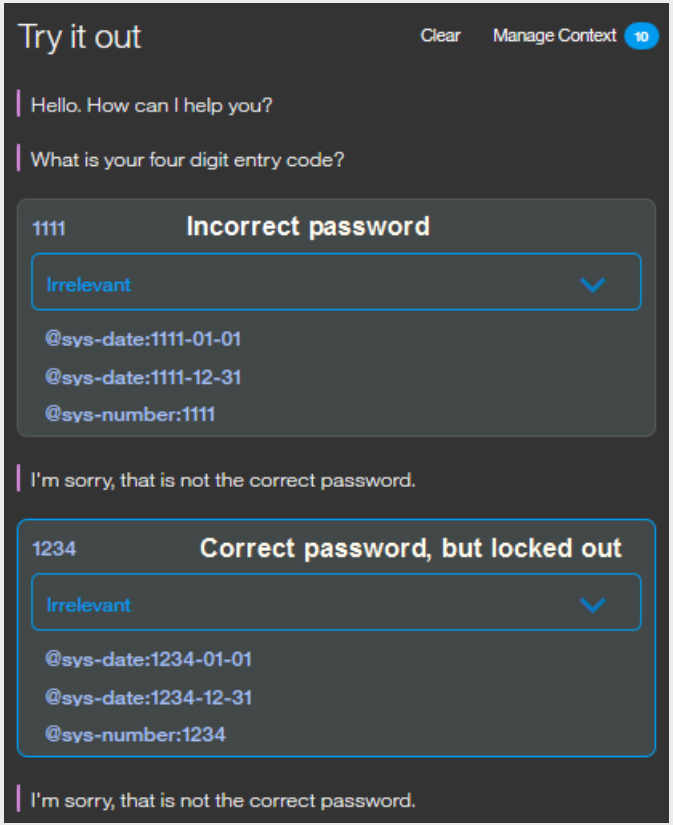
1. Create a child of the Invalid Password node (anything_else)
2. Name the node Post Password Fail Input
3. Condition as *true;* do not add a response
4. Add a JumpTo from this node to the Password Request

Results – Simple loop

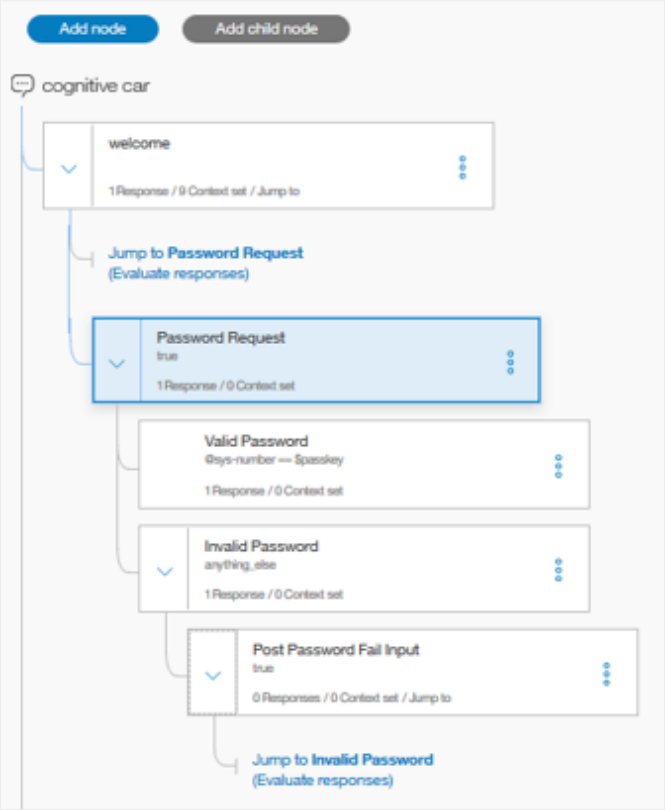


Try it!

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



Results – Multi-turn interaction



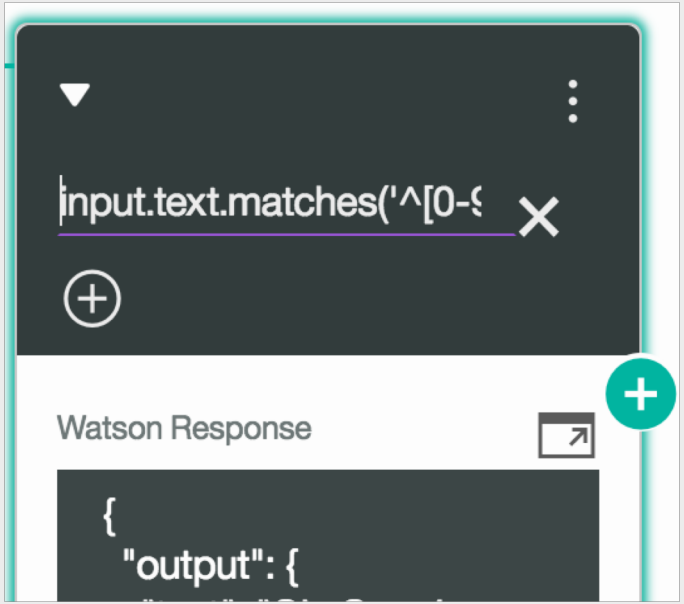
So that was simple – What else can it do?

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:

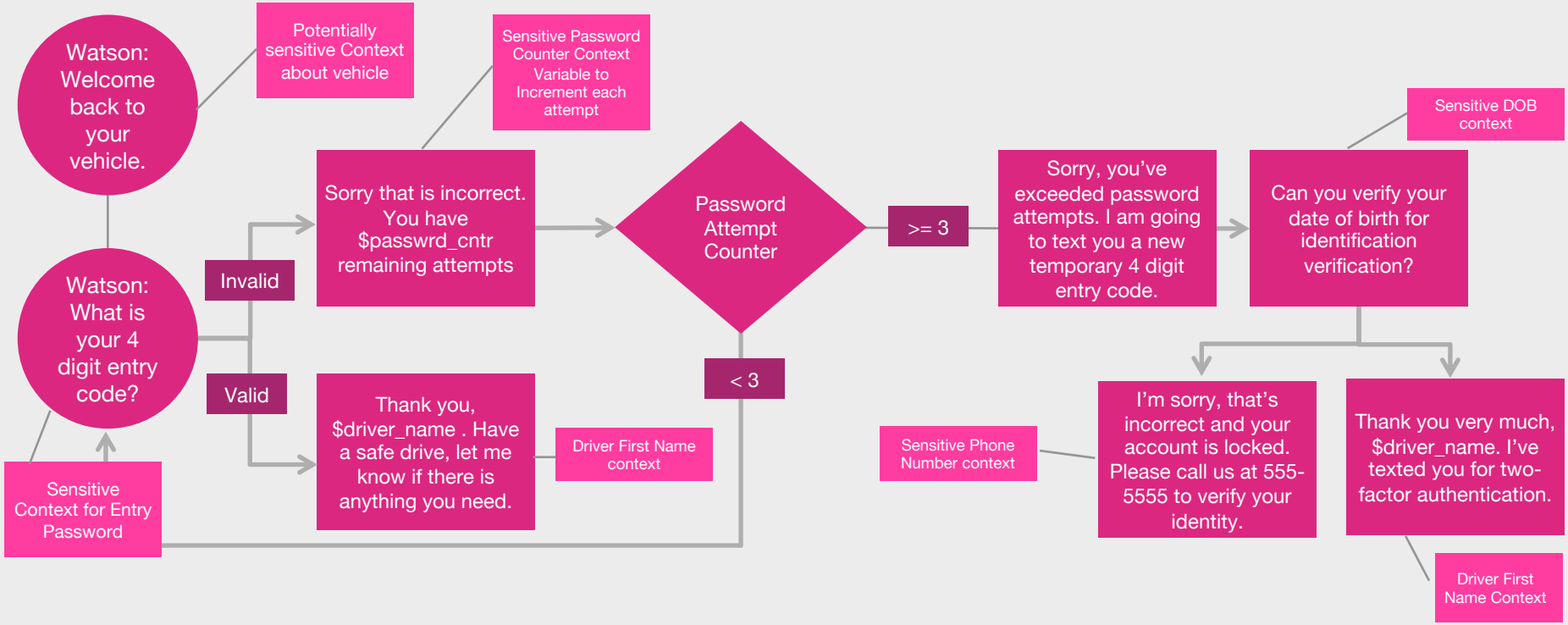
- **Input.text.matches('[0-9]+'), input.text.matches('^[0-9]{4}$')**
- **Input.text.extract(regexp, group_index)**
  - Input.text.extract('[//d]+',0
  - Remember to escape the slashes
- input.text.contains('yes')
- Input.text.equals(String) and input.text.equalsIgnoreCase(String)
- Input.text.length()
- **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



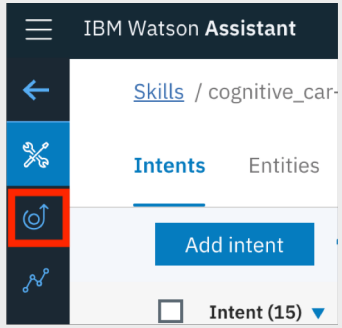
Designing More Complex Multi-turn Interactions

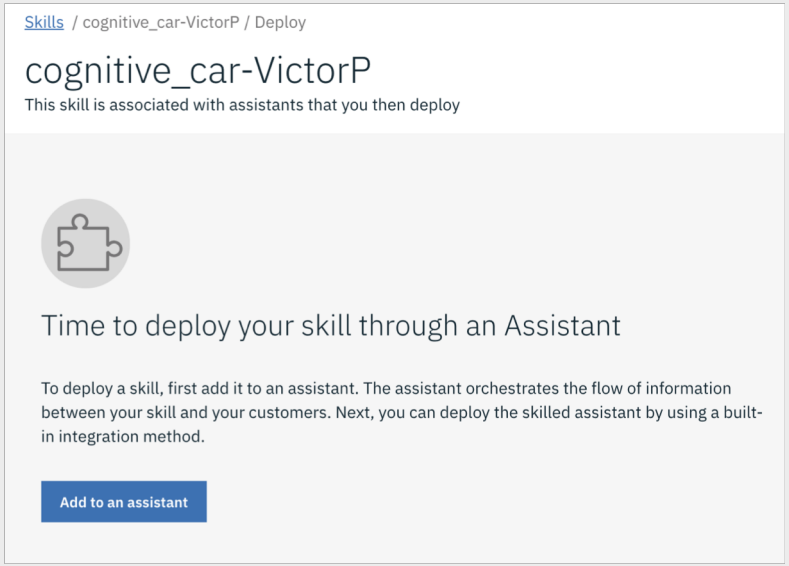


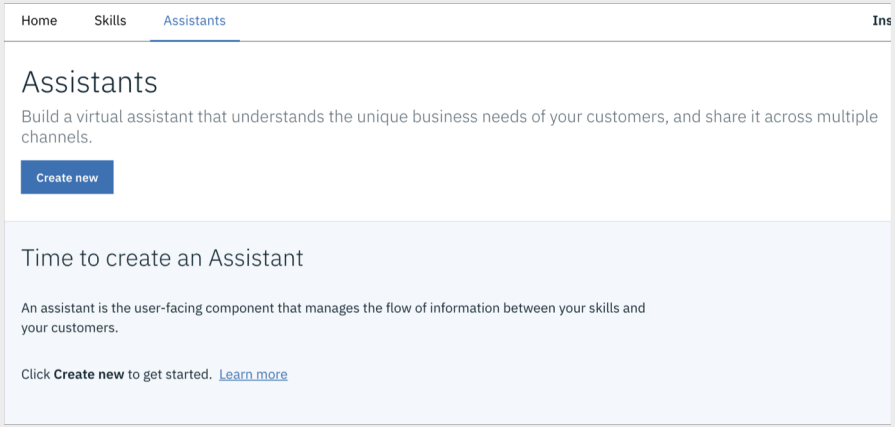
User Interface
Lab

Create the Assistant

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

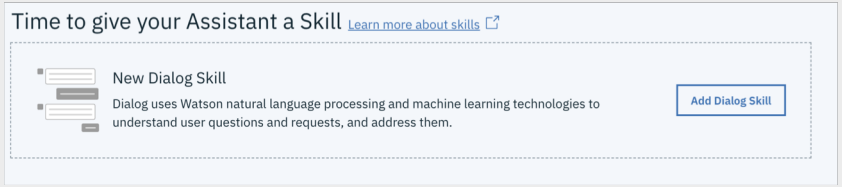




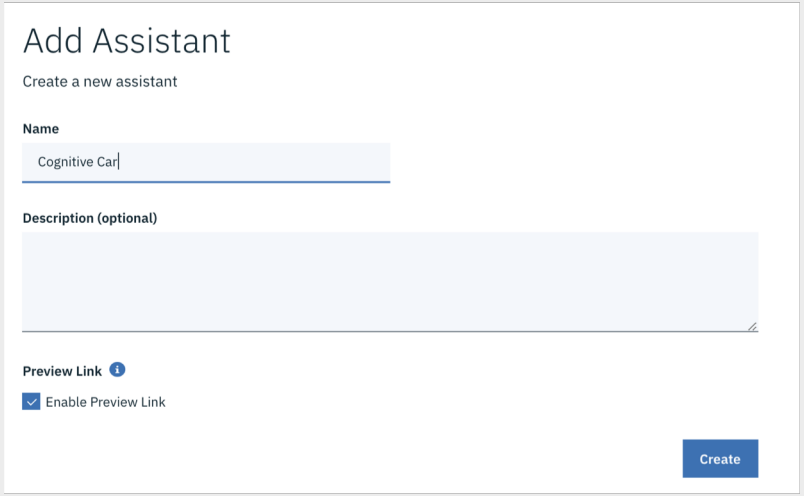


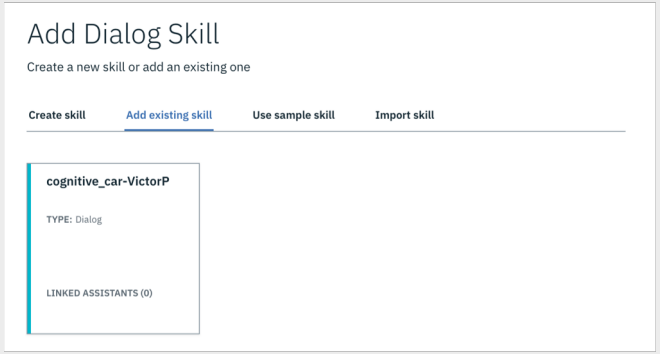
Add the Skill to the Assistant

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



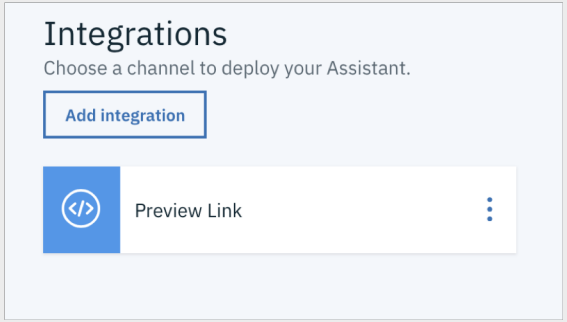
4. Choose “Add existing skill” from the top navigation menu
5. Select the displayed skill

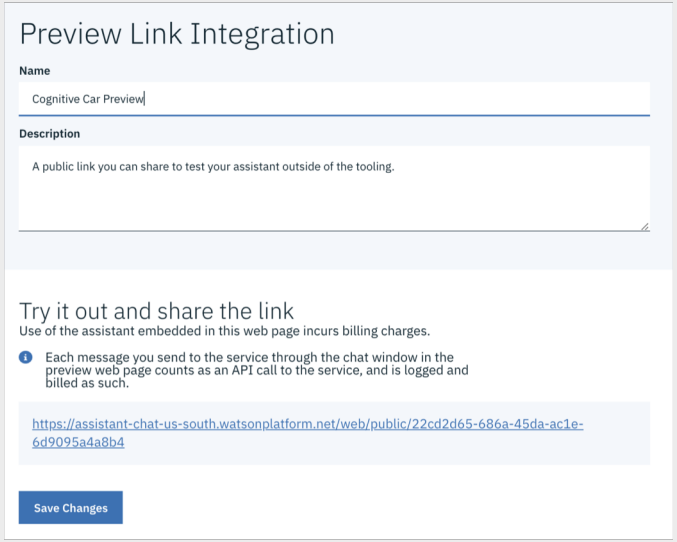




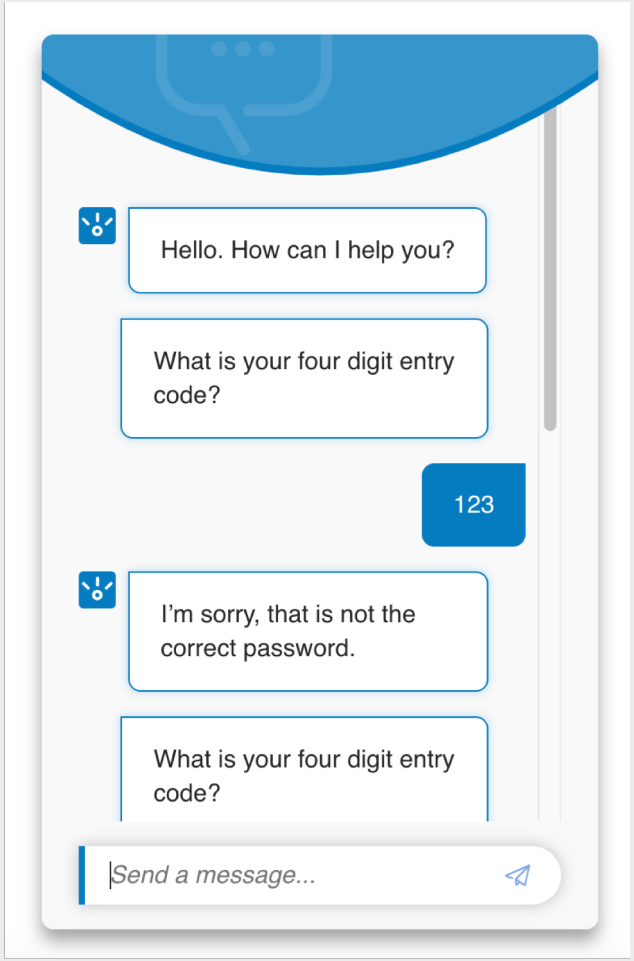
Create a Preview Link

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”





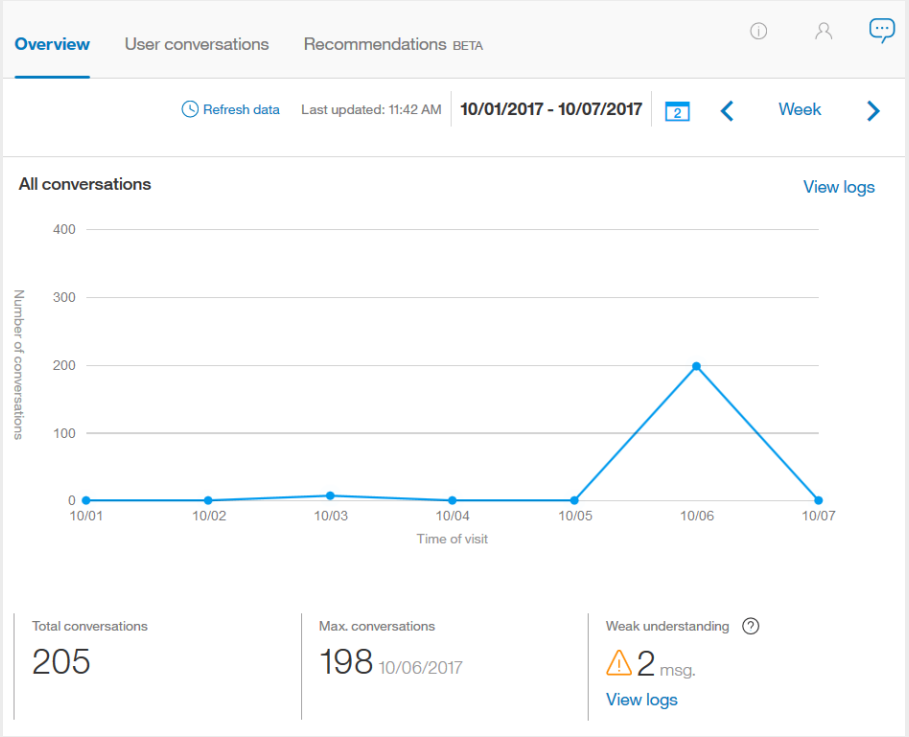
Test the Assistant



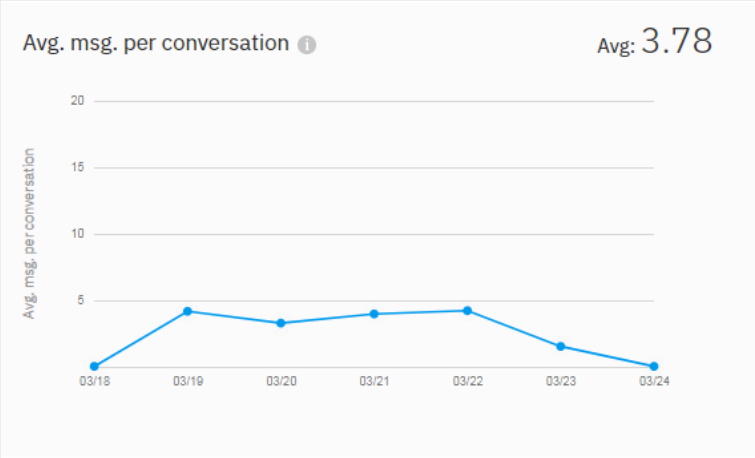
Improve
Lab

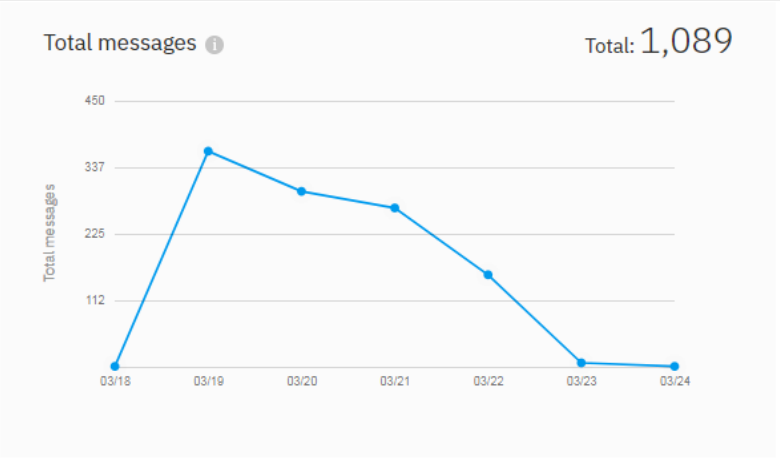
Improve Dashboard

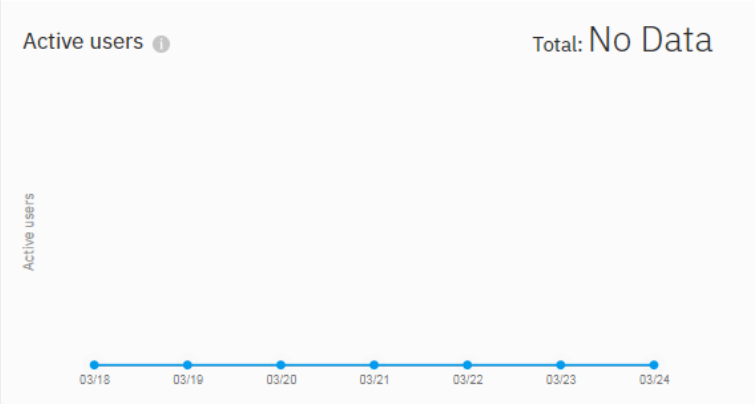
- Tooling to understand your conversational activity at a glance
- Graph of live conversation volumes by day/week/month/year
- Dashboard data providing key metrics on the conversational system including top intents/entities
- Ability to drill into conversation logs directly from the dashboard
- Only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

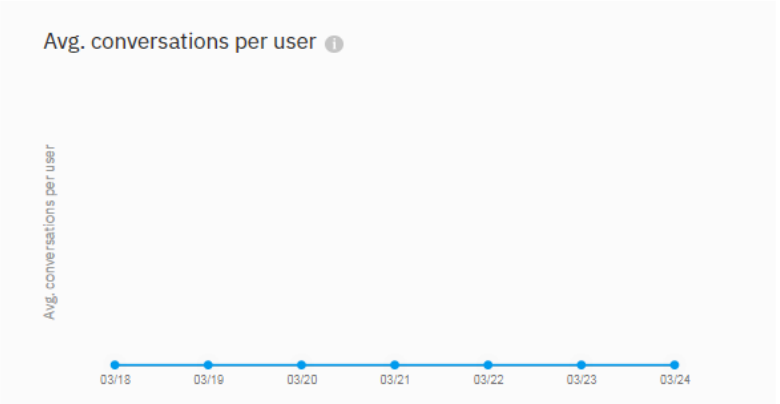


Improve Dashboard - Continued



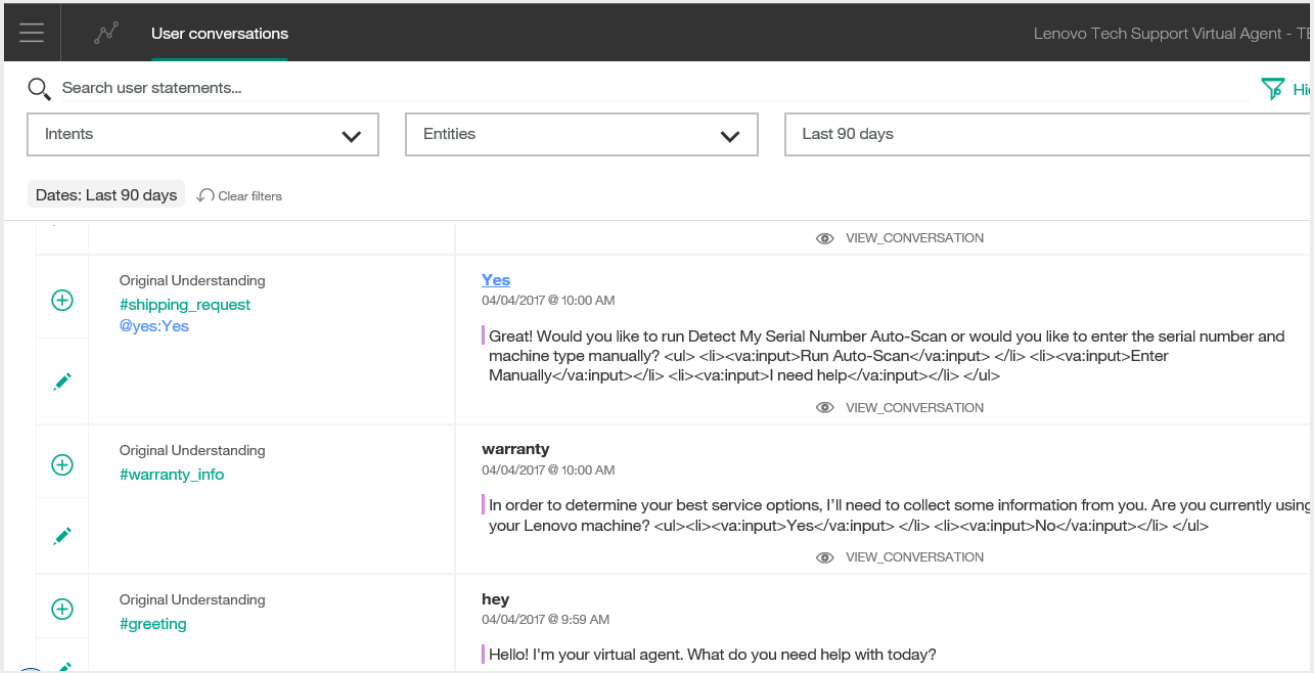






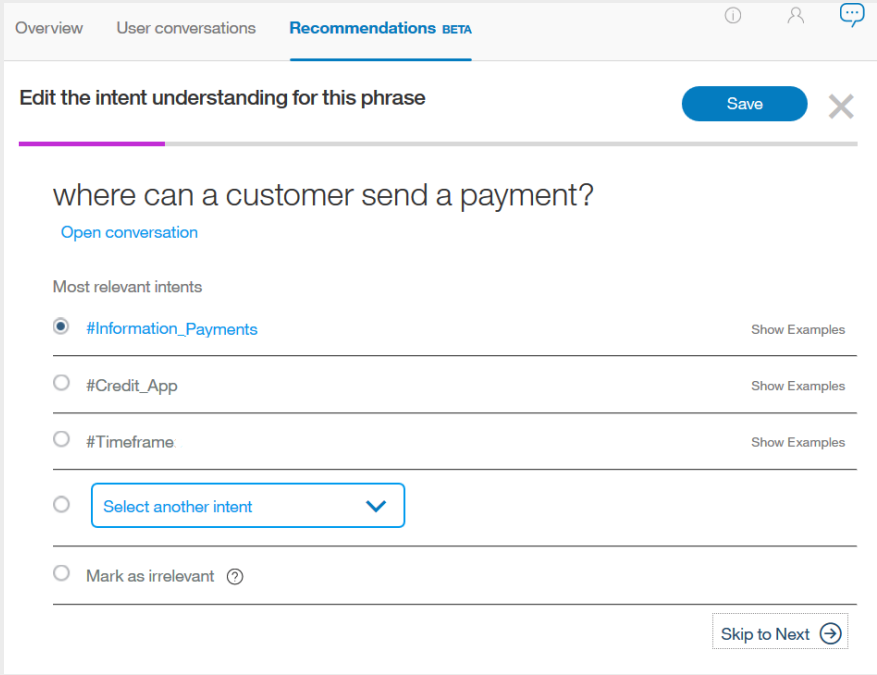
Improve – User Conversations

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.



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.



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."
   }
 }
```