

IBM Watson

Watson Developer Cloud Services

Watson Conversation Deep Dive

IBM



Conversation Overview



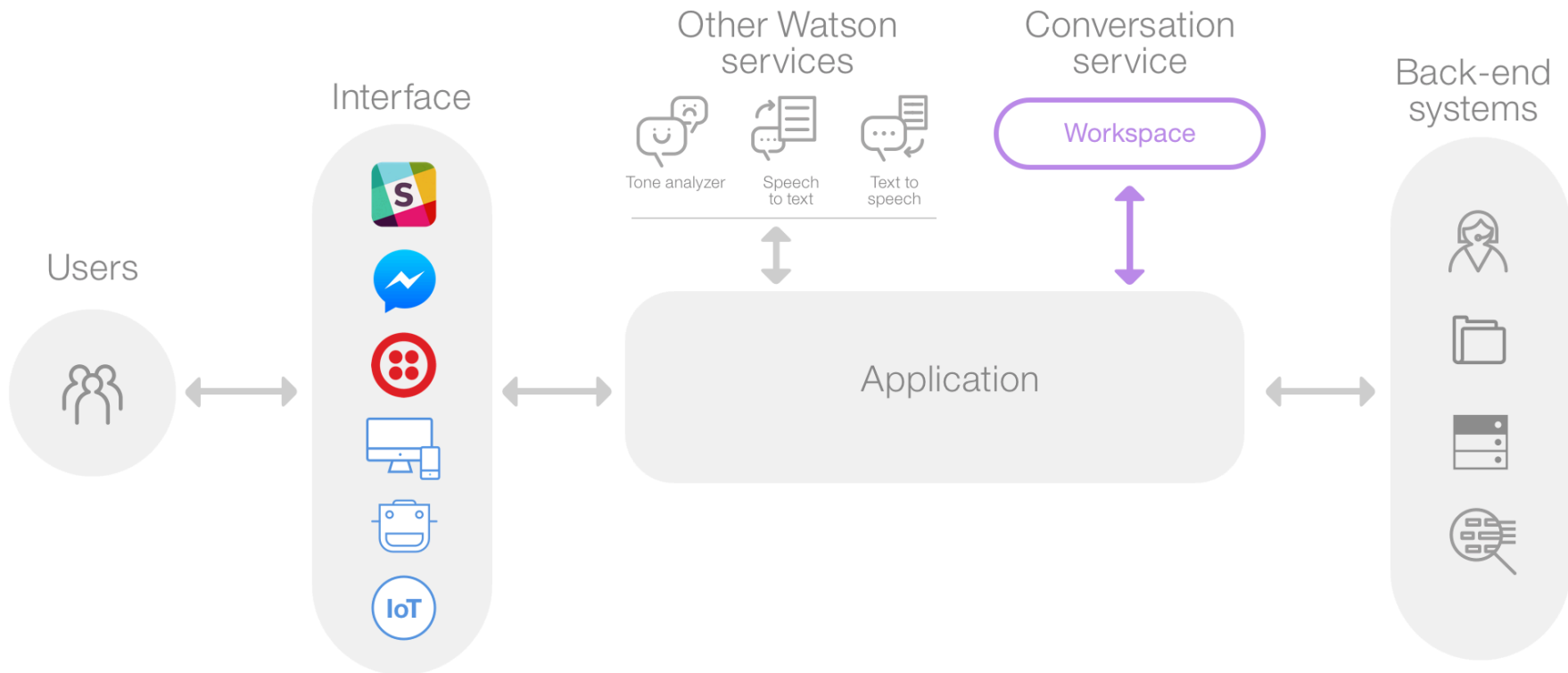
Description

- Enables Developers with Business users to create natural, human-like conversational experiences across all channels (e.g. mobile, messaging, robots, etc.)
- Combines Intents, Entities and Dialog into a seamless experience

Benefits

- Enables customers to self-serve on their terms
- Delivers information and services with a consistent, on-brand and engaging experience
- Reduces costs through deflection of calls to Contact Centers

Watson Conversation



Interpreting Natural Language

“ I’m frustrated, I haven’t
been able to login into your
online billing system... ”

Extract User Intent From Question

“ I’m frustrated, I haven’t been able to login into your online billing system...” ”

Intent Password Reset

Extract Key Information From Question

“ I’m frustrated, I haven’t been able to login into your online billing system...” ”

Intent Password Reset

Entities Online Billing System

Extract Key Information From Question

“ I’m frustrated, I haven’t been able to login into your online billing system...” ”

Intent Password Reset

Entities Online Billing System

Emotional Tone Anger

Add Context Around the Question

“ I’m frustrated, I haven’t been able to login into your online billing system...” ”

Intent Password Reset

Entities Online Billing System

Emotional Tone Anger

Context Bill Smith, 47,
Gold Member, High Value

Context Mobile

Action: Responses come in Different Form

Question

Answer

How do I reset my password?

Dialog

Guide the user through a set of steps

Someone has stolen my credit card.

Deflect

Transfer to human agent

Where is the nearest store?

Map

Application launches map with directions

I need to pay my outstanding invoice.

App Nav.

Bring user to pay bill screen

Can I pay my bills using my credit card?

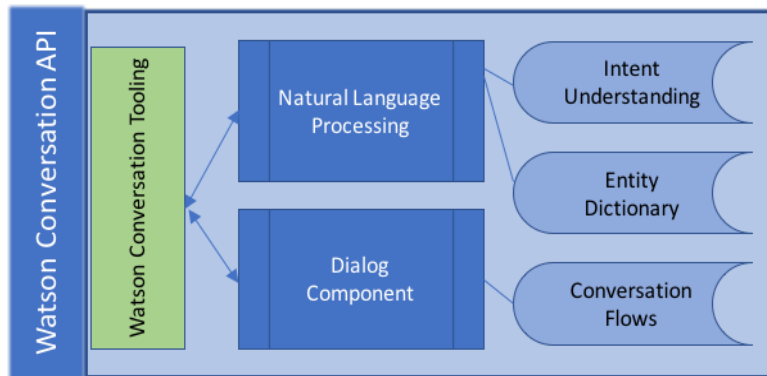
Info. Retrieval

Bring back an answer

Conversation Components

The Building Blocks

- Natural Language Processing
 - Identification/processing for:
 - Intent classification is used to determine the meaning of an utterance from the end user
 - Entity Extraction is used to understand the object(s) of the utterance.
 - Orchestration of modifications, training, active revisions for a workspace
 - New intents, examples or entities are propagated to training automatically without user intervention
- Dialog Interpreter
 - Based on this understanding, dialog is used to take action
 - Providing an answer or engaging the user in a dialogue



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.

19	#capabilities can I manipulate the
40	#goodbyes adieu
35	#greetings aloha
49	#locate_amenity Amenities
#not_specified	
⊕ 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

Intents

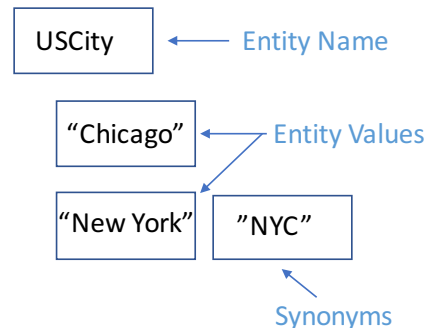
- Identified in tooling via **#intent_name**
 - Allowed: Letters, numbers, hyphen, underscores, dashes and/or dot
 - Prohibited: Spaces, more than 128 characters.
- Intents can be developed as either **core** intents or **full** intents.
 - A **core** intent is a one word intent that refers only to the intent of the user's utterance
 - A **full** intent is a two word intent that allows utterances to be clustered to an intent that refers to both the intent of the user's utterance and the entity that will refine the response

Shorthand syntax	Full syntax in SpEL
#help	intent == 'help'
!#help	intent != 'help'
NOT #help	intent != 'help'
#help OR #!_am_lost	(intent == 'help' intent == '!_am_lost')

```
Tell me the current weather conditions.,weather_conditions
Is it raining?,weather_conditions
What's the temperature?,weather_conditions
Where is your nearest location?,find_location
Do you have a store in Raleigh?,find_location
```

Entities

- Entities are objects user wants to take action on and are inputs that 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
- Once the intent is determined, **an answer can be refined by entities**
- Watson's way of handling significant parts of an input that should be used to slightly alter the way it responds to the intent
- Consists of a **name**, a set of **values**, and set of **synonyms** per value.
- Currently entities are defined as a **closed** list.
 - No generalization, fuzzy matching, etc.
 - You must include all possible variations and examples
 - You can also easily extend your app to use a service like Alchemy for entity extraction

**@amenity**

gas, place, restaurant, restroom

@appliance

ac, fan, heater, lights, music, volume, wipers

@cuisine

burgers, pasta, seafood, tacos

@cuisine_bad

african, american, argentine, asian, austrian, basque, belgian, breton, european, french, galician, german, gluten free, greek, halal, hawaiian, middle eastern, moroccan, paleo, persian, peruvian, pescatarian, portuguese, tuscan, vegan, vegetarian, vietnamese

@genre

classical, jazz, pop, rock

@genre_bad

Alternative, Blues, Childrens, Comedy, Country, Dance, Electronic,

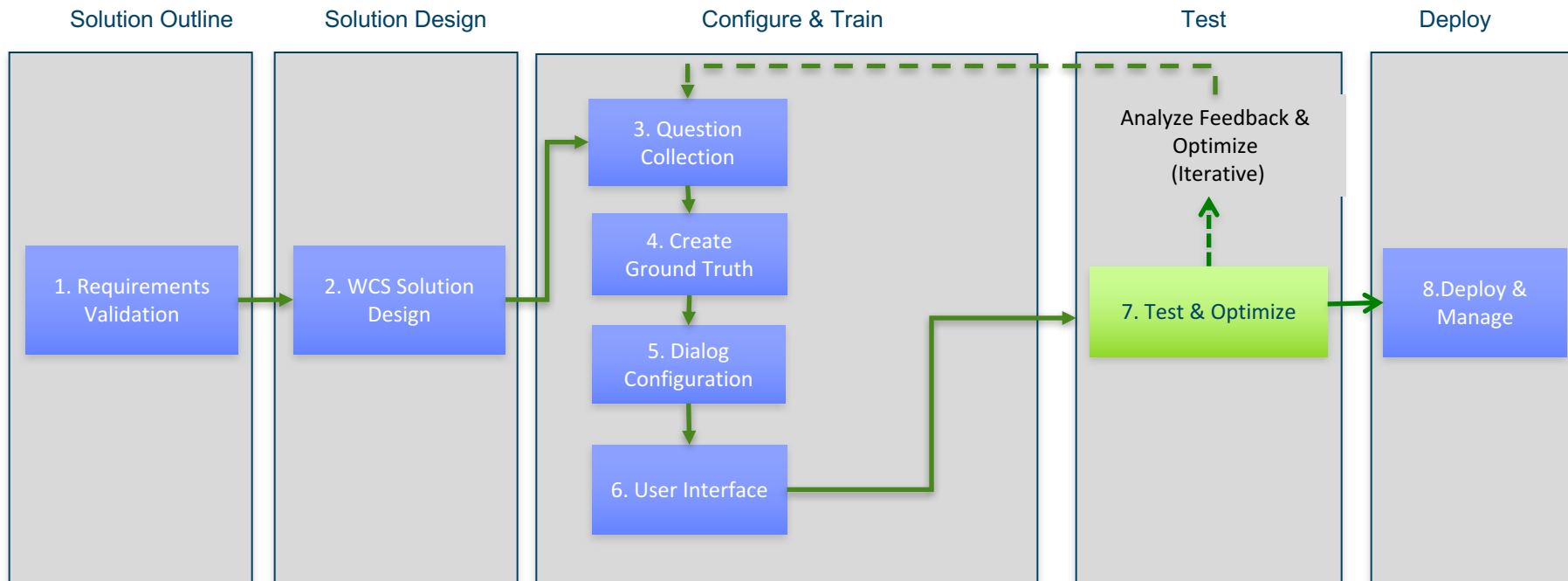
Entities

- 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>

Shorthand syntax	Full syntax in SpEL
@year	entities['year']?.value
@year == 2016	entities['year']?.value == 2016
@year != 2016	entities['year']?.value != 2016
@city == 'Boston'	<ul style="list-style-type: none">entities['city']?.value == 'Boston'entities['city'] == 'Boston'
@city:Boston	<ul style="list-style-type: none">entities['city']?.value == 'Boston'entities['city']?.contains('Boston')
@city:(New York)	<ul style="list-style-type: none">entities['city']?.value == 'New York'entities['city']?.contains('New York')

```
weekday,Monday,Mon
weekday,Tuesday,Tue,Tues
weekday,Wednesday,Wed
weekday,Thursday,Thur,Thurs
weekday,Friday,Fri
weekday,Saturday,Sat
weekday,Sunday,Sun
month,January,Jan
month,February,Feb
month,March,Mar
month,April,Apr
month,May
```

WCS Deployment Methodology



Best Practices for Identifying Intents and Entities

Intents

Exercise should be completed by SMEs

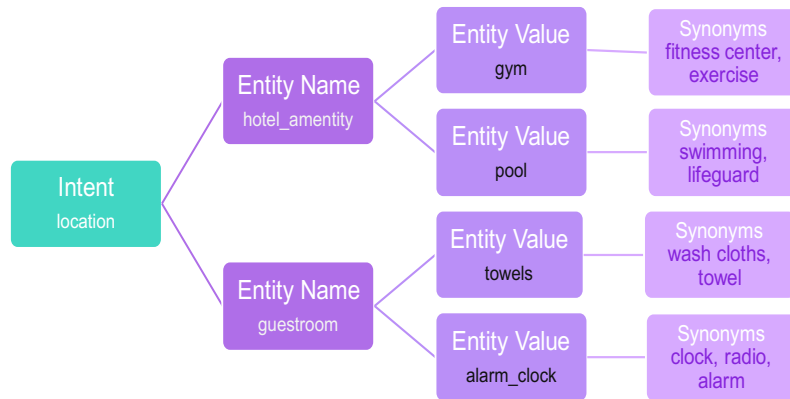
Define intents

Group questions by similar intents

Review questions to ensure it is grouped by the most similar intent

Consistency is the key to confidence!

Entities



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 WCS 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.

Question Collection

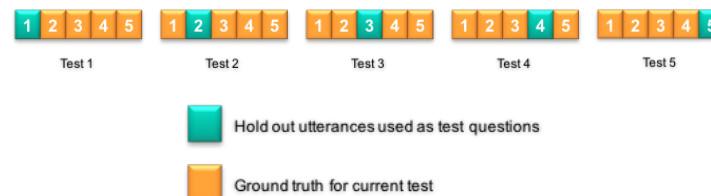
Question collection is the process of collecting utterances that represent the user scenario

- 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
- Use the questions to train and test WCS and divide into groups:
 - **Training set:** This is the full set of utterances used to create the Ground Truth (GT) for production.
 - **Hold out tests:** For testing purposes you will divide the training set into 5 randomly selected partitions of equal size. Multiple tests will be run each using 1/5 of the questions as untrained questions, and 4/5 as Ground Truth.
- 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.

Question Sets Defined

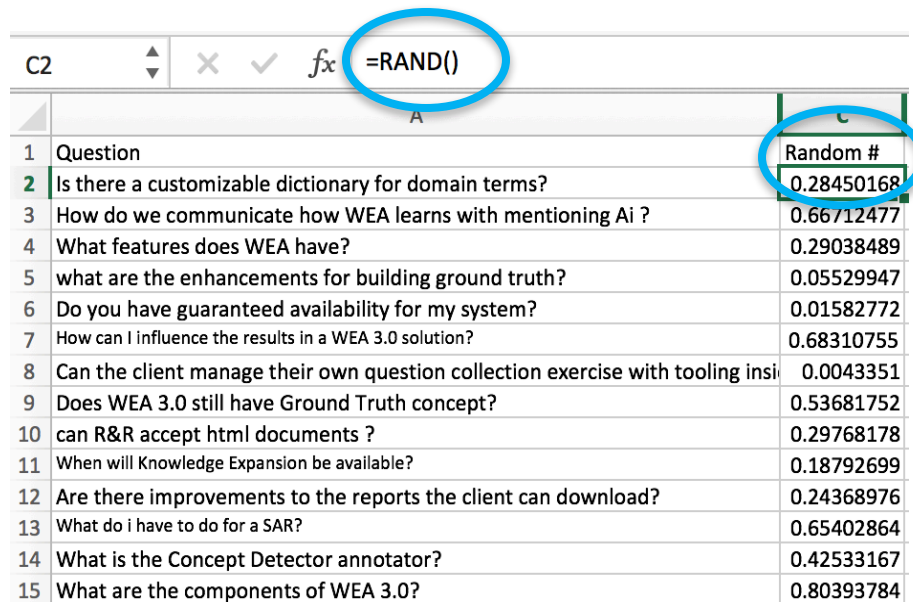
- In order to train and test WCS during a domain adaptation, you work with different question sets that are **randomly obtained during each experiment**.
- Collect as many end-user examples as possible, with 2000 being the minimum for a Production deployment, and 400 being the minimum for a Pilot deployment.
- All examples will be used to train production Ground Truth.
- Run a set of 5 hold out tests each time you run an experiment. Each test uses one partition as the question source, and the other 4 partitions as Ground Truth.
- Keep the same partition markers to obtain a baseline of performance and measure improvement against it over time.

Collected End-User Examples				
Production minimum = 2000, Pilot minimum = 400				
Training Set – Production Ground Truth				
2000+ examples Or 100% of collected questions				
Partition 1	Partition 2	Partition 3	Partition 4	Partition 5
400 examples Or 20% of questions	400 examples Or 20% of questions	400 examples Or 20% of questions	400 examples Or 20% of questions	400 examples Or 20% of questions



Question Management – Obtaining test sets

- Once you have collected end user representative questions, the next step is to create question sets for training and blind testing
- It is critical for training and blind test sets to be obtained *randomly*.
- Export all collected questions into .csv file
- Use the **rand()** function in Excel to obtain **random** question sets
- Keep track of each end-user example and its assigned set:
 - Assign an ID to each question for tracking
 - After all collected questions have been classified with intents and entities, then separate your question lists into train and blind sets.
- Never use an utterance for more than one purpose (test, training)
 - If possible, do not use the entire set of collected questions
 - Obtain additional end-user examples if necessary



	Question	Random #
1	Question	
2	Is there a customizable dictionary for domain terms?	0.28450168
3	How do we communicate how WEA learns with mentioning Ai ?	0.66712477
4	What features does WEA have?	0.29038489
5	what are the enhancements for building ground truth?	0.05529947
6	Do you have guaranteed availability for my system?	0.01582772
7	How can I influence the results in a WEA 3.0 solution?	0.68310755
8	Can the client manage their own question collection exercise with tooling insi	0.0043351
9	Does WEA 3.0 still have Ground Truth concept?	0.53681752
10	can R&R accept html documents ?	0.29768178
11	When will Knowledge Expansion be available?	0.18792699
12	Are there improvements to the reports the client can download?	0.24368976
13	What do i have to do for a SAR?	0.65402864
14	What is the Concept Detector annotator?	0.42533167
15	What are the components of WEA 3.0?	0.80393784

Organizing Intents

To create an import file (preferred method) for intents and entities, you can start with a spreadsheet to organize your thoughts.

1. When you are ready to upload the file, save it as a CSV file.
2. Open CSV file in a text editor to make sure that unwanted commas and symbols have been removed.
3. Save as UTF-8.
4. Organize questions after intents and entities have been determined in your Ground Truth file in a pivot table
5. Can minimize and maximize the high level intent sections of the pivot table to review all questions that have been categorized with each intent

Spreadsheet setup for intent import

1	What are the hotel Restaurant hours of operation?	place_hours
2	What local restaurants have live music?	place_recommendation
3	How can I arrange transportation?	place_procedure
4	Can I walk to that restaurant or do I need to take a cab?	place_location
5	What time can I grab dinner?	place_hours
6	Which floor is the gym/pool on?	place_location
7	I want to eat at an Italian restaurant. What are the best	place_location
8	Can I bring food to the pool?	hotel_info
9	Whereabouts is the gym?	place_location
10	Is there any where I can get food to go around here?	place_recommendation

3	Row Labels	Count of Text Questions
4	hotel_info	12
5	place_hours	16
6	place_location	27
7	place_procedure	10
8	Hey can I get some new towels up to my room	1
9	How can I arrange transportation?	1
10	How can I book a cab?	1
11	How can I book a taxi?	2
12	How do I adjust the heat/AC in the guestroom?	1
13	How do I arrange a shuttle or a taxi to the airport?	1
14	How do I arrange a shuttle or taxi to the airport?	1
15	How do I book a large reservation in the restaurant?	1
16	How do I set the clock alarm in the guestroom?	1
17	place_recommendation	32
18	Can you make a dinner recommendation - we are thinking Italian.	1
19	Can you recommend a restaurant?	1
20	Can you recommend a good Italian restaurant?	1
21	Can you recommend a good restaurant for dinner?	1
22	Can you recommend a Japanese restaurant?	1
23	Can you recommend a restaurant for dinner? Perhaps chinese food, or italia	1
24	can you recommend a restaurant nearby? i'm looking for "x" cuisine and "x"	1
25	Can you recommend a restaurant that is not a chain for Italian cuisine?	1
26	Can you recommend some great upscale restaurants outside of the hotel?	1
27	Could you give me a recommendation for a good restaurant with Mexican fc	1
28	Good sitdown reaturants in the area.	1
29	Hello I'm starving I'm looking for a place to eat	1
30	Hey I was looking for someplace interesting to go eat, do you know anyplace	1

Guidance – Intents

- 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
 - 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

Planning Responses (Using Core Intents and Entities)

A **response** is what the Conversation 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 Conversation 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
local_recommend			No response for this high level intent
	places		For a list of restaurants near the hotel, please see a concierge at the front desk.
hotel_procedure			No response for this high level intent
	transportation		You can arrange transportation by contacting the front desk.
	guestroom	heat/ac	For instructions how to adjust your room's temperature, please call our front desk.
	guestroom	towels	If you need more towels or wash cloths in your room, please call our front desk.
	guestroom	alarm clock	For instructions to program your room's alarm clock radio, please call our front desk.
hotel_info			No response for this high level intent
	hotel_amenity	pool	For more information about the pool, please see our website.
	hotel_amenity	gym	For more information about the fitness center, please see our website.
hotel_locations			No response for this high level intent
	hotel_amenity	pool	The pool is on the second floor.
	hotel_amenity	gym	The gym is in the front lobby.
	hotel_amenity	hotel restaurant	The hotel restaurant is located on the ground floor of the hotel.
	hotel_amenity	sauna	This hotel location does not have a sauna.
hotel_hours			No response for this high level intent
	hotel_amenity	gym	The hotel gym is open from 5am to 12pm.
	hotel_amenity	pool	The hotel pool is open from 5am to 8pm.
	hotel_amenity	hotel restaurant	The hotel restaurant is open from 6am to 9pm.

Entity import file setup

Spreadsheet setup for entity import

1	places	breakfast					
2	places	fast_food					
3	places	food_to_go	take away	carry out			
4	places	lunch					
5	places	sushi					
6	places	chinese	chinese restaurant				
7	places	coffee_shop	coffee place	coffee shop	Dunkin	Dunking	Starbucks
8	places	dinner	supper				
9	places	italian_restaurant	italian cuisine	italian food	italian place	italian restaurant	
10	places	japanese	japanese food	japanese restaurant			
11	places	pizza	pizza restaurant				
12	places	sitdown	sit down	eat in			
13	hotel_amenity	gym	bike	elliptical	fitness center	lift weights	weights

Text editor setup for entity import

entity_name1, entity_value1, synonym1, synonym2, synonym3, synonym4

entity_name1, entity_value2, synonym1, synonym2

entity_name2, entity_value1, synonym1

entity_name2, entity_value2, synonym1, synonym2, synonym3, synonym4

Completed Ground Truth

- **Question Collection** – create Training and Blind/Test set
 - WCS_QuestionCollection.xlsx
 - Assign IDs and use RAND() function to separate questions
- **Ground Truth Creation**
 - WCS_GroundTruth.xlsx
 - Create Analyze Worksheet – pivot table based on intents
 - Cluster questions
 - Import entities

B6 What is the length of the Pool?

	A	B	C	D	E	F	G
1	ID	Text Questions	High Level Intent	Entity Value:Synonym	Entity Type	Full Intent	Intent
2	1	Do you have an elliptical?	hotel_info	gym:elliptical	hotel_amenity	hotel_info_gym	hotel_info
3	2	Do you have a bike?	hotel_info	gym:bike	hotel_amenity	hotel_info_gym	hotel_info
4	3	Do you have weights?	hotel_info	gym:weights	hotel_amenity	hotel_info_gym	hotel_info
5	4	Do you have water in your gym?	hotel_info	gym	hotel_amenity	hotel_info_gym	hotel_info
6	5	What is the length of the Pool?	hotel_info	(none)	(none)	hotel_info_pool	hotel_info_pool
7	6	How deep is the pool?	hotel_info	(none)	(none)	hotel_info_pool	hotel_info_pool
8	7	What size is the pool?	hotel_info	(none)	(none)	hotel_info_pool	hotel_info_pool
9	8	Is the pool heated?	hotel_info	(none)	(none)	hotel_info_pool	hotel_info_pool
10	9	Do you have an outdoor/ indoor pool?	hotel_info	(none)	(none)	hotel_info_pool	hotel_info_pool
11	10	Is there a lifeguard?	hotel_info	(none)	(none)	hotel_info_pool	hotel_info_pool
12	11	Can I bring food to the pool?	hotel_info	(none)	(none)	hotel_info_pool	hotel_info_pool
13	12	Can I order food at the pool?	hotel_info	(none)	(none)	hotel_info_pool	hotel_info_pool
14	13	When is the pool open?	place_hours	pool	hotel_amenity	place_hours_pool	place_hours
15	14	hours fo pool	place_hours	pool	hotel_amenity	place_hours_pool	place_hours
16	15	What are the pool and Fitness Center hours ?	place_hours	gym:fitness center	hotel_amenity	place_hours_gym	place_hours
17	16	What are the hours of the gym?	place_hours	gym	hotel_amenity	place_hours_gym	place_hours
18	17	When is the gym open?	place_hours	gym	hotel_amenity	place_hours_gym	place_hours
19	18	When does the gym close?	place_hours	gym	hotel_amenity	place_hours_gym	place_hours
20	19	What time does the lounge/restaurant open	place_hours	(none)	(none)	place_hours_restaurant	place_hours_restaurant
21	20	When are you serving dinner?	place_hours	(none)	(none)	place_hours_restaurant	place_hours_restaurant
22	21	What's the lunch time?	place_hours	(none)	(none)	place_hours_restaurant	place_hours_restaurant
23	22	What time does the restaurant close?	place_hours	(none)	(none)	place_hours_restaurant	place_hours_restaurant
24	23	What are the hotel Restaurant hours of operation?	place_hours	(none)	(none)	place_hours_restaurant	place_hours_restaurant
25	24	What time does the food place open?	place_hours	(none)	(none)	place_hours_restaurant	place_hours_restaurant
26	25	What time can I grab dinner?	place_hours	(none)	(none)	place_hours_restaurant	place_hours_restaurant
27	26	What are the hours of your restaurant?	place_hours	(none)	(none)	place_hours_restaurant	place_hours_restaurant
28	27	what's open right now	place_hours	(none)	(none)	place_hours_restaurant	place_hours_restaurant
29	28	What are your hours of the restaurant?	place_hours	(none)	(none)	place_hours_restaurant	place_hours_restaurant
30	29	any good time to go swimming	place_location	pool:swimming	hotel_amenity	place_hours_pool	place_location

◀ ▶ Analysis v1 Train Set v1 IntentImport v1 Test Set 1 v1 Test Set 2 v1 Test Set 3 v1 Responses v1 Snapshot v2 Analysis v2 Train Set v2 IntentImport v2 Test Set 1 v2 Test Set 2 v2 Test Set 3 v2 Responses v2 +

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

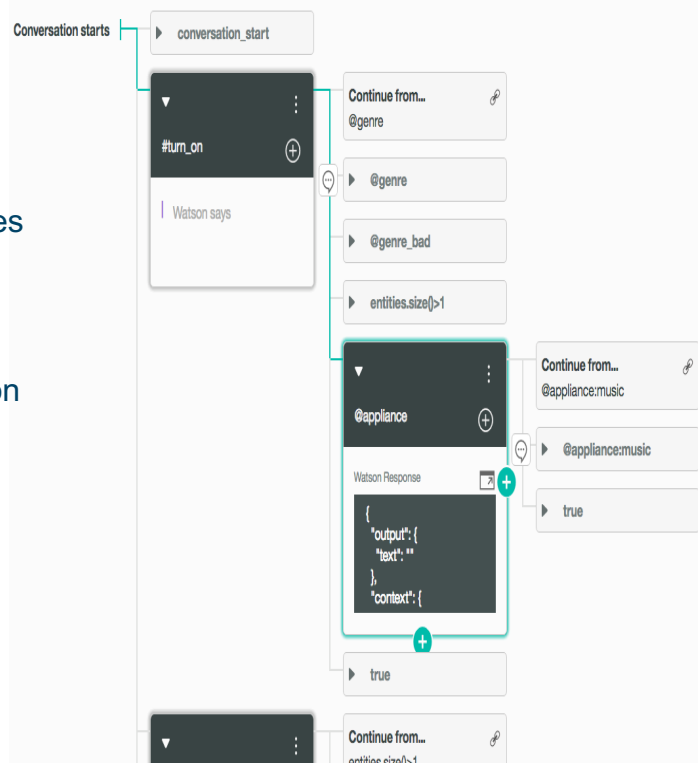
@sys-currency.literal
@sys-currency.numeric_value
@sys-currency.unit –
Currenty unit (Euro, USD, etc)

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

@sys-number.literal –
String text representation
@sys-number.numeric_value –
Number representation

Dialog

- The model of your user engagement / interaction (responses and prompts).
 - Made up of dialog nodes (container objects) chained together.
 - Tree will have parent nodes, children nodes, sibling nodes, leaf nodes
 - The condition is the portion of the dialog node that determines whether the node is used in the conversation.
- Input (Conditions) - determines whether that node is used in the conversation
 - Matching rules against entities, intents, variables
 - Valid expressions in conditions are written in the Spring Expression (SpEL) language
- Output (Response):
 - Can be what the system should say back to the user (answer) or request for data (prompt) or echo some part of the user input (@intent_name or \$context_variable_name)
 - Can use randomized output in dialog nodes using selection_policy
 - Random – system randomly selects output text from the values array
 - Sequential – run through each item in values array



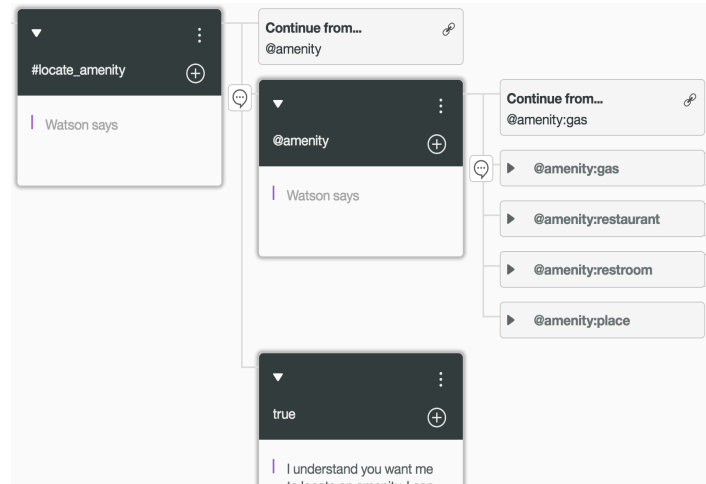
```
{"output": {"text":{"values":["Hello.","Hi."],"selection_policy":"random"}}}
```

Dialog (cont.)

- Order Matters
 - Dialog nodes evaluated in order they appear in UI. **Tip:** Place general responses at the bottom, specific near top
- Special Node conditions:
 - conversation_start - Triggers anytime a conversation was initiated.
 - anything_else - End of the dialog to be processed when the user input does not match any other nodes.
 - True – Execute no matter what. Used at the end of a list of nodes or responses to catch any responses that did not match any of the previous conditions
 - False – Does not get "normally" executed. Used at the top of a branch that is under development, to prevent it from being used
- "Jump_To" can be used to alter default flow, to route the flow or simplify dialog logic. It determines whether the system will wait for user input or continue directly to the next node. If you do not use a Jump to node, WCS will wait for user input.
 - Targets can be sections of another dialog node :
 - Response – just run the output of the node
 - Condition – check the nodes condition and update context node.
 - User Input – gather input before further evaluation.
 - Jump_To statements are processed after context variables are updated and output is evaluated.
 - If neither the target node nor its peers is evaluated as true, the dialog turn is ended. Any response that has been generated is returned to the user, and an error message is returned to the application

Dialog (cont.)

- Dialog evaluation process based on contextual node (defines which node to start the condition search)
 - If the last node evaluated has children, then it is the contextual node. If it doesn't, then top level node (conversation start) becomes the contextual node.
 - Stored in `context.system.dialog_stack` (last node in the stack is the contextual node)
 - First attempts to find a condition match in the child nodes of contextual node. If no match, then attempt search at top level dialog nodes.
 - Make sure you have an `anything_else` node at top level.
- More features to come: reusable sub-dialogs, procedural form filling, problem resolution support



Dialog Node Editor

Name this node... >

Triggered by ⓘ
if Enter a condition

Fulfill with a response ⓘ Jump to...

+ Add response condition {...}

Enter a response...

+ Create another response

Name this node... ×

Triggered by ⓘ
if conversation_start - +

Fulfill with a response ⓘ Jump to...

{...}

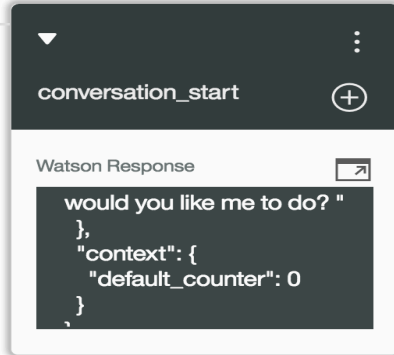
```
{
  "output": {
    "text": {
      "values": [
        "Hello, my name is Watson? What is your name?"
      ],
      "selection_policy": "sequential"
    }
  }
}
```

+ Create another response

Context

- Mechanism to store information between your bot and the application that submits the user input (variables)
 - Can be any supported JSON types (strings, numbers, JSON arrays, JSON objects).
- Access the dialog context by typing `context.variable_name`, where `variable_name` is the name of the variable on the context you want to access, or you can use the shorthand syntax `$variable_name`.
- Updates to context variables:
 - Booleans, numbers, strings JSON array are overwritten.
 - JSON Objects are merged
- Can have conditions act on variables:
 - `$age_number > 18`,
 - `$toppings_array.size() > 2`
 - `$complex_object.user_firstname.contains('Peter')`

```
"context":
{
  "my_dessert_string": "ice-cream",
  "toppings_array": [ "onion", "olives" ],
  "age_number": 18,
  "complex_object": {
    "user_firstname" : "Peter",
    "user_lastname" : "Pan",
    "has_card" : false
  }
}
```



Shorthand syntax	Full syntax in SpEL
<code>\$card_type:VISA</code>	<code>context['card_type'] == 'VISA'</code>
<code>\$card_type:(MASTER CARD)</code>	<code>context['card_type'] == 'MASTER CARD'</code>

Analytics

- Use production data to review service.
 - Log Viewer
 - Last 90 days of data (varies by plan - standard/free)
- Search/Filter based on keywords (of requests not response), Intents, Entities, and Date Range
- Requests/response through the /message endpoint (not try it out panel)
- Requires log data to be stored (no opt out header)

The screenshot displays the IBM Watson Analytics interface for the 'Car_Dashboard' dataset. The top navigation bar includes 'Intents', 'Entities', 'Dialog', and 'Improve' tabs. Below the navigation bar is a search bar and filter controls. The 'Filter by:' section shows 'Intents' and 'Entities' dropdown menus, with a 'Last 90 days' date range selected. The results section shows 7 results, with the first four displayed in a table. Each row contains a 'System classification' (e.g., '#locate_amenity', '@amenity:gas') and a corresponding user query (e.g., 'is there a gas station near me?'). A 'View conversation' link is provided for each result.

System classification	Query	Action
#locate_amenity @amenity:gas	is there a gas station near me? There are a few gas stations nearby. Which one would you like to drive to?	View conversation
#	Hi. It looks like a nice drive today. What would you like me to do?	View conversation
#turn_on @appliance:music	what about music ? Sure thing! Which genre would you prefer? Jazz is my personal favorite.	View conversation
@amenity:gas @option:closest @genre:rock @appliance:music	find the closest gas station and also play some rock music ? Great choice! Playing some rock music for you.	View conversation

Developing Your Application

- Create a wrapper to communicate with WCS
 - an object you will use to send input to, and receive output from, the service
 - specify the authentication credentials from the service key, as well as the version of the Conversation API
- Maintaining state
 - Conversation service is stateless
 - The application is responsible for maintaining state information
 - Maintained using the context which is a JSON object passed back and forth between the application and WCS.
 - send back the response.context object received in the previous round to ensure that the context is maintained from one turn to the next
- Implement app actions
 - output object in the response JSON to signal when the application needs to carry out an action, based on the detected intents.
 - action flags are sent using the action property

```
// Example 4: implements app actions.

var prompt = require('prompt-sync')();
var ConversationV1 = require('watson-developer-cloud/conversation/v1');

// Set up Conversation service.
var conversation = new ConversationV1({
  username: 'USERNAME', // replace with username from service key
  password: 'PASSWORD', // replace with password from service key
  path: { workspace_id: 'WORKSPACE_ID' }, // replace with workspace ID
  version_date: '2016-07-11'
});

// Start conversation with empty message.
conversation.message({}, processResponse);

// Process the conversation response.
function processResponse(err, response) {
  if (err) {
    console.error(err); // something went wrong
    return;
  }

  var endConversation = false;

  // Check for action flags.
  if (response.output.action === 'display_time') {
    // User asked what time it is, so we output the local system time.
    console.log('The current time is ' + new Date().toLocaleTimeString());
  } else if (response.output.action === 'end_conversation') {
    // User said goodbye, so we're done.
    console.log(response.output.text[0]);
    endConversation = true;
  } else {
    // Display the output from dialog, if any.
    if (response.output.text.length !== 0) {
      console.log(response.output.text[0]);
    }
  }

  // If we're not done, prompt for the next round of input.
  if (!endConversation) {
    var newMessageFromUser = prompt('>> ');
    conversation.message({
      input: { text: newMessageFromUser },
      // Send back the context to maintain state.
      context : response.context,
    }, processResponse)
  }
}
```

Practices and Patterns

Guidance – Entities and Dialog

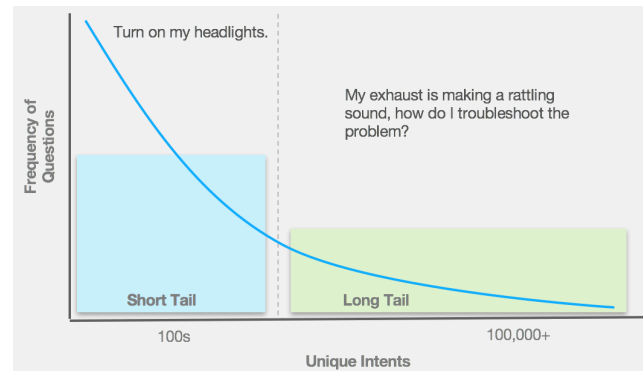
- Entities
 - Focus on classifier first before tackling entities
 - Capture only the entities that are necessary to fulfil user request.
 - Infer entities from real user data and capture at the right granularity
 - account_type vs checking_account or savings_account
- Dialog
 - Don't try to respond to every possible permutation or dialogs
 - Use care when wording responses is critical.
 - Use multiple values with selection policy to prevent “robotic” feeling.
 - Where/when possible, decouple “answers” from dialog
 - Use continue-from to reduce dialog complexity
- Continuously train and iterate
 - Create Train / test / blind sets.
 - Perform automated testing (i.e. Confusion matrix)
 - Improve by refining intent groups, removing/adding questions, etc

Patterns

- Hand-off or fall back strategies:
 - Call to agent / ticketing system
 - Information Retrieval
 - When knowledge base exists.
- Enrichment (pre and post processing)
 - Pre-process questions to set metadata or parse information used by conversation in context
 - Alchemy language to extract open entities.
 - Post-processing to augment data returned to user (data lookups or service calls)
- Orchestration is responsibility of application
 - Based on confidence of conversation, a flag set by conversation or pre-processing logic (i.e question analysis).

Majority of questions hit a small number of intents

Different Approach required to bring back appropriate response given number of answers



Watson Conversation



Here Watson uses reasoning strategies that focus on the language and context of the **question**.

Watson Discovery



Here Watson uses reasoning strategies that focus on identifying the most appropriate **answer**.

Patterns

- Parallel Conversations
 - Set a context flag (unset when you no longer want to return)
 - Application stores full input payload at each step if flag is set
 - When user asks another question, the application sees the stack returns to root.
 - Can then generate a new conversation ID.

The screenshot displays the IBM Watson Conversation service interface, illustrating a parallel conversation pattern. The left pane shows the 'Intents' and 'Entities' sections, with a 'conversation_start' intent and a 'pizza' entity. The right pane shows the 'Dialog' section, which contains a flowchart with nodes for 'conversation_start', 'order_pizza', and 'toppings'. The 'toppings' node is highlighted, showing its configuration. The bottom pane shows the 'Watson understands' section, which displays the input text and the resulting JSON payload. The JSON payload includes the 'input' object, the 'context' object, and the 'dialog_stack' array, which contains the current state of the conversation.

```
{
  "input": {
    "text": "I want a small pizza"
  },
  "context": {
    "conversation_id": "b2b2c3ab-20f9-4b28-b1b8-b0d4c7d7",
    "dialog_stack": [
      "root"
    ],
    "dialog_turn_counter": 1,
    "dialog_request_counter": 1
  },
  "entities": [
    {
      "entity": "size",
      "start": 14,
      "end": 16,
      "value": "small"
    }
  ],
  "intents": [
    {
      "intent": "order_pizza",
      "confidence": 0.945054230958
    }
  ]
}
```

Resources

Resources

- Sample Applications
 - **Conversation simple app - <http://conversation-simple.mybluemix.net/>**
 - The Node.js app shows how the Conversation service uses intents in a simple chat interface. It shows the conversation with an end user and the JSON responses at each turn of the conversation.
 - **Conversation Enhanced app - <http://conversation-enhanced.mybluemix.net/>**
 - This Java app demonstrates the combination of the Conversation service and the Retrieve and Rank service. First, users pose questions to the Conversation service. If Conversation is not able to confidently answer, a call is executed to the Retrieve and Rank service to provide the user with a list of helpful answers. Use this user interface to accelerate the development of your own conversational apps.
 - **Cognitive car demo - <https://conversation-demo.mybluemix.net/>**
 - This Node.js app is a fully developed example of the type of app you can build with the Conversation service. It shows how the Conversation service uses intents, entities, and dialog.
- Watson Community - <https://developer.ibm.com/watson/>

Ground Truth Creation Sequence

- Question Collection – create Training and Blind/Test set
 - WCS_QuestionCollection.xlsx
 - Assign IDs and use RAND() function to separate questions
- Ground Truth Creation
 - WCS_GroundTruth.xlsx
 - Create Analyze Worksheet – pivot table
 - Cluster questions

Appendix

Syntax Examples

- `#help` `intent == 'help'`
- `! #help` `intent != 'help'`
- `NOT #help` `intent != 'help'`
- `#(help me)` `intent == 'help me'`
- `@year` `entities['year']?.value` (entity value is present)
- `@year == 2017` `entities['year']?.value == 2017`
- `@year != 2017` `entities['year']?.value != 2017`
- `@city == 'Boston'` `entities['year']?.value == 'Boston'`
- `@city:Boston` `entities['year']?.value == 'Boston'`
- `@(car make)` `entities['car make']?.value`
- `#help OR #i_am_lost` `(intent == 'help' || intent == 'i_am_lost')`
- `$gold_member` `context['gold_member']` (same as `context.gold_member`)
- `$(gold member)` `context['gold member']`
- `$card_type:VISA` `context['card_type'] == 'VISA'`
- `$card_type:(MASTER CARD)` `context['card_type'] == 'MASTER CARD'`

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**
- Things to think about:
 - What behaviors will achieve the defined purpose of the solution?
 - Proactive statements – reaching out to get the user's interest and engage them at specific intervals in the conversation and to guide them to the content you want them to engage with.
 - Process flows – branching node structures that lead the user through a turn-taking conversation which allows them to choose the options that work best for them (in order to make recommendations)
 - Profiling – remembering things the user has said and options they have selected to tailor the conversation for each user.

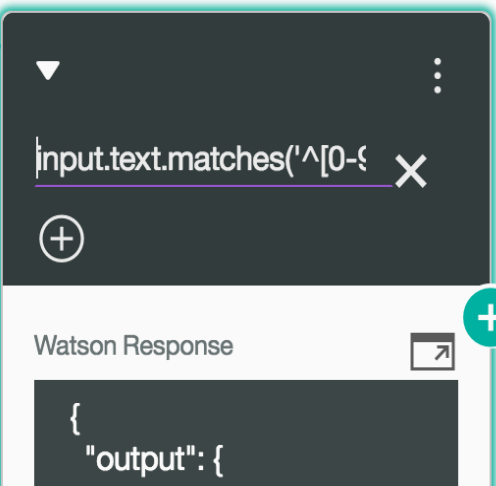
Expression Language / Notation

- Spring Expression (SpEL) language: short-hand notation enables concise node conditions such as
 - #help same as: intent == 'help' and @year:2017 same as: entities['year']?.value == 2017
 - Under the hood this uses the [Spring Expression Language \(SpEL\)](#)
- Condition notation:
 - Matching on multi-term values requires use of parenthesis in short hand
 - @appliance:(air conditioner) | @appliance == 'air conditioner'
 - Specifying just the concept (@entity_name of #intent_name) means the variable is not null (has any value)
- Evaluation syntax denoted by <? Some expression ?>
 - <? \$age_number ?> - output the variable data
 - <? \$toppings.array.append('Cheese') ?>
 - <? \$toppings_array.removeValue('Onion') ?>

Expression Language / Notation

- Objects have their own properties/metadata that may be useful
 - Accessing entity synonym - `<?entities["Account_Type"][0].literal ?>`
 - Accessing confidence scores – `intents[int_position].confidence`
- Objects have properties/methods useful for validation and operations (not an exhaustive list)
 - String
 - `input.text.matches('[0-9]+')`, `input.text.matches('^ [0-9]{4}$')`
 - `Input.text.extract(regexp, group_index)`
 - `Input.text.extract('[//d]'+',0`
 - Remember to escape the slashes
 - `input.text.contains('yes')`
 - `Input.text.equals(String)` and `input.text.equalsIgnoreCase(String)`
 - `Input.text.length()`
 - `Input.text.isEmpty()` -> check if a context variable or input is set
 - Json objects
 - `JSONObject.has(string)` – check if a property exists
 - `$context_var.has("property_name")`
 - JsonArray
 - `JsonArray.size()`

▶ `intents[0].confidence>0.5`



Expression Language / Notation (Java)

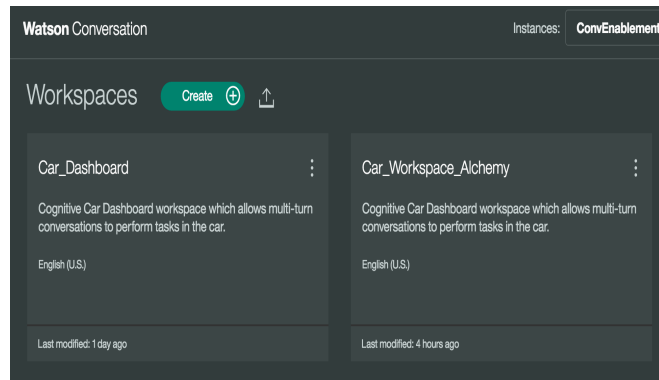
- Some Java constructs available in dialog
 - Accessed by a constructor [new java.util.Date()] or by type locator [T(java.util.Date)] in SpEL
 - `<? T(java.lang.Math).random() ?>`
 - `<? new java.util.Date() ?>`
 - Example:
 - `context": {"max_num": "<? T(java.lang.Math).max($var1, $var2) ?>"}`
- Inline 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."  
  }  
}
```

Interaction: Development

- Development environment through web based UI.
 - Single tool to provides access to any provisioned instance the user has access to.
 - Collaboration controlled via Bluemix organization/spaces
 - Built in testing environment
- Limit on number of workspaces, intent, entities based on service plan
- Workspace
 - Container for conversation development artifacts
 - Language identified in workspace is used to train intents/entity values
 - Workspaces may be deleted after long period of inactivity.



Items	Standard plan	Free plan
Workspaces	20	3
Intents	2000	25
Dialog nodes	100,000	25,000
Monthly queries	Unlimited	1000 API calls
Production data	30 days	7 days

Interaction: Runtime

- All runtime interactions currently through a single REST endpoint.
 - workspace identifier used to target developed conversation
- Basic Auth using service credentials from provisioned conversation service.
- Service is completely stateless
 - Data is not persisted from one request to the next, application must maintain state/context.
- Response formatting is responsibility of the application
- SDKs exist for Java, Node, Python.

POST /v1/workspaces/{workspace_id}/message

Response Class (Status 200)
Successful request

Model Model Schema

```
"context": {  
  "conversation_id": "1b7b67c0-90ed-45dc-8508-9488bc483d5b",  
  "system": {  
    "dialog_stack": [  
      "root"  
    ],  
    "dialog_turn_counter": 2,  
    "dialog_request_counter": 2  
  }  
},  
"entities": [
```

Response Content Type

Parameters

Parameter	Value	Description	Parameter
workspace_id	0a0c06c1-8e31-4655-9067-58fcac5134f1	Unique identifier of the workspace.	path
version	2016-07-11	Release date of the API version in YYYY-MM-DD format.	query
body	<div></div>	The user's input, with optional intents, entities, and other properties from the response.	body