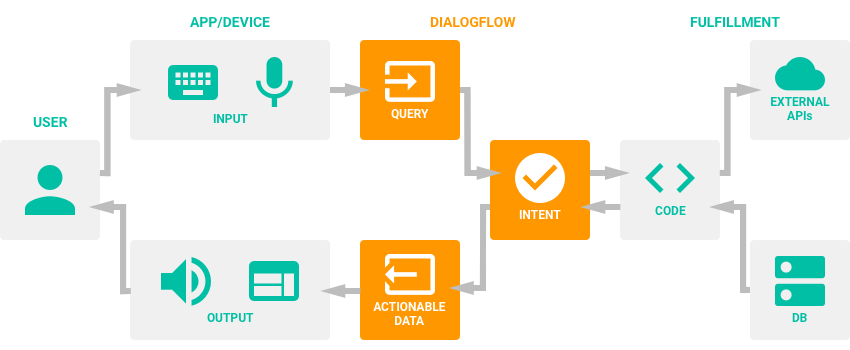
Agents

Agents are best described as NLU (Natural Language Understanding) modules. These can be included in your app, product, or service and transform natural user requests into actionable data.

This transformation occurs when a user input matches one of the intents inside your agent. [Intents](https://dialogflow.com/docs/intents) are the predefined or developer-defined components of agents that process a user’s request.

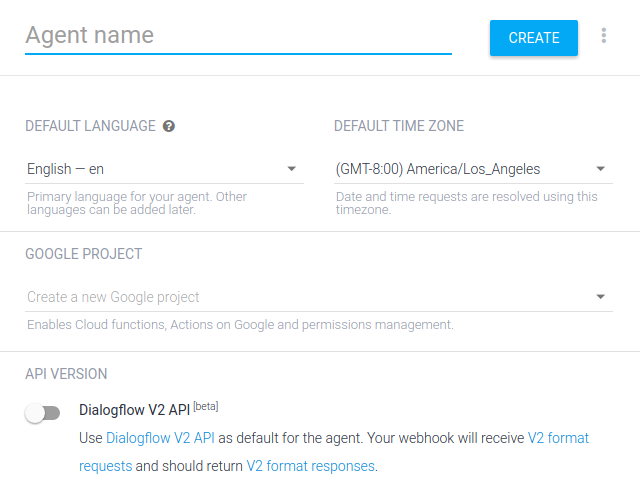
Agents can also be designed to manage a conversation flow in a specific way. This can be done with the help of [contexts](https://dialogflow.com/docs/contexts), intent priorities, slot filling, responsibilities, and fulfillment via webhook.



**Note:** Agents are platform agnostic, so you only have to design an agent once. From there, you can integrate it with a variety of platforms using SDKs and Integrations.

## **Creating**

To create an agent, click on **Create Agent** in the left menu.



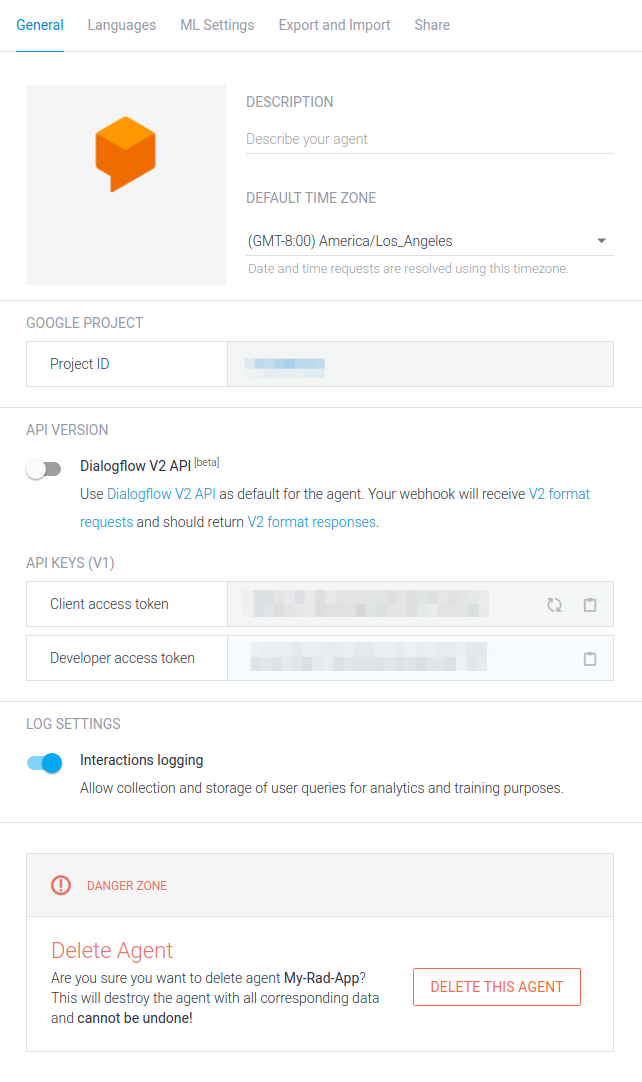
Choose a unique name for your agent and set the relevant settings.

* **Default Language** - Select the default language for your agent.
* **Default Time Zone** - Select the default time zone for your agent.
* **Google Project** - Choose an existing Google Cloud Project or leave on "Create a new Google project" to do so.
* **API Version** - Enable this option to use the beta V2 API.

**Note:** For more information, see our [comparison guide](https://dialogflow.com/docs/reference/v2-comparison).

## **Settings**

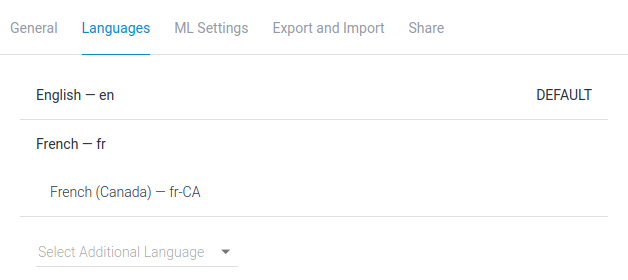
You can access the settings for the agent by clicking the gear icon settings next to the agent name.

**Note:** You will see a **Google Project** option when creating a new agent. This is a project for [Google Cloud Platform](https://cloud.google.com/). If you have an existing Google Cloud Project you can select it from the drop down list. Leaving the default option will create a new Google Cloud Project using the name of the agent.**General**

* **Description & Update Avatar** - These will only be displayed in the [Web Demo](https://dialogflow.com/docs/integrations/web-demo) for your agent.
* **Language** - The default language for your agent.
* **Default Time Zone** - Select the default time zone for your agent. This timezone will be used if you don't pass the timezone parameter in [query requests](https://dialogflow.com/docs/reference/agent/query#query_parameters_and_json_fields).
* **Google Project** - The Google Cloud Project linked to the agent. This project is created and linked by default.
* **Client Access Token** - For end-user interaction with /query, /context and /userEntites endpoints only.
* **Developer Access Token** - Used for agent modification. **Do not share**.
* **Log Settings** - This option disables logging. [Read more in Training](https://dialogflow.com/docs/training#disabling_interaction_logs).
* **Delete Agent** - Completely deletes agent (cannot be undone).

**Note:** If the agent is shared with other users, those users must be removed from the agent before you can delete it.

### **Languages**

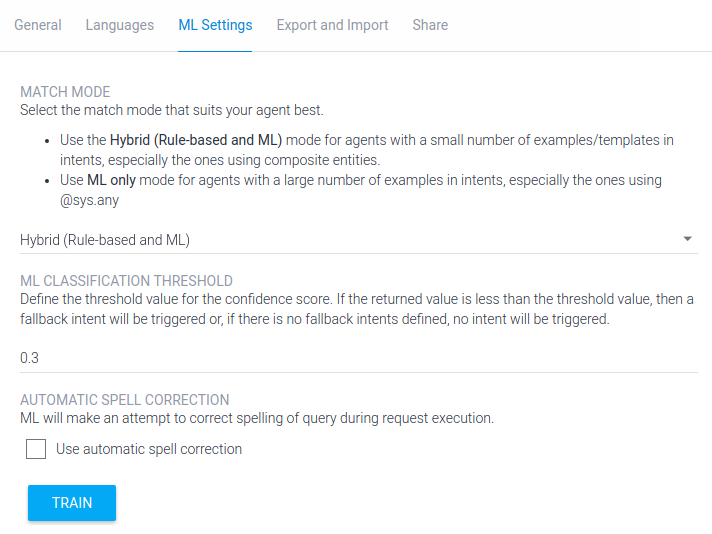


Add multiple languages and their respective locales to make your agent multilingual.

Choose a language from the list and click the **Save** button. To add a locale, if available, hover over the listed language and click **+ Add locale**.

[More on Multi-language Agent Support](https://dialogflow.com/docs/multi-language).

### **ML Settings (Machine Learning)**



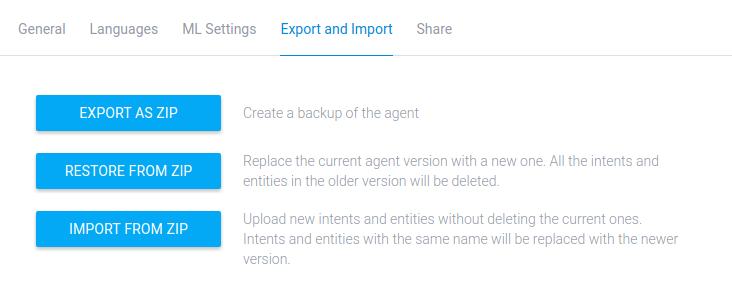
* **Match Mode** - This setting defines what algorithm should be used for machine learning. The setting applies to all the intents in which Machine Learning is enabled.
  + **Hybrid** - This mode fits best for agents with a small number of examples in intents and/or wide use of composite entities.
  + **ML Only** - This mode can be used for agents with a large number of examples in intents, especially the ones using the system entity @sys.any.
* **ML Classification Threshold** - Defines a threshold value between 0 and 1 for triggering [fallback intents](https://dialogflow.com/docs/intents#fallback_intents)and returns the "score" in the JSON response to the query. If no fallback intents are provided, no intent will be triggered.

**Note:** Fallback intents return **"score": 1** and no fallback intent returns a **"score": 0**.

* **Automatic Spell Correction** - When enabled, this allows a misspelled input from a user to be accepted for matching.

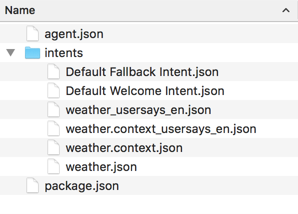
**Note:** This feature is currently only available for English agents.**Caution:** While this allows an incorrect term to be matched, the misspelled input will still be returned in the logs, training and JSON.**Warning:** Misspellings that are close to another viable term may be matched, potentially triggering the wrong intent. If you run into matching issues, try disabling this option.

### **Export and Import**



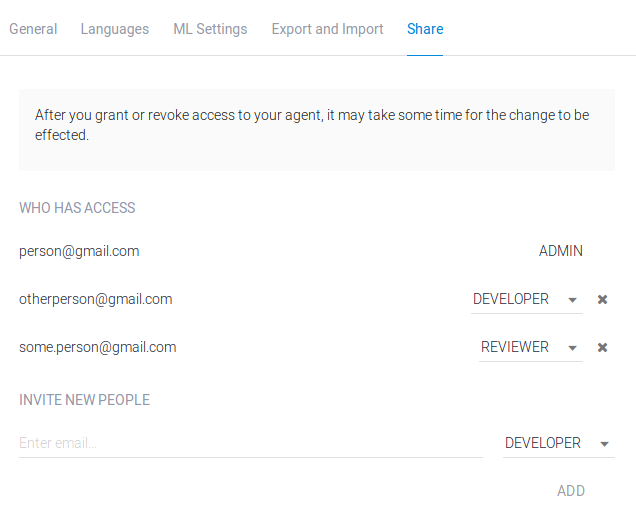
* **Export as ZIP** - Exports agent as .ZIP file (JSON includes [these fields](https://dialogflow.com/docs/reference/agent-json-fields)).
* **Restore from ZIP** - Replaces the current agent with the supplied .ZIP file.
* **Import from ZIP** - Upload intents and entities from an existing exported agent.

Below is an example of the files and structure of an exported agent:



**Note:** Import and export should be used for backing up agents or transferring them from one account to another. While you can edit the JSON files directly and re-import them, editing should be done using the developer console or API. This ensures the changes are validated by the system, and keeps troubleshooting to a minimum.

### **Share**



This page is used to share access to the agent with other Dialogflow users.

Enter the email address of the other user you want to share the agent with, choose one of the following roles for that user, click **Add**, and then click the **Save** button:

* **Developer** - Ability to edit intents, entities, fulfillment, integrations and training for the agent. This is ideal for team members like developers and linguists.
* **Reviewer** - Read-only access to intents and entities. This is ideal for project managers, legal and anyone who just needs to review content.

# Prebuilt Agents

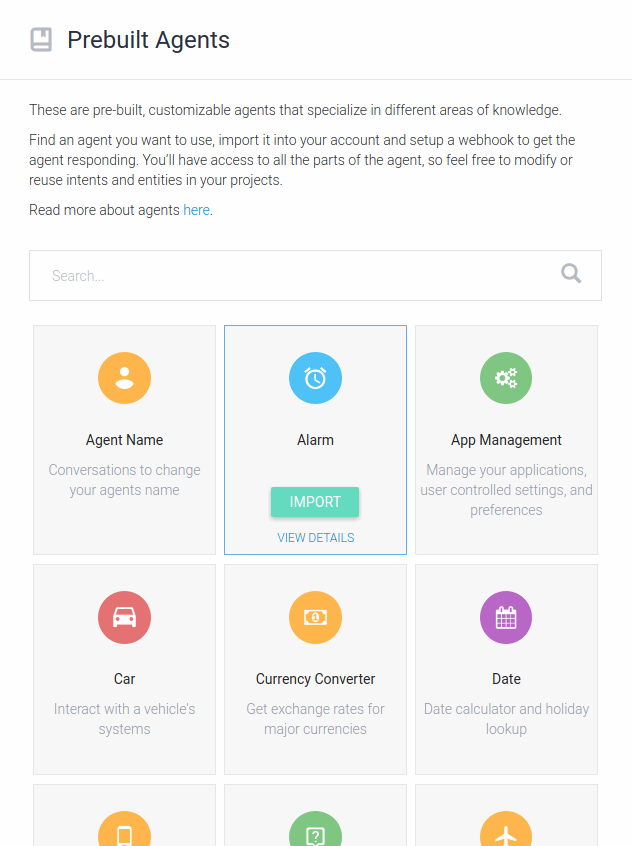
**Prebuilt agents** are a collection of agents developed by the Dialogflow team. In the developer console, you can download agents to your account that fit your use case and use them as a base for building a conversational interface for your app, bot, or device.

Prebuilt agents are available in:

* Brazilian Portuguese
* Chinese (Cantonese)
* Chinese (Simplified)
* Chinese (Traditional)
* Danish
* English
* French
* German
* Hindi
* Indonesian
* Italian
* Japanese
* Korean
* Norwegian
* Portuguese
* Russian
* Spanish
* Swedish
* Thai

## **How to Import Prebuilt Agents**

In the developer console, click **Prebuilt Agents** in the left hand menu. You can search the agents by keywords related to your desired use case. Click on the card of a specific agent for more information and examples. To import a specific agent to your account, click the **IMPORT** button.



## **Prebuilt Agents Structure**

Prebuilt agents are ready to use for user input classification, that is, transforming a user's natural language into actionable data.

Agent [responses](https://dialogflow.com/docs/intents#response) are not provided since they either may depend on particular use cases or may need to be retrieved from external sources (from a web service connected to your agent via a webhook or from external APIs or databases connected to your web service). For example, for providing weather forecast for the users, you need to retrieve it from a 3rd party Weather API.

## **How to Customize Prebuilt Agents**

Prebuilt agents are fully customizable, as you’ll have access to edit everything.

### **Customizing the Small Talk prebuilt agent**

The [Small Talk prebuilt agent](https://console.dialogflow.com/api-client/#/agent//prebuiltAgents/Small-Talk), provides causal conversation responses. This agent can be fully customized to include branding or product specific information and responses. See more info on [Small Talk](https://dialogflow.com/docs/small-talk).

**Note:** Responses for the Small Talk agent are only supported in English (en), Russian (ru), French (fr), and Italian (it) at this time.

### **Customize Topics Coverage and Classification Model**

Prebuilt agents cover the most popular user inputs. You may want to cover additional topics related to the same area or, on the contrary, cover less user questions. For these purposes, you can add or delete [intents](https://dialogflow.com/docs/intents) and [entities](https://dialogflow.com/docs/entities).

You can copy or move intents and entities between agents, as well as delete intents and entities in bulk. For more information, read the respective sections about batch operations with [intents](https://dialogflow.com/docs/intents#batch-operations) and [entities](https://dialogflow.com/docs/entities#batch-operations).

### **Customize Agent’s Responses**

#### **Defining Responses in Intents**

For simple use cases, you can define agent [responses](https://dialogflow.com/docs/intents#response) directly in intents. If you use our one-click integrations with some messaging platforms (currently, [Actions on Google](https://dialogflow.com/docs/integrations/actions-on-google), [Facebook Messenger](https://dialogflow.com/docs/integrations/facebook), [Kik](https://dialogflow.com/docs/integrations/kik), [Slack](https://dialogflow.com/docs/integrations/slack), [Telegram](https://dialogflow.com/docs/integrations/telegram), [Viber](https://dialogflow.com/docs/integrations/viber), and [Skype](https://dialogflow.com/docs/integrations/skype)), you can define responses not only as text, but as [Rich Messages](https://dialogflow.com/docs/rich-messages) as well.

#### **Defining Responses in Your Fulfillment Service**

You can consider creating your web service and connecting it to your agent via [webhook](https://dialogflow.com/docs/fulfillment). In your web service, you can define responses based on the actionable data received from your agent, perform operations, and retrieve information from external sources (databases or 3rd-party APIs).

If you already have your backend service, you may prefer sending [query](https://dialogflow.com/docs/reference/agent/query) requests to your agent and defining responses in your business logic. See the [list of our SDKs](https://dialogflow.com/docs/sdks) designed for sending query requests and getting responses in different platforms/languages.

# Multi-language Agents

This feature lets you support multiple root languages and locales within the same agent, allowing you to specify different **Training Phrases** and **Responses** for each language your agent supports.

In most cases, your intent structure and parameters won't change from language to language, so you'll just add additional languages to the same agent and customize the **Training Phrases** and **Responses** as needed. If you do need to change the intent structure or parameters to facilitate another language, you should setup a new agent. This makes it easier to manage the differences, since [newly created intents and entities are copied](https://dialogflow.com/docs/multi-language#adding_an_intent_or_entity_to_a_multi-language_agent) to all languages.

## **Root languages and locales**

Dialogflow agents support [15 root languages](https://dialogflow.com/docs/reference/language), which are used for the main development of your app.

Some root languages include locale languages, which cover region or country specific versions of a root language. For example, French is a root language, but there are differences between how it's spoken in France and French Canadian provinces.

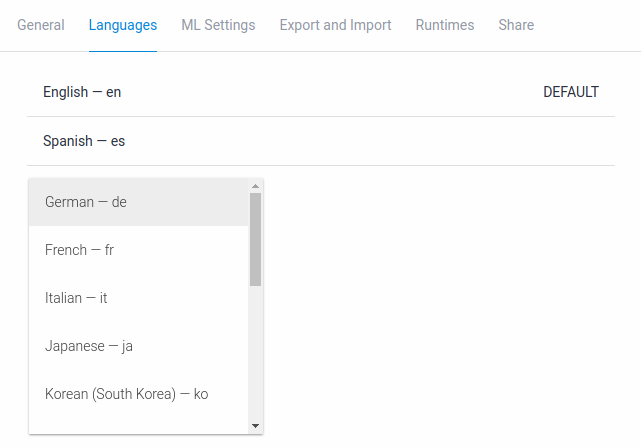
Ideally, you should develop primarily around a common root and only customize for specific locales if needed.

### **Adding a root language**

**Key Point:** As a best practice you should create all of the intents and entities you think you'll need, prior to adding an additional root language. Once a root language is added, all subsequent intents and entities you create, will need to be created in all languages.

Whether you’re creating an agent or already have an existing one, you can add root languages within the [agent’s settings](https://dialogflow.com/docs/agents#settings). To add a root language to an agent:

1. Click on the **gear icon** next to the agent name.
2. Click on **Languages** tab.
3. Click on the **Select Additional Language** dropdown and choose a language from the list.
4. Click the **Save** button.



**Note:** When you add a new root language, copies of all the intents and entities in your agent will be added into the new language, but these copies will not have existing data like **Training Phrases** and **Responses**. You'll need to add localized data for each selected language.

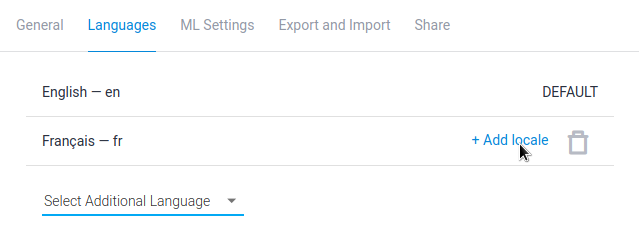
### **Adding a locale**

**Key Point:** As a best practice you should create all of the intents and entities you think you'll need, prior to adding locales. Once a locale is added, all subsequent intents and entities you create, will need to be created in all locales.**Note:** If a specific locale is passed to an agent, and that agent only has a root language set, the root language related to the locale will handle the request. For example, if **en-GB** is sent as the language in a request to an agent that only uses **en**, the request will be handled by **en**.

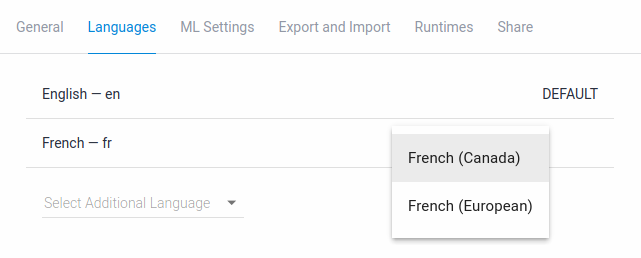
Additional locales (if available) can be added to encompass different versions of a root language.

To add a locale:

1. Hover over a root language and click **+ Add locale**. Available locales will be listed.



1. Click on a listed locale to add it. This will show up as a separate language for your agent.

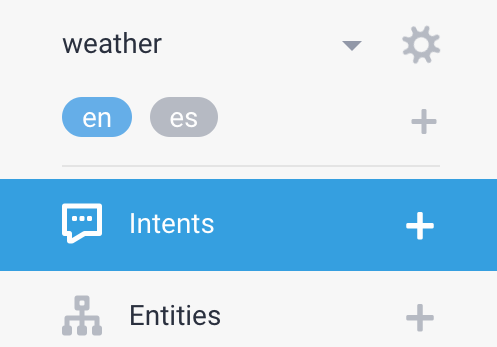


1. Click the **Save** button.

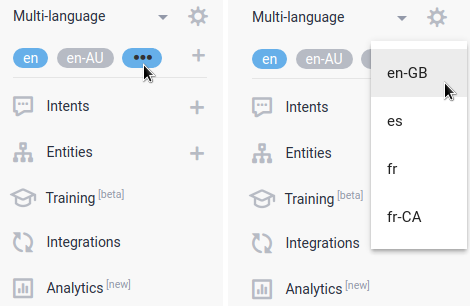
**Note:** When you add a locale to a root language, copies of all existing intents and entities in your agent will be added into the new language, along with their existing data like **Training Phrases** and **Responses**. Intents and entities added after the locale will be empty.

## **Testing languages in the console**

When testing in the Dialogflow test console, you can only test the selected language for the agent. For example, in the image below, English is the currently selected language. If you need to test in other languages, click on the corresponding language chip (**es** in the image) to switch languages.



Language chips will overflow into a "more" chip. This list is alphabetical, with the currently selected language moved to the first position.



## **Testing languages via API**

When sending requests via the API, the language is set by the lang parameter. This should correspond to the listed language in [this reference page](https://dialogflow.com/docs/reference/language).

{  
  "lang": "en-GB",  
  "query": "I need apples",  
  "sessionId": "12345"  
}

## **Importing and exporting a multi-language agent**

When [exporting](https://dialogflow.com/docs/agents#export) a multi-language agent, the generated ZIP file contains the same folder structure as a single language agent. There is one JSON file that represents the intent and one file per language for **Training Phrases**. For instance, an intent named "weather" that uses English and Spanish would include weather.json, weather\_usersays\_en.json, and weather\_usersays\_es.json.

Exported entities will also include two files. If you have an entity called "cities", it would include cities.jsonand cities\_entries\_es.json.

[Importing](https://dialogflow.com/docs/agents#export) an agent, whether multi-language or not, now specifies one language as the default language. This is the language specified when you create the agent.

## **Adding an intent or entity to a multi-language agent**

When you add a new [intent](https://dialogflow.com/docs/intents) or [entity](https://dialogflow.com/docs/entities) to an agent that is already supporting more than one language, the newly created intent or entity gets added to all added languages. Similar to the behavior when adding a new language, adding a new intent or entity in one language only creates a blank version in all other languages and doesn't copy any other data automatically.

# Intents

An **intent** represents a mapping between what a user says and what action should be taken by your software.

Intent interfaces have the following sections:

* Training Phrases
* Action
* Response
* Contexts

This agent has three intents with specific [Training Phrases](https://dialogflow.com/docs/intents#user-says) that will trigger the intent.

|  |  |
| --- | --- |
| Intent | Training Phrases |
| Dog | "dog", "pooch", "puppy" |
| Cat | "cat", "kitty", "kitten" |
| Mouse | "mouse" |

**To try it out, enter one of the**[**Training Phrases**](https://dialogflow.com/docs/intents#user-says)**to trigger an intent.**

You can also try entering something different to see the [Fallback intent](https://dialogflow.com/docs/intents#fallback_intents) pick up on unrecognized entries.

## **Training Phrases**

### **Example (“) and Template (@) Modes**

Each **User says** expression can be in one of two modes: Example Mode (indicated by the " icon) or Template Mode (indicated by the@ icon).

**Examples** are written in natural language and annotated so that parameter values can be extracted. You can read more on annotation below.

**Templates** contain direct references to [entities](https://dialogflow.com/docs/entities) instead of annotations, i.e., entity names are prefixed with the @sign.

To toggle between modes, click on the " or @ icon.

We recommend using examples rather than templates, because it’s easier and Machine Learning learns faster this way. And remember: the more examples you add, the smarter your agent becomes.

## **Example Annotation**

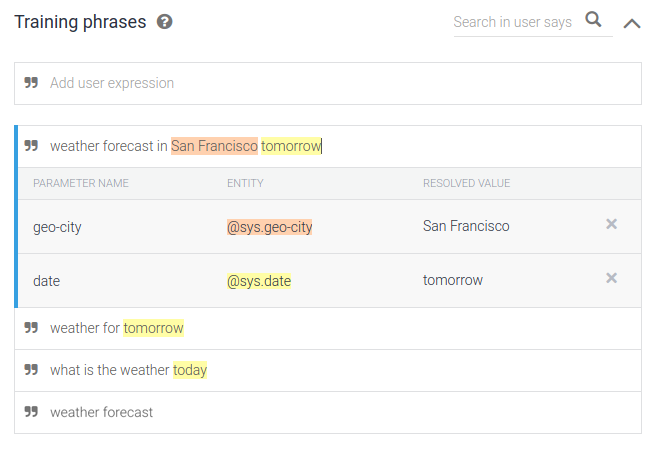
**Annotation** is a process (and also the result of such process) of linking a word (or phrase) to an entity.

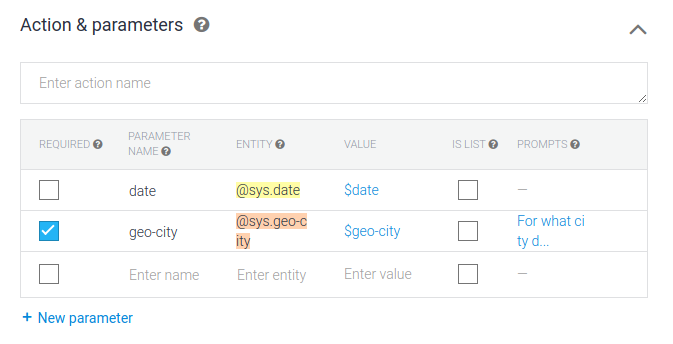
### **Automatic Annotation**

When you add examples to the ‘User says’ section, they are annotated automatically. The system detects the correspondence between words (or phrases) and existing developer and system entities and highlights such words and phrases. It also automatically assigns a parameter name to each detected entity.

### **Editing Automatically Annotated Examples**

You can edit the linked entity and the parameter name assigned to it in either the review window that opens when you click on the annotated example, or the parameter table of the ‘Action’ section.





The results of the automatic annotation in the review window and parameter table are synchronized. If you change something in the review window, the respective changes will automatically occur in the parameter table, and vice versa.

Note that changes made in the review window and in the parameter table have different scopes:

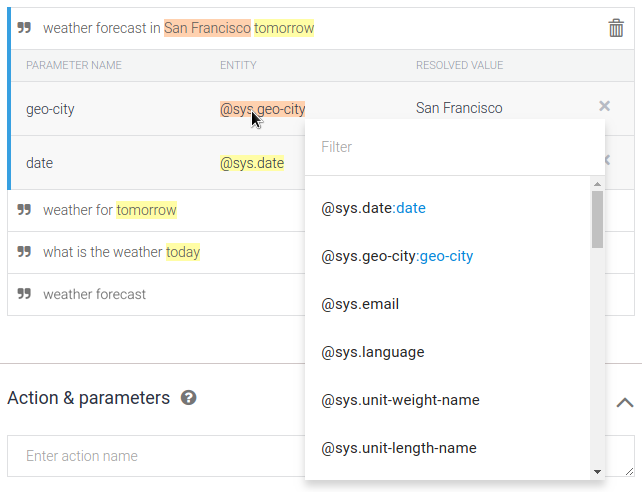
* Changes in the review window won’t affect other examples containing the same annotations.
* Changes in the parameter table will affect all ‘User says’ examples with the same annotation.

You can do 3 type of changes:

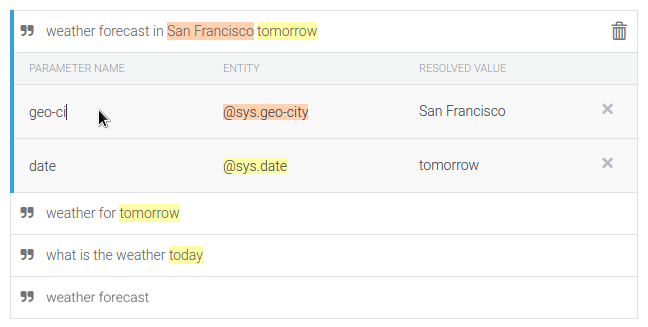
* Assign a different entity to an annotated part of the example
* Edit the parameter name
* Delete the annotation (i.e., delete the link between the word and the entity).

**Local changes (in one example)**

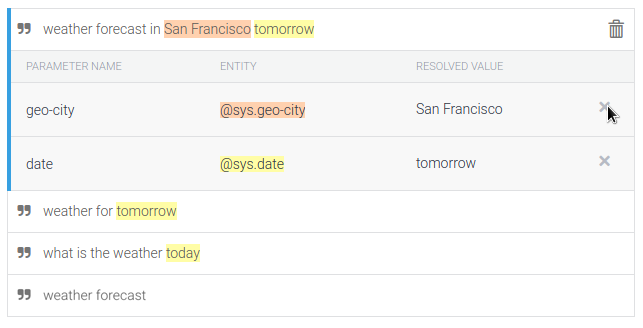
* To assign a different entity to the annotation in one example, click on the highlighted phrase. A pop-up window will appear, where you can choose from the list of existing system or developer entities.



* In order to change a parameter name in one example, click on the example and edit the parameter name in the review window.

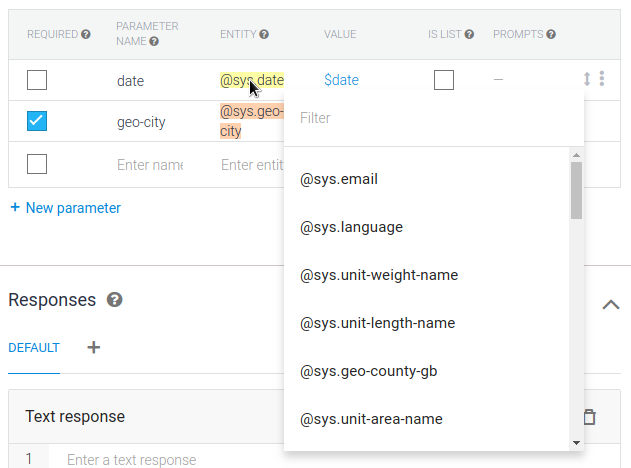


* To delete the annotation, click on either the bin icon in the pop-up window or the **X** icon in the corresponding row.

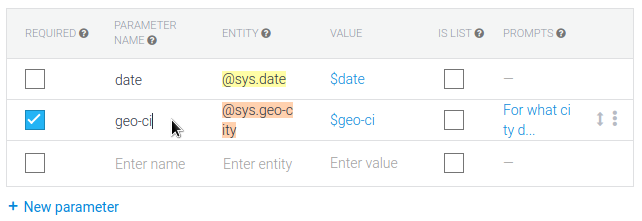


**Changes for entire intent (in the parameter table)**

* To assign a different entity to the parts highlighted with the same color in all examples, click on the entity in the parameter table. In the pop-up window, choose from the list of the existing system or developer entities.



* To change the parameter name for the annotations through all examples, edit the parameter name in the parameter table.



* To delete a particular annotation from all examples, click on the menu icon on the right side of the respective row in the parameter table and select **Delete** from the drop-down menu.

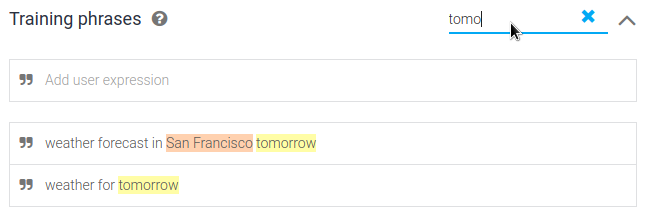


### **Manual Annotation**

If necessary, you can annotate examples manually by selecting a word or phrase and choosing an existing entity or creating a new one in the pop-up window.

## **Search Option**

If you need to find a specific example or template, you can search for keywords in the **User says** section.



## **Action**

This section consists of the action name field and the parameter table.

The **action name** is defined manually. It will be the trigger word for your app to perform a particular action.

**Parameters** can be filled in automatically from the ‘Users says’ examples and templates, or added manually.

Read more in [Action and Parameters](https://dialogflow.com/docs/actions-and-parameters).

## **Response**

In this section, you can define **your agent’s responses** which will be provided by your application when the intent is triggered.

### **Text Response**

You can improve your agent eloquence by adding several variations of the text response per intent. When the same intent has been triggered more than once, different text response variations will be unrepeatable until all options have been used. It'll help make your agent speech more human-like.

#### **References to Parameter Values**

Responses can contain **references to parameter values**.

If a parameter is present in the parameter table, use the following format to refer to its value in the ‘Text response’ field: $parameter\_name.

There are special parameter value types that don’t appear automatically in the parameter table.

If you need to refer to such a special type of value, you'll have to add a new parameter to the parameter table, define its value manually, and then reference it in the response as $parameter\_name.

Use the following formats:

* $parameter\_name.original – to refer to the original value of the parameter
* $parameter\_name\_for\_composite\_entity.inner\_alias – to refer to a value of one of the composite entity components
* #context\_name.parameter\_name – to reference a parameter value collected in some other intent with defined context

#### **Special Characters**

If you need the **dollar sign $** or the **number sign #** to appear in your agent's text response, use braces around the value that follows $ or #. For example: ${100} – where 100 is a constant value ${$number} – where $number is a reference to the parameter value #{channel} – where channel is a string #{#channel.name}– where #channel.name is the reference to the parameter value "name" from the context "channel"

If you need to use braces in text responses, use double braces: {{ will be displayed as { and }} will be displayed as }.

#### **Handling Empty Parameter Values**

If an intent is designed in such a way that some parameters can return empty values after the intent has been triggered, the variants of Text response that contain references to the parameters with empty values won't be given as text responses. Make sure to define different variations of Text response for such intents.

For example, if an intent has 2 parameters and may return both or any of them with empty value, and you want to reference parameter values in Text response, make sure to define at least 4 variants of Text response:

* referencing both parameter values,
* referencing the 1st parameter value,
* referencing the 2nd parameter value,
* without any reference to the parameter values.

#### **Emojis**

If you want your agent to display emojis in responses, just copy and paste a desired emoji in the **Response** field.

### **Rich Messages**

If you use one of the following one-click integrations – Facebook Messenger, Kik, Slack, or Telegram – you can define your bot responses as rich messages (images, cards, quick replies etc.) directly in intents.

For more information, please read the [documentation on Rich Messages](https://dialogflow.com/docs/rich-messages).

## **Contexts**

**Contexts** are designed for passing on information from previous conversations or external sources (e.g., user profile, device information, etc). Also, they can be used to manage conversation flow.

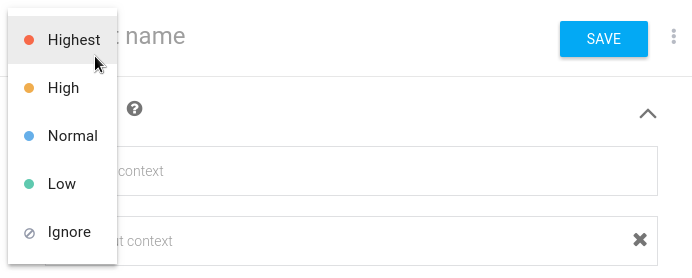
To define the contexts, click on ‘Define contexts’ right below the intent name.

Input contexts serve as a prerequisite for the intent to be matched; i.e., the intent will participate in matching only when **all** the contexts in the input context field are active.

[Read more on contexts](https://dialogflow.com/docs/concept-contexts).

## **Intents priority**

Intent priority allows you to assign more weight to one of the intent in case an input phrase matches multiple intents. Intents priority can be changed by clicking on the blue (default) dot on the left of the intent name and selecting the priority from the drop-down menu.



## **Fallback Intents**

Fallback intents are triggered if a user's input is not matched by any of the regular intents or enabled built-in Small Talk.

When you create a new agent, a default fallback intent is created automatically. You can modify or delete it if you wish.

If you’d like to create contingency actions or responses for various input contexts, you can add your own fallback intents. Make sure to set a unique set of input contexts for each fallback intent.

To add a new fallback intent:

* Go to **Intents** from the left side menu
* Click on the three dot “more options” menu next to ‘Create Intent’
* Choose ‘Create Fallback Intent’



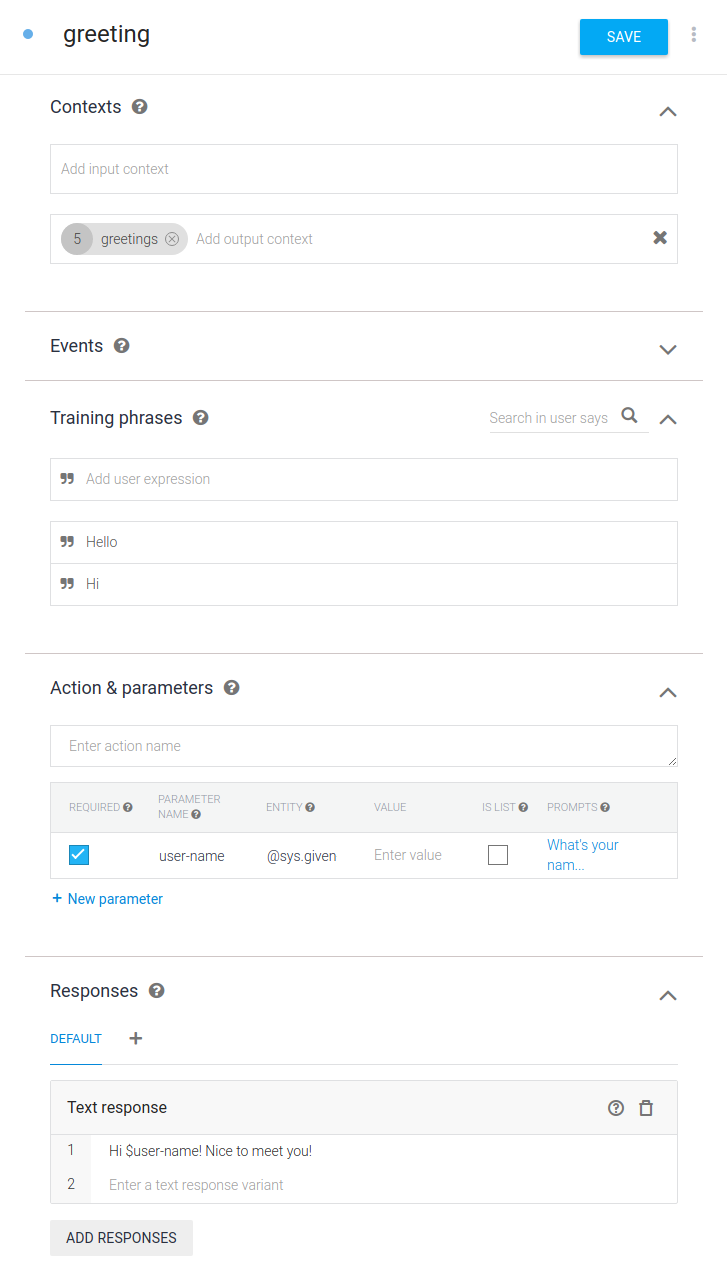
Fallback intents can reference parameter values from context directly in the ‘Text response’ field in this format: #context\_name.parameter\_name. See example [below](https://dialogflow.com/docs/intents#referencing_parameter_values_in_fallback_intents).

If you’d like certain phrases and their variations to be classified as a fallback intent, you need to create a regular intent, add those phrases to the ‘User says’ section, and define the same action and Text response as in the fallback intent.

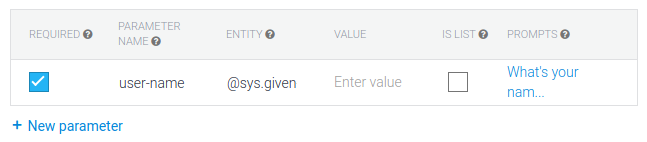
## **Referencing Parameter Values in Fallback Intents**

The following example shows how to use parameter values from context in the fallback intent.

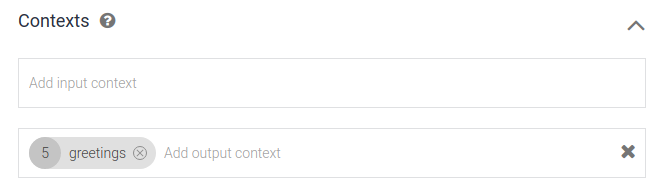
First, we create a regular intent, ‘Greetings’.



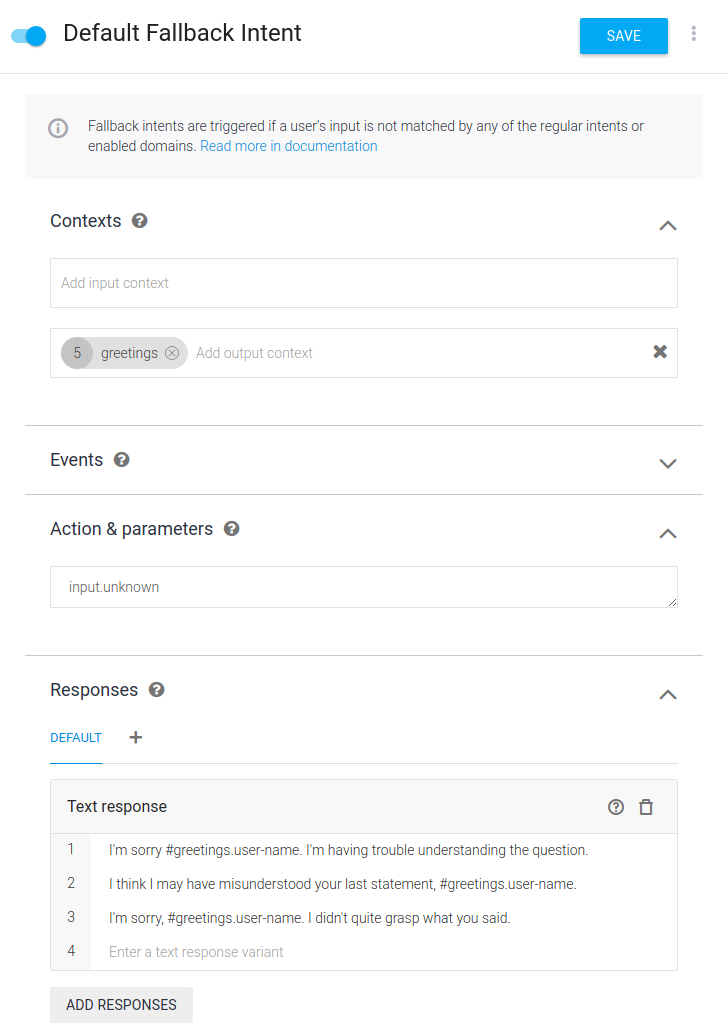
You'll collect the user’s name in the parameter ‘user-name’ using the slot filling feature (marking the parameters as required and defining a prompt to inform the user what info you're looking for).



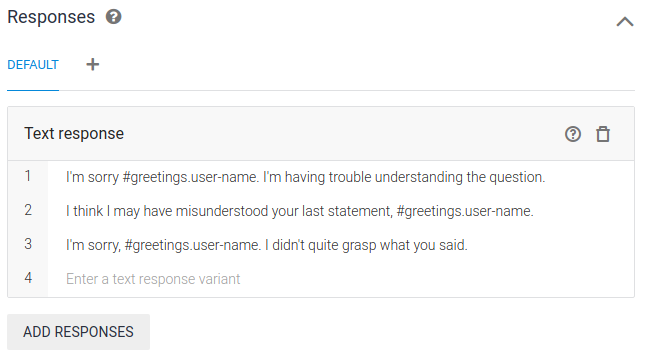
We also define an output context ‘greetings’ to store the parameter value.



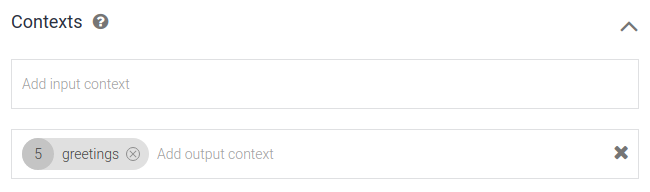
Then, we make some modifications in the Default Fallback Intent which was automatically created with the agent.



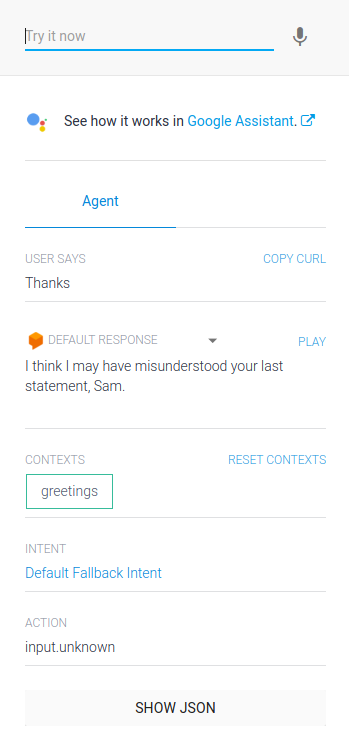
We’ll reference the user’s name as #greetings.user-name in the Text response variations.



To keep the ‘greetings’ context alive, we add it as an output context to this intent as well.



Now, any input other than greetings will be classified as Default Fallback Intent.



## **Follow-up intents**

Natural human dialogs are filled with follow-ups and confirmations, for example:

User: Get me a cab.   
Agent: Your cab can meet you at the corner of 5th and Broad street in 12 mins.   
User: Ahh, can you repeat that please?

or

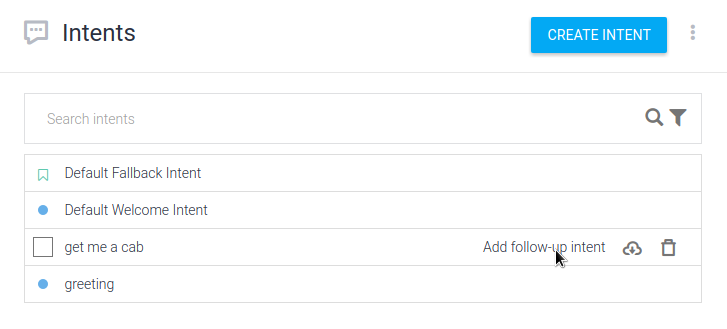
User: I want 2 hamburgers and one Cola.   
Agent: Do you want fries with that?   
User: Sure, sounds good, thanks

**Follow-up intents** make these types of natural conversation flows easy to build and customize. With this feature of Dialogflow you can define dialog situations that may follow any given “parent” intent you are currently working on.

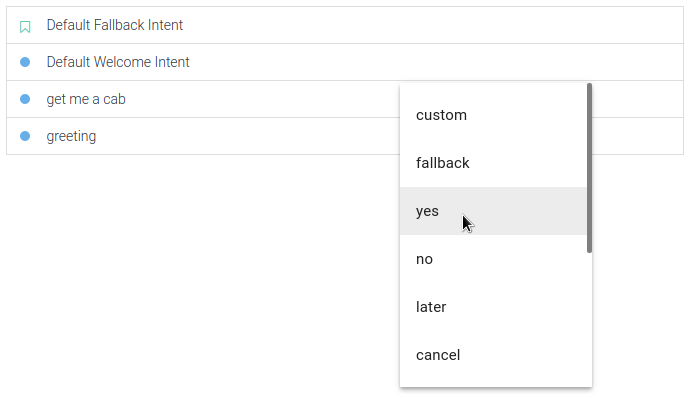
While you can also do this (and much more) by manually defining contexts, follow-up intents make the process of building sample dialogs easier and more visual. We’ve thought of most popular follow-up scenarios and taken care of building the [contexts](https://dialogflow.com/docs/contexts) for you automatically.

### **Adding a Follow-up Intent**

To start building Follow-up intents, go to the ‘Intents’ page, hover the mouse over intent names, tap “Add follow-up intent” link.



Select one of the [available follow-up intents](https://dialogflow.com/docs/reference/follow-up-intent-expressions):



Follow-up intents are nested under parent intents which makes it easier to see your agent conversation structure. To see the follow-up click on the arrow next to the parent intent. You can nest multiple follow-up intents under one another if your use case requires.

### **Working with Follow-up Intents**

Follow-up intents are like any other intent in Dialogflow: you can edit their expressions, responses, and fulfilment methods. See our documentation on intents for more information about the actions you can perform.

### **Follow-ups as Fallback Intents**

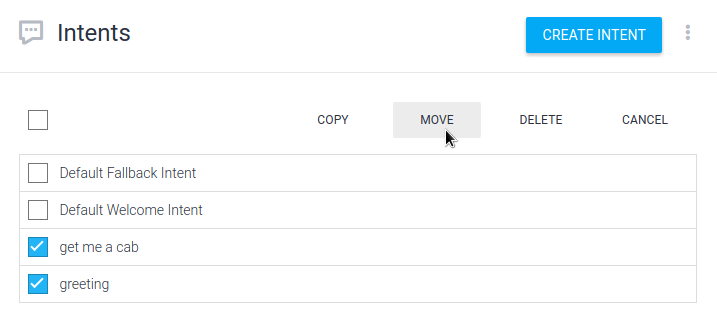
A follow-up intent can act as a fallback if no other expression with the parent context is triggered. This can be used for handling unexpected user input. For more information about how to use [fallbacks in our documentation](https://dialogflow.com/docs/intents#fallback_intents).

## **Batch operations with intents**

For efficiency you can perform operations on multiple intents at once.

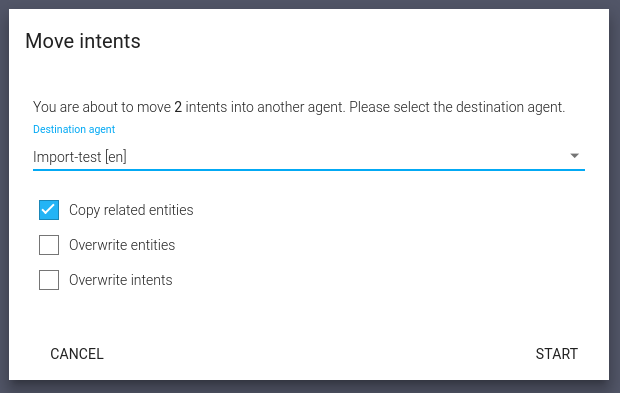
### **Copying or moving multiple intents**

To copy or move intents into another agent, go to ‘Intents’ in the left hand menu and hover the mouse over the intent list. You’ll see checkboxes on the left of the intent names. Check one of them to enable the batch operations tool. Then select the intents you want to copy or move to another agent. If you need to copy or move all intents, select the checkbox at the top of the list. Once all the desired items are selected, click ‘COPY’ or ‘MOVE’.



**Note:** Using the ‘MOVE’ option will remove the intents from the current agent and move them into the selected destination agent. If you want to keep them in the current agent, use the ‘COPY’ option.

Choose the destination agent from the drop down list. This will list all of the agents in your account in the alphabetical order.



Check any additional options you’d like to perform as part of the copy or move and click ‘START’.

Batch operation options for intents include:

* Copy related entities – This is generally an option you want to check, as it will also copy or move over entities related to the intents being copied or moved.
* Overwrite entities – This option will overwrite entities with the same name.
* Overwrite intents – This option will overwrite intents with the same name.

When all selected intents are copied/moved, you can either go to the destination agent by clicking ‘PROCEED TO AGENT’ or stay in the current agent by clicking ‘DONE’.

### **Deleting multiple intents**

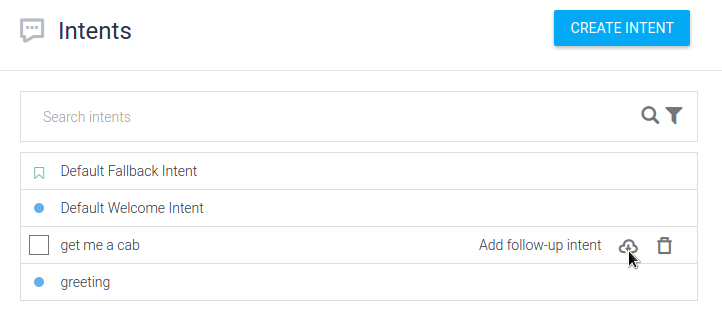
To delete multiple intents, enable the batch operations tool by hovering the mouse over the intent list and checking one intent. Then select additional intents you want to delete. If you want to delete all of the intents, check the checkbox at the top of the list. Once all of the desired items are selected, click ‘DELETE’. You will be asked to confirm this action by clicking the ‘DELETE’ button.

**Warning:** Please note that this cannot be undone! Deleting intents or entities is permanent! Check the number of intents you are about to delete before confirming the action.

## **Download and Upload Intents**

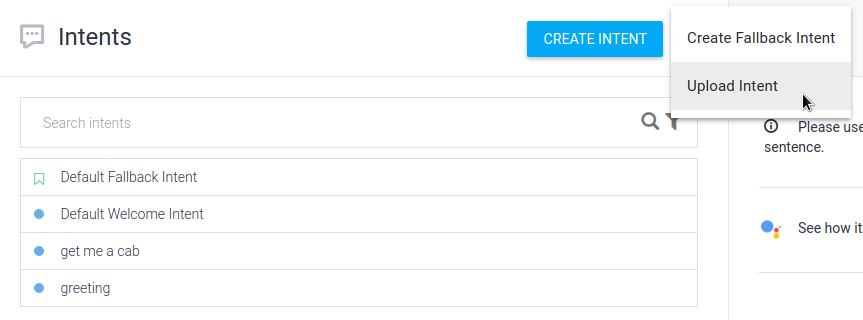
### **Download Intents**

You can download intents in JSON format. To do so, go to **Intents** from the left side menu, move the cursor over the intent, and click the little cloud sign.



### **Upload Intents**

You can upload intents as JSON files. To do so, click on the **More options** button next to the **Create Intent** button, choose **Upload intent**, and follow the instructions on the page.



# Entities

Entities are powerful tools used for extracting [parameter values](https://dialogflow.com/docs/actions-and-parameters#parameters) from natural language inputs. Any important data you want to get from a user's request, will have a corresponding entity.

The entities used in a particular agent will depend on the parameter values that are expected to be returned as a result of the agent functioning. In other words, a developer does not need to create entities for every possible concept mentioned in the agent – only for those needed for actionable data.

There are 3 types of entities: [system](https://dialogflow.com/docs/entities#system_entities) (defined by Dialogflow), [developer](https://dialogflow.com/docs/entities#developer_entities) (defined by a developer), and [user](https://dialogflow.com/docs/entities#user_entities) (built for each individual end-user in every request) entities. Each of these can be classified as mapping (having reference values), enum (having no reference values), or composite (containing other entities with aliases and returning object type values) entities.

## **System Entities**

[System Entities](https://dialogflow.com/docs/reference/system-entities) are pre-built entities provided by Dialogflow in order to facilitate handling the most popular common concepts. Below are examples of the different types of system entities.

This agent uses the [@sys.color](https://dialogflow.com/docs/reference/system-entities#other) system entity to find the color in your reply.

**Enter your favorite color.**

On the surface it looks like the agent just repeated what you said, but the intent uses the @sys.color system entity which allows it to pick out the color you mentioned.

If you had said “My favorite color is Crimson”, the agent would still be able to find the color, since "My favorite color is ..." is provided as a [Training Phrase](https://dialogflow.com/docs/intents#training-phrases).

A reply that includes a color not registered with the system entity or a word that isn't a color, will trigger the [fallback intent](https://dialogflow.com/docs/intents#fallback-intents).

### **System Mapping**

These system entities have reference values. For example, @sys.date matches common date references such as "January 1, 2015" or "The first of January of 2015" and returns a reference value in ISO-8601 format: 2015-01-01T12:00:00-03:00.

### **System Enum**

These entities have no reference value. For example, @sys.color matches most popular colors and returns the matched color as it is without mapping it to any reference value. For example, shades of red, such as "scarlet" or "crimson", won't be mapped to "red" and will return their original values "scarlet" and "crimson".

### **System Composite**

These entities can contain other entities with aliases and return object type values. For example, @sys.unit-currency is meant for matching amounts of money with indication of currency name, e.g., "50 euros" or "twenty dollars and five cents". It returns an object type value consisting of two attribute-value pairs: {"amount":50,"currency":"EUR"}

## **Developer Entities**

You can create your own entities for your agents, either through web forms, uploading them in JSON or CSV formats, or via API calls.

**Note:** Developer entity names are unique for each agent. Entity names should start with a letter and can contain the following: A-Z, a-z, 0-9, \_ (underscore), - (dash).

This agent uses a custom developer entity to recognize coffee drinks.

**Enter your favorite coffee house beverage.**

Developer entities can and should be used for anything that's not covered by system entities, and should contain values you can expect from your users.

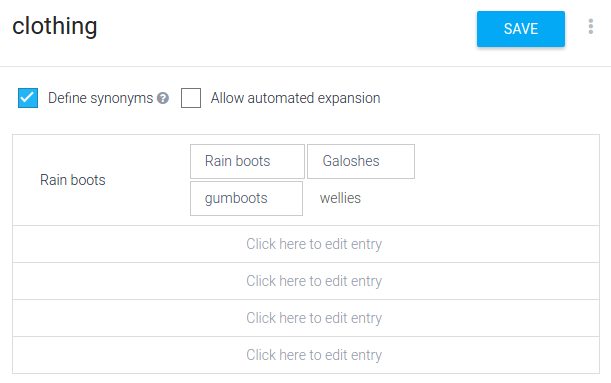
Like [Training Phrases](https://dialogflow.com/docs/intents#training-phrases), the more entities you provide, the more your agent will understand.

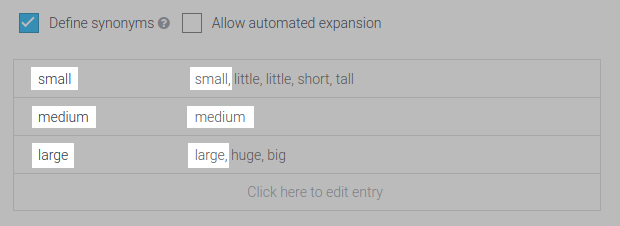
Providing a reply that doesn't include a coffee drink will trigger the [fallback intent](https://dialogflow.com/docs/intents#fallback-intents).

### **Dev Mapping**

These entities allow for the mapping of synonyms to a reference value. For example, a food type entity could have an entry with a reference value of "vegetarian" with synonyms of "veg" and "veggie".

To create a mapping entity, leave the checkbox **Define Synonyms** checked and add a reference value and synonyms.

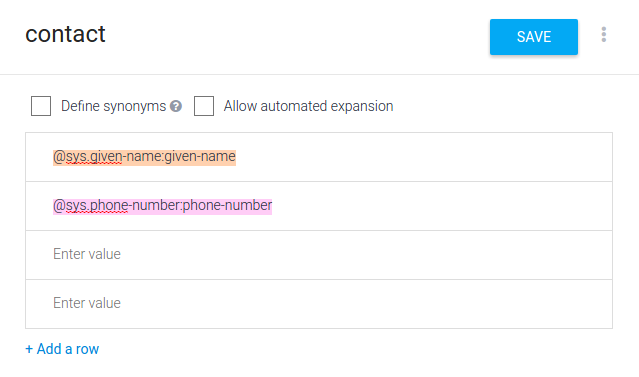


**Note:** In order for a reference value to be matched, it needs to be included as a synonym itself. When you first enter a reference value, it will be automatically added as a synonym. If for some reason this is removed, make sure to add the original value as a synonym so it can be matched.

### **Dev Enum**

Enum type entities contain a set of entries that do not have mappings to reference values. Entries can contain simple words or phrases, or other entities.

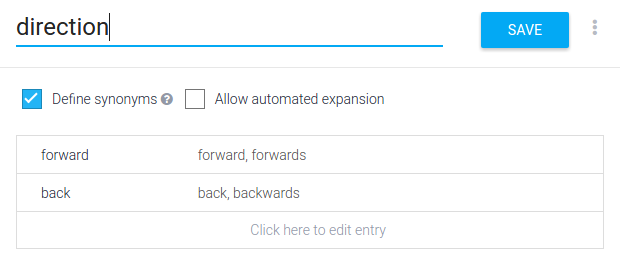
To create an enum type entity, uncheck the **Define Synonyms** option and add entries.



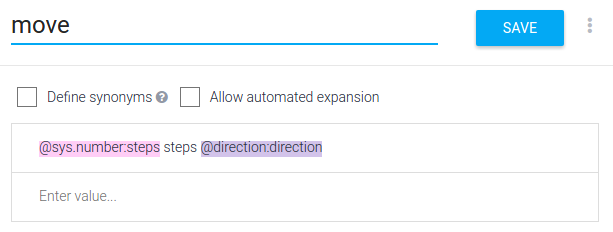
### **Dev Composite**

When an enum type entity contains other entities and they are used with aliases, we call it a composite entity. Composite entities are most useful for describing objects or concepts that can have several different attributes.

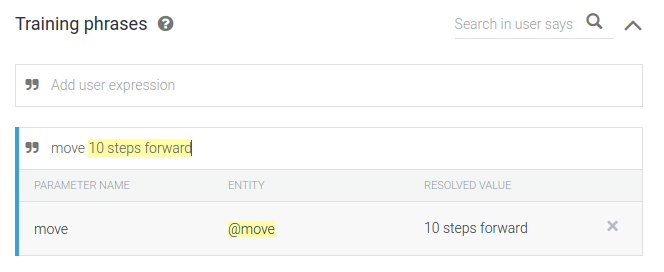
For example, a robot's movement can have two characteristics: direction and number of steps. In order to describe all possible combinations, you first need an entity that captures the direction.



Then you create a second entity that refers to the direction and the number of steps. Make sure to uncheck **Define Synonyms**.



By providing the example "Move 10 steps forward", the @move entity is annotated automatically.



## **User Entities**

User entities can be redefined on a session ID level, allowing for specific concepts, like a user's playlists.

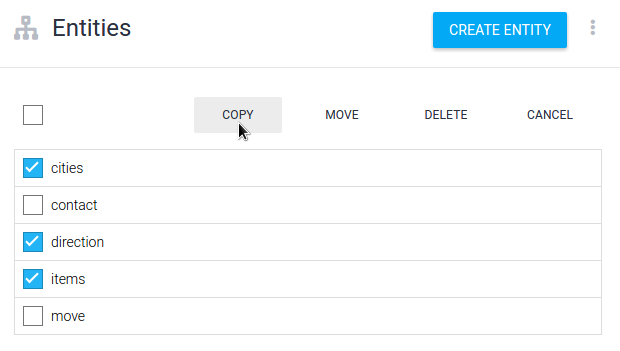
See [/userEntities](https://dialogflow.com/docs/reference/agent/userentities) for more information.

**Note:** User entities and sessions last 30 minutes.

## **Batch Operations**

You can move, copy, or delete multiple entities at once using batch operations.

To select multiple entities, click on the **Entities** section in the left menu, hover over the list and check one or more entities. This will reveal the options available for the selection. Choose the desired action and check the desired related options.



| Operation | Description |
| --- | --- |
| **Copy** | This will keep the selected entities in the current agent, and move copies into the destination agent. |
| **Move** | This option will remove the selected entities from the current agent, and move them into the destination agent. |
| **Delete** | This will permanently delete the selected entities. |

**Caution:** Use the **delete** option with caution, as it cannot be undone.

Additional options for copy and move operations include:

* **Copy related entities** - This is generally an option you want to check, as it will move or copy entities used in the composite entities being affected.
* **Overwrite entities** - This option will overwrite entities with the same name.

## **Export/Upload Entities**

### **Export**

You can download entities in either JSON or CSV format. To do this, click on **Entities** in the left menu, hover over the entity, and click on the cloud/download icon cloud\_download. This will prompt you to choose a format for the download.

### **Upload**

You can upload entities in JSON or CSV format. To do this, click on **Entities** in the left menu, click on the More icon more\_vert, then **Upload entity** and choose the file you want to upload.

#### **JSON Format**

The JSON file should correspond to the [entity object format](https://dialogflow.com/docs/reference/agent/entities#entity_json_object).

#### **CSV Format**

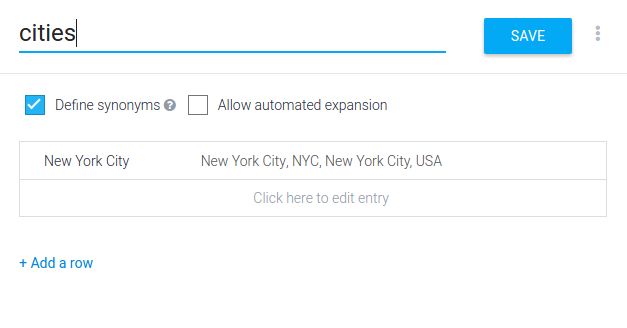
The CSV file should have the following format:

* Each entry corresponds to a new line.
* The reference value and synonyms should be separated by commas.
* Each reference value and synonym should be enclosed in double-quotes.
* The reference value should be at the beginning of the line.
* Include the reference value twice of you want it to be matched by the entity.

For example, this is an item in a CSV:

"New York City", "New York City", "NYC", "New York City, USA"

And this is the result, once uploaded to an agent:

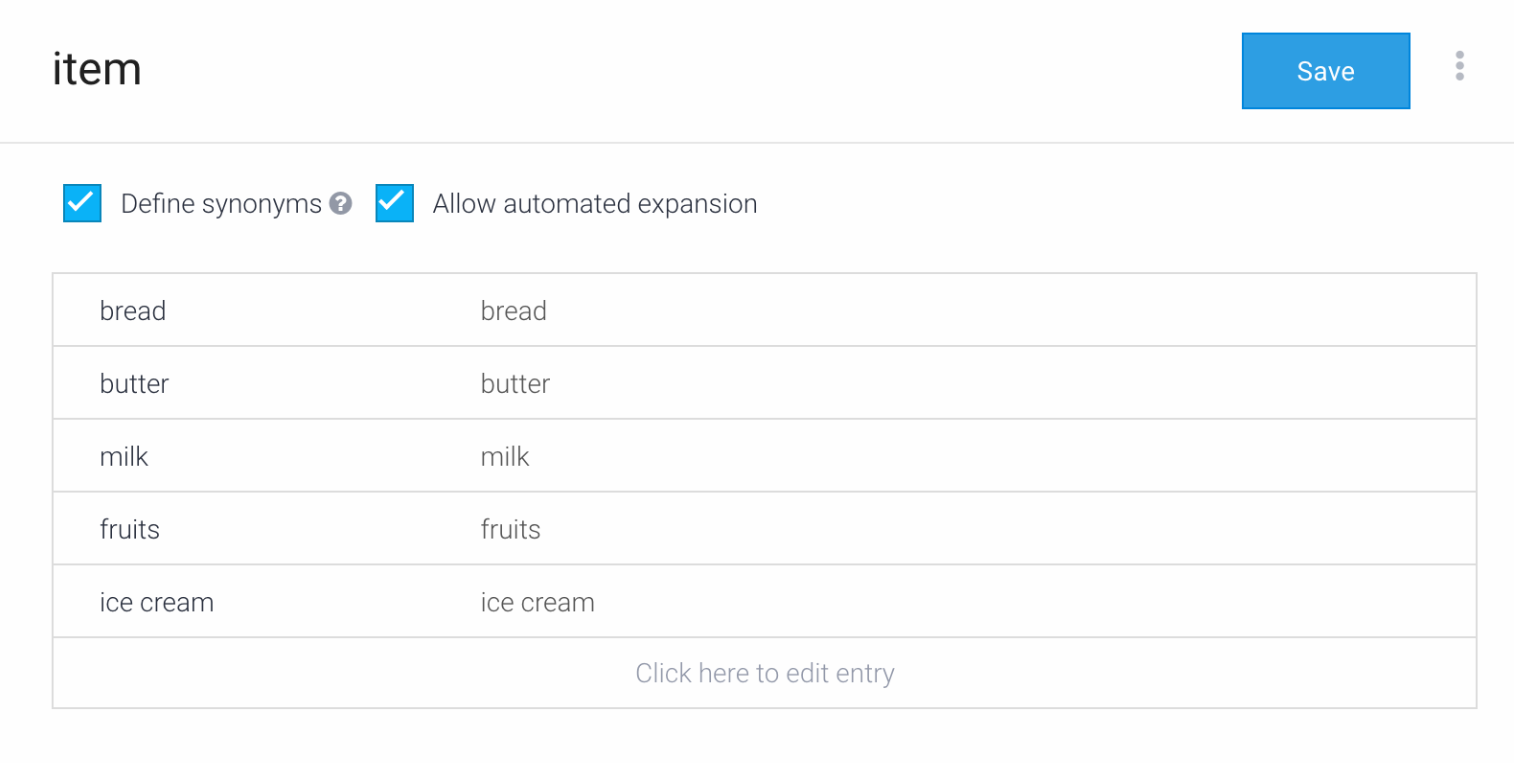


## **Allow Automated Expansion**

This feature of developer mapping entities allows an agent to recognize values that have not been explicitly listed in the entity.

If a user's request includes an item that isn't listed in the entity, Automatic Expansion recognizes the undefined item as a parameter in the entity. The agent sees the user's request is similar to the examples provided, so it can derive what the item is in the request.

For example, consider a shopping list with items to buy:



If a user says "I need to buy some vegetables", "vegetables" will be picked up as a value, even though it's not included in the @item entity. With Automated Expansion enabled, the NLU sees the user's query is similar to the Training Phrases provided in the intent and can pick out what should be extracted as a new value.

The closer the user's input is to the examples provided in the Training Phrases section, the better the results the Automated Expansion feature will provide. This is another reason to provide as many examples as possible.

**Note:** Use this feature with caution:

* For finite lists, stick to creating entities containing complete lists instead of providing a partial list and checking **Allow automated expansion**.
* If you enabled **Allow automated expansion** in more than one entity in the same agent, it may cause conflicts and unexpected classification results.

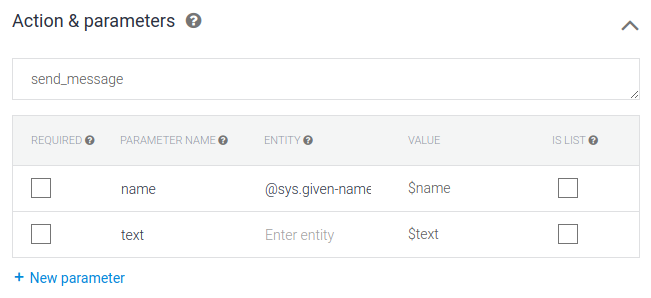
# Actions and Parameters

## **Actions**

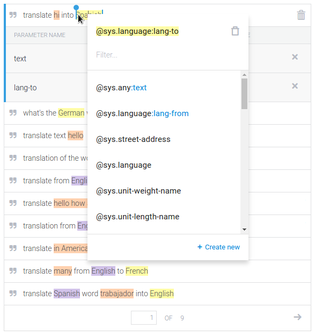
An action corresponds to the step your application will take when a specific intent has been triggered by a user’s input. Actions can have parameters for extracting information from user requests and will appear in the following format in a JSON response:

    {“action”:”action\_name”}  
    {“parameter\_name”:”parameter\_value”}

Both the action name and its parameters are defined in the **Action** section of an intent. For example, if you are building an application for sending messages, the **Action** section would have the name of the action, as well as any [automatically](https://dialogflow.com/docs/actions-and-parameters#automatic_default) defined or [manually](https://dialogflow.com/docs/actions-and-parameters#manual) added parameter values.



## **Parameters**



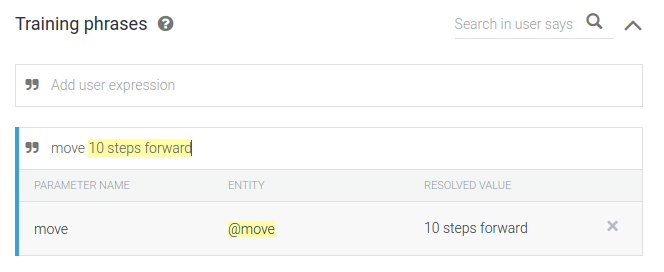
Parameters are elements generally used to connect words in a user’s response, to entities. In JSON responses to a query, parameters are returned in the following format:

    {“parameter\_name”:”parameter\_value”}

Parameters appear in two different areas. In the **Training Phrases** section, parameters related to known entities will be highlighted (annotated) after an example is added. Clicking on an annotated word in an example, will reveal a table with data on the chosen entity.

A table of the parameters in the intent, is located under the **Action & Parameters** section of the intent page. This table represents all of the parameters used throughout all of the **Training Phrases** examples in the current intent.

### **Defining Parameters**



#### **Automatic (Default)**

When you enter an example in the **Training Phrases** section of an intent, words that correspond to system and developer entries will be highlighted (annotated), and parameters will automatically appear in the parameter table.

#### **Manual**

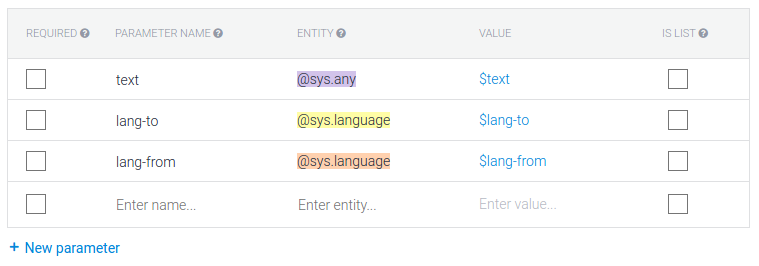
There are some cases when you may need to define or edit parameters and their values manually. For example, if your application controls a robot’s step motor, you may want to use fixed values for the parameters. You need to enter these values manually.

### **Deleting Parameters**

You can delete the association between a word and an entity by removing the annotation from a single **Training Phrase**. Doing this will only affect this particular example.

If you need to delete an association from the entire intent, you can delete the parameter row by hovering over the parameter, click on the menu button and choose **Delete**.

## **The Parameter Table**



Entries in both the **Training Phrases** and **Action & Parameter** tables can be manually defined or edited. Changes to parameters in the **Training Phrases** table, only affect the parameters in the selected **Training Phrases**example. Changes to parameters in the **Action & Parameter** table affect the parameter throughout the current intent.

### **Required**

Checking this option will mark the parameter as required, which means the parameter is needed to perform the action in the intent. When this option is enabled, a **Prompts** column will be revealed. This is the response the agent will give when a required parameter isn’t collected.

### **Parameter Name**

This property defines how the parameter will show up in the JSON response:

{**“parameter\_name”**:”parameter\_value”}

### **Entity**

This property is the entity (system or developer) that is linked to the parameter. The linked entity can be changed by clicking on the current property and choosing a new entity from the searchable list.

### **Value**

This property is the value assigned to the parameter. In the **Training Phrases** table, this will show the resolved value from the specific example.

In the **Action** table, the value is represented as $ and the parameter name. Clicking on value will either show you additional options (automatic) or allow you to edit the value (manual). This is how the value will show up in the JSON response:

{“parameter\_name”:**”parameter\_value”**}

### **Is List**

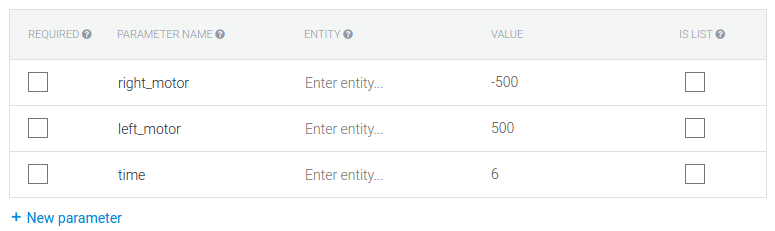
This property marks the parameter as having it’s value as a list. This is ideal for situations where a user’s input contains an enumeration, like “I would like apples, bananas, and oranges.” More on [defining list values](https://dialogflow.com/docs/actions-and-parameters#list).

## **Defining Values**

### **Constant**

Your application may need parameters with fixed values.

For example, if your application controls a robot’s step motor, you may want to use fixed values for the parameters. You would need to enter these values manually.



### **Default**

To define a default parameter value, hover over the parameter row, click on the menu button, and select **Default Value**. This is ideal for situations where you can expect a default value without the user specifying.

For example, your user could say “I want to order a hamburger.” A default value of 1 would be used since a specific quantity wasn’t provided. The returned parameters would look like this:

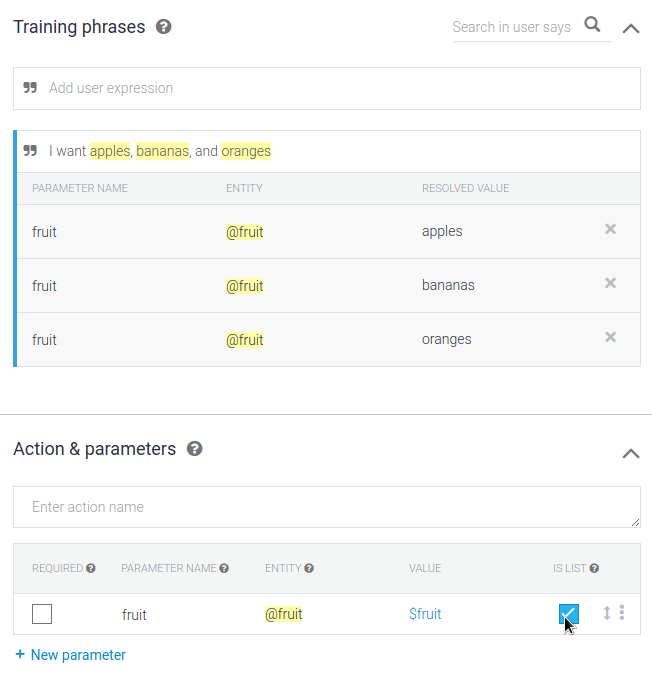
“resolvedQuery”: “I want to order a hamburger”,  
    “parameters”: {  
        “item”: “hamburger”,  
        “number”: “1”  
    }

### **List**

List values can contain an indefinite number of elements. You can define simple **Training Phrases**, which allow for more inputs to be matched.

For example, if you expect users to say "I want apples, bananas, and oranges." or "I want apples and oranges.", you could add the example "I want apples" and then highlight the word “apples” and set the @fruit entity and check the **Is List** property for the parameter.

When a user says "I want oranges, apples, and bananas." it will recognize all items as related to the @fruitentity.



### **@sys.date**

By default, when you expect a date from a **Training Phrase**, the system entity @sys.date is used. If the user provides an incomplete date like “Monday” or “July 8th”, the system will return it’s best estimation at the nearest future date.

For example, if today is November 3rd, 2017 and the user says “Book a ticket for Monday”, the system will return ”date”: “2017-11-6” which is the closest Monday in the future.

You can change the how incomplete dates are handled by setting the parameter value’s option. Clicking on the value for the parameter will give you the following options:

* **$date.recent** - This option estimates the closest past date. If it's November 3rd, 2017 and a user asks “What happened on Monday?” the system will return ”date”: “2017-10-30” which is the closest Monday in the past.
* **$date.partial** - This option accepts the partial date provided by the user, in cases where you want to handle the data with your own logic. If it’s November 3rd, 2017 and a user asks “What happened on April 23rd?” the system would return ”date”: “UUUU-04-23” which is the unaltered data provided by the user.

**Caution:** **$date.partial** doesn’t work with days of the week (e.g. Monday, Tuesday, etc).

## **Extracting Values**

### **Original**

You may want to extract the original value from the user’s input to give a grammatically correct response.

For example, a user may ask for one or more items and could use a singular or plural form, like “I want one hamburger” versus “I want five hamburgers”. To provide a proper response, you can define a new parameter with the original value. Click on the value in the “Action” table to see the $parameter\_value.original option.

**Note:** [@sys.any](https://dialogflow.com/docs/reference/system-entities#generic) will strip out special characters. To preserve special characters use **.original**.

### **From Composite Entities**

By default, composite entities return object-type values. In order to extract string values from "sub-entities", you will need to manually define parameters with the following format:

$composite\_entity\_parameter\_name.subentity\_alias

### **From Contexts**

To refer to a parameter value contained within an active context, you will need to manually define parameters with the following format:

#context\_name.parameter\_name

For example, if you have an active context called “greeting” that contains a parameter called name and you want to reference that parameter's value in another parameter or in a response, you'd use the following format:

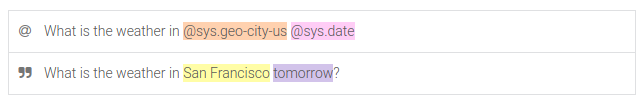
#greeting.name

## **Managing Parameters with Template Mode**

In some rare cases, you may want to use Template Mode (indicated by the @ icon at the beginning of the line) instead of the suggested Example Mode.

In templates, entities can be directly referenced in **Training Phrases**. Entities should be referenced by their names, prefixed with the @ sign, and can be used with or without aliases: @entity:alias or @entityrespectively.

For example, while using Example Mode, a **Training Phrase** could be “What is the forecast for tomorrow?”. This same example in Template Mode would be “What is the @condition for @sys.date?”



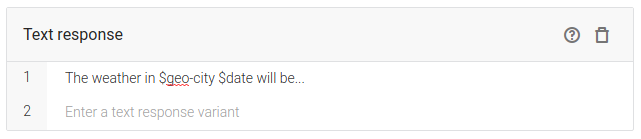
The main case for using Template Mode is to take advantage of [special syntax](https://dialogflow.com/docs/reference/system-entities#generic).

### **Referring to Parameter Values in Responses**

To reference parameter values in the responses, use the following format:

$parameter\_name

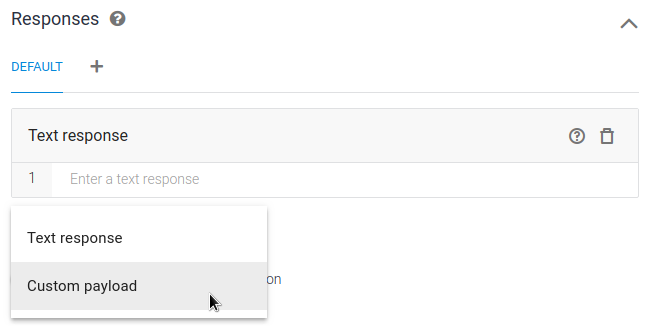
If your agent needs to confirm what your user just ordered, and the items use the $item value, a response would look like this:



# Rich Messages

In the [**Response**](https://dialogflow.com/docs/intents#response) section, you can add tabs for some of our supported integrations. This allows you to define default or integration-specific responses. In each tab, you can add up to 10 of the same or different message types. The **DEFAULT** tab and the integration tabs offer different message types. The integration tabs allow you to add images, cards, and quick replies.

To add integration tabs, either enable the respective integrations on the **Integrations** page or click on the **+** sign next to the **DEFAULT tab**. To add message elements, click the **ADD MESSAGE CONTENT** button.



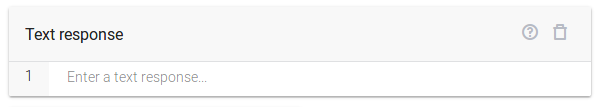
Your agent’s response can consist of up to 10 sequential messages, you can reordered in the UI. If you create intents via the [/intents](https://dialogflow.com/docs/reference/agent/intents) endpoint, the order of messages will correspond to the [messages](https://dialogflow.com/docs/reference/agent/intents#intent_json_object) array elements order.

## **Types of Responses**

You can see example code for these message types in our [message objects reference](https://dialogflow.com/docs/reference/agent/message-objects#one-click_integration_message_objects).

**Note:** The code for Actions on Google rich responses is a bit different, so it has it's own section [here](https://dialogflow.com/docs/reference/agent/message-objects#actions_on_google_message_objects).

### **Text**



Text responses are available in all platforms. Your agent can send up to 10 sequential text messages in response to a user input (assuming no other message types are defined in the intent).

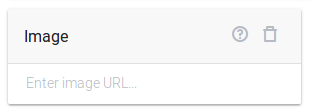
To add a new line in the UI, press Shift+Enter. If you [create intents via the API](https://dialogflow.com/docs/reference/agent/intents), use \n in the [intent object](https://dialogflow.com/docs/reference/agent/intents#intent_json_object).

**Note:** Line breaks are currently supported for the following integrations:

* Facebook Messenger
* Kik
* LINE
* Skype
* Slack
* Telegram
* Viber

You can reference parameter values as described [here](https://dialogflow.com/docs/intents#referencing_parameter_values_in_fallback_intents).

### **Image**



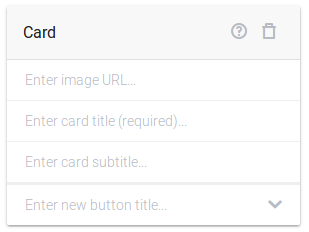
The image message type lets your agent send images in integrations for [Facebook Messenger](https://dialogflow.com/docs/integrations/facebook), [Kik](https://dialogflow.com/docs/integrations/kik), [LINE](https://dialogflow.com/docs/integrations/line), [Skype](https://dialogflow.com/docs/integrations/skype), [Slack](https://dialogflow.com/docs/integrations/slack), [Telegram](https://dialogflow.com/docs/integrations/telegram), and [Viber](https://dialogflow.com/docs/integrations/viber).

All you need to do is to store the image and provide the public URL in the response for the agent.

**Note:** Platforms support different image formats and sizes:

* Facebook Messenger: jpg, png and gif (animated gifs are supported)
* Kik: jpg, 1 MB max size (to send animated gifs, use video format in Custom payload)
* LINE: jpeg, 1 MB max size, 240 x 240 max dimensions
* Skype: 20 MB max size
* Slack: gif, jpeg, png, and bmp
* Telegram: 5 MB max size
* Viber: jpeg, 1 MB max size

### **Card**



Cards consist of an image, a card title, a card subtitle, and interactive buttons (for sending user queries or opening links). These elements can be combined depending on the platform's requirements:

▸

#### **Facebook Messenger**

▸

#### **Kik**

▸

#### **LINE**

▸

#### **Skype**

▸

#### **Slack**

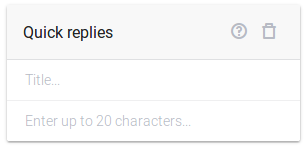
▸

#### **Telegram**

▸

#### **Viber**

### **Quick Replies**



Quick replies are displayed in messengers as clickable buttons with pre-defined user responses. When clicked, the button text is sent to the agent as a user query.

The following sections explain how to set up Quick Replies on each platform.

**Requirements/Limitations:**

* One Quick reply element per intent.
* Maximum of 10 replies.
* 20 character limit per reply.

▸

#### **Facebook Messenger**

▸

#### **Kik**

▸

#### **LINE**

▸

#### **Skype**

▸

#### **Slack**

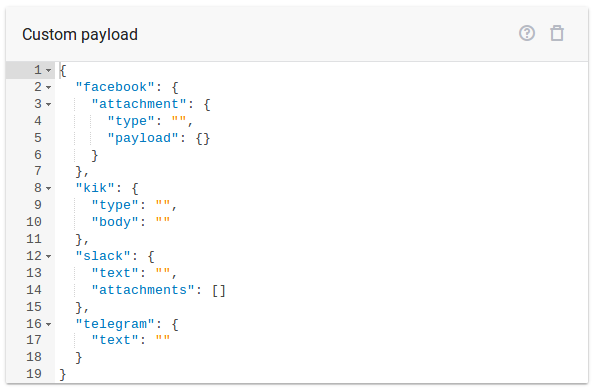
▸

#### **Telegram**

▸

#### **Viber**

### **Custom Payload**



You can send custom payloads in the JSON format provided in the platforms documentation.

To send a custom payload to Facebook Messenger, Kik, Slack, and Telegram one-click integrations, use the following format:

{  
  "facebook": {   
  },  
  "kik": {  
  },  
  "line": {  
  },  
  "skype": {  
  },  
  "slack": {  
  },  
  "telegram": {  
  }  
  "viber": {  
  }  
}

You can also send a custom payload to self-developed integrations. It won’t be processed by Dialogflow, so you'll need to handle it in your own business logic.

See the integration's developer docs for more information:

* [Facebook Messenger](https://developers.facebook.com/docs/messenger-platform)
* [Kik](https://dev.kik.com/#/docs/getting-started)
* [Skype](https://docs.microsoft.com/en-us/bot-framework/)
* [Slack](https://api.slack.com/)
* [Viber](https://developers.viber.com/docs/)

▸

#### **Facebook**

▸

#### **Kik**

▸

#### **LINE**

▸

#### **Skype**

▸

#### **Slack**

▸

#### **Telegram**

▸

#### **Viber**

# Contexts

Contexts represent the current context of a user's request. This is helpful for differentiating phrases which may be vague or have different meanings depending on the user’s preferences, geographic location, the current page in an app, or the topic of conversation.

For example, if a user is listening to music and finds a band that catches their interest, they might say something like: “I want to hear more of them”. As a developer, you can include the name of the band in the context with the request, so that the agent can use it in other intents.

Or let’s say you’re a manufacturer of smart home devices, and you have an app that remotely controls lights and household appliances. A user might say, "Turn on the front porch light", followed by “Turn it off”. By setting a context the app will understand that the second phrase is referring to the light from the first request. Later, if the user says, "Turn on the coffee machine", and then “Turn it off”, it will result in different action than before, because of the new context.

This agent uses a context to carry a parameter value from one intent to another.

**Enter "Ask me" to get started.**

The intent that collects your favorite food uses an output context to "remember" what you said. The same context is used, but as the input to the next intent.

When referring to the first parameter, the agent's response used #context-name.parameter-name. In this case, the context is named "food-context" so the following is used in the response: #food-context.dishes.

Providing a reply that doesn't include a valid food or drink will trigger the [fallback intent](https://dialogflow.com/docs/intents#fallback-intents), and start the process over.

## **Adding Contexts**

To define and add a context to an intent, click in the **Add input context** or **Add output context** field and type the desired name of the context and hit Enter to commit it.

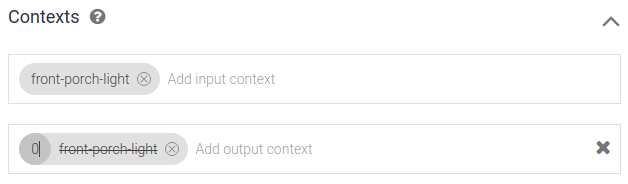
**Note:** When naming contexts, keep the following points in mind:

* Use alphanumeric names.
* Use "-" or "\_" instead of spaces.
* Context names are **not** case sensitive. For example, you can add "AaBbCc123" and "aabbcc123" to an intent, but they're considered the same context.

### **Lifespan**

By default, contexts in intents expire after either five requests or ten minutes from the time they were activated. Contexts in [Follow-up intents](https://dialogflow.com/docs/intents#follow_up_intents) have a default lifespan of two requests. Intents that renew the context will reset the counter and clock to give an additional five requests and ten minutes.

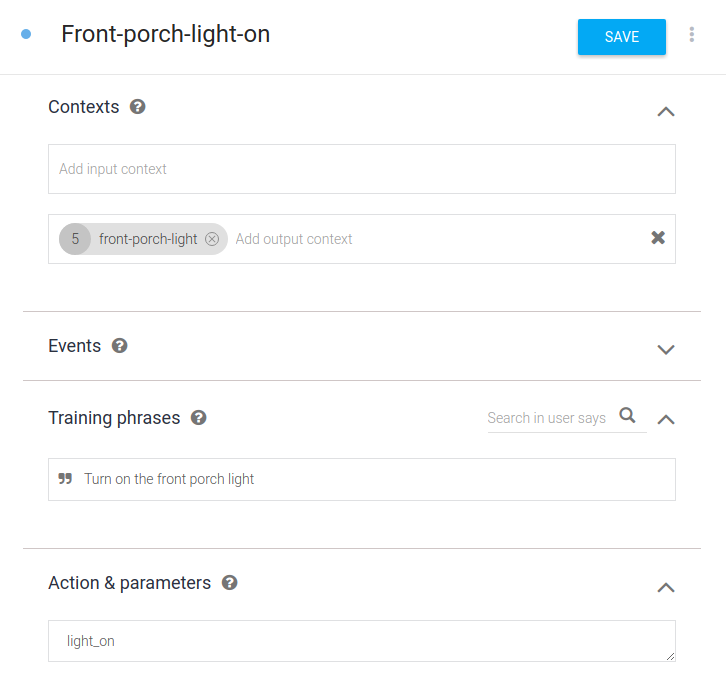
You can change the lifespan of the context by clicking on the current lifespan and entering the desired number of requests. Setting the lifespan to 0 will reset the content when it is matched.



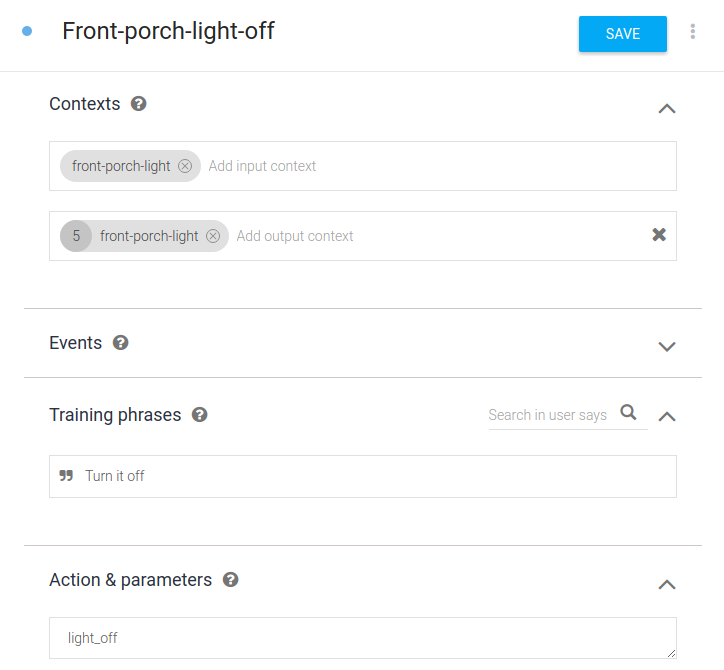
### **Output Contexts**

Contexts are tied to user sessions (a session ID that you pass in API calls). If a user expression is matched to an intent, the intent can then set an output context to be shared by this expression in the future. You can also add a context when you send the user request to your Dialogflow agent.

In our appliance app example, saying "Turn on the front porch light" should set an output context to "front-porch-light".



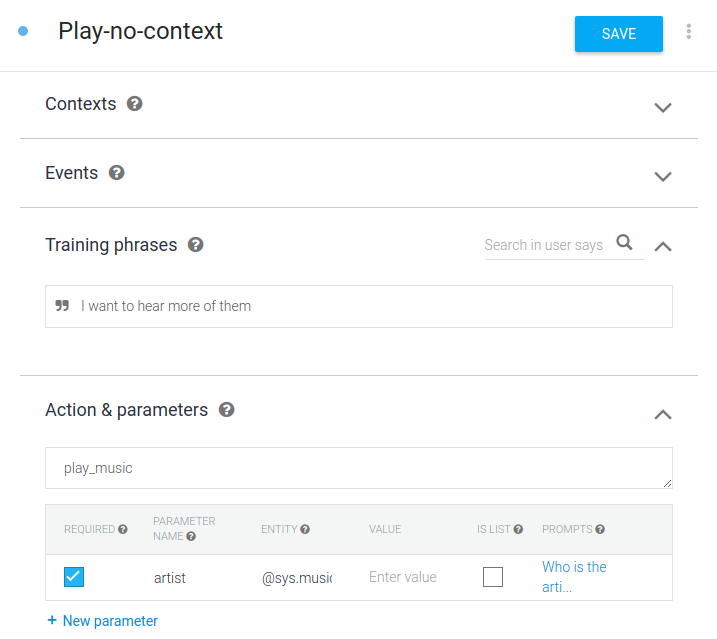
There may be several intents that respond to "Turn it off", each with a different input context. Since the output context is set to "front-porch-light", the intent for "Turn it off" that has a matching input context of "front-porch-light" will be executed, and all others will not.



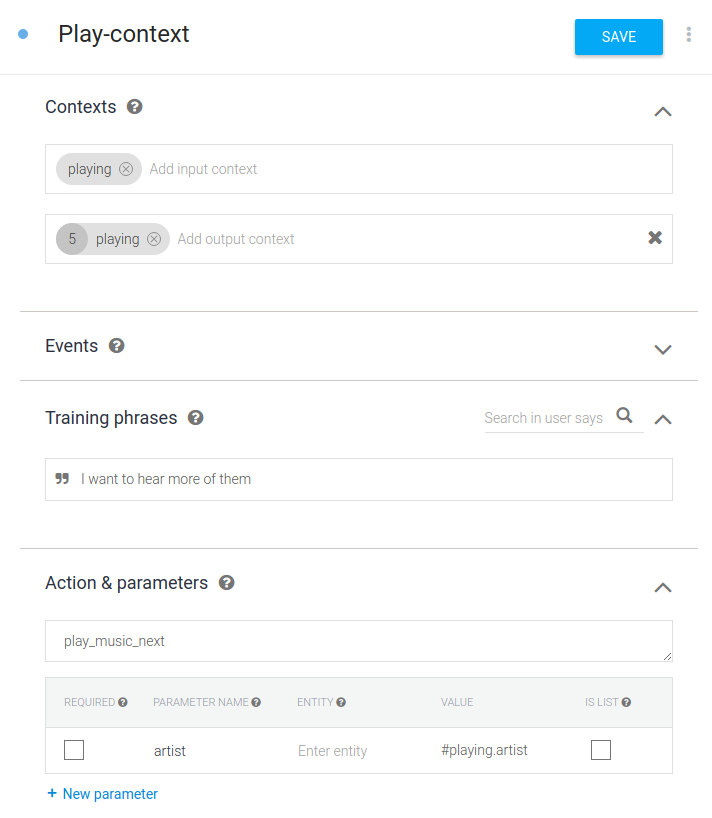
### **Input Contexts**

Input contexts limit intents to be matched only when certain contexts are set.

For the music example, we would create two intents for requests like “I want to hear more of them”. The intent without context will match the user’s request when the app has no information about the artist, so it will need to get that info from the user. This is done by marking parameters as required and providing prompts.



The other intent that we create has an input context, which contains information about which artist or band is meant by the user’s request.



**Note:** In case of multiple input contexts defined in a single intent, all these contexts should be active in order for this intent to be matched.

## **Extracting Parameter Values from Contexts**

To refer to a parameter value that has been extracted in an intent with a defined output context, use the following format in the **VALUE** column: #context\_name.parameter\_name

# Events

Events is a feature that allows you to invoke intents by an event name instead of a user query.

First, you define event names in intents. Then, you can trigger these intents by sending a [/query](https://dialogflow.com/docs/reference/agent/query) request containing an event parameter.

**Note:** Note: The request format is different when using the V2 API. See the [QueryInput](https://dialogflow.com/docs/reference/api-v2/rest/v2beta1/projects.agent.sessions/detectIntent" \l "queryinput) doc for relevant info.

## **Event Name**

Event name is a string up to 50 characters long. It may contain Latin letters (a-z A-Z), digits (0-9), underscore (\_), and hyphen (-). Event names are case insensitive.

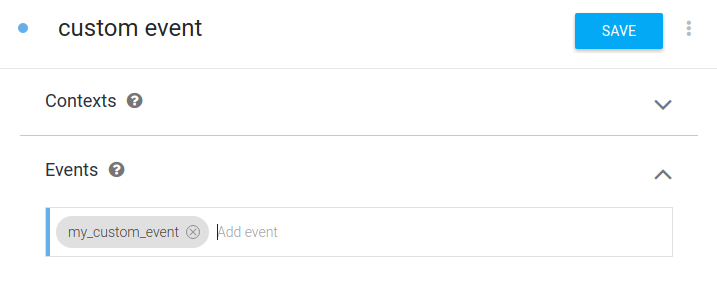
Some strings are reserved. For example, event names that invoke events in the supported messaging platforms integrated into your agent with the help of [one-click integrations](https://dialogflow.com/docs/integrations/).

| Event Name | Platform |
| --- | --- |
| **WELCOME** | Google Assistant, Facebook Messenger, Telegram, Kik. See more in [Default Welcome Intent](https://dialogflow.com/docs/events#default_welcome_intent) |
| **FACEBOOK\_WELCOME** | Facebook Messenger |
| **FACEBOOK\_LOCATION** | Facebook Messenger - See more [below](https://dialogflow.com/docs/events#facebook_location) |
| **GOOGLE\_ASSISTANT\_WELCOME** | Google Assistant (Actions on Google) |
| **KIK\_WELCOME** | Kik |
| **SLACK\_WELCOME** | Slack |
| **SKYPE\_WELCOME** | Skype |
| **TELEGRAM\_WELCOME** | Telegram |

## **Defining Event Names in Intents**

You can define an event name in an intent in the developer console or via the API with the help of the [/intents](https://dialogflow.com/docs/reference/agent/intents)endpoint.

While in an intent, click **Events** above the **Training Phrases** section and enter the event name(s).



To add events via the API, use the events field in the [intent object](https://dialogflow.com/docs/reference/agent/intents#intent_json_object) in the following format:

"events": [  
    {  
        "name": "<EVENT\_NAME>"  
    }  
]

## **Invoking an Event via Query Request**

To trigger a specific intent by event name, send a [/query](https://dialogflow.com/docs/reference/agent/query) request containing an event parameter value that corresponds to the event name.

A query request should contain either the query or event parameter.

### **GET /query Request**

If you use the GET method to invoke an event, you can send the event name in the following format: e=<EVENT\_NAME>.

For example:

curl \  
-H "Authorization: Bearer YOUR\_CLIENT\_ACCESS\_TOKEN" \  
"https://api.dialogflow.com/v1/query?v=20150910&e=event\_name&timezone=Europe/Paris&lang=en&sessionId=1234567890"

### **POST /query Request**

If you use the POST method to invoke an event, the event parameter value is an object of the following format:

"event":{    
  "name":"<EVENT\_NAME>",  
  "data":{  
      “<PARAMETER\_NAME>”:”<PARAMETER\_VALUE>”    
  }  
}

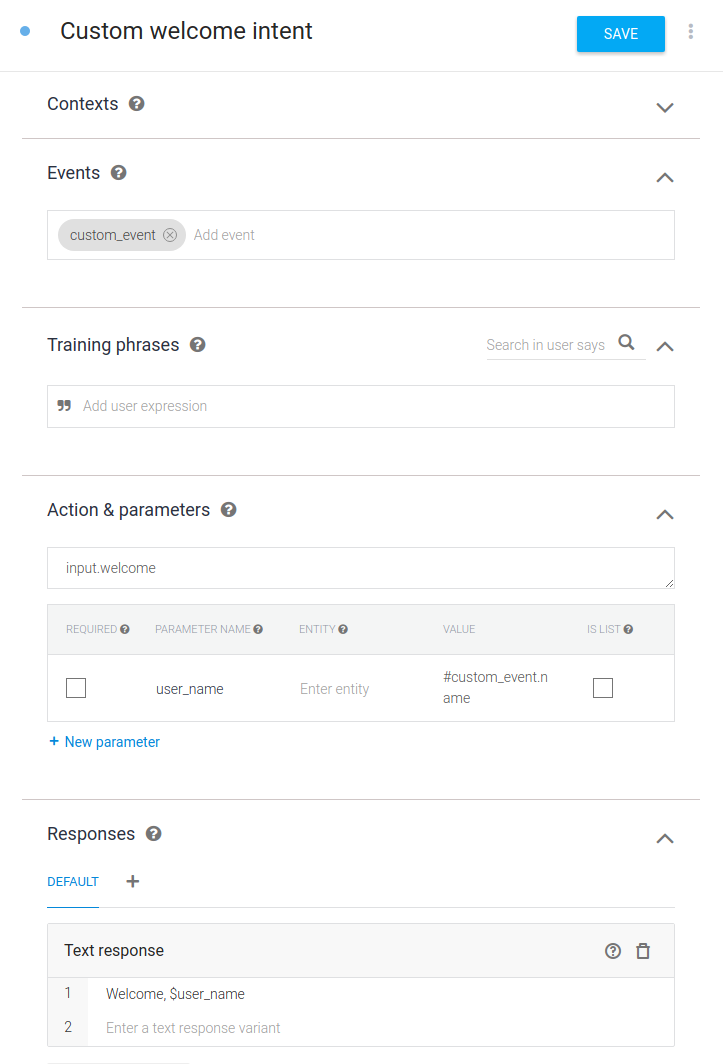
### **Sending Parameters in a Query Request**

When you send a query request with an event parameter, Dialogflow creates a context with the same name as the event name and context "lifespan": 0. This lifespan value means that the context is active only during the current request.

You can use this context to pass parameter values from the data object to the parameters manually defined in the **Action** section of the intent or reference these parameter values in the **Response** section of the triggered intent. To do so, use the following format: #event\_name.parameter\_name\_from\_data

The following example sends a user name to your custom welcome intent invoked by the event and references this name in the text response.

1. Define the event name, parameter value, and text response in the following way:



1. Send a POST /query request with the event parameter containing the user name:

curl \  
-X POST -H "Content-Type: application/json; charset=utf-8" \  
-H "Authorization: Bearer YOUR\_CLIENT\_ACCESS\_TOKEN" \  
"https://api.dialogflow.com/api/query?v=20150910" \  
--data "{  
'event':{'name': 'custom\_event', 'data': {'name': 'Sam'}},  
'timezone':'America/New\_York',  
'lang':'en',  
'sessionId':'1321321'}"

The JSON response to this query will look like this:

{  
  "id": "a3a27316-572a-443f-bdb9-ca65dd2325d6",  
  "timestamp": "2016-12-01T19:46:07.379Z",  
  "result": {  
    "source": "agent",  
    "resolvedQuery": "custom\_event",  
    "action": "welcome",  
    "actionIncomplete": false,  
    "parameters": {  
      "user\_name": "Sam"  
    },  
    "contexts": [  
      {  
        "name": "custom\_event",  
        "parameters": {  
          "user\_name": "Sam",  
          "name": "Sam",  
          "user\_name.original": ""  
        },  
        "lifespan": 0  
      }  
    ],  
    "metadata": {  
      "intentId": "ade506c7-851b-4f62-ba85-2f33023d079a",  
      "webhookUsed": "false",  
      "webhookForSlotFillingUsed": "false",  
      "intentName": "Custom welcome intent"  
    },  
    "fulfillment": {  
      "speech": "Welcome, Sam!",  
      "messages": [  
        {  
          "type": 0,  
          "speech": "Welcome, Sam!"  
        }  
      ]  
    },  
    "score": 1.0  
  },  
  "status": {  
    "code": 200,  
    "errorType": "success"  
  },  
  "sessionId": "1321321"  
}

## **Default Welcome Intent**

When you create a new agent, a Default Welcome Intent is automatically added. Such intents have a pre-defined WELCOME event and text responses.

The WELCOME event is a generic event for supported one-click integrations. It’s a short way to set the following events: GOOGLE\_ASSISTANT\_WELCOME, FACEBOOK\_WELCOME, TELEGRAM\_WELCOME, KIK\_WELCOME.

When the end-user triggers a welcome intent from a supported messaging platform, the relevant event is sent to Dialogflow. If there is no other intent with a defined event specific to a particular messaging platform, the Default Welcome Intent is triggered.

For example, if a user clicks the **Get started** button in Facebook Messenger, a query containing {"event": {"name": "FACEBOOK\_WELCOME", "data": {}} is sent to the agent. First, Dialogflow checks for intents containing the FACEBOOK\_WELCOME event. If no such intent is found, the Default Welcome Intent is triggered.

## **Invoking Event from Webhook**

You can invoke events via [fulfillment](https://dialogflow.com/docs/fulfillment) by sending event name with the followupEvent parameter in the [response from your web service](https://dialogflow.com/docs/fulfillment#section-format-of-response-from-the-service).

The followupEvent parameter value is an object of the following format:

{  
   "followupEvent": {  
      "name": "<EVENT\_NAME>",  
      "data": {  
         "<PARAMETER\_NAME>":"<PARAMETER\_VALUE>"  
      }  
   }  
}

When the followupEvent parameter is sent from the web service, the system ignores speech, displayText, and data fields.

When an intent which sends a request to the web service to trigger the follow-up intent afterwards is triggered, the metadata field in the [JSON response](https://dialogflow.com/docs/reference/agent/query#get_and_post_responses) contains the executionSequence field. Its value is an array of objects corresponding to the intent sequence. The first object in the sequence contains information about the intent that is responsible for firing the sequence, i.e., sending the request from Dialogflow to the web service. The webhookTriggeringEvent value in the first object is an event name that triggers the follow-up intent from the web service.

The following example shows that the sequence started from the “Amount” intent in which webhook is enabled. That intent invoked the my\_custom\_event event from the web service. By this event, the “Event” intent (in which the event is defined) was triggered.

{  
    "metadata": {  
      "intentId": "d213b0be-2699-43d5-a4e7-5952919906b9",  
      "executionSequence": [  
        {  
          "source": "agent",  
          "intentId": "816d4f13-b677-4ced-b5c4-ae5b810a55f0",  
          "action": "amount",  
          "intentName": "Amount",  
          "webhookUsed": true,  
          "webhookTriggeringEvent": "my\_custom\_event"  
        },  
        {  
          "source": "agent",  
          "intentId": "d213b0be-2699-43d5-a4e7-5952919906b9",  
          "action": "event",  
          "intentName": "Event",  
          "webhookUsed": false  
        }  
      ],  
      "webhookUsed": "false",  
      "webhookForSlotFillingUsed": "false",  
      "intentName": "Event"  
    }  
}

If you send parameters in the followupEvent object, you can reference parameter values in the 'Response' section in the following format: #event\_name.parameter\_name.

## **FACEBOOK\_LOCATION**

The FACEBOOK\_LOCATION event allows you to get a location from Facebook Messenger.

1. Create an intent to request the location, using a custom payload:

{  
  "facebook": {  
    "text": "give me your location please",  
    "quick\_replies": [  
      {  
        "content\_type": "location"  
      }  
    ]  
  }  
}

1. Create another intent to process the location and set the event to FACEBOOK\_LOCATION.
2. In the webhook, you'll receive the latitude and longitude from the originalRequest field:

{  
  "source": "facebook",  
  "data": {  
    "postback": {  
      "data": {  
        "lat": 37.428434,  
        "long": -122.0723816  
      },  
      "payload": "FACEBOOK\_LOCATION"  
    },  
  }  
}

# Dialogs

There are two types of dialogs to consider when building voice interaction scenarios:

* **Linear dialogs** - the aim of which is to collect the information necessary to complete the required action (e.g. find the best hotel, turn on the right light bulb, or play the desired song)
* **Non-linear dialogs** - which may have several branches, depending on users’ answers

## **Linear Dialog**

To build linear dialogs, we use a feature known as slot filling. The following example is an agent for a hotel booking app.

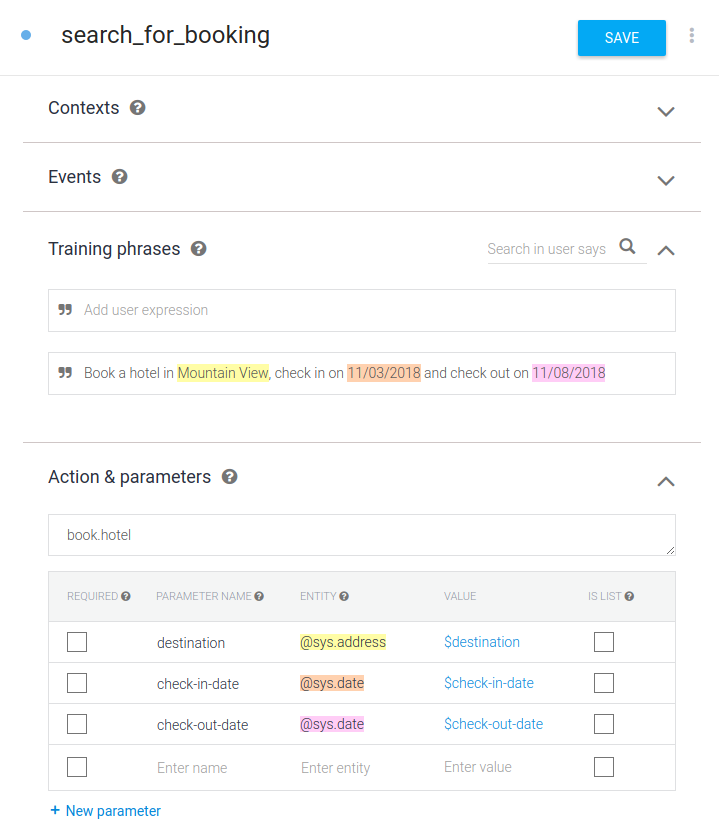
Since our app understands natural language, our travelers can request a hotel in many ways. Depending on how they word their request, there may be several different search parameters:

* “I need to book a hotel."
* “Find a cheap hotel in Rome.”
* “Book a hotel in New York, check in November 10, check out November 15.”

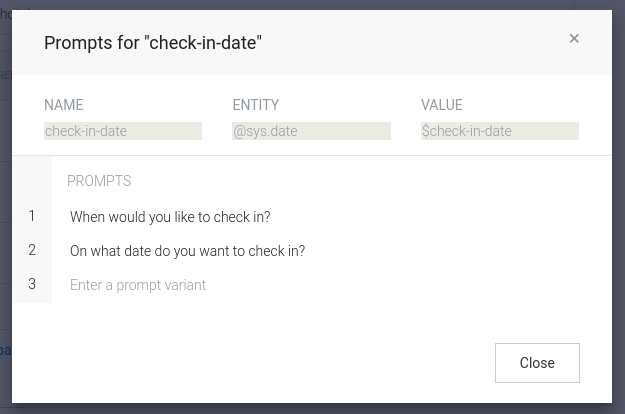
Let’s say our app requires at least three parameters to start the search:

* destination
* check-in date
* check-out date

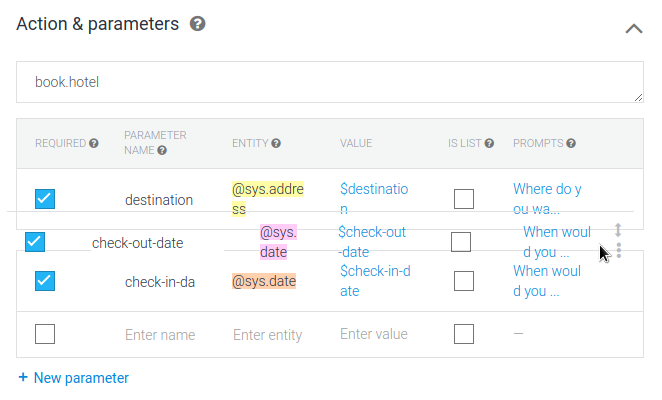
This is where the slot filling feature comes in. Even before you start listing examples in the **Training Phrases**section, you can list all of the parameters your app can use to perform a search, both required and optional.



When you mark a parameter as **Required**, you’ll be asked to write prompts that your app will address to your user when they make a request that does not include that parameter.



You can change the order in which your agent will ask these questions by dragging and dropping the parameters in the list.



Your agent will continue to ask these prompts until it has collected information for all required parameters. At any time, users can say "Cancel" and start from the beginning.

## **Non-linear Dialogs**

Let’s look at an example of a non-linear dialog. In this example, our agent handles a customer satisfaction survey for a hotel.

It starts with these two questions:

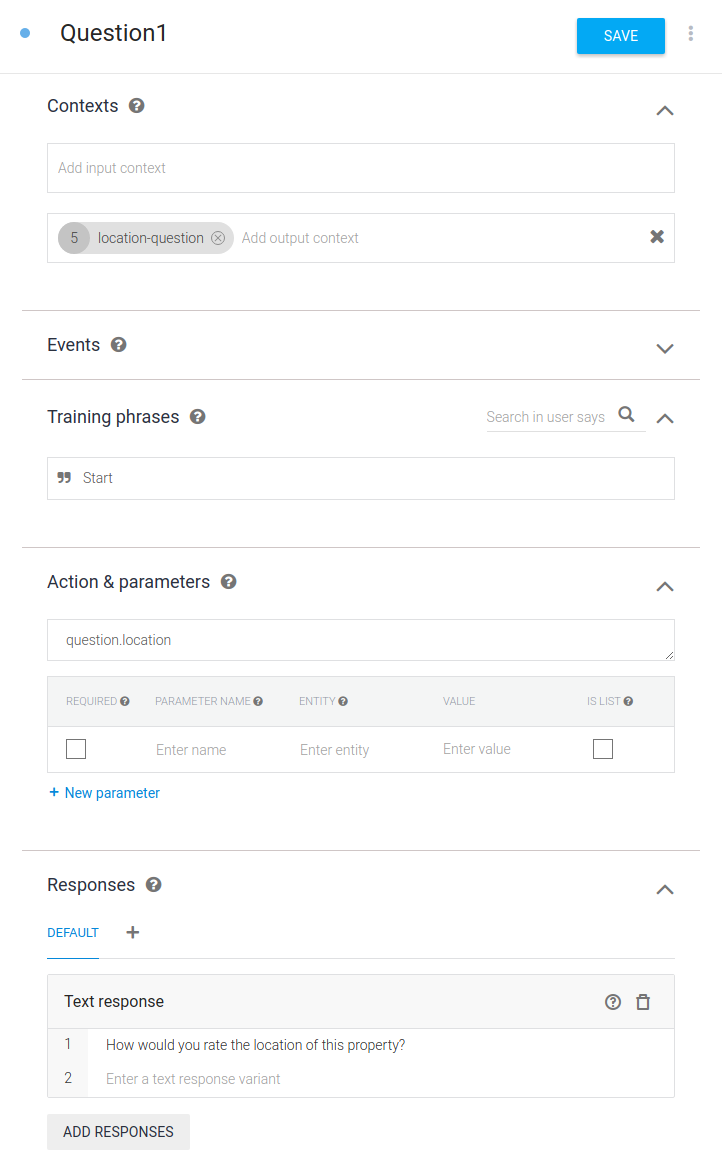
* How would you rate the location of the property?
* How would you rate the facilities at the hotel?

For each question, there are four accepted answers:

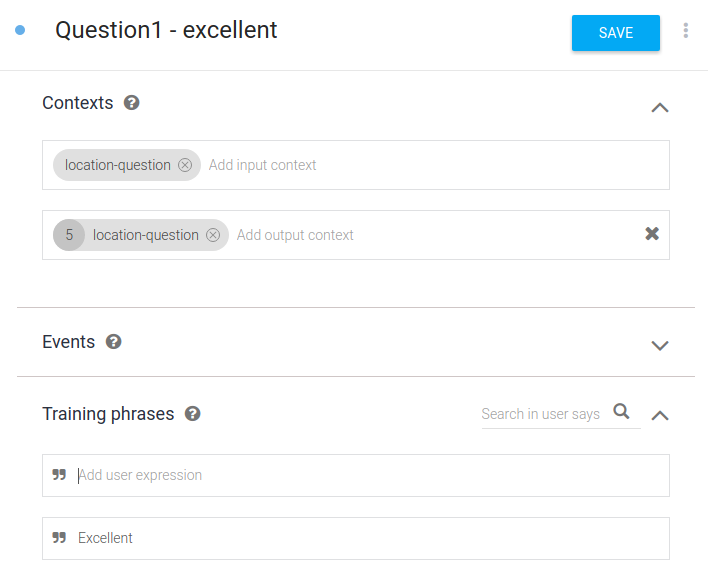
* poor
* fair
* good
* excellent

To build this dialog we’ll need to use contexts. Here’s how it’s done:

First, we create an intent which reacts to the “start” command and triggers the dialog. In this intent we’ll ask the first question and set the outgoing context as “location-question”. As a result, the preferred intents for the answers to this question will have this as the incoming context.

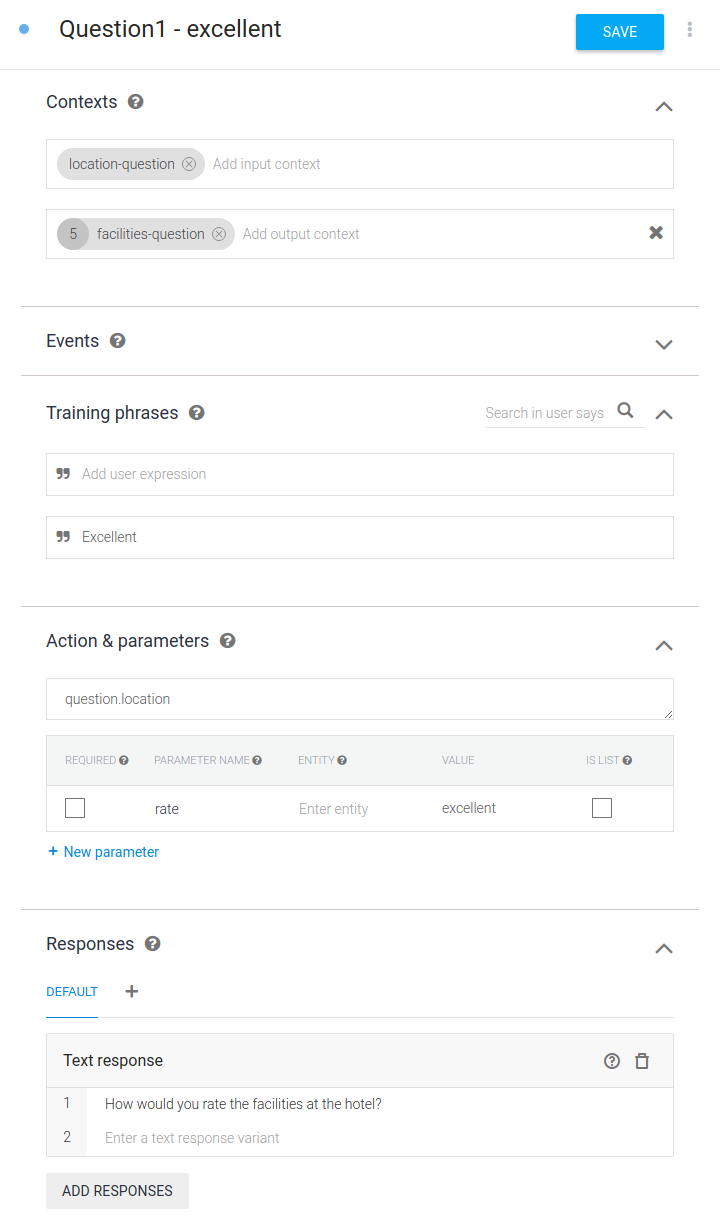


Next we create intents for each of the four expected response variations, with the incoming context “location-question”. These four questions will work only while this context is active.

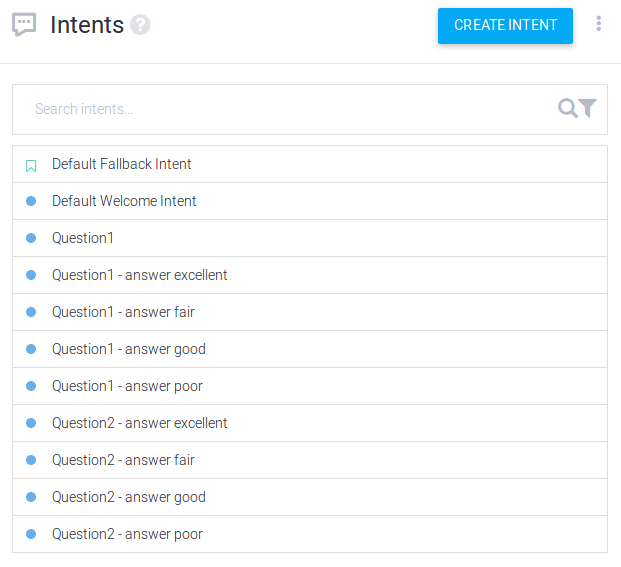


Now we need setup moving to the next question. To do this we make the following changes to the intents we just created:

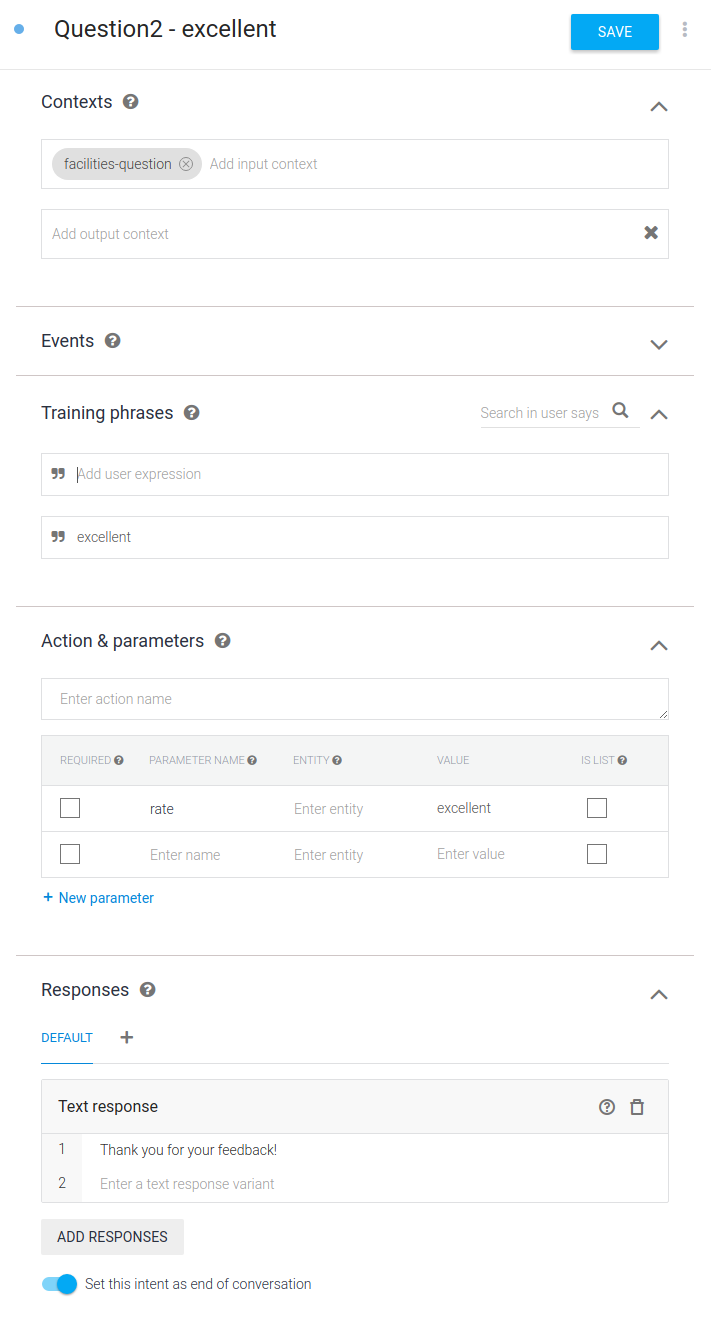
* Remove "location-question" as the output context and add "facilities-question"
* Set the **Action** to question.location
* Add a parameter with the name as "Rate" and the value as the corresponding rating (poor, fair, good, excellent)
* Add a text response for the next question ("How would you rate the facilities at the hotel?")



Then we create four more intents for the possible responses, and we mark “facilities-question” as the incoming context so that these answers will match our new question.



Once again, we mark the action and parameters we want to receive when these new intents are matched, and add the final response in the **Speech Response** field.



# Fulfillment

## **Webhook**

Setting up a webhook allows you to pass information from a matched intent into a web service and get a result from it.

### **Request**

Your web service receives a POST request from Dialogflow. This is in the form of the response to a user query, matched by intents with webhook enabled. Be sure that your web service meets all the webhook requirements specific to the API version enabled in this agent.

If a request is sent from one of the messaging platforms, the originalRequest field is added to the response to a query.

This format is chosen in order to simplify the response parsing on the service side with the help of Dialogflow SDKs.

The request to the service can have the following fields:

### **V2 API**

### **V1 API**

| Name | Type | Description |
| --- | --- | --- |
| responseId | String | Unique id for request. |
| session | String | Unique session id. |
| queryResult | Object | Result of the conversation query or event processing. |
| ↳ queryText | String | The original text of the query. |
| ↳ parameters | Object | Consists of parameter\_name:parameter\_value pairs. |
| ↳ allRequiredParamsPresent | Boolean | Set to false if required parameters are missing in query. |
| ↳ fulfillmentText | String | Text to be pronounced to the user or shown on the screen. |
| ↳ fulfillmentMessages | Object | Collection of [rich messages](https://dialogflow.com/docs/reference/api-v2/rest/v2beta1/projects.agent.intents#message) to show the user. |
| ↳ outputContexts | Object | Collection of output [contexts](https://dialogflow.com/docs/reference/api-v2/rest/v2beta1/projects.agent.sessions.contexts#resource-context). |
| ↳ intent | Object | The [intent](https://dialogflow.com/docs/reference/api-v2/rest/v2beta1/projects.agent.intents#Intent) that matched the user's query. |
| ↳ intentDetectionConfidence | Number 0-1 | Matching score for the intent. |
| ↳ diagnosticInfo | Object | Free-form diagnostic info. |
| ↳ languageCode | String | The language that was triggered during intent matching. |
| originalDetectIntentRequest | Object | Full request coming from an integrated platform. (Facebook Messenger, Slack, etc.) |

#### **Sample Request to the Service**

### **V2 API**

### **V1 API**

POST https://my-service.com/action  
  
Headers:  
//user defined headers  
Content-type: application/json  
  
POST body:  
{  
  "responseId": "ea3d77e8-ae27-41a4-9e1d-174bd461b68c",  
  "session": "projects/your-agents-project-id/agent/sessions/88d13aa8-2999-4f71-b233-39cbf3a824a0",  
  "queryResult": {  
    "queryText": "user's original query to your agent",  
    "parameters": {  
      "param": "param value"  
    },  
    "allRequiredParamsPresent": true,  
    "fulfillmentText": "Text defined in Dialogflow's console for the intent that was matched",  
    "fulfillmentMessages": [  
      {  
        "text": {  
          "text": [  
            "Text defined in Dialogflow's console for the intent that was matched"  
          ]  
        }  
      }  
    ],  
    "outputContexts": [  
      {  
        "name": "projects/your-agents-project-id/agent/sessions/88d13aa8-2999-4f71-b233-39cbf3a824a0/contexts/generic",  
        "lifespanCount": 5,  
        "parameters": {  
          "param": "param value",  
        }  
      }  
    ],  
    "intent": {  
      "name": "projects/your-agents-project-id/agent/intents/29bcd7f8-f717-4261-a8fd-2d3e451b8af8",  
      "displayName": "Matched Intent Name"  
    },  
    "intentDetectionConfidence": 1,  
    "diagnosticInfo": {},  
    "languageCode": "en"  
  },  
  "originalDetectIntentRequest": {}  
}

### **Response**

The response from the service should have the following fields:

### **V2 API**

### **V1 API**

| Name | Type | Description |
| --- | --- | --- |
| fulfillmentText | String | Optional. The text to be shown on the screen. This value is passed directly to QueryResult.fulfillment\_text. |
| fulfillmentMessages[] | Object ([Message](https://dialogflow.com/docs/reference/api-v2/rest/v2beta1/projects.agent.intents#Message)) | Optional. The collection of rich messages to present to the user. This value is passed directly to QueryResult.fulfillment\_messages. |
| source | String | Optional. This value is passed directly to QueryResult.webhook\_source. |
| payload | Object ([Struct](https://developers.google.com/protocol-buffers/docs/reference/google.protobuf#google.protobuf.Struct)) | Optional. This value is passed directly to QueryResult.webhook\_payload |
| outputContexts[] | Object ([Context](https://dialogflow.com/docs/reference/api-v2/rest/v2beta1/projects.agent.sessions.contexts#Context)) | Optional. The collection of output contexts. This value is passed directly to QueryResult.output\_contexts. |
| followupEventInput | Object ([EventInput](https://dialogflow.com/docs/reference/api-v2/rest/v2beta1/projects.agent.sessions/detectIntent" \l "EventInput)) | Optional. Makes the platform immediately invoke another sessions.detectIntent call internally with the specified event as input. |

#### **Sample Response from the Service**

### **V2 API**

### **V1 API**

{  
"fulfillmentText": "This is a text response",  
"fulfillmentMessages": [  
  {  
    "card": {  
      "title": "card title",  
      "subtitle": "card text",  
      "imageUri": "https://assistant.google.com/static/images/molecule/Molecule-Formation-stop.png",  
      "buttons": [  
        {  
          "text": "button text",  
          "postback": "https://assistant.google.com/"  
        }  
      ]  
    }  
  }  
],  
"source": "example.com",  
"payload": {  
  "google": {  
    "expectUserResponse": true,  
    "richResponse": {  
      "items": [  
        {  
          "simpleResponse": {  
            "textToSpeech": "this is a simple response"  
          }  
        }  
      ]  
    }  
  },  
  "facebook": {  
    "text": "Hello, Facebook!"  
  },  
  "slack": {  
    "text": "This is a text response for Slack."  
  }  
},  
"outputContexts": [  
  {  
    "name": "projects/${PROJECT\_ID}/agent/sessions/${SESSION\_ID}/contexts/context name",  
    "lifespanCount": 5,  
    "parameters": {  
      "param": "param value"  
    }  
  }  
],  
"followupEventInput": {  
  "name": "event name",  
  "languageCode": "en-US",  
  "parameters": {  
    "param": "param value"  
  }  
}

}

**Caution:** The header must be “Content-type: application/json”.**Note:** To send formatted messages to bots, use the following format for the **data** field:

* **"data": {"slack": {<slack\_message>}}**
* **"data": {"facebook": {<facebook\_message>}}**
* **"data": {"kik": {<kik\_message>}}**
* **"data": {"telegram": {<telegram\_message>}}**

### **Errors**

The service should be able to handle the following errors:

* **400 Bad Request** – The request was invalid or cannot be served.
* **401 Unauthorized** – The request requires user authentication.
* **403 Forbidden** – The server understood the request but refuses to take any further action or the access is not allowed.
* **404 Not found** – There is no resource behind the URI.
* **500 Server fault** – Internal Server Error.
* **503 Service Unavailable** – Internal Server Error.

In case of these or other errors (timeout limit exceeded, service is not available), the “status” in the response sent to the client will contain the following:

"status": {  
    "code": 206,  
    "errorType": "webhook call failed with %error Code% error"  
}

where error Code is the error ID received from the service.

**Note:** In the case of an error or empty response, Dialogflow will send a standard response to the [/query](https://dialogflow.com/docs/reference/agent/query#get_and_post_responses) to the client. In this case, the agent response will contain the **Response** content defined in the triggered intent.

### **Limits**

* Timeout for service response – 5 seconds.
* Data received in the response from the service – up to 64K.

### **Authentication**

Authentication can be done in 2 ways:

* Basic authentication with login and password.
* Authentication with additional authentication headers.

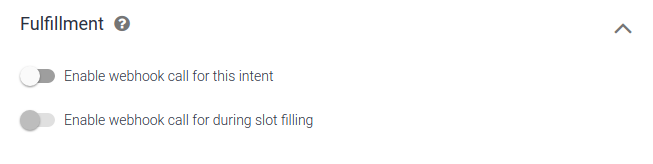
If the integrated service doesn’t require any authentication, leave the authentication fields blank.

The service must use HTTPS and the URL must be publicly accessible.

**Caution:** Dialogflow does not support self-signed SSL certs. For information on SSL setup, please refer to [this documentation](https://developers.google.com/web/fundamentals/security/encrypt-in-transit/enable-https).

### **Slot Filling**

If you want to send requests for required parameters from Dialogflow to your web service, check the option **Use webhook for slot filling** in the **Fulfillment** section of the intent.



### **Example**

Check out the second part of our getting started guide, [Basic Fulfillment and Conversation Setup](https://dialogflow.com/docs/getting-started/basic-fulfillment-conversation), for steps on setting up fulfillment.

## **Cloud Functions for Firebase**

For simple webhook testing and implementation, you can use the Cloud Functions for Firebase area of the **Fulfillment** page. In most cases the free "Spark" tier of Firebase is all you'll need. Tier limitations and pricing information for the other tiers can be found on the [Firebase pricing page](https://firebase.google.com/pricing/).

To enable:

1. Click on **Fulfillment** in the left menu.
2. Click the switch for **Inline Editor**.

Some basic, functional code provided to help you get started.



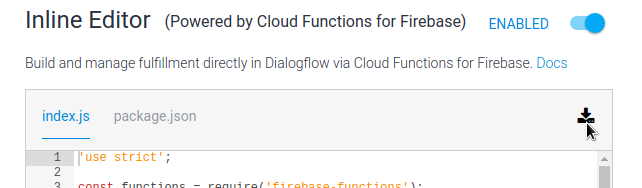
To deploy your fulfillment, click the **Deploy** button under the code editor.

After clicking **Deploy**, Dialogflow will save your Cloud Function for Firebase and begin the deployment process. When deployment is complete, you'll see a timestamp next to the **Deploy** button, signifying the last time the function was deployed.

The link under the code editor can be used to visit the function's logs in the Firebase console. This is used to debug issues and get information on your function.

### **"Graduating" your code**

Once you're ready to move your code out of the **Fulfillment** page, you can use the **Download** button to get a .ZIP file of your code in it's current state. Using the Firebase CLI, you can unzip your code locally and deploy more complex fulfillment code.



### **Limitations**

Here is a list of things to be aware of when using the Cloud Functions for Firebase option:

* If you modify your function outside of code editor, you can no longer use the Dialogflow editor to modify your Cloud Function for Firebase. Your function will continue to provide fulfillment for your agent, but if you need to make changes, you will need to do so in the Firebase console.
* The function must be named "dialogflowFirebaseFulfillment". If you change the name of your function it will not deploy via Dialogflow. If you want to change the name of your function you can download the code, change the function name, and deploy through Firebase's CLI.
* Code modified in the dialogflow editor cannot be saved or downloaded without being deployed.
* Dialogflow's editor only supports two files: index.js and package.json (modifying package.json will install any dependencies you specify upon deployment)
* Network calls originating from your Cloud Function for Firebase to destinations outside Google's network require billing to be enabled for the underlying [Google Cloud or Firebase project](https://cloud.google.com/functions/pricing#networking).

# Analytics

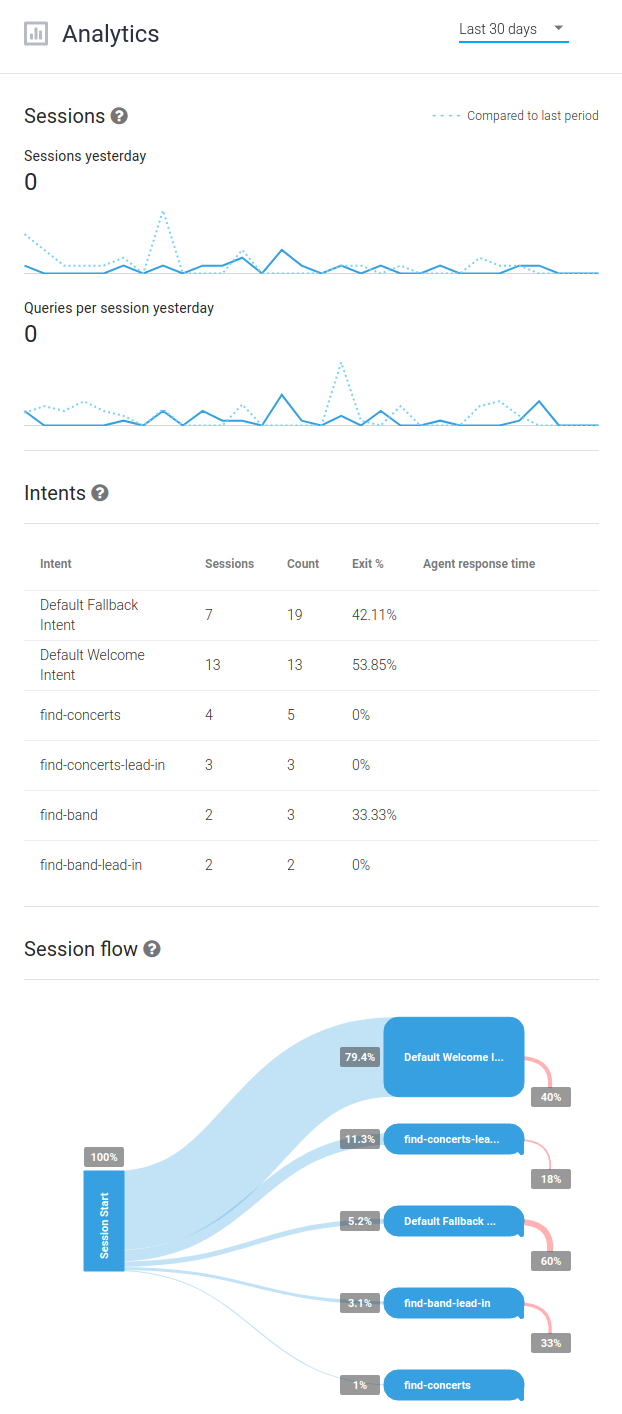
The analytics page gives you insight into how well your agent is performing, so you can work to further improve the user experience you’re providing.

We show two types of data related to the agent and the conversations it’s been a part of:

* Usage data – number of sessions and queries per session.
* NLU data – most frequently used intents and exit percentages.

## **Dashboard**

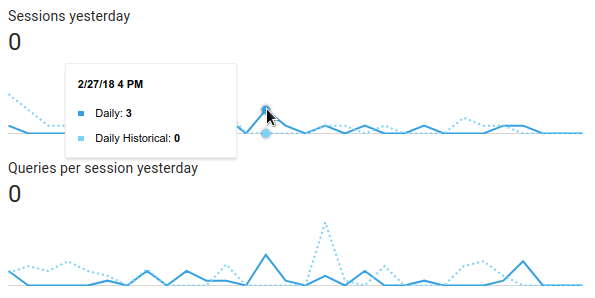
Clicking on **Analytics** in the left hand menu will take you to UI Dashboard. Here you can review statistics relevant to the specific agent.



### **Filtering**

Using the dropdown menu in the upper right hand corner, you can filter the data by chosen date range (1 day, 7 days, or 30 days).

### **Sessions**



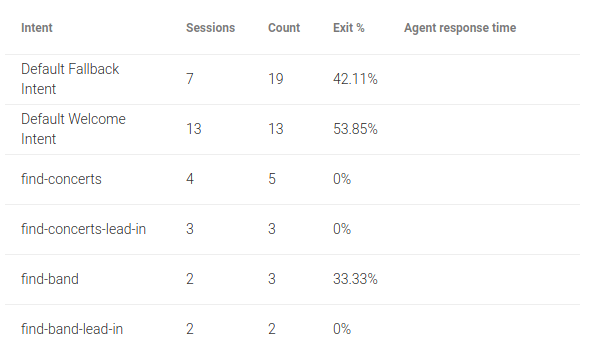
#### **Sessions Yesterday**

This graph represents daily sessions with your agent. One day values are plotted over time, based off 6am UTC timezone. The values above the graph represent the data collected from the previous day (yesterday). The blue line in the graph shows the data for the current day or time period and the secondary, dotted line (light blue) indicates the data from the previous day or time period, so you can compare recent changes.

#### **Queries Per Session Yesterday**

This section shows the average messages from the user, per day. The secondary, dotted line (light blue) line indicates the change from the last time period.

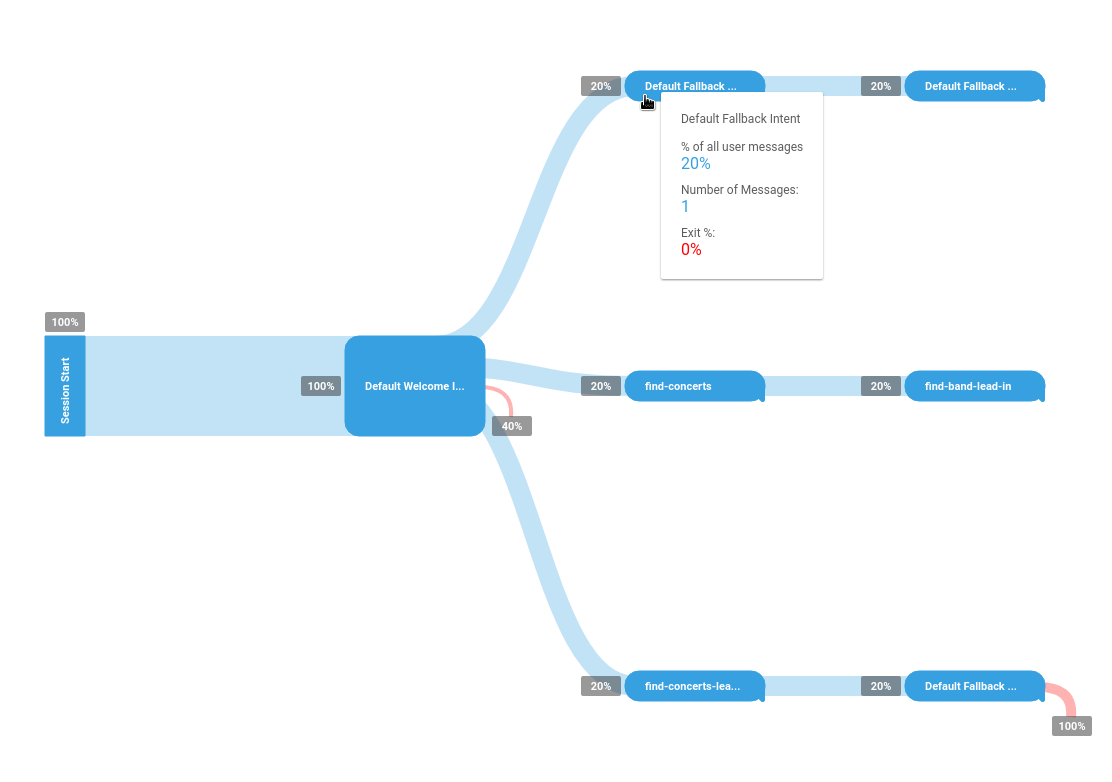
### **Intents**



This table shows the agent’s intents in order of popularity. Data in the table includes:

* **Intent name**
* **Sessions** - The number of sessions the intent was initialized
* **Count** - The number of times the intent was used (total from all sessions)
* **Exit %** - The percentage of sessions where a user exited the conversation in the specified intent. - This is taken from the total number of sessions where this same intent was triggered.
* **Agent Response Time** - Average response time to user requests (90% response time in blue, median response time in light blue)

### **Session flow**



This chart visually summarizes the conversational paths your users have taken when interacting with your agent.

Hover your cursor over the intent names (blue boxes) to see the following information:

* The percentage of all users that matched the intent.
* The number of requests the intent was matched to.
* The percentage of users that exit the intent.

Click **Explore** to view the chart in a larger format. While viewing the chart in this format, you can click on an intent to expand the next matched intents.