Google DialogFlow

Basics

The process a Dialogflow agent follows from invocation to fulfillment is similar to someone answering a question, with some liberties taken of course. In the example scenario below, the same question is being asked, but we compare the "human to human" interaction with a conversation with an Dialogflow agent.

|  |  |
| --- | --- |
| Welcome | Invocation |
|  |  |
| Bill's friend Harry wants to ask him a question. So as not to be rude, Harry says "Hello" to Bill first. | In order to start a conversation with an agent, the user needs to invoke the agent. A user does this by asking to speak with the agent in a manner specified by the agent's developer. |
| Request | [Intent](https://dialogflow.com/docs/intents) |
|  |  |
| Harry asks Bill "What's the weather supposed to be like in San Francisco tomorrow?" Because Bill is familiar with the city and the concept of weather, he knows what Harry is asking for. | A user asks the agent "What's the weather supposed to be like in San Francisco tomorrow?" In Dialogflow, an intent houses elements and logic to parse information from the user and answer their requests. |
| ... | [Training Phrases](https://dialogflow.com/docs/intents#elements_of_an_intent) |
|  |  |
|  | For the agent to understand the question, it needs examples of how the same question can be asked in different ways. Developers add these permutations to the Training Phrases section of the intent. The more variations added to the intent, the better the agent will comprehend the user. |
| ... | [Entities](https://dialogflow.com/docs/entities) |
|  |  |
|  | The Dialogflow agent needs to know what information is useful for answering the user's request. These pieces of data are called entities. Entities like time, date, and numbers are covered by system entities. Other entities, like weather conditions or seasonal clothing, need to be defined by the developer so they can be recognized as an important part of the question. |
| Fulfillment | [Fulfillment Request](https://dialogflow.com/docs/fulfillment) |
|  |  |
| Armed with the information Bill needs, he searches for the answer using his favorite weather provider. He enters the location and time to get the results he needs. | Dialogflow sends this information to your webhook, which subsequently fetches the data needed (per your development). Your webhook parses that data, determines how it would like to respond, and sends it back to Dialogflow |
| Response | [Response](https://dialogflow.com/docs/rich-messages) |
|  |  |
| After scanning the page for the relevant info, Bill tells Harry "It looks like it's going to be 65 and overcast tomorrow." | With the formatted reply "in hand", Dialogflow delivers the response to your user. "It looks like it's going to be 65 and overcast tomorrow." |
| Context | [Context](https://dialogflow.com/docs/contexts) |
|  |  |
| Now that the conversation is on the topic of weather, Bill won't be thrown off if Harry asks "How about the day after that?" Because Harry had asked about San Francisco, follow up questions will more than likely be about the same city, unless Harry specifies a new one. | Similar to Bill's scenario, context can be used to keep parameter values, from one intent to another. Contexts are also used to repair a conversation that has been broken by a user or system error, as well as branch conversations to different intents, depending on the user's response. |

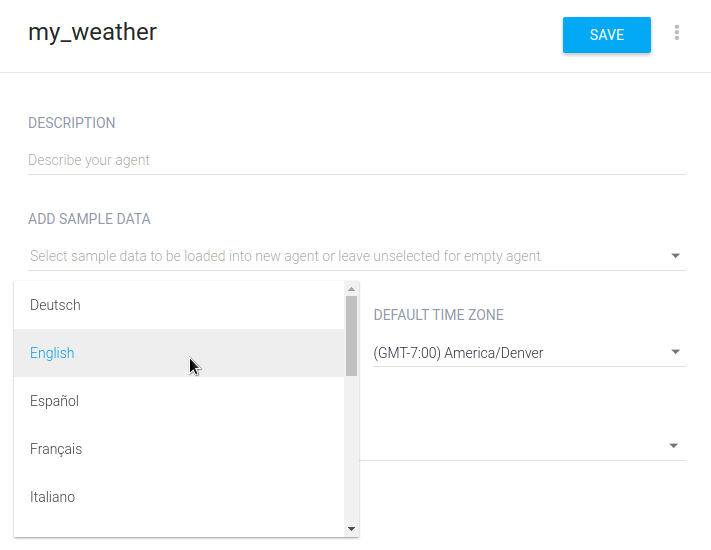
# Building Your First Agent

In this example, you'll build a basic weather agent that provides simple, built- in responses to user's requests. An [agent](https://dialogflow.com/docs/agents) is essentially the container or project and it contains [intents](https://dialogflow.com/docs/intents), [entities](https://dialogflow.com/docs/entities), and the responses you want to deliver to your user. Intents are the mechanisms that pick up what your user is requesting (using entities) and direct the agent to respond accordingly.

For simple replies that don't include information gathered outside of the conversation, you can define the responses directly in the intents. More advanced responses can be made using your own logic and [webhook](https://dialogflow.com/docs/fulfillment) for fulfillment. In later sections, you'll add fulfillment so the agent can reply with information it gathers from an external weather API call. For now you'll cover the basics.

## **Create an agent**

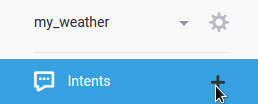
A Dialogflow agent represents the conversational interface of your application, device, or bot. To create an agent:

1. If you don't already have a Dialogflow account, [sign up](https://console.dialogflow.com/api-client/authorize_url_google/nopopup). If you have an account, [login](https://console.dialogflow.com/api-client/#/login).
2. Click on **Create Agent** in the left navigation and fill in the fields.  
   **Note:** Every agent you create must have a [Google Project](https://console.cloud.google.com/) associated with it. 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. You'll use this in the next section of the guide.
3. Click the **Save** button.

## **Create an intent**

An [intent](https://dialogflow.com/docs/intents) maps what a user says with what your agent does. This first intent will cover when the user asks for the weather.

To create an intent:

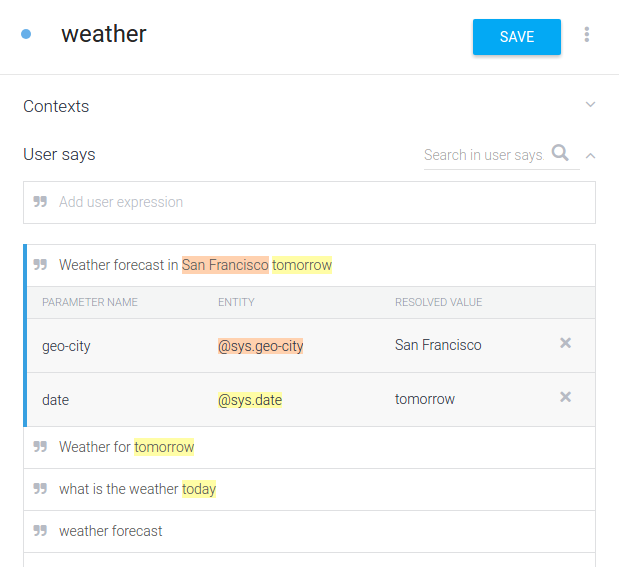


1. Click on the plus icon add next to **Intents**. You will notice some default intents are already in your agent. Just leave them be for now.
2. Enter a name for your intent. This can be whatever you'd like, but it should be intuitive for what the intent is going to accomplish.
3. In the **Training Phrases** section, enter examples of what you might expect a user to ask for. Since you're creating a weather agent, you want to include questions about locations and different times. The more examples you provide, the more ways a user can ask a question and the agent will understand.

Enter these examples:

* + What is the weather like
  + What is the weather supposed to be
  + Weather forecast
  + What is the weather today
  + Weather for tomorrow
  + Weather forecast in San Francisco tomorrow

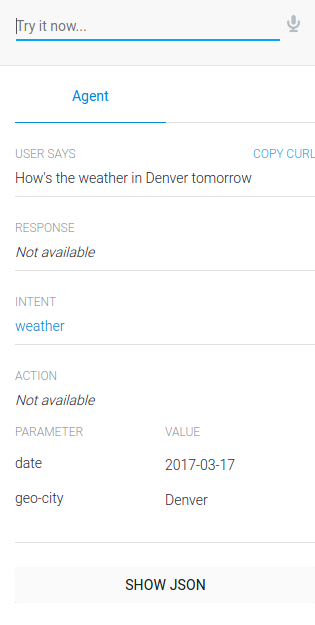
In the last three examples you'll notice the words today and tomorrow are highlighted with one color, and San Francisco is highlighted with another. This means they were annotated as parameters that are assigned to existing date and city [system entities](https://dialogflow.com/docs/reference/system-entities). These date and city parameters allow Dialogflow to understand other dates and cities the user may say, and not just "today", "tomorrow", and "San Francisco".



1. Once you're done, click the **Save** button.

## **Try it out**

Now that your agent can understand basic requests from the user, try out what you have so far.



In the console on the right, type in a request. The request should be a little different than the examples you provided in the **Training Phrases**section. This can be something like "How's the weather in Denver tomorrow". After you type the request, hit "Enter/Return".

You won't get a conversational response, but you should see data in the following fields of the console:

* **Response** - "Not Available" because the agent doesn't have any actual responses set up yet
* **Intent** - weather means the request hit the "weather" intent
* **Parameters** - date and geo-city have their respective values from the request (e.g. tomorrow's date and "Denver")

## **Add responses**

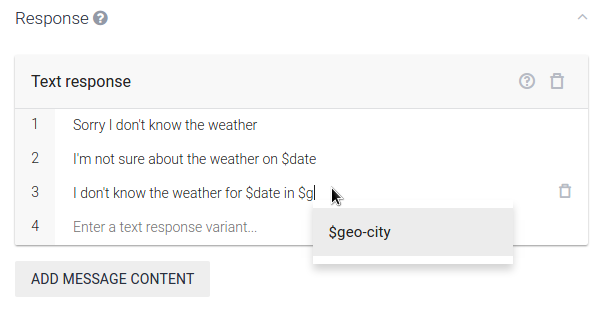
Now you'll add basic responses to the intent so the agent doesn't just sit there in awkward silence. As mentioned before, responses added to an intent don't use external information. So this will only address the information the agent gathered from the user's request.

If you've navigated away from the "weather" intent, return to it by clicking on **Intents** and then the "weather" intent.

1. In the same way you entered the **Training Phrases**, add the lines of text below in the [**Response**](https://dialogflow.com/docs/intents#elements_of_an_intent) section:
   * Sorry I don't know the weather
   * I'm not sure about the weather on $date
   * I don't know the weather for $date in $geo-city but I hope it's nice!

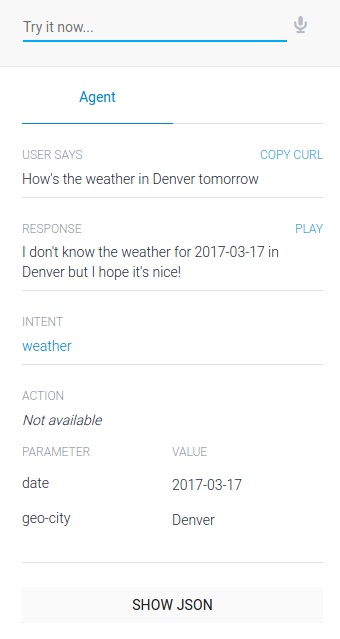
You can see the last two responses [reference entities](https://dialogflow.com/docs/actions-and-parameters#referring_to_parameter_values_in_text_response) by their value placeholders. $date will insert the date from the request, and $geo-city will insert the city.

When the agent responds, it takes into account the parameter values gathered and will use a reply that includes those values it picked up. For example, if the request only includes a date, the agent will use the second response from the list.



1. Once you're done, click the **Save** button.

## **Try it out, again**



Back in the console on the right, enter the same request or enter a new one. You should see the following data in the console fields:

* **Response** - shows an appropriate response from the ones provided
  + The response chosen is based off of the values you provide in the query (e.g. By providing only the date, the agent should respond with the option that only includes the date)
* **Intent** - weather again a successful trigger of the intent
* **Parameter** - the values you provided in your query, should be reflected in the appropriate response

## **What's next?**

In the next part, you'll add fulfillment to get relevant weather information via an API call. You'll also ensure the conversation with your users goes smoothly, even if they wander off your conversational path, with branching.

# Basic Fulfillment and Conversation Setup

In the previous guide, you built a basic weather agent that can recognize requests from users. In order to serve the actual information the user is requesting, you'll need to setup [fulfillment](https://dialogflow.com/docs/fulfillment), which requires deploying a service and calling an API.

Additionally, you want the agent to manage and repair the conversation if it doesn't go as expected, so you'll add some [contexts](https://dialogflow.com/docs/contexts) and [fallback intents](https://dialogflow.com/docs/intents#fallback_intents).

Some of the information in this guide is more developer-centric, but don't be discouraged. If you follow the steps as they're outlined, you will have a deployed, working weather agent.

Fulfillment (Webhook)

In order to add response logic or include the results of an API call in your agent's response, you need to setup [fulfillment](https://dialogflow.com/docs/fulfillment) for the agent. This includes some basic JavaScript and setting up and hosting the files in a cloud service. This guide uses a [Google Cloud Project](https://console.cloud.google.com/) for the hosting and deployment.

Create a Starter JS File

To start, create a directory on your local system for the code:

* Linux or Mac OS X:
* mkdir ~/[PROJECT\_NAME]
* cd ~/[PROJECT\_NAME]
* Windows:
* mkdir %HOMEPATH%[PROJECT\_NAME]
* cd %HOMEPATH%[PROJECT\_NAME]

Then create an index.js file in the project directory you just created, with the following code:

/\*  
\* HTTP Cloud Function.  
\*  
\* @param {Object} req Cloud Function request context.  
\* @param {Object} res Cloud Function response context.  
\*/  
exports.helloHttp = function helloHttp (req, res) {  
  response = "This is a sample response from your webhook!" //Default response from the webhook to show it's working  
  
  
  res.setHeader('Content-Type', 'application/json'); //Requires application/json MIME type  
  res.send(JSON.stringify({ "speech": response, "displayText": response   
  //"speech" is the spoken version of the response, "displayText" is the visual version  
  }));  
};

Setup Google Cloud Project

1. Follow ["Before you begin"](https://cloud.google.com/functions/docs/quickstart) steps 1-5

**Note:** For step 1 at the link above, make sure to select the Google Project related to the agent when you first created it.

To locate the Google Project name, make sure you are viewing the correct agent and click the gear icon settings. The project name is under **Google Project**.

Use this project name in all the steps of the processes outlined in the linked page above.

1. Deploy the function

gcloud beta functions deploy helloHttp --stage-bucket [BUCKET\_NAME] --trigger-http

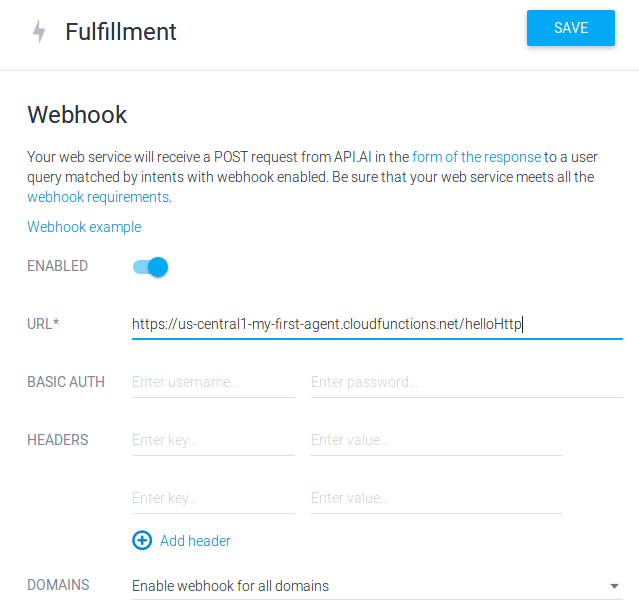
* + helloHttp is the name of our project. You should have created your project in [step 1](https://cloud.google.com/functions/docs/quickstart#before-you-begin) and set the project at the end of [step 4](https://cloud.google.com/sdk/docs/) when you initialized gcloud.**Note:** The project name is also used in the **index.js** code above. You will need to update the two mentions of **helloHttp** to use your project name.
  + --stage-bucket [BUCKET\_NAME] can be found by going to your related Google Cloud project and click on **Cloud Storage** under the Resources section.
  + --trigger-http [More information](https://cloud.google.com/functions/docs/calling/http)

**Note:** If you need to make edits the index.js file for any reason, you will need to deploy the function again to update index.js in your Google Cloud Project. Use the exact same command to redeploy the same function.

Once completed, the status and information related to the function will be displayed. Make note of the httpsTrigger url. It should look something like this:

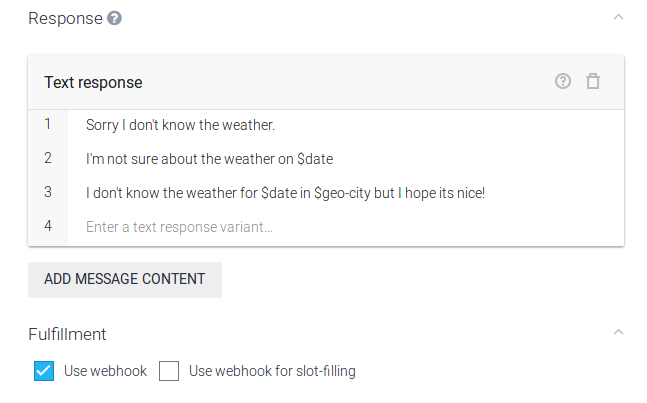
https://[REGION]-[PROJECT\_ID].cloudfunctions.net/helloHttp

Enable Webhook in Dialogflow



1. In Dialogflow, make sure you're in the correct agent and click on **Fulfillment** in the left hand menu
2. Toggle the switch to enable the webhook for the agent
3. In the **URL** text field, enter the httpTrigger url you got when you deployed your function
4. Click **Save**

Enable Fulfillment in Intent



1. Navigate to the "weather" intent
2. Expand the **Fulfillment** section at the bottom of the page
3. Check the **Use Webhook** option
4. Click **Save**

Try it out

In the Dialogflow test console, enter "weather". You will see the webhook response we defined in the function. This means the webhook is working! You should also see the two parameters we need from our user, date and geo-city.

Setup Weather API

Get API Key

For this sample, we use the WWO (World Weather Online) service, so you'll need to [get an API key](https://developer.worldweatheronline.com/api/). Once you register, login and make note of your API key.

Update Code

Now that we have an API key, we can make requests of the weather service and get actual data back from them.

Replace the current code in index.js with the code below. This adds communication with the weather service, our API key, and functions to handle our queries.

**Note:** Add your WWO API key in line 19 **const wwoApiKey = ''**

// Copyright 2017, Google, Inc.  
// Licensed under the Apache License, Version 2.0 (the 'License');  
// you may not use this file except in compliance with the License.  
// You may obtain a copy of the License at  
//  
//    http://www.apache.org/licenses/LICENSE-2.0  
//  
// Unless required by applicable law or agreed to in writing, software  
// distributed under the License is distributed on an 'AS IS' BASIS,  
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
// See the License for the specific language governing permissions and  
// limitations under the License.  
  
'use strict';  
const http = require('http');  
const host = 'api.worldweatheronline.com';  
const wwoApiKey = '[YOUR\_API\_KEY]';  
exports.weatherWebhook = (req, res) => {  
  // Get the city and date from the request  
  let city = req.body.result.parameters['geo-city']; // city is a required param  
  // Get the date for the weather forecast (if present)  
  let date = '';  
  if (req.body.result.parameters['date']) {  
    date = req.body.result.parameters['date'];  
    console.log('Date: ' + date);  
  }  
  // Call the weather API  
  callWeatherApi(city, date).then((output) => {  
    // Return the results of the weather API to Dialogflow  
    res.setHeader('Content-Type', 'application/json');  
    res.send(JSON.stringify({ 'speech': output, 'displayText': output }));  
  }).catch((error) => {  
    // If there is an error let the user know  
    res.setHeader('Content-Type', 'application/json');  
    res.send(JSON.stringify({ 'speech': error, 'displayText': error }));  
  });  
};  
function callWeatherApi (city, date) {  
  return new Promise((resolve, reject) => {  
    // Create the path for the HTTP request to get the weather  
    let path = '/premium/v1/weather.ashx?format=json&num\_of\_days=1' +  
      '&q=' + encodeURIComponent(city) + '&key=' + wwoApiKey + '&date=' + date;  
    console.log('API Request: ' + host + path);  
    // Make the HTTP request to get the weather  
    http.get({host: host, path: path}, (res) => {  
      let body = ''; // var to store the response chunks  
      res.on('data', (d) => { body += d; }); // store each response chunk  
      res.on('end', () => {  
        // After all the data has been received parse the JSON for desired data  
        let response = JSON.parse(body);  
        let forecast = response['data']['weather'][0];  
        let location = response['data']['request'][0];  
        let conditions = response['data']['current\_condition'][0];  
        let currentConditions = conditions['weatherDesc'][0]['value'];  
        // Create response  
        let output = `Current conditions in the ${location['type']}   
        ${location['query']} are ${currentConditions} with a projected high of  
        ${forecast['maxtempC']}°C or ${forecast['maxtempF']}°F and a low of   
        ${forecast['mintempC']}°C or ${forecast['mintempF']}°F on   
        ${forecast['date']}.`;  
        // Resolve the promise with the output text  
        console.log(output);  
        resolve(output);  
      });  
      res.on('error', (error) => {  
        reject(error);  
      });  
    });  
  });  
}

Deploy Function (again)

Now that our function is different, we need to deploy again and alter the command. This is due to the code having a new function name exported.

gcloud beta functions deploy weatherWebhook --stage-bucket [BUCKET\_NAME] --trigger-http

**Note:** You can use the same **[BUCKET\_NAME]** from the earlier steps.

Once the new function is deployed, make a note of the new httpTrigger url. It should look something like this:

https://[REGION]-[PROJECT\_ID].cloudfunctions.net/weatherWebhook

Update Fulfillment in Dialogflow

1. Return to Dialogflow and click on **Fulfillment** in the left hand menu
2. Replace the current URL with the new httpTrigger url.

https://[REGION]-[PROJECT\_ID].cloudfunctions.net/weatherWebhook

1. Click **Save**

Conversation Branching

We can't expect users will always provide all the information our agent needs to fulfill their request. In the case of our weather agent, a city and a date are required as inputs for our function. If no date is provided, we can assume the user is referring to "today" or the current date, but there's no way to gather the user's location or city through Dialogflow so we need to make sure we collect that.

Making Location Required

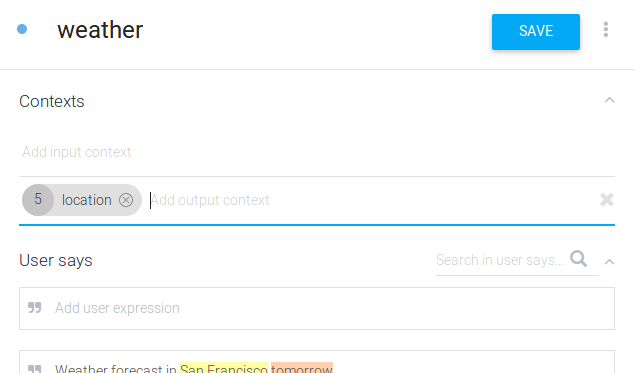
1. In the "weather" intent, locate the geo-city parameter and check the **Required** option. This will reveal an additional column called **Prompts**. These are responses the agent will give when this specific data isn't provided by the user.
2. Click on **Define prompts** and enter the following response:
   * For what city would you like the weather?
3. Click **Close**
4. Click **Save** on the Intent page

Give it a go!

Again, enter "weather" into the console and you should see the prompt to collect the city.

Add Location Context

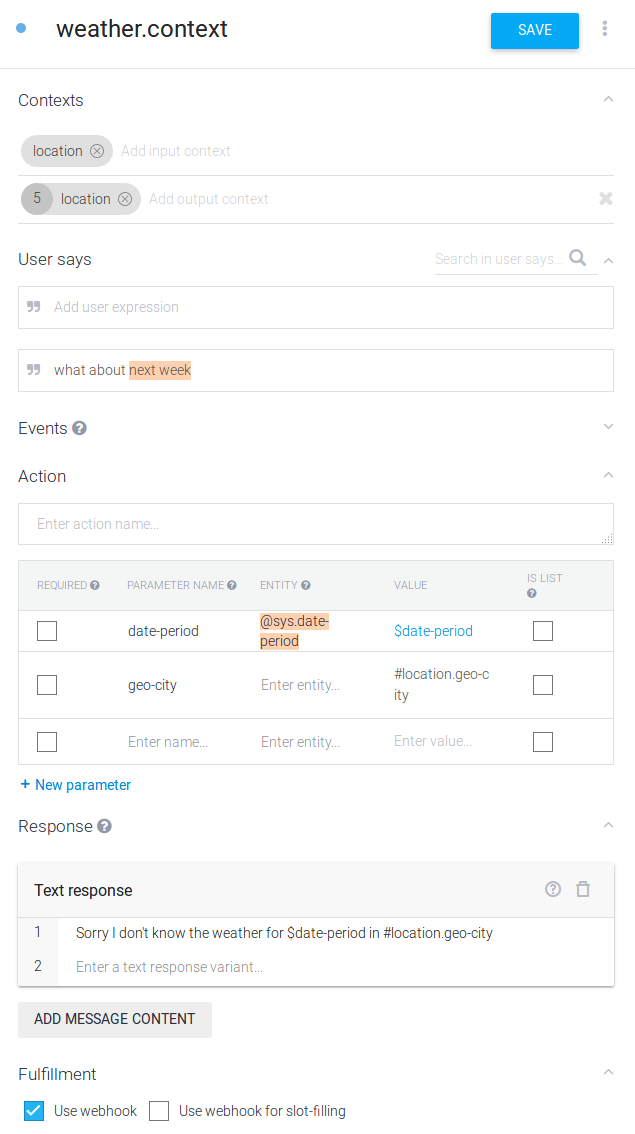
In order to refer to data collected in a previous intent, we need to set an [output context](https://dialogflow.com/docs/contexts#output_contexts). In this case we want to "reuse" the value collected for location.



1. In the "weather" intent, click on **Contexts** to expand the section
2. In the **Add output context** field, type "location" and press "Enter" to commit the context
3. Click **Save**

Create a New Intent for Context

We want to be able handle additional questions using the same location, without asking the user for the data again. Now that we've set an output context, we can use it as the [input context](https://dialogflow.com/docs/intents#input_contexts) for the intent that handles the additional questions.



1. Click on **Intents** in the left hand menu or click the plus icon add to create a new intent
2. Name the intent "weather.context"
3. Set the input and output context as "location"
4. Add the following **Training Phrase**:
   * What about tomorrow
5. Add a new parameter with the following information:
   * **Parameter Name:** geo-city
   * **Entity:** *empty*
   * **Value:** #location.geo-city
6. Add the following reply in the **Response** section:
   * "Sorry I don't know the weather for $date-period in #location.geo-city"
7. Click on **Fulfillment** in the menu to expand the section and check the **Use webhook** option
8. Click **Save**

Take it for a test drive!

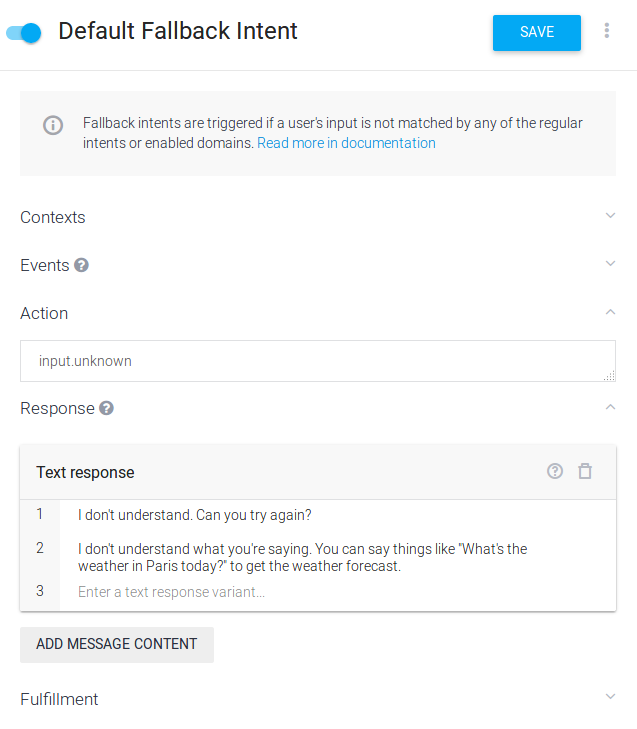
In the Dialogflow console, enter "weather" and get the reply asking for the city. Then enter a city of your choosing. You'll see a response like the one above, which includes the data retrieved from the service. You can then ask questions like "What about tomorrow" to retrieve the forecast for that date.

Conversation Repair

Now that the main conversational function of our agent is complete, we want to make sure our agent welcomes the user and knows how to respond to requests that are not related to the weather.

Editing the Default Fallback Intent

When our user responds with an unrelated query, we want our agent to reply gracefully and direct the user back into "familiar territory." For this we'll edit the existing [Default Fallback Intent](https://dialogflow.com/docs/intents#fallback_intents).



**Note:** You could leave Default Fallback Intent as it is, but it's best to customize your agent's responses to fit the primary function of your application.

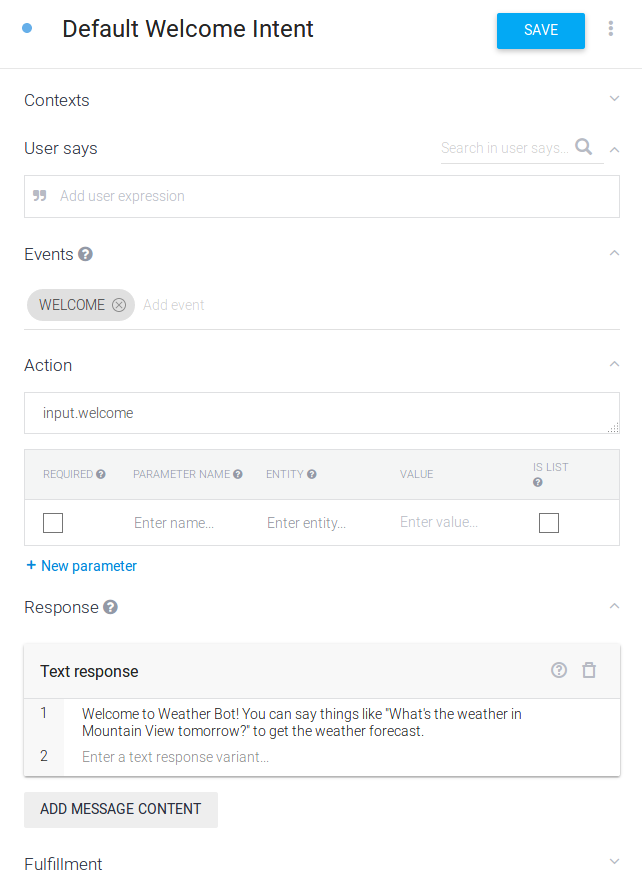
1. Click on **Intents**, then **Default Fallback Intent**
2. Click on the trash can icon delete in the upper right hand corner of the **Text Response** table
3. Click on **Add Message Content** and choose **Text Response**
4. Enter the following responses:
   * I didn't understand. Can you try again?
   * I don't understand what you're saying. You can say things like "What's the weather in Paris today?" to get the weather forecast.
5. Click **Save**

One more time!

Enter an unrelated request into the console and you'll get one of the two fallback responses.

Editing the Welcome Intent

Finally, we want our agent to greet users and maybe provide some ideas as to what they can ask.



**Note:** You could leave Default Welcome Intent as it is, but it's best to customize your agent's responses to fit the primary function of your application.

1. Click on **Intents**, then **Default Welcome Intent**
2. Click on the trash can icon delete in the upper right hand corner of the **Text Response** table
3. Click on **Add Message Content** and choose **Text Response**
4. Enter the following response:
   * Welcome to Weather Bot! You can say things like "What's the weather in Mountain View tomorrow?" to get the weather forecast.
5. Click **Save**