IBM Watson

# Watson Developer Cloud Services

## Watson Conversation Deep Dive

*Javier R. Torres*
*Solutions Architect*

IBM

# Watson for Customer Engagement

- Watson offers better user experience.
  - Allows you to build a single application across multiple platform.
- Able to resolve calls through integrated actions.
  - Minimizes call deflections

# 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

# Key Messaging

## Value proposition

For the developer who wants to develop a natural language interface for their application (often to build a conversational app, chat bot, or virtual agent to improve customer experience and enhance support operations), IBM is their partner.

With the new Conversation capability on Watson Developer Cloud, developers can quickly and easily create a conversational app that enables productive and consistent communication with end users across a variety of channels.

To jumpstart conversational app development, IBM offers an array of software development kits, code examples, and simplified tooling created for developers, by developers.

## Key messaging

- **Build, test and deploy a system straight away**; start for **free**
- App development is **quick and easy** with Conversation's **intuitive tooling interface**
- **Minimal coding or machine learning** expertise is required
- Build once with Conversation – but **deploy wherever** you want. You can integrate your bots into a mobile or web environment as easily as you can into **messaging channels, mobile, web, robots etc.**
- **Enhance** your conversational app with **complementary Watson APIs** like Tone Analyzer and Speech to Text / Text to Speech to give it more human-like qualities
- Extend the capabilities of your app with Retrieve & Rank, enabling it to **answer more questions** than it usually could

“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

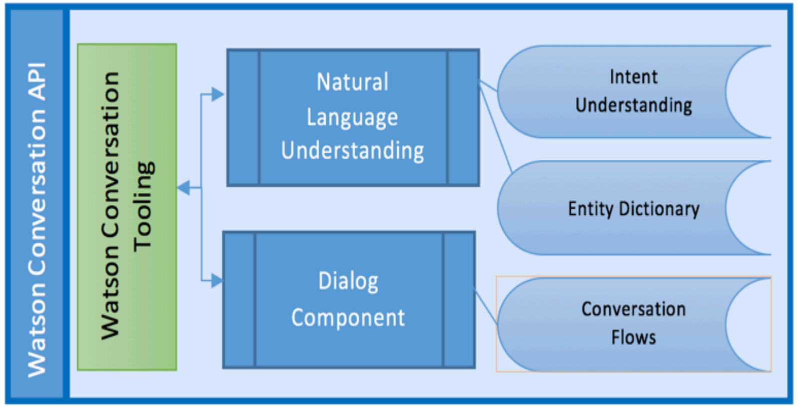
| Question | | Answer |
|---|---|---|
| How do I reset my password? | Dialog | Guide the user through a set of steps |
| Someone has stolen my credit card. | Deflect | Transfer to human agent |
| Where is the nearest store? | Map | Application launches map with directions |
| I need to pay my outstanding invoice. | App Nav. | Bring user to pay bill screen |
| Can I pay my bills using my credit card? | Info. Retrieval | Bring back an answer |

IBM **Watson**

IBM

## Conversation Components

# The Building Blocks

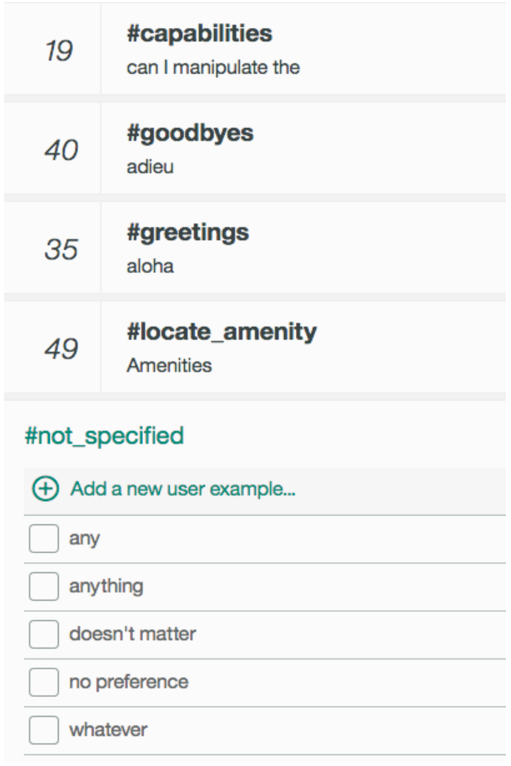


- Natural Language Understanding (NLU)
  - Identification/processing for:
    - Intent classification is used to determine the meaning of an utterance from the end user.
    - Entity Extraction is used to understand the object(s) of the utterance.
  - Orchestration of modifications, training, active revisions for a workspace
    - New intents, examples or entities are propagated to training automatically without user intervention.
    - Deletion of workspace

- Dialog Interpreter
  - Based on this understanding, dialog is used to take action
    - Providing an answer or engaging the user in a dialogue

## Intent

- The action a user wants to take (verb)
- Deep Learning state of the art classifier for understanding the intent behind a user's utterance
  - Train **convolutional neural networks** for text classification with word embedding as input
  - Uses entities (synonyms) as training elements
  - Relative confidence values for intents



| | |
|---|---|
| 19 | **#capabilities** can I manipulate the |
| 40 | **#goodbyes** adieu |
| 35 | **#greetings** aloha |
| 49 | **#locate_amenity** Amenities |

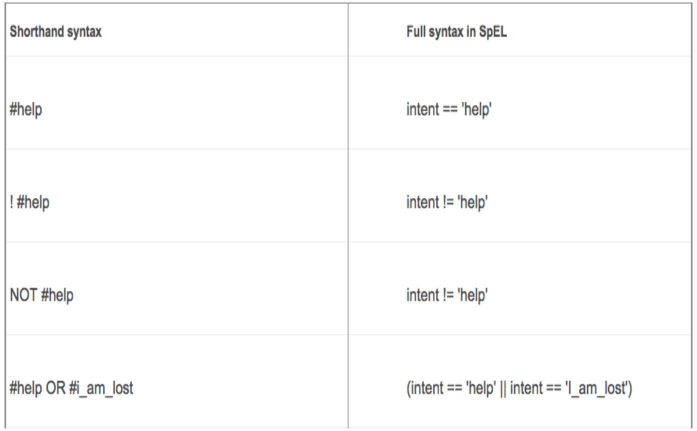
#not_specified

Add a new user example...

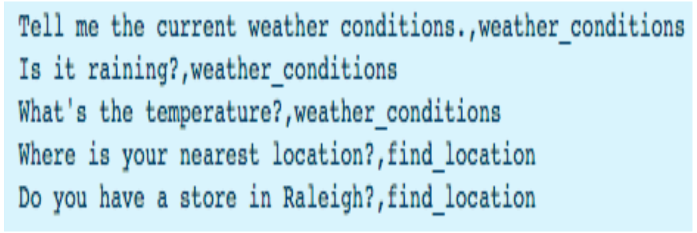
- any
- anything
- doesn't matter
- no preference
- whatever

## Intent

- Identified in tooling via **#intent_name**
  - Allowed: Letters, numbers, hyphen, underscores, dashes and/or dot
  - Prohibited: Spaces, more than 128 characters.
- Import intents via CSV
  - <example>, <intent>
- Multi-lingual: supporting English, French, Italian, Spanish, Portuguese, and more to come.



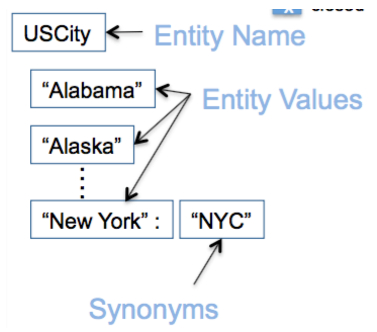
| Shorthand syntax | Full syntax in SpEL |
|---|---|
| #help | intent == 'help' |
| ! #help | intent != 'help' |
| NOT #help | intent != 'help' |
| #help OR #i_am_lost | (intent == 'help' \|\| intent == 'I_am_lost') |

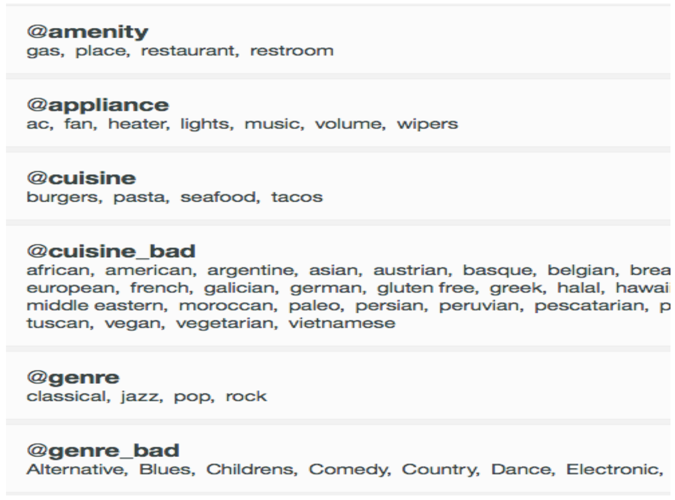


```
Tell me the current weather conditions.,weather_conditions
Is it raining?,weather_conditions
What's the temperature?,weather_conditions
Where is your nearest location?,find_location
Do you have a store in Raleigh?,find_location
```

# Entities

- Entities are objects user wants to take action on
  - “Where is the pool?” Entity = @Pool
  - “I need to reset my password to my email” – Entity = @Email
- 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

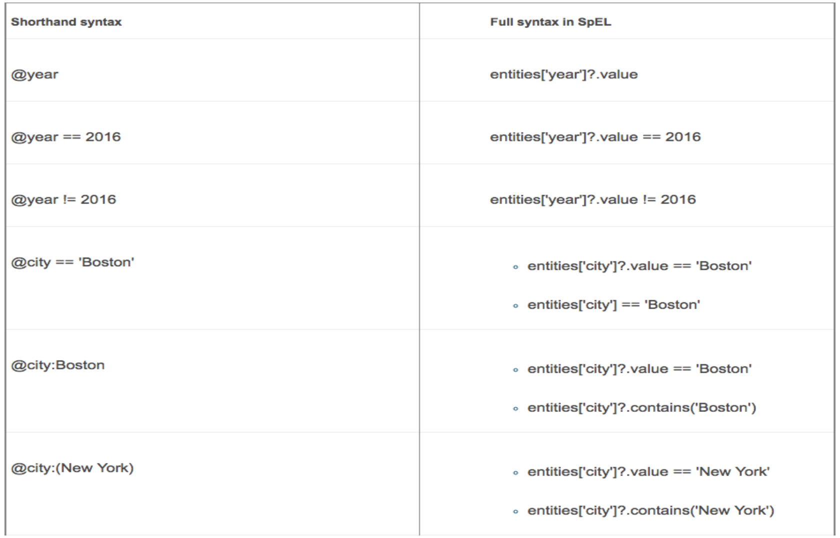
**@cuisine_bad**
african, american, argentine, asian, austrian, basque, belgian, brea
european, french, galician, german, gluten free, greek, halal, hawai
middle eastern, moroccan, paleo, persian, peruvian, pescatarian, p
tuscan, vegan, vegetarian, vietnamese

**@genre**
classical, jazz, pop, rock

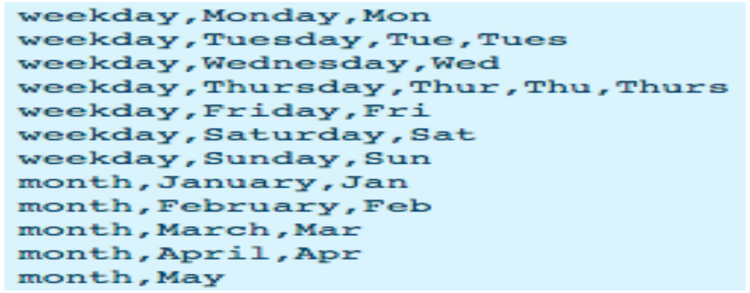
**@genre_bad**
Alternative, Blues, Childrens, Comedy, Country, Dance, Electronic,

## 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' | ◦ entities['city']?.value == 'Boston'<br>◦ entities['city'] == 'Boston' |
| @city:Boston | ◦ entities['city']?.value == 'Boston'<br>◦ entities['city']?.contains('Boston') |
| @city:(New York) | ◦ entities['city']?.value == 'New York'<br>◦ entities['city']?.contains('New York') |



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

## 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

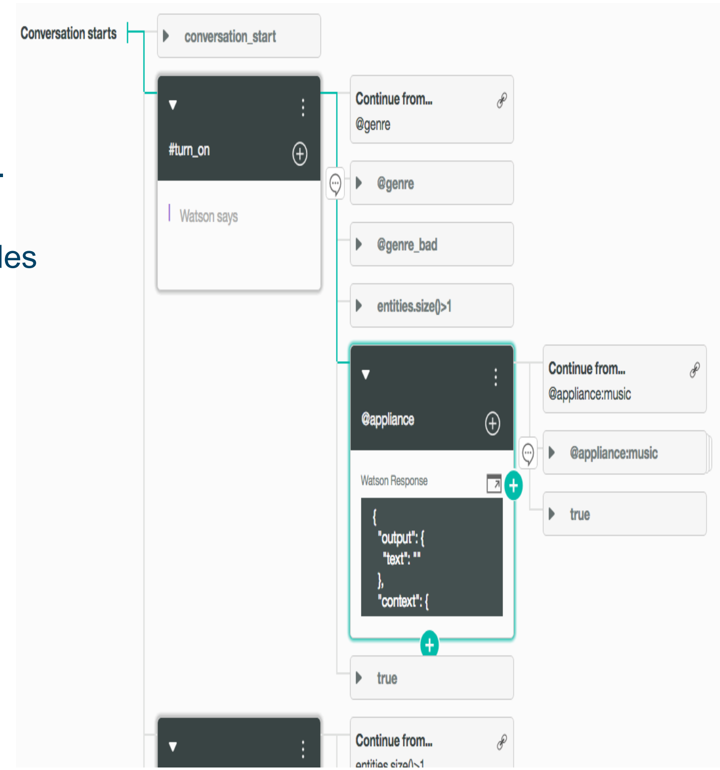
IBM Watson

# 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

- Input (Conditions)
  - Matching rules against entities, intents, variables
  - Supports ==, !=, &&, || operators
    - Default operand is logical AND
    - AND has precedence over OR

- Output:
  - 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": ""
 },
 "context": {
```

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

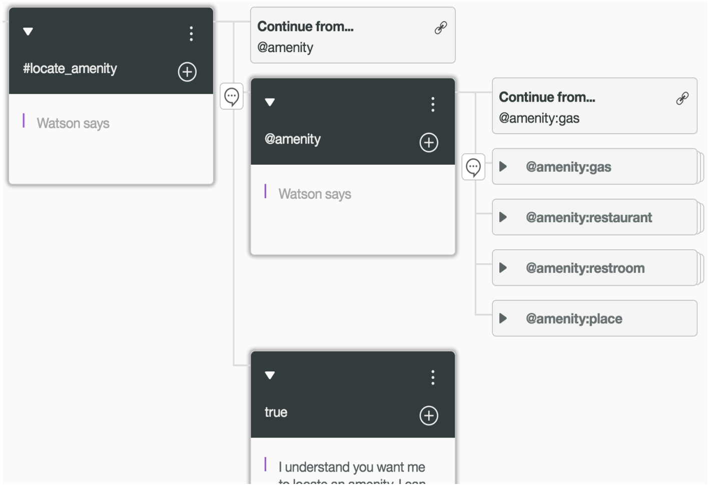
## Dialog (cont.)

- Order Matters
  - Dialog nodes evaluated in order they appear in UI.

- 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 mater what.
  - False – Does not get ”normally” executed.

- “Continue-from” can be used to alter default flow, to route the flow or simplify dialog logic.
  - 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.
  - Continue from statements are processed after context variables are updated and output is evaluated.

# Dialog (cont.)



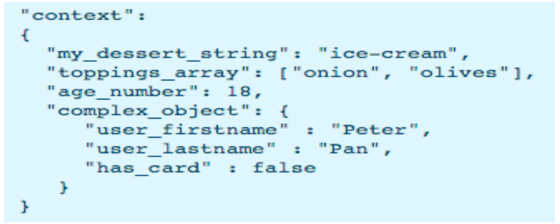
- Dialog evaluation process based on contextual node (defines which node to start the condition search)
  - If the last node evaluated has children, then it is the contextual node. If it doesn't, then top level node (conversation start) becomes the contextual node.
  - Stored in context.system.dialog_stack (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.

- #, @, $, ? are special characters
  - Escape each with a backslash (\)

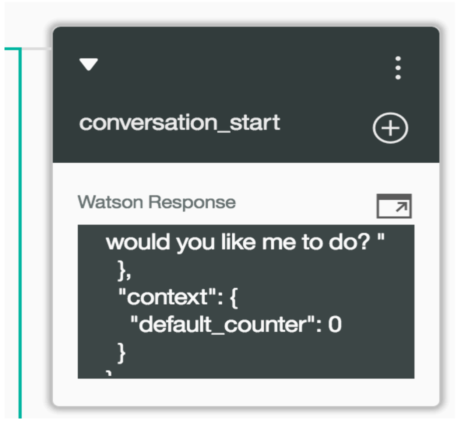
- More features to come: reusable sub-dialogs, procedural form filling, problem resolution support

# Context

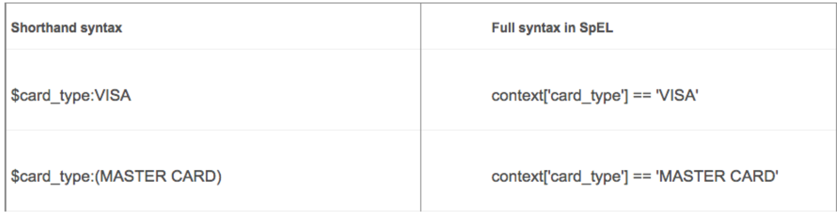
- Mechanism to store information (variables)
  - Can be any supported JSON types (strings, numbers, JSON arrays, JSON objects.
- Accessed by putting node into ‘Advanced’ mode.
- Updates to context variables:
  - Booleans, numbers, strings JSON array are overwritten.
  - JSON Objects are merged
- Can 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
   }
}
```



```
  would you like me to do? "
   },
   "context": {
     "default_counter": 0
   }
```



| Shorthand syntax | Full syntax in SpEL |
|---|---|
| $card_type:VISA | context['card_type'] == 'VISA' |
| $card_type:(MASTER CARD) | context['card_type'] == 'MASTER CARD' |

# Expression Language / Notation

- Simple expression language: short-hand notation enables concise node conditions such as
  - #help same as: intent == 'help' and @year:2016 same as: entities['year']?.value == 2016
  - Under the hood this uses the Spring Expression Language (SpEL)

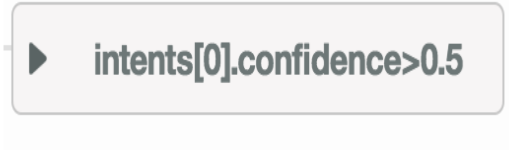
- Condition notation:
  - Matching on multi-term values requires use of parenthesis in short hand
    - @applicance:(air conditioner) | @applicance == 'air conditioner'
  - Specifying just the concept (@entity_name of #intent_name) means the variable is not null (has any value)

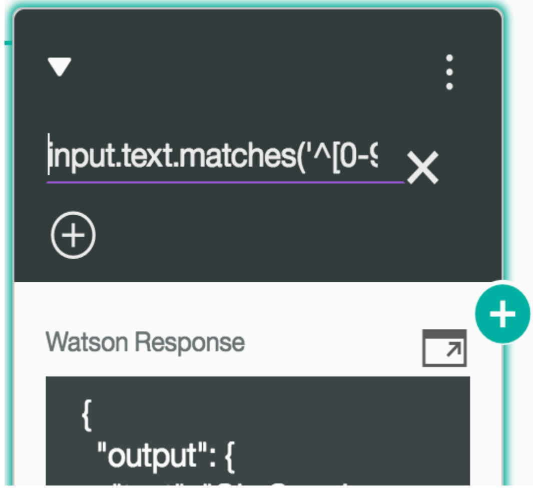
- Evaluation syntax denoted by <? Some expression ?>
  - <? $age_number ?> - output the variable data
  - <? $toppings.array.append('Cheese') ?>
  - <? $toppings_array.removeValue('Onion') ?>

# 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


input.text.matches('^[0-

```
{
  "output": {
```

# 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 |

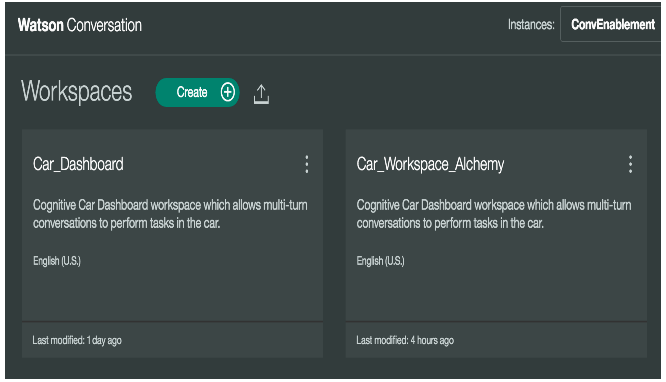
```
"context": {
   ”precanned_resp": {
     ”joke": ”Knock Knock”
     ”weather": "I don’t have the
weather.",
     ”feeling": "I feel great."
   }
 }
```

# 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.



Car_Dashboard

Cognitive Car Dashboard workspace which allows multi-turn conversations to perform tasks in the car.

English (U.S.)

Last modified: 1 day ago

Car_Workspace_Alchemy

Cognitive Car Dashboard workspace which allows multi-turn conversations to perform tasks in the car.

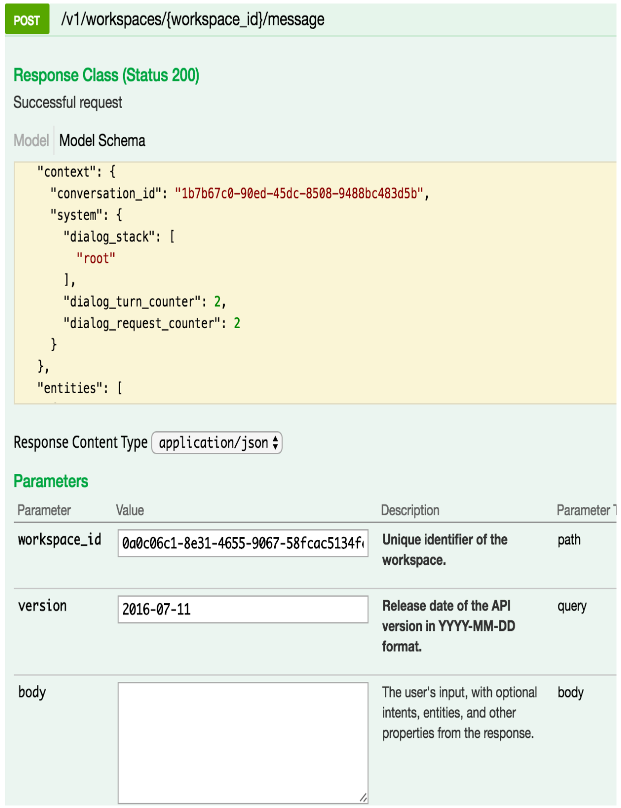
English (U.S.)

Last modified: 4 hours ago

| 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": [
```

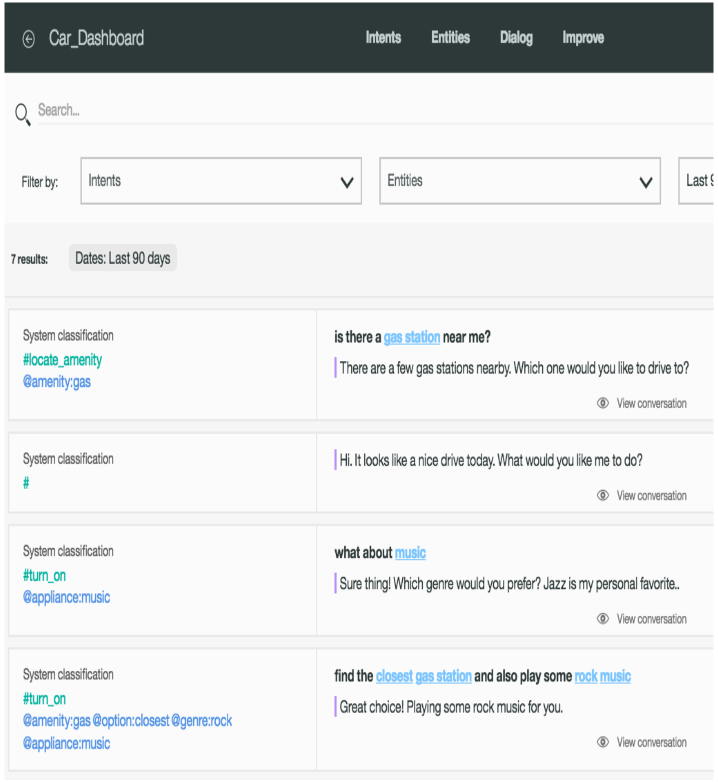
Response Content Type application/json

Parameters

| Parameter | Value | Description | Parameter T |
|---|---|---|---|
| workspace_id | 0a0c06c1-8e31-4655-9067-58fcac5134f | **Unique identifier of the workspace.** | path |
| version | 2016-07-11 | **Release date of the API version in YYYY-MM-DD format.** | query |
| body | | The user's input, with optional intents, entities, and other properties from the response. | body |

# 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)



| | |
|---|---|
| System classification #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 |
| System classification # | Hi. It looks like a nice drive today. What would you like me to do? View conversation |
| System classification #turn_on @appliance:music | what about music Sure thing! Which genre would you prefer? Jazz is my personal favorite.. View conversation |
| System classification #turn_on @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 |

IBM **Watson**

## Practices and Patterns

## 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 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

## 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

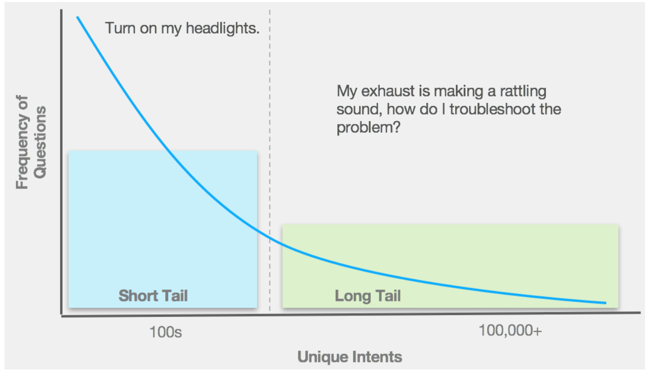
- Hand-off or fall back strategies:
  - Call to agent / ticketing system
  - Information Retrieval
    - When knowledge base exists.

- 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**.

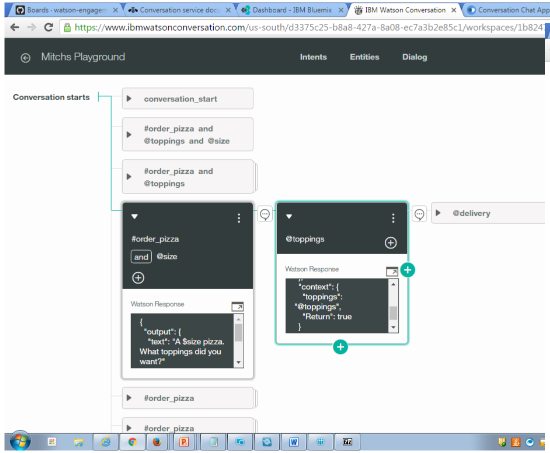
**Retrieve and Rank**

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

IBM Watson

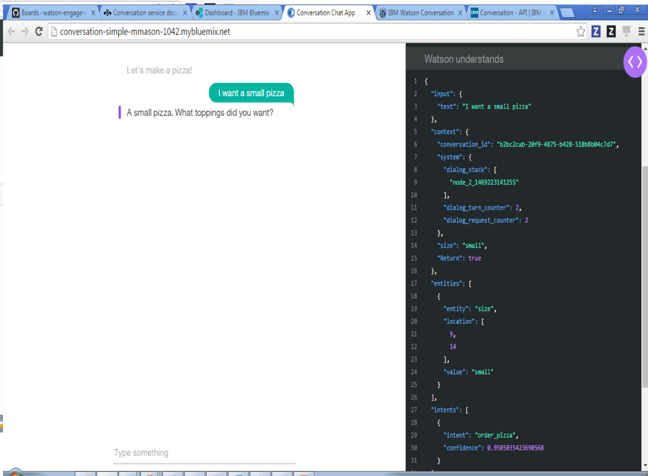
## Patterns

- 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.


https://www.ibmwatsonconversation.com/us-south/d3375c25-b8a8-427a-8a08-ec7a3b2e85c1/workspaces/1b824
Mitchs Playground
Intents
Entities
Dialog
Conversation starts
conversation_start
#order_pizza and @toppings and @size
#order_pizza and @toppings
#order_pizza
and @size
@toppings
@delivery
Watson Response
#order_pizza
#order_pizza

```
{
 "output": {
  "text": "A $size pizza. What toppings did you want?"
```

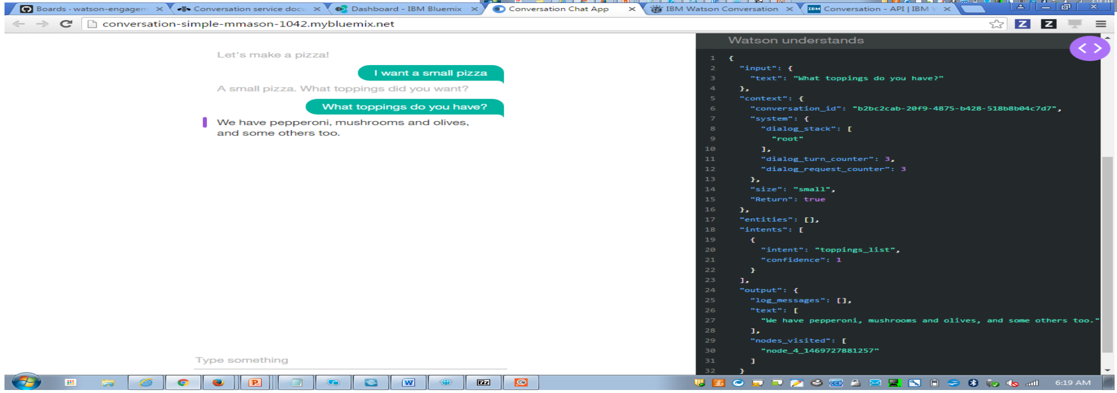
```
 "context": {
  "toppings": "@toppings",
  "Return": true
 }
```



I want a small pizza

A small pizza. What toppings did you want?

```
{
  "input": {
    "text": "I want a small pizza"
  },
  "context": {
    "conversation_id": "b2bc2cab-20f9-4875-b428-518b8b04c7d7",
    "system": {
      "dialog_stack": [
        "node_2_1469223141255"
      ],
      "dialog_turn_counter": 2,
      "dialog_request_counter": 2
    },
    "size": "small",
    "Return": true
  },
  "entities": [
    {
      "entity": "size",
      "location": [
        9,
        14
      ],
      "value": "small"
    }
  ],
  "intents": [
    {
      "intent": "order_pizza",
      "confidence": 0.9585035423690568
    }
```



I want a small pizza

A small pizza. What toppings did you want?

What toppings do you have?

We have pepperoni, mushrooms and olives, and some others too.

```
{
  "input": {
    "text": "What toppings do you have?"
  },
  "context": {
    "conversation_id": "b2bc2cab-20f9-4875-b428-518b8b04c7d7",
    "system": {
      "dialog_stack": [
        "root"
      ],
      "dialog_turn_counter": 3,
      "dialog_request_counter": 3
    },
    "size": "small",
    "Return": true
  },
  "entities": [],
  "intents": [
    {
      "intent": "toppings_list",
      "confidence": 1
    }
  ],
  "output": {
    "log_messages": [],
    "text": [
      "We have pepperoni, mushrooms and olives, and some others too."
    ],
    "nodes_visited": [
      "node_4_1469727881257"
    ]
  }
}
```

IBM **Watson**

## 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/

IBM **Watson**

# Thank You

# 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 == 2016 | entities[‘year’]?.value == 2016 | |
| @year != 2016 | entities[‘year’]?.value != 2016 | |
| @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' | |