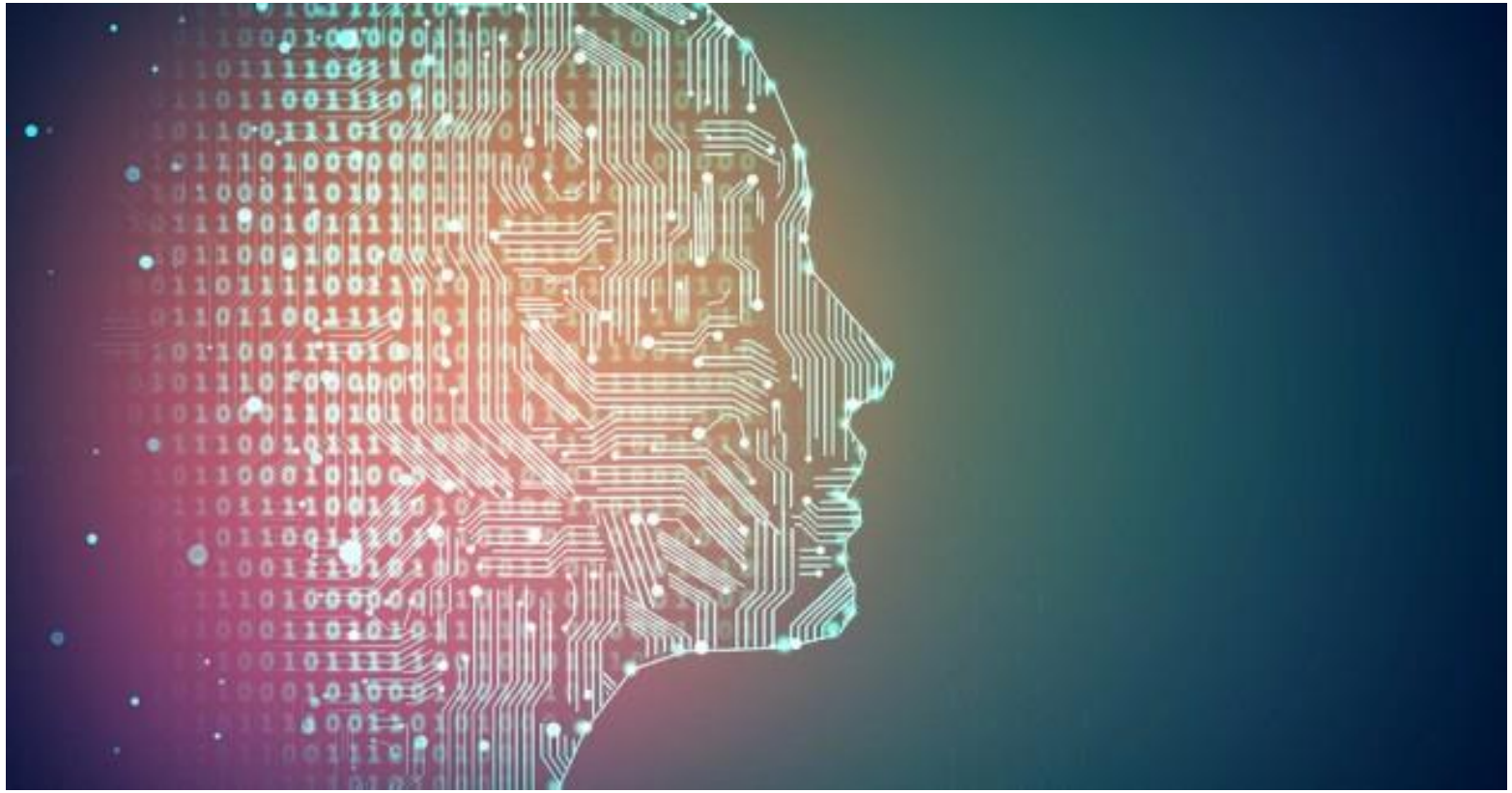
# Conversational AI: Dialogues

## GCP: Dialogflow CX: Entity Types & Entities

(Source Credit: GCP's Dialogflow CX Documentation)

Thuan L Nguyen, PhD

# Slide 2: Dialogflow CX: Entity Types & Entities



*AI Deep learning (Source: mindovermachines.com*)

# Slide 3: Dialogflow CX: Entity Types & Entities

1. Dialogflow CX: Entity Types & Entities: Overview
2. Dialogflow CX: Entity Types & Entities: Terminology
3. Dialogflow CX: Entity Types & Entities: Options
4. Dialogflow CX: Entity Types & Entities: Session Entities
5. Dialogflow CX: Entity Types & Entities: Custom Entities
6. Dialogflow CX: Entity Types & Entities: Automated Expansion
7. Dialogflow CX: Entity Types & Entities: Fuzzy Matching

# Slide 4: Dialogflow CX: Entity Types & Entities

## Entity Types & Entities: Overview

**Entity types** are used to control **how data** from end-user input is **extracted**.

There are **two types of entities** that can be used in Dialogflow CX: **System Entities** and **Custom Entities**

- **System entities**:
  - Dialogflow provides predefined system entities that can match many common types of data

    For example
    --) System entities for matching dates, times, colors, email addresses, to name a few.

- **Custom entities**:
  - It is possible to create custom entities for matching custom data.

    For example:
    --) It is possible to define a vegetable entity type that includes types of vegetables available in a grocery store.

# Slide 5: Dialogflow CX: Entity Types & Entities

## Entity Types & Entities: Terminology

The term **entity** describes the general concept of entities.

There are other important entity-related terms:

- **Entity type**
  - Defines the type of information you want to **extract** from **user input**.
  - For example
    - **Vegetable** could be an entity type that represents a group of types of vegetables such as lettuce, cabbage, carrots, spinach, and so on.

- **Entity entry**
  - For each entity type, there are many **entity entries**.
  - Each entity entry provides a set of words or phrases that are considered equivalent.
  - For example,
    - if vegetable is an entity type, you could define these three entity entries: Carrot, bell pepper, …

- **Entity reference value and synonyms**
  - Some entity entries have multiple words or phrases that are considered equivalent.
  - For example: Car and automobile

  - For these entity entries, you provide one **reference value** and one or more **synonyms**.

# Slide 6: Dialogflow CX: Entity Types & Entities

## Entity Types & Entities: Entity Options

There are many entity options that can be defined and used with Dialogflow CX.

Major **entity options** are:

- **Map** Entity
- **List** Entity
- **Composite** Entity
- **Regexp** Entity

# Slide 7: Dialogflow CX: Entity Types & Entities

## Entity Types & Entities: Entity Options: Map Entity

**Map entities** provide a **map** from **reference** values to **synonyms** for **each** reference **value**.

**Each** map entity entry contains a **single** reference value and a **list** of synonyms.

For example, each of the following rows are entity entries for a size entity type:

- S → S, small, tiny, little
- M → M, medium, average
- L → Large, huge, big

**IMPORTANT NOTES**:

- Reference value must be included in the synonym list for each entity entry.
- For a reference value to be matched, it needs to be included as a synonym itself.

To **create a map entity**:

- Using the console:
  - Uncheck the **Regexp entities** option and uncheck the **Entities only** option.

- Using the API:
  - Set the EntityType.kind field to KIND_MAP.

# Slide 8: Dialogflow CX: Entity Types & Entities

## Entity Types & Entities: Entity Options: List Entity

**List entities**

- Provide a **list** of **single value** entity entries.
- **Not** have reference values and synonyms.

For example:
--) Each of the following rows are entity entries for a material entity type:

- Value: Fabric, wood, metal
  - **NOTES**: *No reference values or synonyms like Map Entity type*

If any value is matched for an end-user input part:

- The value is extracted for the match and is used to resolve the associated parameter value.

To **create a map entity**:

- Using the console:
  - Uncheck the **Regexp entities** option and uncheck the **Entities only** option.

- Using the API:
  - Set the EntityType.kind field to KIND  LIST.

# Slide 9: Dialogflow CX: Entity Types & Entities

## Entity Types & Entities: Entity Options: Composite Entity

A **composite entity** is a special kind of **list** entity.

- Entity entries for list entities typically contain simple words or phrases
  - But they may also contain other entity types.

- When an entity type is referenced in **another** entity type, the reference is called an **alias**.

- When a **list** entity contains **aliases** to other entity types, it is called a **composite** entity.

**HOWTO create an alias**:

- Provide the **name** of the **referenced** entity type and a **property** name of your choosing.

- When a composite entity is matched at runtime, the extracted value is returned as a JSON object, with alias property names used as JSON property names.

- The format for entering an **alias** is: **@entity-name:property-name**

- For example:
  - To create a **place** entity type that matches either a city or a state, it is possible to make it match with both following entity entries:
    - **@sys.geo-city:city**
    - **@sys.geo-state:state**

# Slide 10: Dialogflow CX: Entity Types & Entities

## Entity Types & Entities: Entity Options: Composite Entity

**COMPOSITE ENTITY entry with multiple entity aliases:**

It is possible to use **multiple** entity **aliases** in a **composite** entity entry.

Consider the following **move custom** entity type that contains **aliases** of **two** different entity types:

- Aliases to a **direction** entity type
- The **@sys.number** system entity type:

**Direction custom map entity type:**

- Reference value: **Forward** → Synonyms: Forward, forwards
- Reference value: **Back** → Synonyms: Back, backward, backwards

**Move custom list entity type:**

- **Value: @sys.number:steps steps @direction:direction**

- If the move entity is matched to an end-user **input** that contains "five steps backward":
  - The move entity will be matched.
  - The extracted value is returned as: {"steps": 5, "direction": "back"}.

**Many system entities** are **composite entities**.

For example:

- The @sys.unit-currency system entity is used for matching amounts of money with a currency name.
  - It matches end-user inputs like "50 euros" or "twenty dollars and five cents".
  - The extracted value is returned as a JSON object like: {"amount": 50, "currency": "EUR"}

# Slide 11: Dialogflow CX: Entity Types & Entities

## Entity Types & Entities: Entity Options: Regexp Entity

### Regexp Entity: Overview

With **regexp** entities, it is possible to provide **regular expressions** for matching.

- Some entities need to match patterns rather than specific terms.
  - For example: National ID numbers, IDs, license plates, and so on.

### Compound Regular Expression

**Each** regexp entity corresponds to a single pattern.

- However, it is possible to provide **multiple** regular expressions
  - If they **all** represent variations of a single pattern.

During agent **training**:

- **All** regular expressions of a single entity are combined with the alternation operator (|)
  - To form one **compound** regular expression.

# Slide 12: Dialogflow CX: Entity Types & Entities

## Entity Types & Entities: Entity Options: Regexp Entity

- For example:
  --) It is possible to provide the following regular expressions for a **phone number**:
  - ^[2-9]\d{2}-\d{3}-\d{4}$
  - ^(1?(-?\d{3})-?)?(\d{3})(-?\d{4})$

  The **compound** regular expression becomes:
  - ^[2-9]\d{2}-\d{3}-\d{4}$|^(1?(-?\d{3})-?)?(\d{3})(-?\d{4})$

The **ordering** of regular expressions matters.

- **Each** of the regular expressions in the compound regular expression are processed in **order**.
- Searching stops once a valid match is found.

- For example:
  - For an end user expression of "Seattle":
    - Sea|Seattle matches "Sea"
    - Seattle|Sea matches "Seattle"

# Slide 13: Dialogflow CX: Entity Types & Entities

## Entity Types & Entities: Entity Options: Regexp Entity

### HOWTO Create Regexp Entities

To **create a regexp entity**:

- Using the console:
  - Check the **Regexp entities** option.

- Using the API:
  - Set the EntityType.kind field to KIND_ REGEXP.

### Regexp Entities: Limitations

The following limitations apply to regexp entities:

- **Fuzzy matching** cannot be enabled for regexp entities. These features are mutually exclusive.
- **Each** agent can have a maximum of **50** regexp entities.
- The **compound** regular expression for an entity has a **maximum** length of **1024** characters.

# Slide 14: Dialogflow CX: Entity Types & Entities

## Entity Types & Entities: System Entities

Dialogflow provides many *system entities* to extract common types from end-user expressions.

- For example:
  - @sys.color type can be used to extract values like "red" or "blue".

- The full list of system entities is discussed at this link:
  - https://cloud.google.com/dialogflow/cx/docs/reference/system-entities

System entity support differs for languages and regions. See the system entity limitations for details.

For most applications, the values provided by system entities work well.

- However, your application may need to add additional values for system entities.
  - For example, some user may need to add "blue-green" to the list of values for @sys.color.

Some system entities can be extended for this purpose.

- The System entity reference lists the system entities that can be extended.

HOWTO extend a system entity: See the manuals

- Link: https://cloud.google.com/dialogflow/cx/docs/concept/entity-system

# Slide 15: Dialogflow CX: Entity Types & Entities

## Entity Types & Entities: Custom Entities

HOWTO create custom entity: See the manuals

- Link: https://cloud.google.com/dialogflow/cx/docs/concept/entity-custom

The following limitations are applied to custom entities:

- Custom entity type display names are unique for each agent.
- Entity type display names should start with a letter and can contain the following:
  - A-Z, a-z
  - 0-9
  - _ (underscore)
  - - (dash).
  - **NOTES**: *Entity **reference** and **synonym** values have no such limitation.*

# Slide 16: Dialogflow CX: Entity Types & Entities

## Entity Types & Entities: Session Entities

A **session** represents a conversation between a Dialogflow agent and an end-user.

It is possible to create special entities, called **session entities**, or **user entities**, during a session.

- Session entities can extend or replace custom entity types
- Session entities only exist during the session that they were created for.
- **All session data**, including session entities, is **stored** by Dialogflow for **30 minutes**.

For example:

- An agent has a @fruit entity type that includes "pear" and "grape"
- That entity type could be updated to include "apple" or "orange", depending on the information your agent collects from the end-user.
- The updated entity type would have the "apple" or "orange" entity entry for the rest of the session.

# Slide 17: Dialogflow CX: Entity Types & Entities

## Entity Types & Entities: Entities: Automated Expansion (Limited Use)

It is possible to enable **automated expansion** for a **custom** entity type.

- When enabled,
  - The agent can recognize values that have **not** been explicitly provided.

For example, consider a shopping **list entity** type:

- **Value**: Bread, butter, milk, apple, ice cream.
- If an end-user input is "I need to buy some **carrots**":
  - "**carrots**" will be matched for this entity type, even though it's not provided.
  - The agent recognizes that "carrots" is contextually similar to other values.

It is good to follow the best practices when considering automated expansion:

- Enabling automated expansion does **not** guarantee entity extraction.
- For a **finite** list, it is good to provide the **complete** list.
  - Instead of providing a partial list and enabling automated expansion.

- If **automated** expansion is enabled in **more than one** entity
  - It may cause conflicts and unexpected classification results.

- To ensure **better** parameter extraction quality
  - It is crucial to provide diverse training data which cover all the use-cases in which a given entity can be found in the expected agent's traffic.
  - With not enough examples, automated entities expansion might not work as expected.

# Slide 18: Dialogflow CX: Entity Types & Entities

## Entity Types & Entities: Entities: Automated Expansion (Limited Use)

### HOWTO Enable Automated Expansion

To **enable automated expansion**:

- Using the console:
  - Check the **Automatically add entities** option.

- Using the API:
  - set the EntityType.autoExpansionMode field to AUTO_EXPANSION_MODE_DEFAULT.

# Slide 19: Dialogflow CX: Entity Types & Entities

## Entity Types & Entities: Entities: Fuzzy Matching (Limited Use)

You can enable **fuzzy matching** for a **custom** entity.

- With fuzzy matching enabled, the **ordering** of the words in a value or synonym does **not** matter.

By default, entity matching requires an exact match for **one** of the entity entries.

- This works well for **single-word** entity entry values and synonyms.

- However, it may present a **problem** for multi-word values and synonyms.
  - For example:
    - Consider a **ball** entity that should be matched for the following end-user expression parts:

- "ball"
- "red ball"
- "ball red"
- "small ball"
- "ball small"
- "small red ball"
- "small ball red"
- "red small ball"
- "red ball small"
- "ball small red"
- "ball red small"

# Slide 20: Dialogflow CX: Entity Types & Entities

## Entity Types & Entities: Entities: Fuzzy Matching (Limited Use)

For a **match** to occur:

- It normally needs to define an **entity entry** value and synonyms for **each** of these permutations.

- With fuzzy matching enabled, the ordering of the words in a value or synonym does not matter.
  - The following will trigger a match for **all** of the examples above:
    - "ball"
    - "red ball"
    - "small ball"
    - "small red ball"

To **enable fuzzy matching**:

- Using the console:
  - Check the **Fuzzy matching** option.

- Using the API:
  - set the EntityType.enableFuzzyExtraction field to true