Step by step guide to DialogFlow (API.AI) Part 1 – Introducing the FAQ bot

Table of Contents for Step by Step Guide

This is the first part of a multi-part guide I am writing on API.AI. This series came about as a result of a question I saw on the API.AI forum, where someone asked if there were any step by step guides. When I began learning API.AI, I sometimes felt that many of the online resources do not really take a step by step approach – here, by step-by-step I mean an approach where they focus on one thing at a time as they teach.

The way bot building frameworks are built, it does look like you need to simultaneously learn many different concepts and isolated concepts are not very helpful. So my first goal is to create what I call "microbots" – they are too focused and even a little too contrived to be of much practical use, but they serve as useful resources for learning.

Requirements

This is intended as a resource and guide aimed mainly at programmers. At the moment, building moderately complex chatbots is not quite "push button". As we go further along, I expect that you will be able to write some code on your own to get your bot working.

If you are completely non-technical, you can still follow along for the first three bots I will be building where I will teach you how intents, entities and contexts work. You can also ask your questions in the API.AI discussion forum. If you are interested in private online coaching, see end of post for contact info.

The FAQ Bot

The Frequently Asked Questions bot is one such microbot which may not have too many practical applications. *An FAQ bot simply returns the best resource/post/article on a given website which matches the user's question.* The FAQ bot is limited on many levels.

1. A search engine can usually do what the FAQ bot can
2. If it only provides "one and done" answers to a user's question, it is not very conversational. There is nothing chatty about such a bot.
3. It is not likely to be as comprehensive as the search on a website

However, even an FAQ bot can actually be an improvement sometimes. For example, the default WordPress search is not very good when the keywords used are not present in the article. Since the FAQ bot uses natural language processing techniques, it is more likely to use word stems, synonyms and such and still provide good answers when the keyword varies from what is usually seen in the article.

In addition, suppose you have a website where people repeatedly ask the same type of questions. An FAQ bot can actually be very helpful for them since you can make it work for many different variants of the same question.The FAQ bot we will be building will be an FAQ bot for API.AI itself.

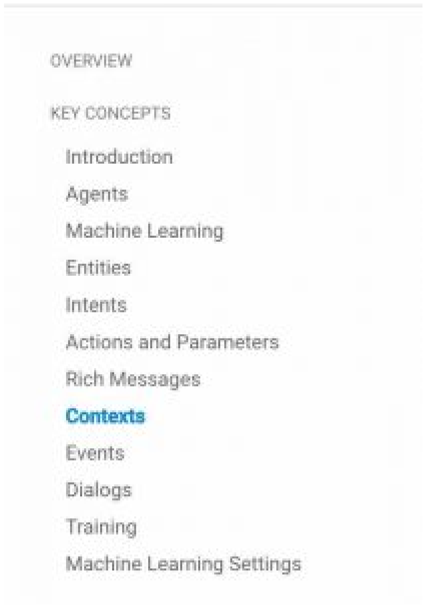
Step by step guide to DialogFlow (API.AI) Part 2 – Preparing the FAQ list

Table of Contents for Step by Step Guide

As mentioned in the previous post, we will be building an FAQ bot which will answer questions about API.AI. So the first thing we need is a resource from where we can get answers about API.AI – and we have one ready: the API.AI documentation.

Preparing the FAQ list

We will look at the different sections in the API.AI documentation, and generate a list of questions that users may have about that section. Since this is for learning purposes and not meant to be exhaustive, we will only consider the high level ideas. We can easily gleam the key concepts by looking at the structure of the API.AI documentation. Here is what the documentation looks like:



For each section under Key Concepts, we will have atleast one FAQ answer. Our goal here is to anticipate what questions should direct users to which answers (sections) on the documentation.

So we will collect each section that can be a standalone answer, and make a list of links in a table. This is what this list looks like for me:

As you can see, the list of links and FAQ questions is not comprehensive. But the idea is straightforward: for each resource (i.e. a link on the left column of the table) you will be anticipating possible questions and adding them into the corresponding FAQ questions in the right column of the table.

In the next post, we will look at understanding intents based on the FAQ list we prepared.

Step by step guide to DialogFlow (API.AI) Part 3 – Creating an agent from the FAQ list

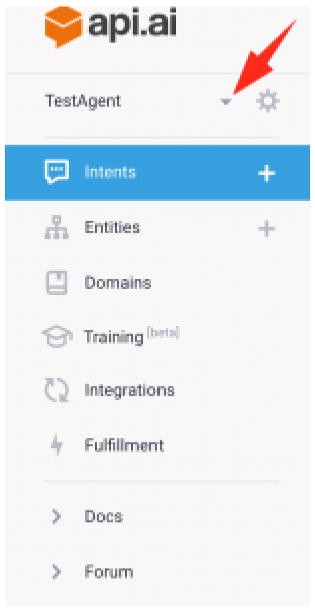
Table of Contents for Step by Step Guide

Now that we have prepared an FAQ list as mentioned in the previous post, we are ready to create our first chatbot.

Log in to api.ai (you can use your Google account if you want).

Create new agent

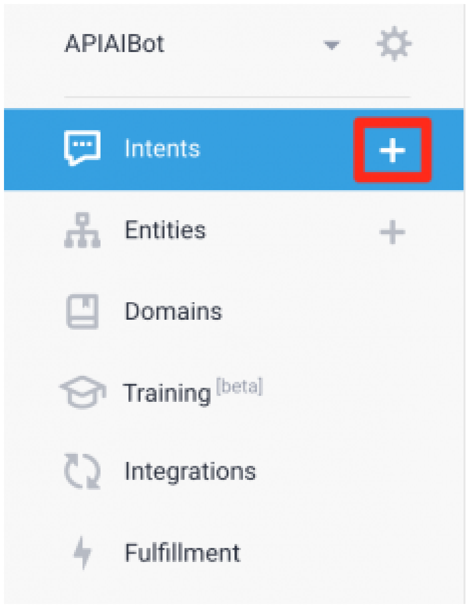
For a fresh account, no agents will be listed. But if you already have some agents created, like mine, you will need to click on the down arrow next to the current agent's name, and scroll down to the bottom of the list and select "Create new agent".



Create a list of intents

Now it is time to populate the intents. As you do this step, it will become clear what intents are – but there is a section to follow which goes into more detail.

1 Click on the + sign next to the Intents in the left pane to create a new Intent

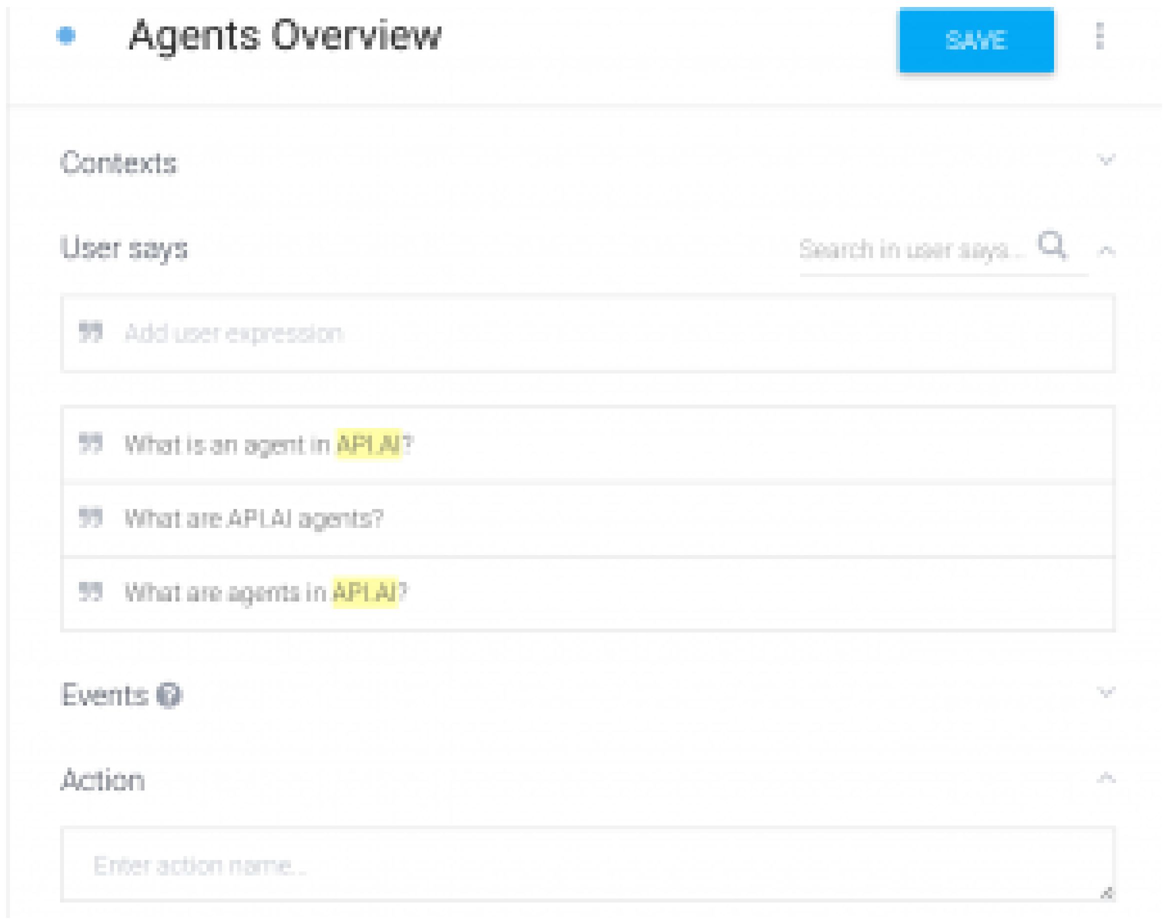


2 Name the Intent

Use the section name as the name of the intent.

3 Add the list of questions

For each item in the right column of the table, add one "User says" entry. Once you are done, your Intent screen should look like below. Ignore the yellow highlights for now.



Agents Overview

Contexts

User says

What is an agent in API.AI?

What are API.AI agents?

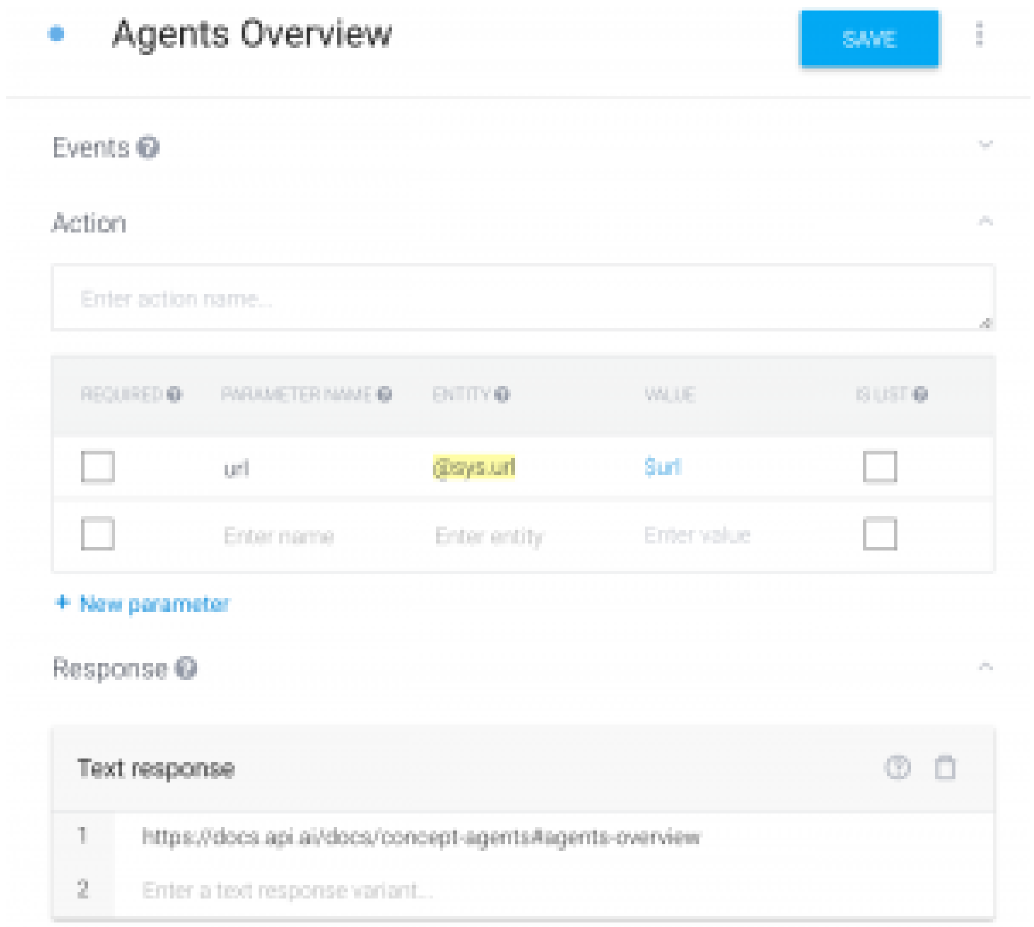
What are agents in API.AI?

Events

Action

4 Add the link in the Response field

Now take the link on the left hand side of the table and add it to the Response text box. Your screen will look like below.



Agents Overview

Events

Action

Enter action name...

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST
☐	url	@sys.url	$url	☐
☐	Enter name	Enter entity	Enter value	☐

+ New parameter

Response

Text response

1 https://docs.api.ai/docs/concept-agents#agents-overview

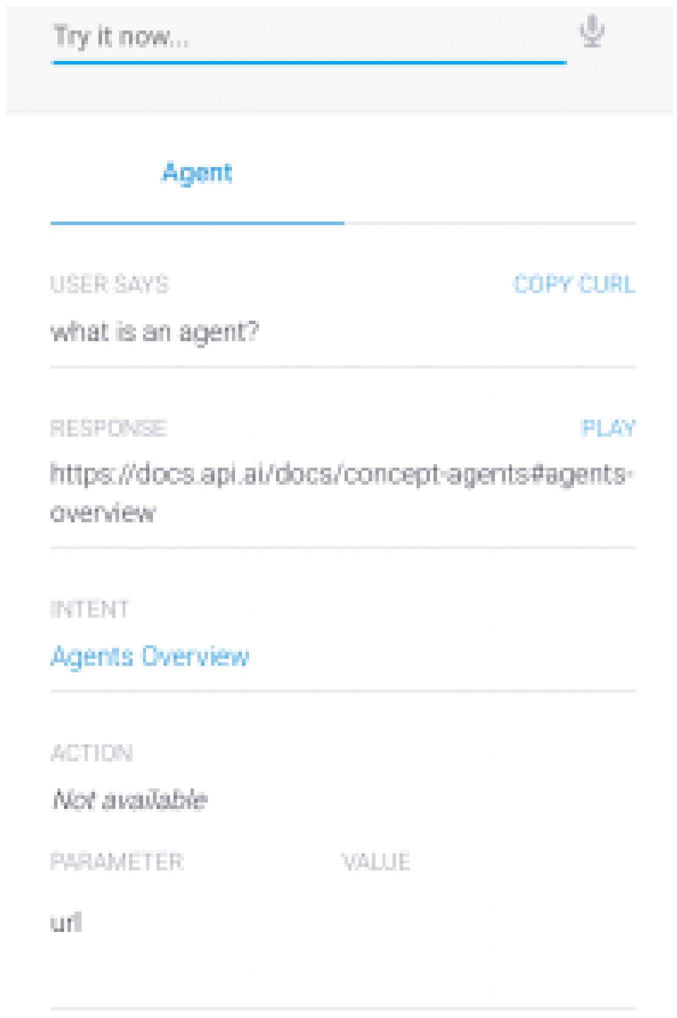
2 Enter a text response variant...

Leave the rest of the fields, like Action and Events, empty.

Repeat steps 1 through 4 for each FAQ link in the table you created from the previous post. When you are finished with all the links, your agent is ready.

Test the agent by using the interactive console on the right hand side

As you are creating the agent, you can keep testing it by using the interactive console on the right side. For example, here is how my agent responds when I ask a question.



Try it now...

Agent

USER SAYS COPY CURL

what is an agent?

RESPONSE PLAY

https://docs.api.ai/docs/concept-agents#agents-overview

INTENT

Agents Overview

ACTION

Not available

PARAMETER VALUE

url

Let us look at this in some detail. First, under User Says, you see what you just typed.

Next, the Response is what the agent will respond with when you deploy it. As you can see, it gives us a link to the API.AI documentation pointing to the definition of agents (as expected).

It also has, under Intent, “Agents Overview” which is the name I gave the intent. So API.AI has determined that the query that the user typed is similar to the User Says phrases which we had used when defining this intent. You should note that the phrase used for testing isn’t an exact match with any of the items in the list of “User says” phrases for that intent, *but it is close enough that API.AI can decide that is the intent it should be mapped to*.

Finally, it also shows a couple more things such as Action and Parameter, which we will ignore for now.

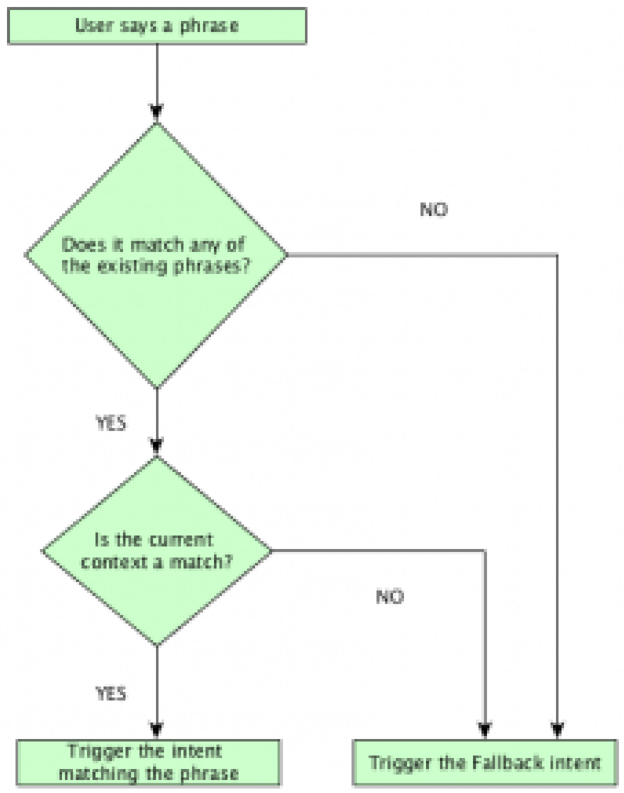
Step by step guide to DialogFlow (API.AI) Part 4 – Intent Mapping

Table of Contents for Step by Step Guide

In the previous post, you used the interactive console on the right hand side to test your agent's responses for different questions from the user.

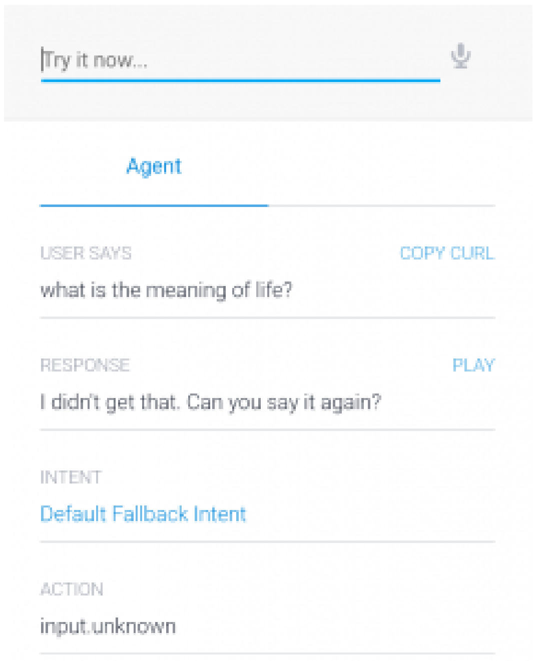
In this post, I will give an overview of what is happening, and what is the expected behavior.

The main feature of a bot building service such as API.AI is to look at the words typed (or said) by the user, and **map it to a list of expected phrases**. This is called intent mapping. While there is a lot of machine learning logic which goes on underneath the hood to make this happen, for the bot programmer, this is like a black box which works as follows.


User says a phrase
Does it match any of the existing phrases?
NO
YES
Is the current context a match?
NO
YES
Trigger the intent matching the phrase
Trigger the Fallback intent
Intent Mapping in API.AI

The first condition in the flowchart – "Does it match any of the existing phrases?" is based off the list of "User says" questions you created for the FAQ. API.AI is looking for similarity.

Suppose, the user were to type "What is the meaning of life?", it will not match any of the "User says" phrases we have used. So the flowchart will select the first (top) NO branch and trigger the fallback intent.


Try it now...

Agent

USER SAYS
what is the meaning of life?

RESPONSE
I didn't get that. Can you say it again?

INTENT
Default Fallback Intent

ACTION
input.unknown

There is also a second condition in the flowchart, as you have noticed. This condition checks if the current context is a match. We will look into contexts in more detail in a later post, but you can think of it as an additional criteria that must be satisfied before an Intent is chosen as the right mapping.

This ends the discussion about our first FAQ bot. The goal of building the FAQ bot was to get a basic idea of what Intents are and how intent mapping works. Next we will be looking at entities.

Step by step guide to DialogFlow (API.AI) Part 5 – The Wimbledon Finals bot

Table of Contents for Step by Step Guide

Hopefully you are familiar with the game of tennis and know something about the Wimbledon tournament, generally considered to be the most prestigious of the Grand Slam tournaments. Even if you are not, you certainly know the concept of a tournament winner and the tournament runner-up.

The WimbledonFinalsBot has only one purpose for its existence. When you ask it “Who was the winner of the women’s Wimbledon 1991?” as an example, it will give you the name of the women’s tennis player who won the tournament that year. Similarly, you can ask the bot about who was the runner up in a particular year. You can also ask the same question for the men’s players. To make it a little easier to create the chatbot, the bot will only provide answers about the tournaments which took place after 1980. We will be using the Wikipedia’s page about men’s winners and women’s winners for reference.

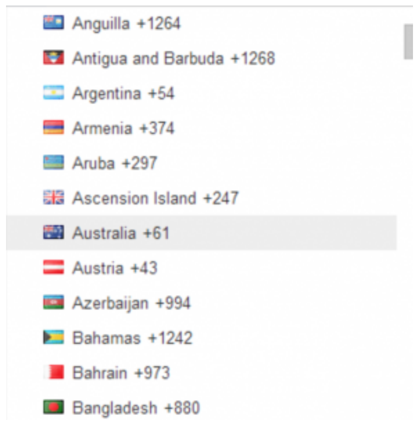
One thing you will notice, when comparing with the FAQ chatbot, is that the answer supplied by the bot has to be *dynamic*. This is where the concepts of entities comes into the picture.

Here is how entities are defined in API.AI's documentation:

Entities represent concepts and serve as a powerful tool for extracting parameter values from natural language inputs.

The broad definition of entities is that they are concepts. For a programmer, a good way to think about entities is that they are instances of a class.

Consider the ubiquitous dropdown around the web that shows a list of country names. Here, the entity is "Country". The actual values you see in that dropdown are the entity values.



A second analogy that programmers can relate to is the concept of the placeholder when you are printing strings. This is actually a very useful analogy, because when you are interacting with API.AI entities you will be using syntax which will look familiar if you are already comfortable using placeholders in your programming.

Step by step guide to DialogFlow (API.AI) Part 6 – Populating the entities

Table of Contents for Step by Step Guide

In the previous post, we learnt a little about the WimbledonFinalsBot (henceforth WFBot) we will be building.

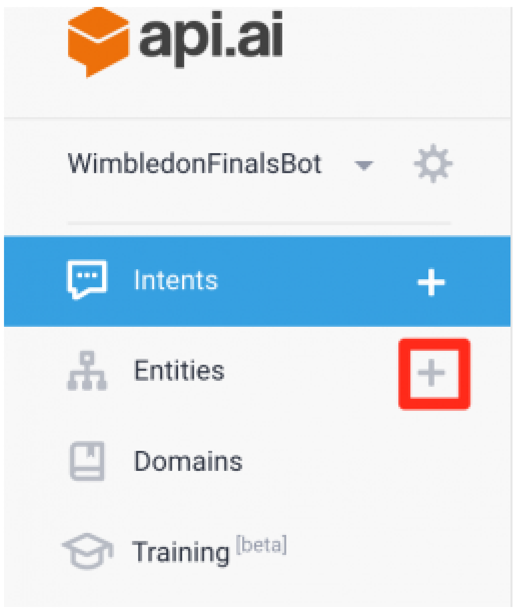
You will start by creating a new agent just like for the FAQ bot. Once you have created an agent, the next step will be to declare the Intents. If you remember from the previous posts, the intents should be the different phrases you expect your user to ask.

In our case, we have only 4 intents.

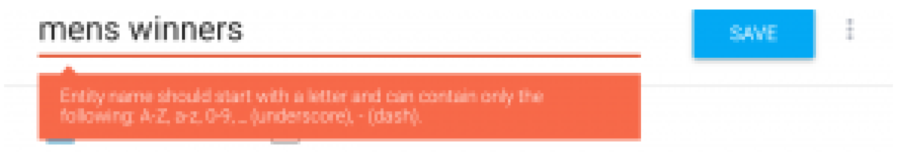
1. Who won the men’s tournament in year X?
2. Who won the women’s tournament in year X?
3. Who was the women’s runnerup in year X?
4. Who was the men’s runnerup in year X?

However, before we can declare our Intents, we should first declare our entities. I mention the Intents first because I am establishing the motivation as to what kind of questions the user will ask the bot, so we can better understand how to declare and populate our entities.

Click on the + sign next to the Entities in the left pane.



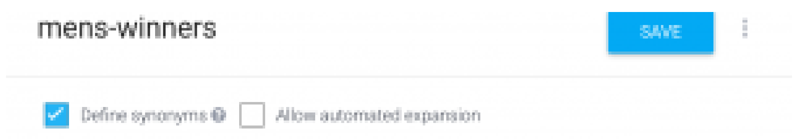
I am going to populate the values for the men’s winners in the first step. Take a look at what happens when I type mens winners into the entity name.



Entity name should start with a letter and can contain only the following: A-Z, a-z, 0-9, _ (underscore), - (dash).

API.AI complains that there should be no spaces in the name. As you will see later, this will be an important constraint which will help us when we use this entity later.

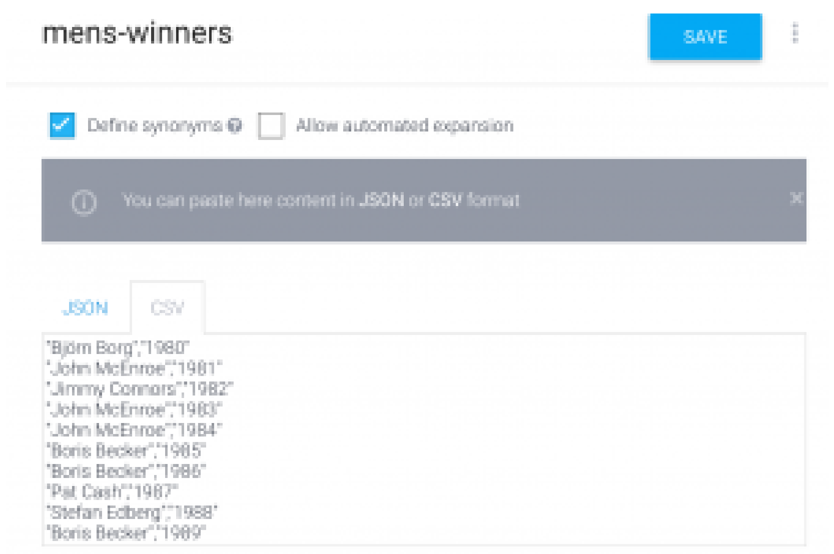
So I will call this entity mens-winners (no spaces).


mens-winners

The next step is to fill out the values.

There is the slow way – you can type out the values one by one. But you can also import these values if you can get the data in CSV or JSON format.

To do this, click on the three vertical dots next to the save button, click on “Switch to raw mode”, and fill out the values in the format of your choice. I have uploaded the input in the CSV format below.


mens-winners

You can paste here content in JSON or CSV format

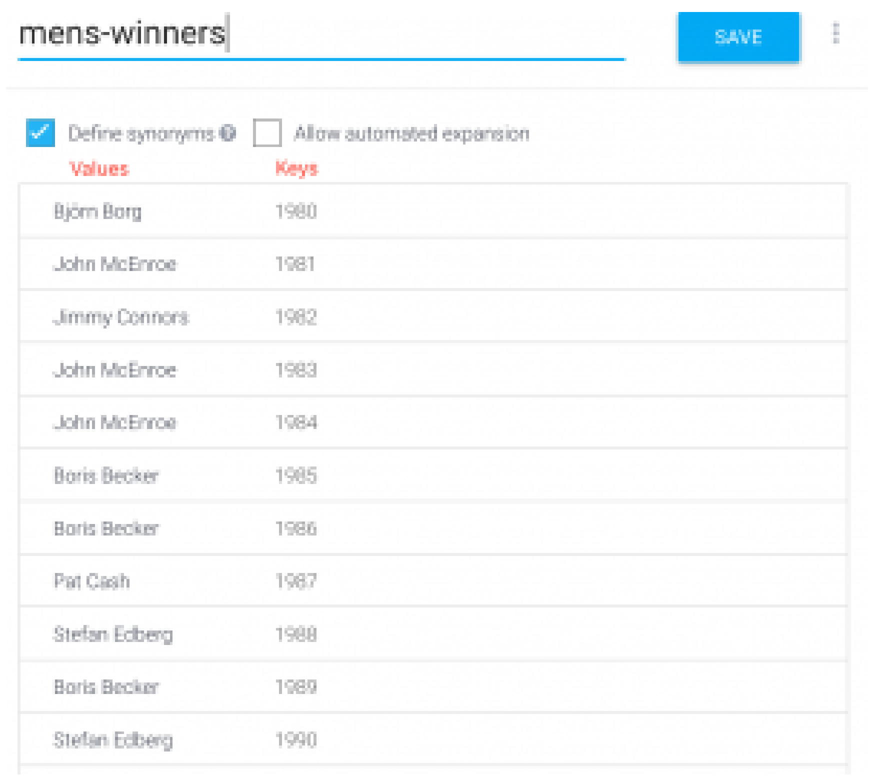
```
"Björn Borg","1980"
"John McEnroe","1981"
"Jimmy Connors","1982"
"John McEnroe","1983"
"John McEnroe","1984"
"Boris Becker","1985"
"Boris Becker","1986"
"Pat Cash","1987"
"Stefan Edberg","1988"
"Boris Becker","1989"
```

Next, we will see how to declare the intents so we can query this data.

Step by step guide to DialogFlow (API.AI) Part 7 – Declaring Intents with Entities

Table of Contents for Step by Step Guide

In the previous post, we populated the mens-winners entity with a list of values. When you see the Entity in the Editor mode (as opposed to the raw mode you used for uploading CSV), you will see the following:



Values	Keys
Björn Borg	1980
John McEnroe	1981
Jimmy Connors	1982
John McEnroe	1983
John McEnroe	1984
Boris Becker	1985
Boris Becker	1986
Pat Cash	1987
Stefan Edberg	1988
Boris Becker	1989
Stefan Edberg	1990

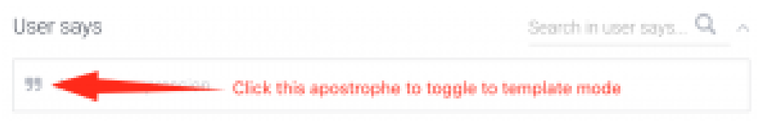
If you notice, I have added the column names Keys and Values to annotate this image. I think of this as a sort of "inverted dictionary" format. The first word you will type is the value of the dictionary (who won?), and the synonyms you will define will contain exactly one item, which is the key of the dictionary (which year). That is, winner[1987] = Pat Cash.

As we go further, you will see why we need to declare it in this inverted dictionary like format.

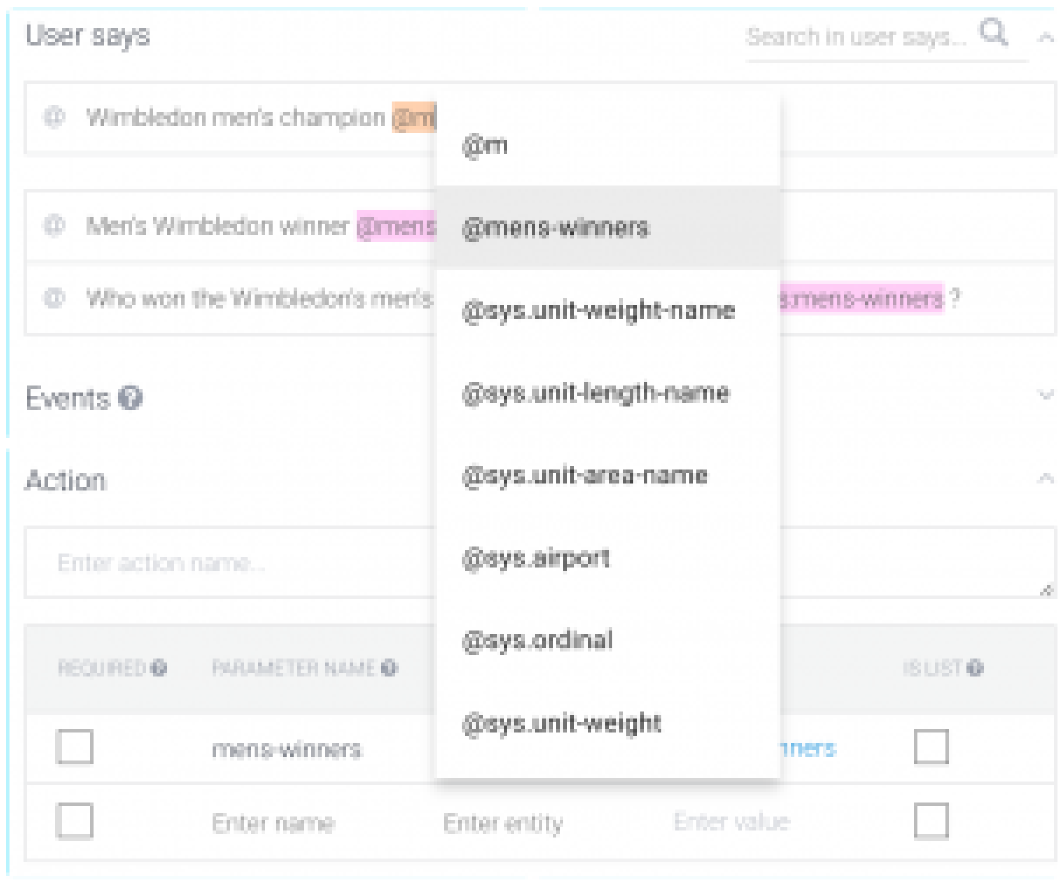
Declaring the Intent

I will start by declaring a single intent corresponding to the entity we have created. We want to know the winner of the Wimbledon's mens title in a particular year.

So create a new Intent, and – **this is very important** – click on the apostrophe at the beginning of the User says textbox and toggle it to the @ sign.



When you click and toggle into the template mode, the apostrophe will turn into an @ sign. More importantly, you will be able to refer to the entities you have already created – by using the @ prefix you will see autocompletion options.



Wimbledon men's champion @m

Men's Wimbledon winner @mens

Who won the Wimbledon's men's ... :mens-winners ?

mens-winners

Now type out the following into the User says box (in three different User says boxes), while remembering to keep the text box in template mode (the default is example mode which has the apostrophe at the beginning)

Wimbledon men’s champion @mens-winners:mens-winners

Men’s Wimbledon winner @mens-winners:mens-winners

Who won the Wimbledon’s men’s championship in @mens-winners:mens-winners ?

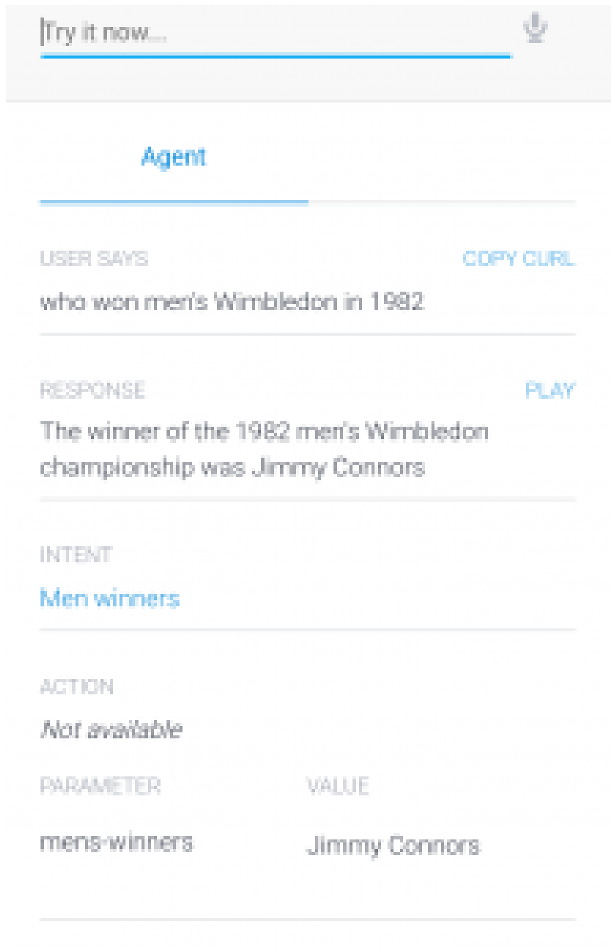
If you remember from the previous FAQ bot, the goal is to create many variants of the same question. This helps API.AI understand the patterns which could be used when people are asking about the same thing.

In the section called Response, input the following:

The winner of the $mens-winners.original men’s Wimbledon championship was $mens-winners

Testing what we have till now

Now let us go to the test console and try the following query, you can see that the response is as expected.


Try it now...

Agent

USER SAYS
who won men's Wimbledon in 1982

RESPONSE
The winner of the 1982 men's Wimbledon championship was Jimmy Connors

INTENT
Men winners

ACTION
Not available

PARAMETER	VALUE
mens-winners	Jimmy Connors

Let us look at the format of the Response we just typed out. When we refer to $mens-winner, API.AI will use it as a *placeholder* for the "Value" of the entity. And if you remember the column names annotated from the picture above, when the user asks the question "who won men's Wimbledon in 1982", a few different things are going on under the hood.

In the next part, we will look "under the hood" and see what is happening.

Step by step guide to DialogFlow (API.AI) Part 8 – Completing the WFBot Intent definitions

Table of Contents for Step by Step Guide

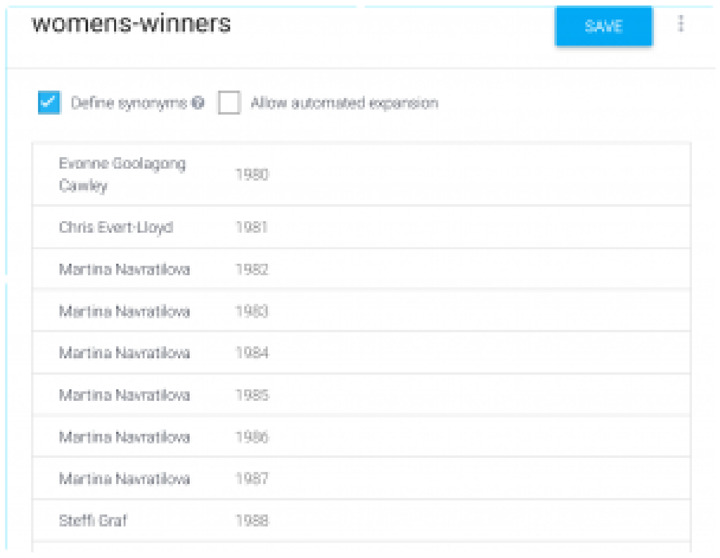
You have seen that for the first type of question (who won the men's Wimbledon in year X?), the agent is able to provide the correct response. All that is left to do to complete the bot is to

a) populate the remaining entities (womens-winners, men-runnersup, women-runnersup)

b) declare the corresponding intents

Populate remaining entities

You can add three more entities, name them womens-winners, men-runnersup and women-runnersup respectively. See the picture for example for what the women-winners entity declaration looks like.

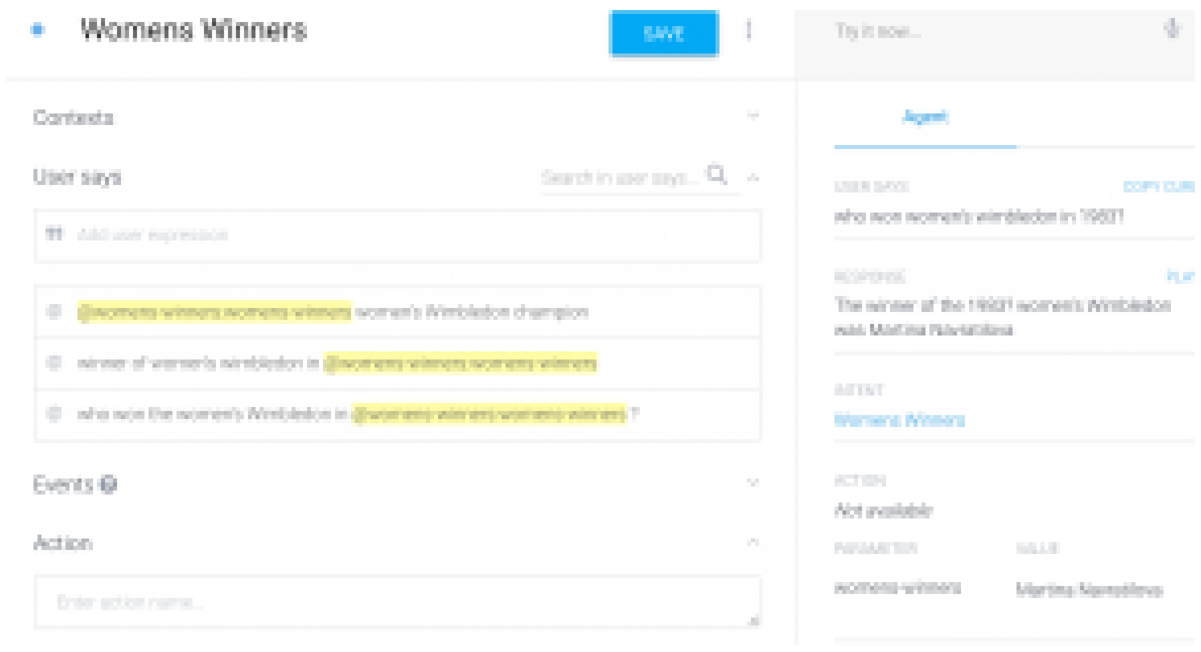

womens-winners

Evonne Goolagong Cawley	1980
Chris Evert-Lloyd	1981
Martina Navratilova	1982
Martina Navratilova	1983
Martina Navratilova	1984
Martina Navratilova	1985
Martina Navratilova	1986
Martina Navratilova	1987
Steffi Graf	1988

Do the same for men-runnersup and women-runnersup

Declare the remaining intents

For women’s winners, here is what the Intent declaration looks like. You can also see that the correct response is provided in the interactive console on the right.



Womens Winners

@womens-winners:womens-winners women's Wimbledon champion

winner of women's wimbledon in @womens-winners:womens-winners

who won the women's Wimbledon in @womens-winners:womens-winners ?

who won women's wimbledon in 1983?

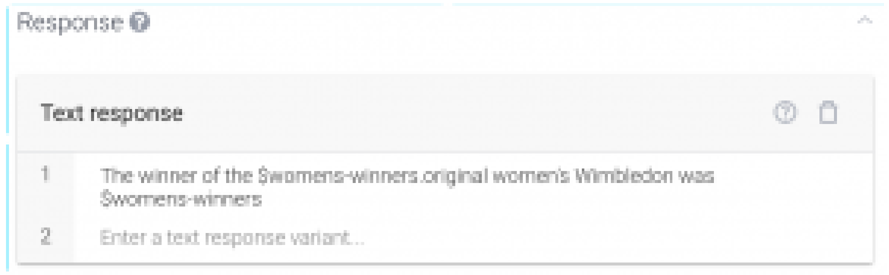
The winner of the 1983? women's Wimbledon was Martina Navratilova

Womens Winners

Not available

PARAMETER	VALUE
womens-winners	Martina Navratilova

The Intent response is declared as follows:



Text response

1 The winner of the $womens-winners.original women's Wimbledon was $womens-winners

2 Enter a text response variant...

By following the same steps for men’s and women’s runner up, you should have the complete WFBot.

Also, do you notice the question mark character next to the year in the agent’s response on the right. This is one of the peculiarities of the way systems like API.AI work today. When entities match, they only need to be *close enough* to already declared entities. At the same time, the value used in the agent’s response is $womens-winner.original, meaning it will simply output what it matched. This is yet another reason that I believe the examples are a little impractical – although they illustrate the point. (Note: You can improve this response and omit the question mark by writing some code on the backend. That is an advanced topic at this point).

Step by step guide to DialogFlow (API.AI) Part 9 – how Intent Mapping works under the hood

Table of Contents for Step by Step Guide

In the previous post you saw how the question “who won men’s Wimbledon in 1982” was correctly answered by the agent. At this point, it would be a good idea to take a little break from building the agent to go into a little detail about what is happening under the hood. It is my view that this knowledge will help you in three different ways.

1 Bot behaves unexpectedly

When your bot doesn’t behave as expected (which will happen sooner or later), knowing some of the underlying concepts can help you better understand what is going on

2 Decide how much custom code you need

As a result of 1, you can often realize the limitations of a bot building platform such as API.AI and decide how much of the code you wish to move outside of the agent and into your custom code

3 Reverse engineering the agent to improve performance

This knowledge will also help you tweak some settings which are at your disposal. Machine Learning, which usually powers these chatbot systems today, is something of a black box. But you can still reverse engineer a few things to improve the results.

The AI in an API.AI chatbot has two tasks:

a) map the phrase to an intent which has already been created, and

b) extract any relevant entities.

When the user types a sentence, it looks at the sentence structure and tries to determine if the sentence structure is a close match to a programmer defined phrase. If one of the programmer-defined phrases in the Intents (let us call them predefined phrases) contains an entity placeholder (the one following the @ symbol), and what the user says also contains an entity value, that is a clue for API.AI (and other bot frameworks) that the user has said something matching one of the predefined phrases.

In the same way, if the remainder of the sentence (excluding the entity) is similar to a predefined phrase, that is ALSO a clue for a match.

As you might imagine, when these two happen in tandem (structural similarity and matching entity value), then API.AI can predict with much greater confidence that the user’s message matched one of the intents. The exact steps for determining this is the secret sauce for any bot framework.

When the user says something completely unrelated, your bot should also figure that out and should trigger the Fallback intent (introduced in Part 4).

The chatbot does not have any other kind of intelligence. For example, it cannot look at the history of conversation it had with a user and make inferences. Look at a discussion I had on the API.AI forum for an example.

Step by step guide to DialogFlow (API.AI) Part 10 – The Vending Machine Bot

Table of Contents for Step by Step Guide

So we have seen two fundamental concepts of a chatbot framework – namely Intents and Entities.

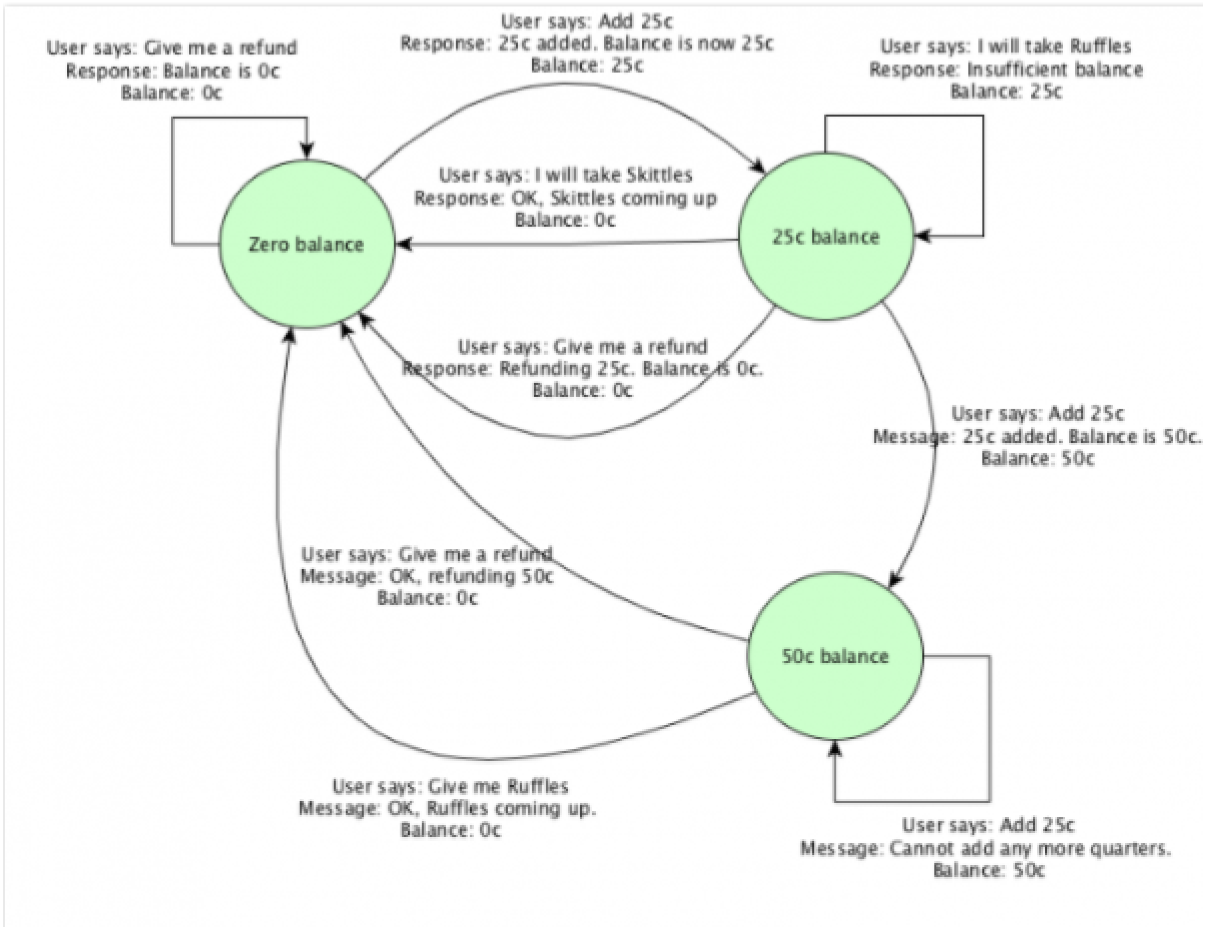
A third important concept is the idea of context. To lay the groundwork for explaining contexts, I will start by describing the Vending Machine bot (VMBot for short).

Think of the vending machine bot as a spoken language interface for the vending machine. This is a limited vending machine, in that it sells exactly two products – Ruffles and Skittles. And you can only insert quarters into the machine. The Skittles costs 25c and the Ruffles costs 50c.

As you use the vending machine bot, you can say one of the following:

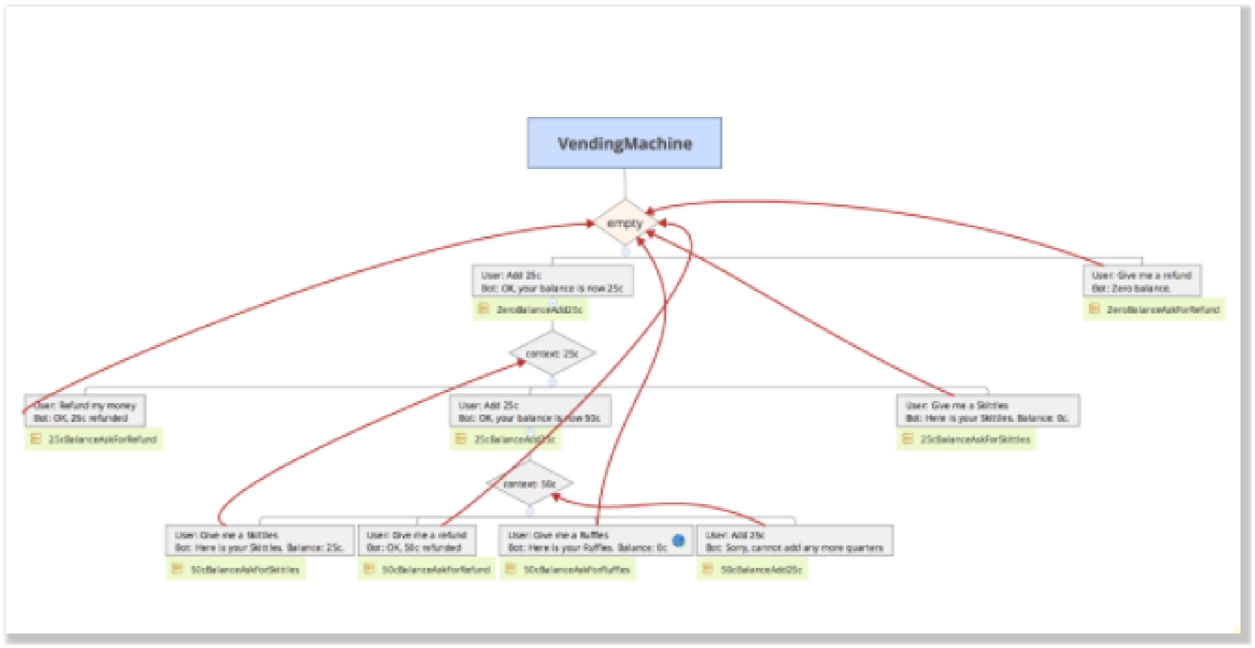
1. Add a quarter
2. Ask for a refund
3. Buy a Skittles
4. Buy a Ruffles

The state diagram of the vending machine would look like this:


User says: Give me a refund
Response: Balance is 0c
Balance: 0c
User says: Add 25c
Response: 25c added. Balance is now 25c
Balance: 25c
User says: I will take Ruffles
Response: Insufficient balance
Balance: 25c
User says: I will take Skittles
Response: OK, Skittles coming up
Balance: 0c
Zero balance
25c balance
User says: Give me a refund
Response: Refunding 25c. Balance is 0c.
Balance: 0c
User says: Add 25c
Message: 25c added. Balance is 50c.
Balance: 50c
User says: Give me a refund
Message: OK, refunding 50c
Balance: 0c
50c balance
User says: Give me Ruffles
Message: OK, Ruffles coming up.
Balance: 0c
User says: Add 25c
Message: Cannot add any more quarters.
Balance: 50c

Here is a flowchart I created using XMind which gives you a representation which is closer to the actual conversation which takes place. You can get the actual mindmap (the .xmind file) from the MBD Bonus Pack if you are interested. In the MBD Bonus Pack, you can see this under the [MM-VendingMachine] folder.

How to read this chart?



Step by step guide to DialogFlow (API.AI) Part 11 – Using contexts to maintain state

Table of Contents for Step by Step Guide

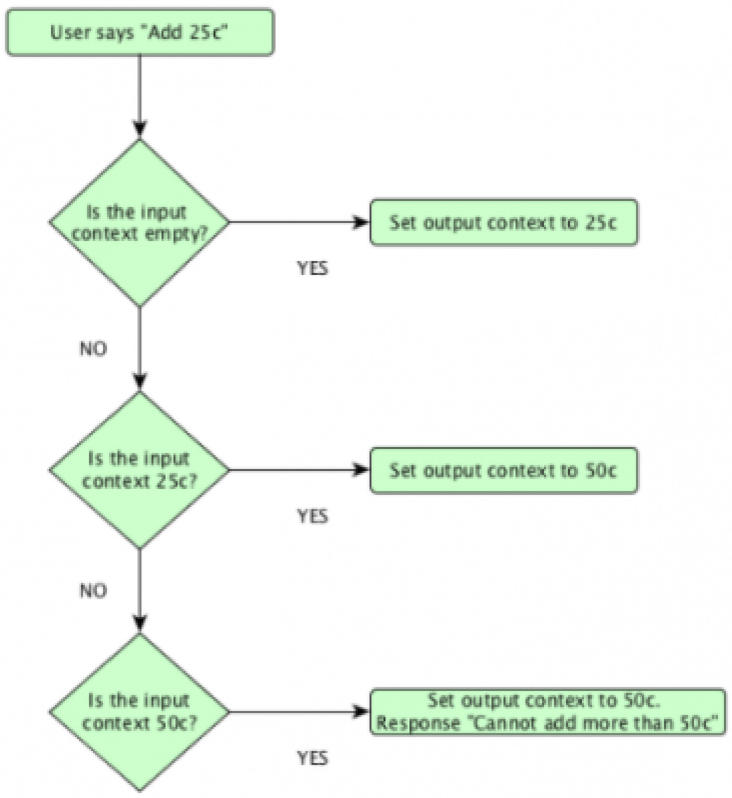
In the last post we saw how the vending machine bot works. The main difference between the previous two chatbots (FAQ Bot and WimbledonFinalBot) and the current one is that we need to keep track of the user's previous messages so that we can use that to decide the current action.

Suppose the user says "Add 25c". Is the current balance 25c or 50c? It depends! If the current balance is already 25c, then the final balance becomes 50c. If the current balance is 0c, then the final balance becomes 25c. The way we know what the current balance is, is by using contexts. In other words, we use contexts to **maintain state**.

At the start of the conversation, if the user says"Add 25c", we will be setting the "output context" of the intent to a string called 25c (since current balance is 0c). Now the output context is used as the "input context" for the next intent. That is, if the input context is already 25c, and the user says "Add 25c" we will be setting the output context of the intent to a string called 50c.

Using this combination of input and output contexts, you can see that it now becomes possible to maintain state even for more complex scenarios.

Here is a flowchart which shows how we set the input and output contexts for the Intents.


User says "Add 25c"
Is the input context empty?
YES
Set output context to 25c
NO
Is the input context 25c?
YES
Set output context to 50c
NO
Is the input context 50c?
YES
Set output context to 50c. Response "Cannot add more than 50c"

The nice thing about API.AI is that the Intents are created in such a way that we can set input and output contexts on our Intents to achieve this result. We will see how in the next part.

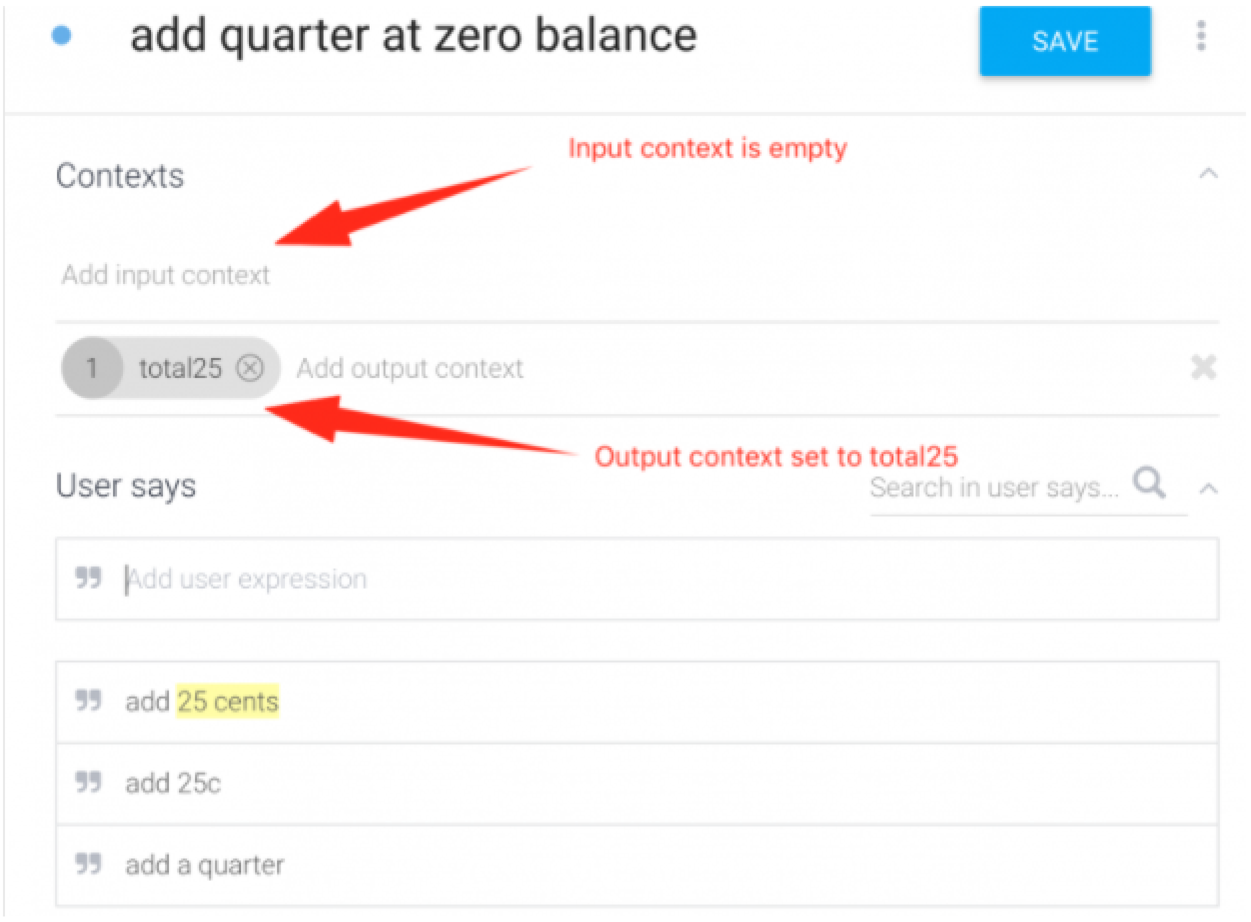
Step by step guide to DialogFlow (API.AI) Part 12 – Using input and output contexts to manage state

Table of Contents for Step by Step Guide

We saw in the previous part that we would like to maintain the state of the current balance in the vending machine so that we can decide what the final balance would be when the user adds another 25c.

You need to remember that the Intent for both zero balance and 25c balance will be defined using more or less the same set of User says phrases (given that in both cases the user will be saying more or less the same phrase). The way API.AI will decide which Intent to trigger will be based on what the Input context is, as we saw in the flowchart of the previous part.

Here is the Intent for adding a quarter at zero balance:


add quarter at zero balance

add 25 cents

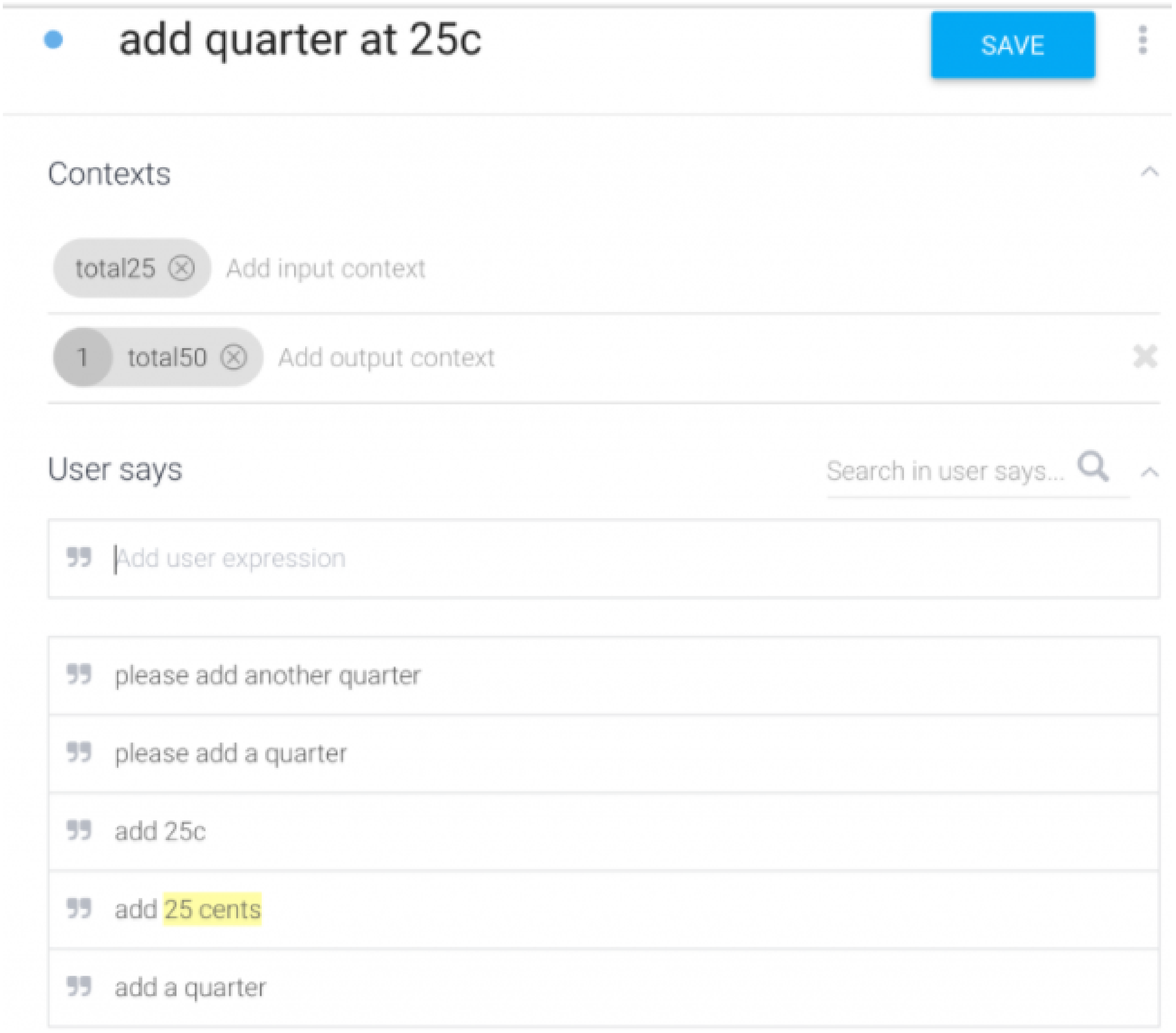
add 25c

add a quarter

There are three things to note here:

1. The input context is empty (indicating start of conversation where balance is zero)
2. The output context is set to total25 (a string value representing the state)
3. There is a 1 on the left of the total25. That is the context lifespan, whose default value is 5. I recommend setting it to 1 and tracking the value carefully (I wrote about this before in my article about context lifespans)

The Intent definition for adding a quarter at 25c balance will have similar phrases, except for the input and output contexts.


add quarter at 25c

SAVE

Contexts

total25 ⊗ Add input context

1 total50 ⊗ Add output context

User says

Search in user says...

Add user expression

please add another quarter

please add a quarter

add 25c

add 25 cents

add a quarter

By using the same idea, you should be able to fill out the remaining Intents for the vending machine chatbot. You can interact with the vending machine chatbot I built at this link.

As I interacted with the chatbot, I noticed that sometimes the response is wrong because the Intent mapping is incorrect. If you face a similar issue, you can provide more User says examples closely matching the request and that should usually fix the problem.

Step by step guide to DialogFlow (API.AI) part 13 – Webhooks

Table of Contents for Step by Step Guide

While the previous example bots were all self-contained (in the sense that all code was executed on the API.AI's service), you will make use of webhooks when you wish to write custom logic which is not supported by API.AI.

My view is that it is unlikely you can build a very functional bot without using webhooks. The best example for learning about webhooks is the API.AI's weather agent

Update July 24th 2017:

API.AI recently updated all their webhook documentation to use the Google Cloud. While the comparison between Google Cloud and Heroku (the previous cloud service which was used for webhook demos) is an article by itself, for now I will just update the above statement. The easiest way (and thus probably the best way) to get started with webhooks right now is to use Heroku, and you can refer to my tutorial on this topic instead of the documentation above.

Extending the Vending Machine bot

Here is a simple thought exercise. Pretend that the Vending Machine bot we saw previously had the facility to add either a quarter, or a nickel, or a dime. Now imagine the number of states we would have had to handle (basically a state for every multiple of 5, upto 50). Instead of having that combinatorial explosion of states, we could simply write some custom logic which would compute the total amount added and have far fewer states.

After implementing the weather agent example, you can try creating some custom code behind a webhook to build out this more full-featured vending machine bot.

In the next part, we will look at another interesting example bot to connect the dots on what we have learnt till now.

Step by step guide to DialogFlow (API.AI) part 14 – The florist bot

Table of Contents for Step by Step Guide

One of the examples already provided in the API.AI documentation is an example bot for a florist shop. I really like this example as a teaching tool because it covers so many different features that you will use as you are building out your chatbots. But the biggest reason why I like this example is that someone made the effort to post the agent's ZIP file online so you can download it, create your own agent with it and play around to get a better picture of the different concepts.

Flowchart

In the excellent book Bot Business 101, Ekim Kaya mentions the following:

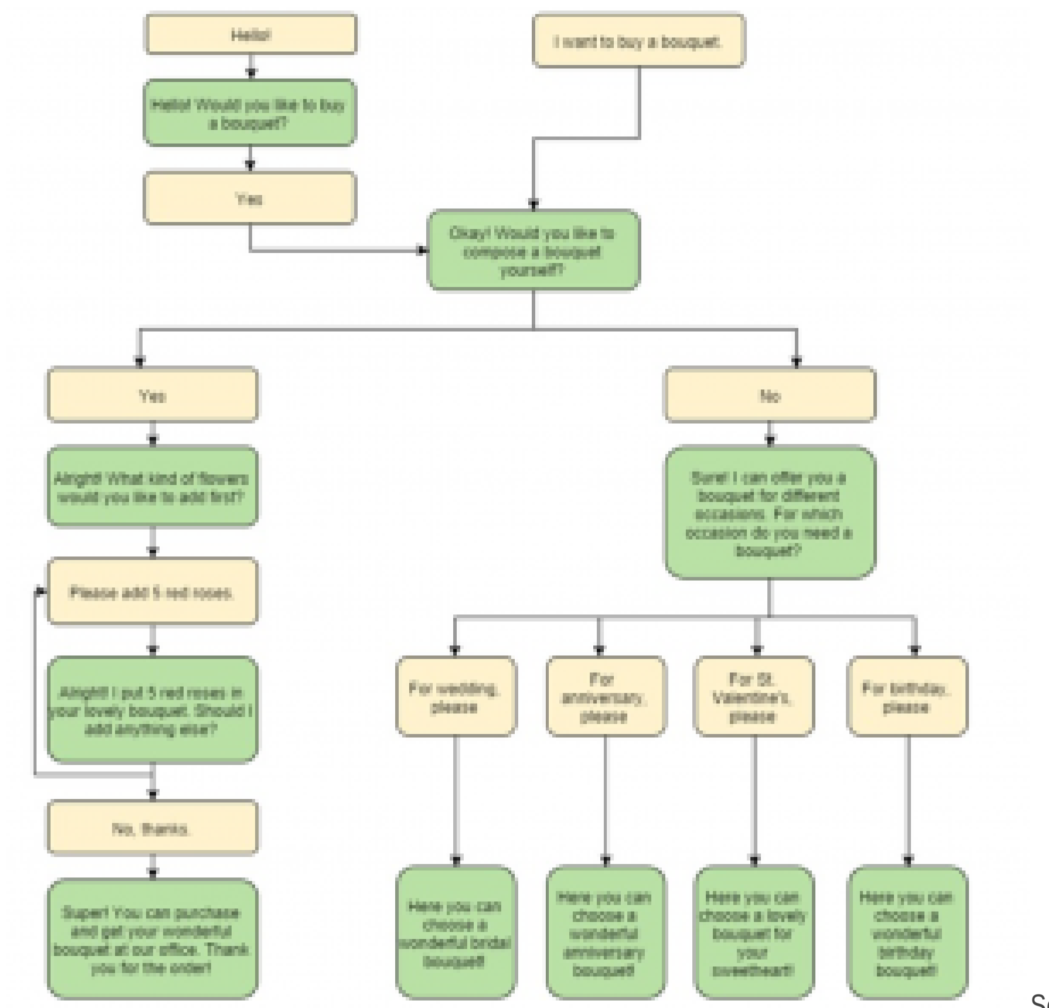
“My biggest recommendation is to have a very, very detailed conversational flow as to how the scenario will be executed….Intent management is not easy and if it does not work on paper, it will not work on the computer either.”

Ekim has been in the bot business for a long time, and you can see from the book’s reviews that he knows what he is talking about.

So designing the bot’s conversation flow in a flowchart is a good idea.

Florist shop flowchart

The florist shop example from API.AI has a detailed conversational flowchart:


Hello!
Hello! Would you like to buy a bouquet?
Yes
I want to buy a bouquet.
Okay! Would you like to compose a bouquet yourself?
Yes
Alright! What kind of flowers would you like to add first?
Please add 5 red roses.
Alright! I put 5 red roses in your lovely bouquet. Should I add anything else?
No, thanks.
Super! You can purchase and get your wonderful bouquet at our office. Thank you for the order!
No
Sure! I can offer you a bouquet for different occasions. For which occasion do you need a bouquet?
For wedding, please
For anniversary, please
For St. Valentine's, please
For birthday, please
Here you can choose a wonderful bridal bouquet!
Here you can choose a wonderful anniversary bouquet!
Here you can choose a lovely bouquet for your sweetheart!
Here you can choose a wonderful birthday bouquet!

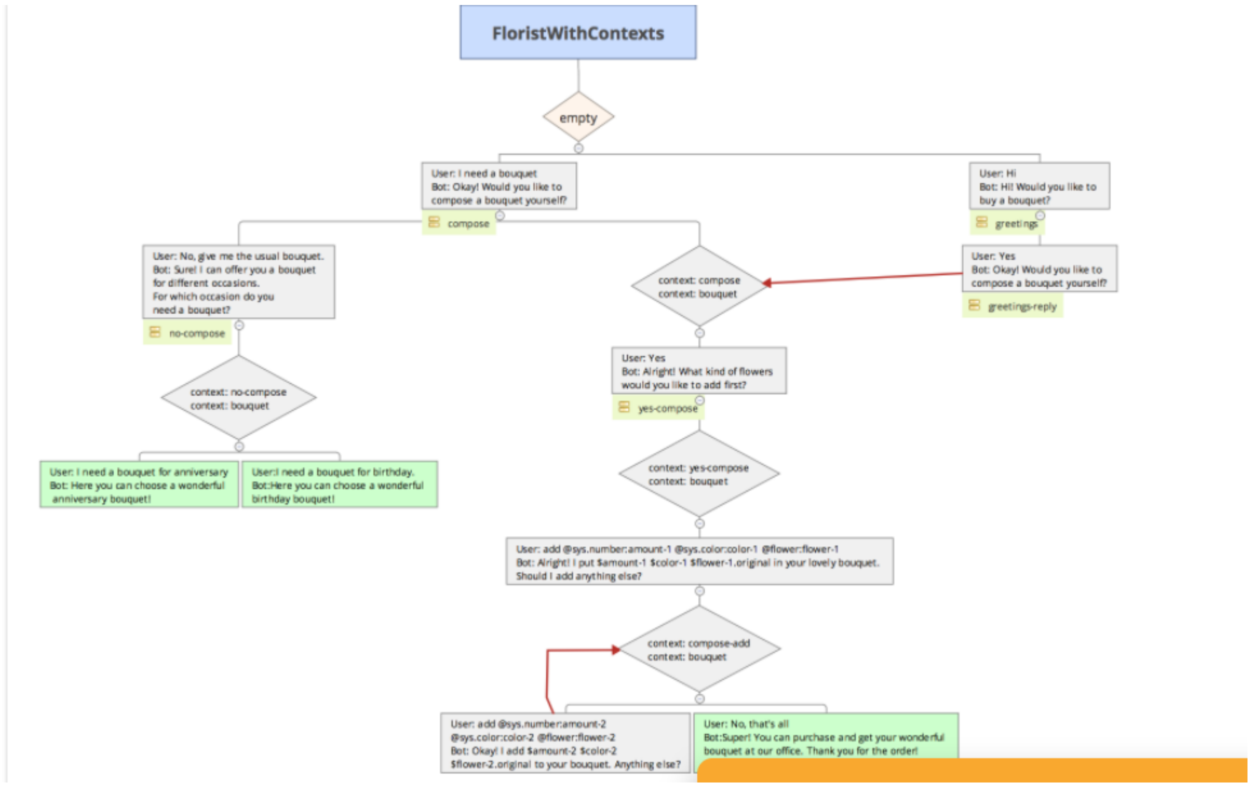
Source: https://api.ai/blog/2015/11/23/Contexts/

Florist shop conversation flowchart using XMind

I have re-created a flowchart for the florist chatbot using XMind. There are some advantages of using XMind for creating the conversation flowcharts for your chatbot. For example by adding input and output contexts into the flowchart itself (but distinguishing them via a different shape), you can add a ton of detail to the conversation flowchart without making it too difficult to follow. I will write about conversation flowcharts in detail in a future article.

If you would like to get a copy of the mindmap file (.xmind format) – you can get it from my MBD Resources course (chapter 11). I have left out a couple of branches for clarity.

How to read this chart?


FloristWithContexts
empty
User: I need a bouquet
Bot: Okay! Would you like to compose a bouquet yourself?
compose
User: Hi
Bot: Hi! Would you like to buy a bouquet?
greetings
User: No, give me the usual bouquet.
Bot: Sure! I can offer you a bouquet for different occasions.
For which occasion do you need a bouquet?
no-compose
context: compose
context: bouquet
User: Yes
Bot: Okay! Would you like to compose a bouquet yourself?
greetings-reply
User: Yes
Bot: Alright! What kind of flowers would you like to add first?
yes-compose
context: no-compose
context: bouquet
User: I need a bouquet for anniversary
Bot: Here you can choose a wonderful anniversary bouquet!
User:I need a bouquet for birthday.
Bot:Here you can choose a wonderful birthday bouquet!
context: yes-compose
context: bouquet
User: add @sys.number:amount-1 @sys.color:color-1 @flower:flower-1
Bot: Alright! I put $amount-1 $color-1 $flower-1.original in your lovely bouquet.
Should I add anything else?
context: compose-add
context: bouquet
User: add @sys.number:amount-2
@sys.color:color-2 @flower:flower-2
Bot: Okay! I add $amount-2 $color-2
$flower-2.original to your bouquet. Anything else?
User: No, that's all
Bot:Super! You can purchase and get your wonderful bouquet at our office. Thank you for the order!

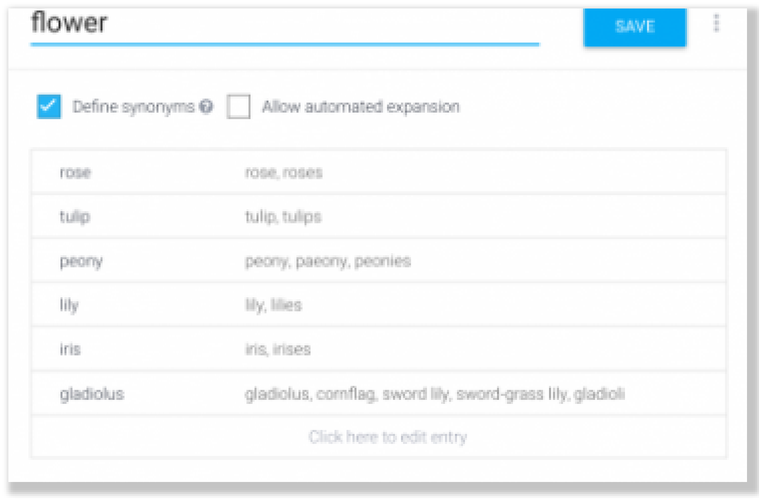
Step by step guide to DialogFlow (API.AI) Part 15 – Identifying intents in the florist bot

Table of Contents for Step by Step Guide

In the previous post, we saw the flowchart for the florist bot.

Identifying entities

The process for identifying entities, in my view, is to start looking for the nouns in your flowchart. In the case of the florist bot, we see the following: red roses, and nouns for events (wedding, anniversary etc.). For this particular case, only the red roses (in general, flowers) are defined as entities since the event-based nouns are only used for static responses in the flowchart.



flower

✓ Define synonyms ☐ Allow automated expansion

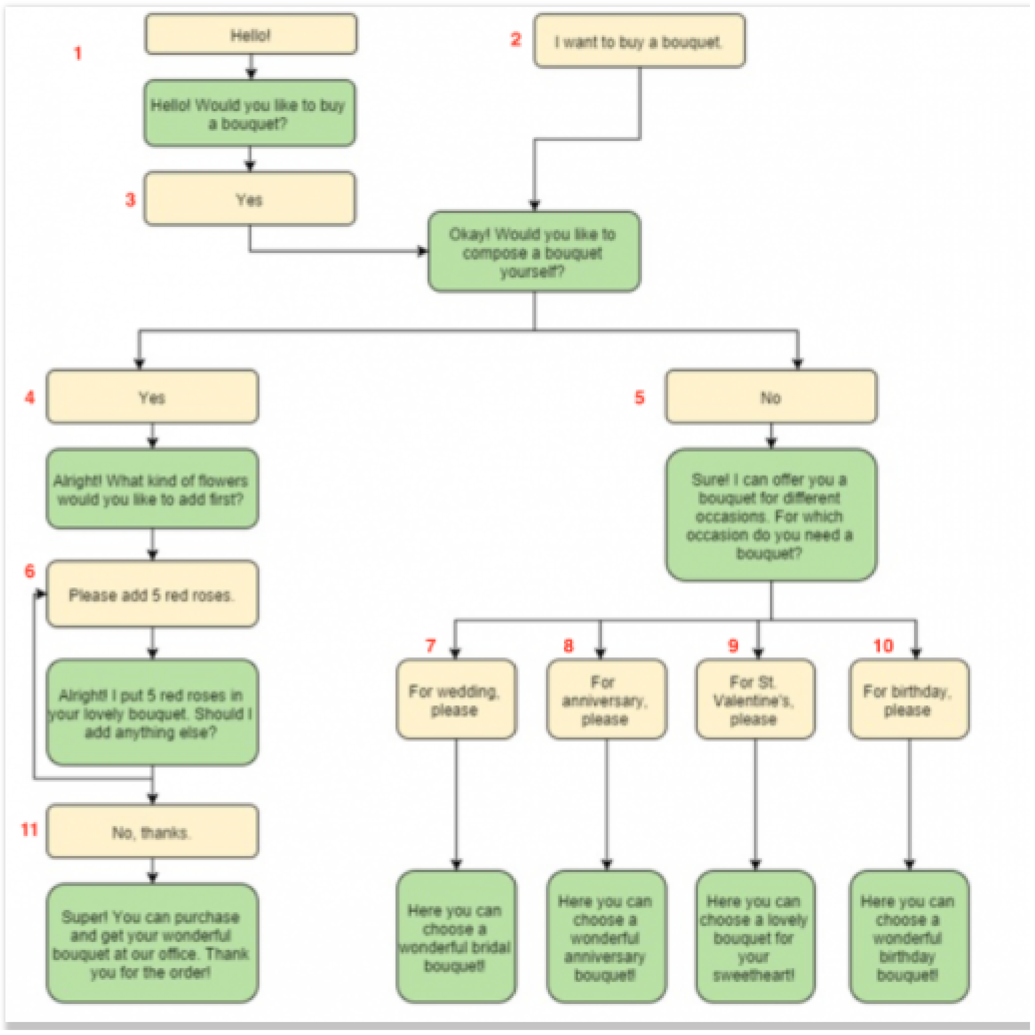
rose	rose, roses
tulip	tulip, tulips
peony	peony, paeony, peonies
lily	lily, lilies
iris	iris, irises
gladiolus	gladiolus, cornflag, sword lily, sword-grass lily, gladioli

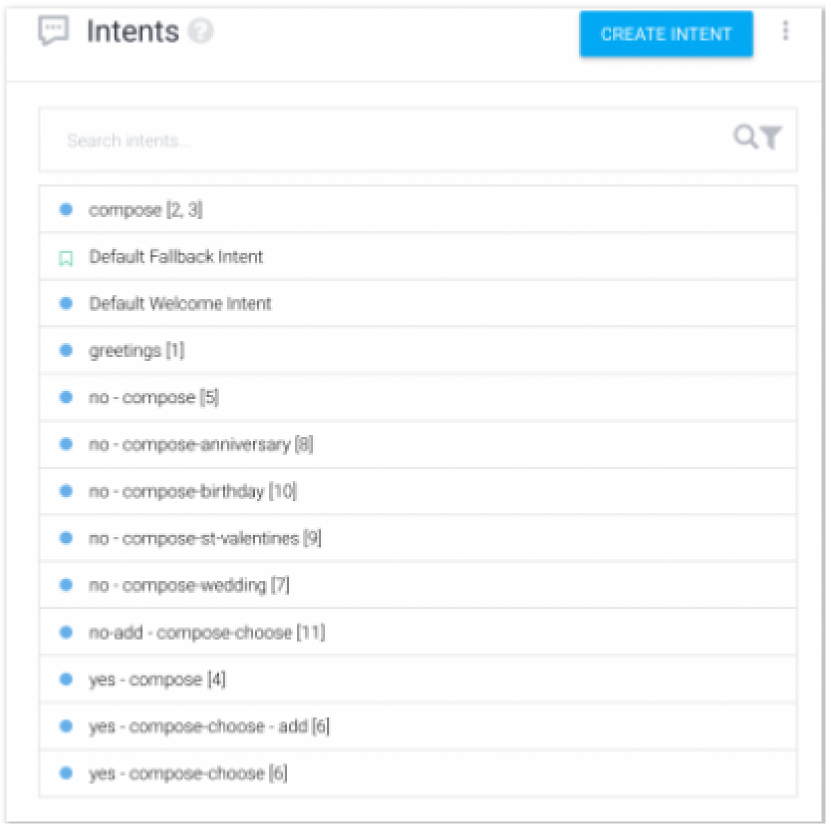
Click here to edit entry

Now let us look at the different intents we need to define.

Identifying Intents

In the flowchart, all the yellow boxes where the user says something is an intent. There are 11 such yellow boxes.


1
Hello!
Hello! Would you like to buy a bouquet?
3
Yes
2
I want to buy a bouquet.
Okay! Would you like to compose a bouquet yourself?
4
Yes
Alright! What kind of flowers would you like to add first?
6
Please add 5 red roses.
Alright! I put 5 red roses in your lovely bouquet. Should I add anything else?
11
No, thanks.
Super! You can purchase and get your wonderful bouquet at our office. Thank you for the order!
5
No
Sure! I can offer you a bouquet for different occasions. For which occasion do you need a bouquet?
7
For wedding, please
8
For anniversary, please
9
For St. Valentine's, please
10
For birthday, please
Here you can choose a wonderful bridal bouquet!
Here you can choose a wonderful anniversary bouquet!
Here you can choose a lovely bouquet for your sweetheart!
Here you can choose a wonderful birthday bouquet!


Intents

CREATE INTENT

Search intents...

- compose [2, 3]
- Default Fallback Intent
- Default Welcome Intent
- greetings [1]
- no - compose [5]
- no - compose-anniversary [8]
- no - compose-birthday [10]
- no - compose-st-valentines [9]
- no - compose-wedding [7]
- no-add - compose-choose [11]
- yes - compose [4]
- yes - compose-choose - add [6]
- yes - compose-choose [6]

I have mapped out the Yellow boxes in the flowchart to the list of intents in the florist zip file.

You should note a couple of things:

1. There are two numbers mapped to the top Intent. This is because you can combine multiple possible phrases into a single intent. (User says "Yes" or User says "I want to buy a bouquet" which both have the same intention)
2. A single number can also become two intents. E.g. Box number 6 is defined using two different intents. Why is this the case? It is because this is the place in the flowchart where you have a loop. And a loop is usually constructed with an intent for the case where the loop has not yet been entered, and another one for the case where the loop has already been entered. If you remember our discussion about contexts, you might remember that they are usually used to distinguish states in these scenarios.

There are also other things you can analyze which will help you decide on the intents.

Does your flowchart have a merge? For example, yellow boxes 2 and 3 merge into a single green Response. This is the reason why they have been declared as a single intent. If you don't yet have any contexts set, you can combine the two merging yellow boxes into a single intent.

Does your flowchart have splits? For example, the previous green box for 4 and 5 is splitting based on the two possible responses (Yes or No). In those cases, you will usually have 1 intent per splitting branch.

Does your flowchart have loops? For example, there is a loop after box 6. You might notice that it is also like a split – except that one of the splitting branches loops back. As a result, you have two intents to represent the loop (the last 2 intents in the list).

In the next post, we will talk about contexts in the “context” of the florist bot.

Step by step guide to DialogFlow (API.AI) Part 16 – How contexts help intent mapping

Table of Contents for Step by Step Guide

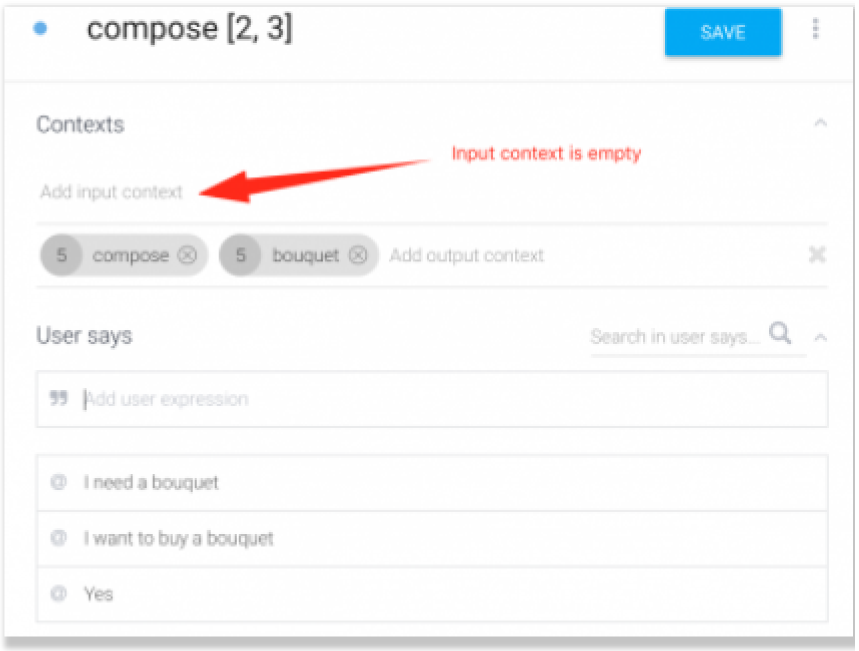
In the previous post, we looked at how the flowchart of the florist bot can be used to understand the entities and intents you will use to create your API.AI agent.

In this post, we will take a closer look at the contexts which have already been defined in the agent and how they influence the agent’s behavior.

Distinguish between two different intents where the user phrase is the same

In the florist bot flowchart, you will notice that boxes 3 and 4 have the same user phrase (“yes”). So if the user says “Yes” in the middle of the conversation, how will the bot know what is the next step?

Now, if we take a look at the intents corresponding to these boxes, we will get a better idea.


compose [2, 3]

SAVE

Contexts

Input context is empty

Add input context

5 compose 5 bouquet Add output context

User says

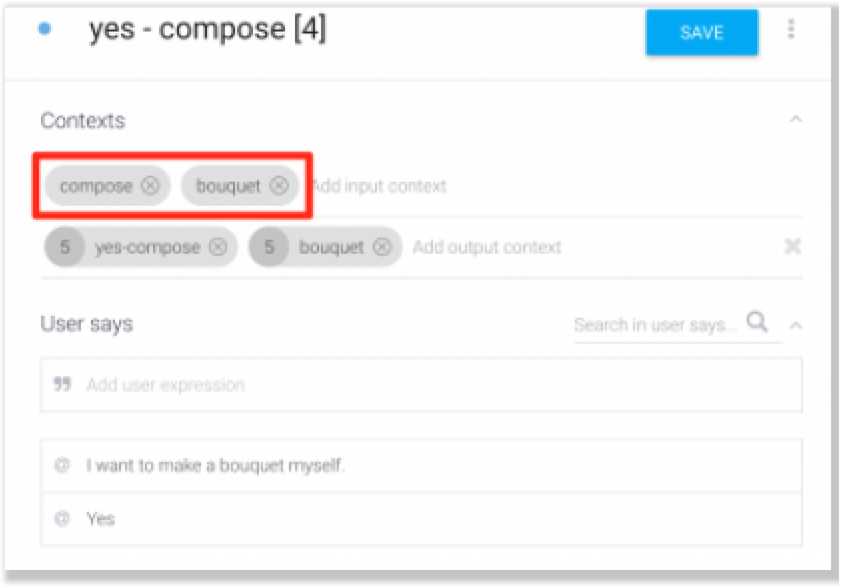
Search in user says...

Add user expression

I need a bouquet

I want to buy a bouquet

Yes


yes - compose [4]

SAVE

Contexts

compose bouquet Add input context

5 yes-compose 5 bouquet Add output context

User says

Search in user says...

Add user expression

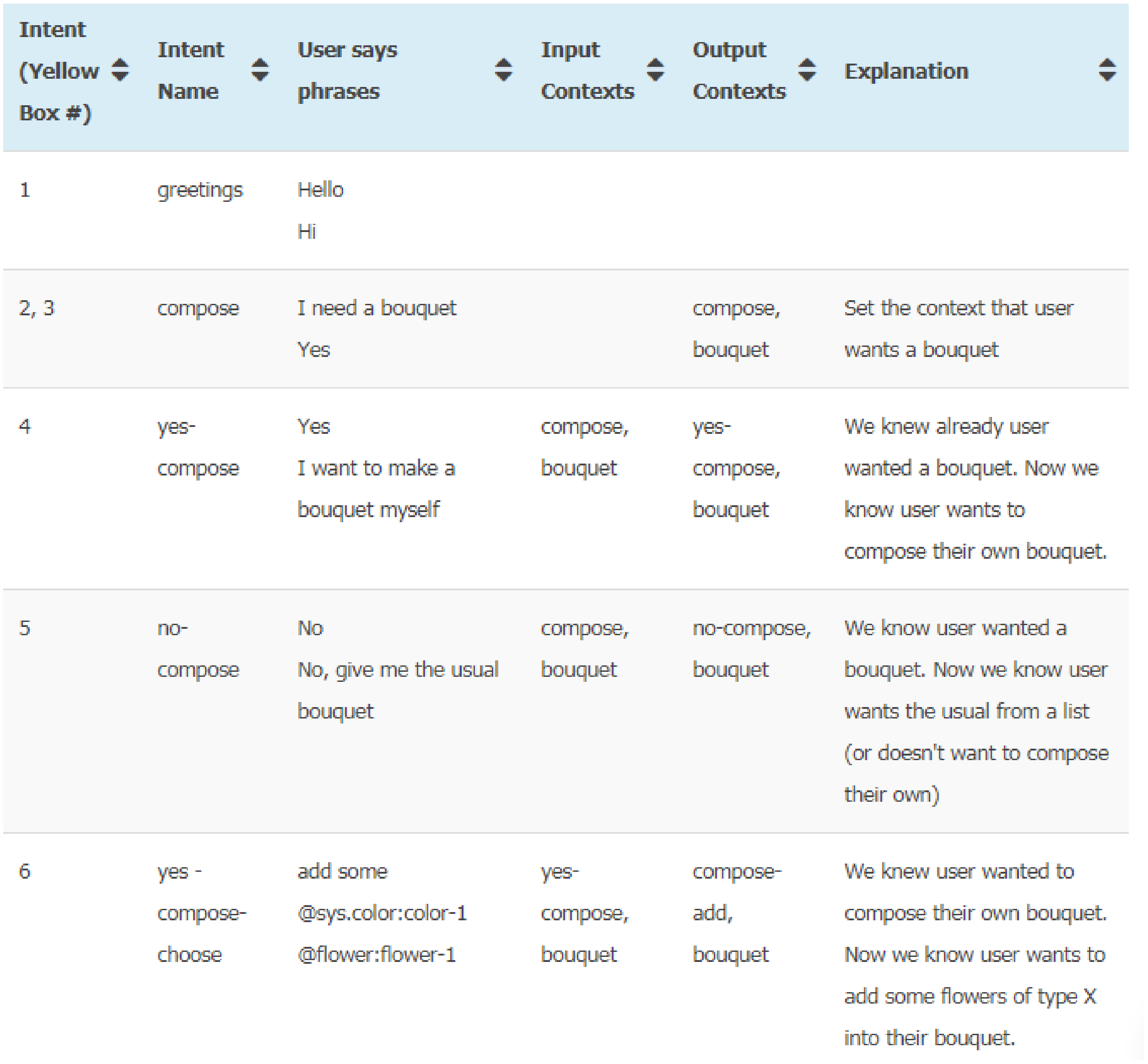
I want to make a bouquet myself.

Yes

For intent corresponding to box 3, there is **no input context**.

For intent corresponding to box 4, **compose and bouquet** have already been defined as the input context.

Now let us look at the different contexts used in the entire list of entities defined in the agent ZIP file.



Intent (Yellow Box #)	Intent Name	User says phrases	Input Contexts	Output Contexts	Explanation
1	greetings	Hello Hi			
2, 3	compose	I need a bouquet Yes		compose, bouquet	Set the context that user wants a bouquet
4	yes-compose	Yes I want to make a bouquet myself	compose, bouquet	yes-compose, bouquet	We knew already user wanted a bouquet. Now we know user wants to compose their own bouquet.
5	no-compose	No No, give me the usual bouquet	compose, bouquet	no-compose, bouquet	We know user wanted a bouquet. Now we know user wants the usual from a list (or doesn't want to compose their own)
6	yes - compose-choose	add some @sys.color:color-1 @flower:flower-1	yes-compose, bouquet	compose-add, bouquet	We knew user wanted to compose their own bouquet. Now we know user wants to add some flowers of type X into their bouquet.

6	yes-compose-choose-add	@ add @sys.number:amount-2 @sys.color:color-2 @flower:flower-2	compose-add, bouquet	compose-add, bouquet	We knew user already has some flowers in their bouquet. Now we know users wants to add more flowers. Note: The input and output contexts are the same. Once user asks to add a certain number of flowers, we will reset the context to be the same. Why do we do this? Because we want to stay in the loop until user is done adding all the flowers they want.
7	no-compose-wedding	For wedding, please I need a bouquet for wedding	no-compose, bouquet		We knew user wanted to get a bouquet from a list. Now we know that user wants a bouquet for a wedding. We clear all the contexts as we are done with the selection and ready to exit the process.

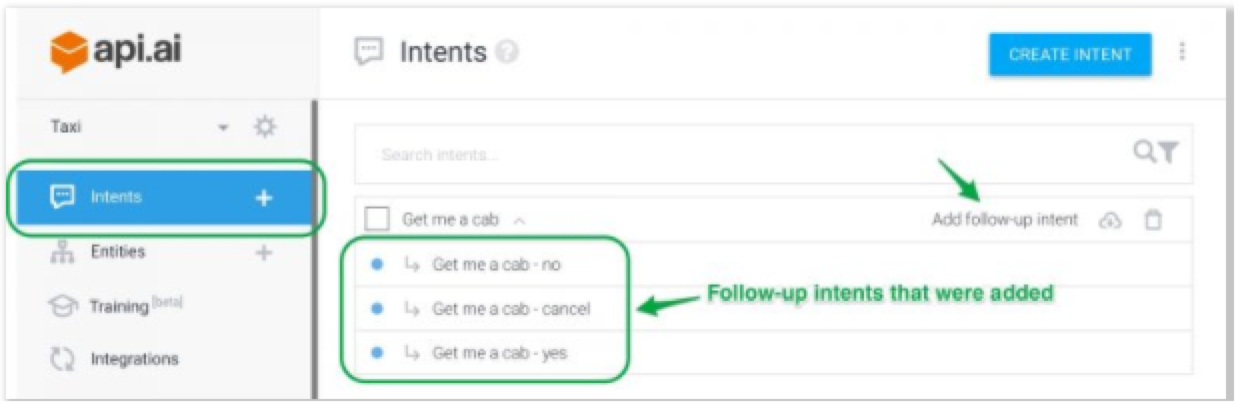
8	no-compose-anniversary	For anniversary I need a bouquet for anniversary	no-compose, bouquet		We knew user wanted to get a bouquet from a list. Now we know that user wants a bouquet for an anniversary. We clear all the contexts as we are done with the selection and ready to exit the process.
9	no - compose-st-valentines	I need a bouquet for Saint Valentine's Day For Saint Valentine's Day	no-compose, bouquet		We knew user wanted to get a bouquet from a list. Now we know that user wants a bouquet for St Valentine's Day. We clear all the contexts as we are done with the selection and ready to exit the process.
10	no - compose-birthday	For birthday, please. I need a bouquet for birthday.	no-compose, bouquet		We knew user wanted to get a bouquet from a list. Now we know that user wants a bouquet for a birthday. We clear all the contexts as we are done with the selection and ready to exit the process.

Intent (Yellow Box #)	**Intent Name**	**User says phrases**	**Input Contexts**	**Output Contexts**	**Explanation**
11	no-add - compose-choose	No, that's all No thanks	bouquet, compose-add		We know user wanted to compose their own bouquet. Now we know user is done and wants to complete the checkout. (Note: it is possible user hasn't added any flowers at all)

Step by step guide to DialogFlow (API.AI) Part 17 – Follow up intents

Table of Contents for Step by Step Guide

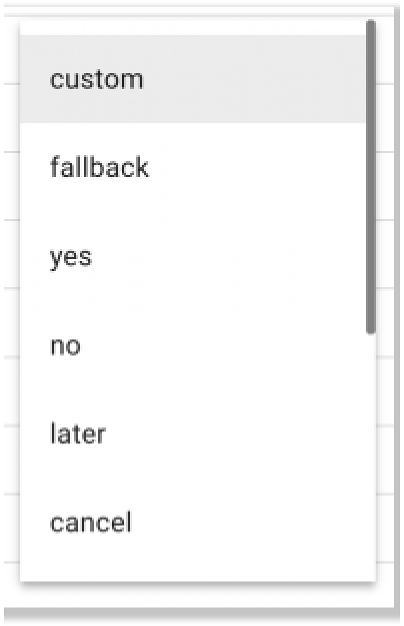
Recently, API.AI announced a new feature called Follow Up Intents. You can add them from the Intents view and they allow you to create a hierarchy of intents.

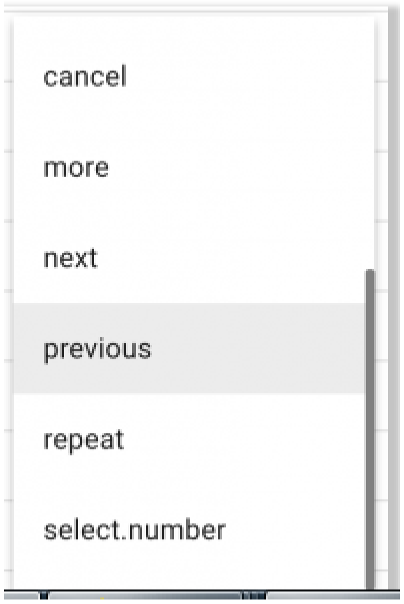


What are the advantages? And when do you need them? Let us take a look.

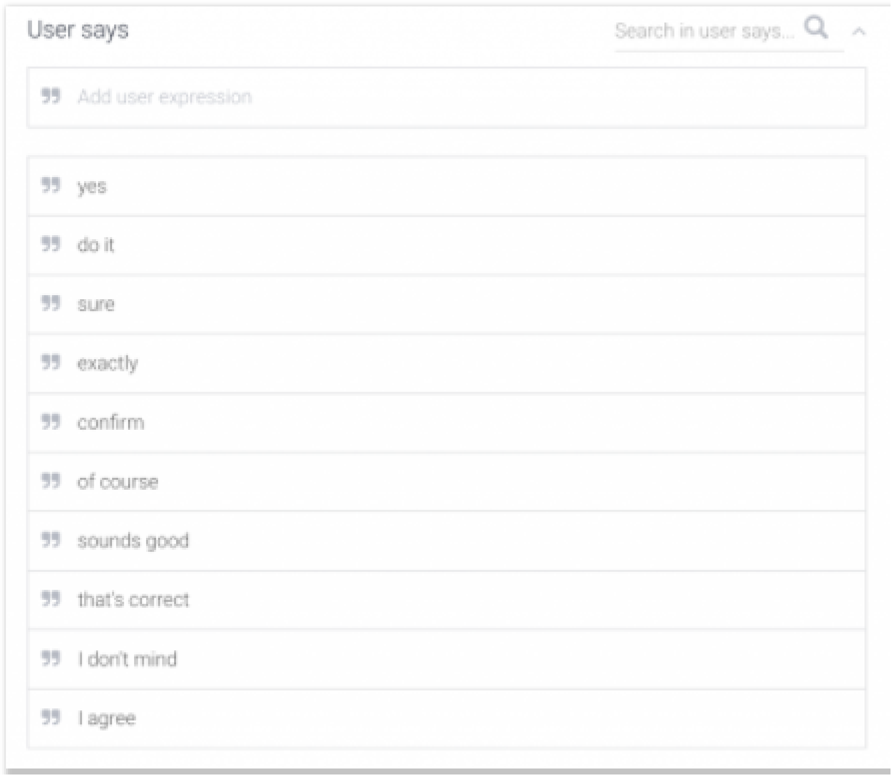
What are follow up intents?

Follow up intents, as their name suggests, are intents you define to be a child intent of an existing intent. Creating a follow up intent is quite straight-forward. As you hover your mouse on any intent in the Intents view, you will see a link which says “Add follow-up intent”. Clicking on the link gives you the following options:





If you choose Custom fallback intent, you will be allowed to enter your own “user says” phrases. If you choose any of the other choices, you will be provided with a prepopulated list of appropriate “user says” phrases which you can then edit to your liking. For example, this is what the prepopulated list for “Yes” choice looks like:



yes
do it
sure
exactly
confirm
of course
sounds good
that's correct
I don't mind
I agree

Can we improve our previous bots with follow up intents?

FAQ Bot

The FAQ bot is too simple to need a follow up intent, given that it acts more or less like a search engine with questions followed by answers, without any need for context.

Wimbledon Finals Bot

Similar to the FAQ bot, the Wimbledon Finals bot also functions like a question and answer bot (although it includes and introduces the concept of entities). Here too, the bot is too simple to need follow up intents.

Vending Machine Bot

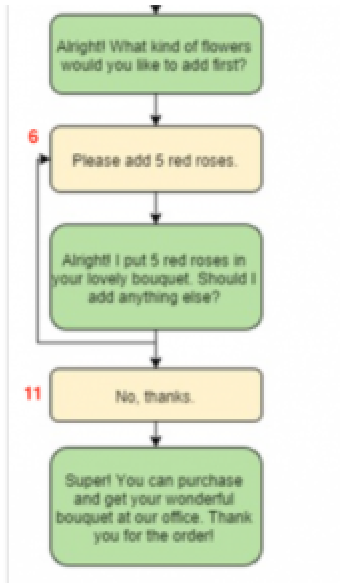
The simple Vending Machine Bot we created to understand contexts is definitely a use case for follow up intents. However, this is also because we had a somewhat oversimplified vending machine. Still, you can rewrite the vending machine bot to use follow up intents and simplify the chatbot quite a bit. In a future post of this series, I will discuss how to rewrite the vending machine bot using follow up intents.

The weather bot (Webhooks)

The weather bot introduced the concept of webhooks, and while there is a fair amount of code which goes into its business logic, the actual user facing portion is still a simple question and answer interface and doesn't need follow up intents.

The florist bot

Here, it gets a little more interesting. On first glance, when you look at the flowchart for the florist bot, you might think follow up intents are very useful for the florist bot and should simplify the overall process a lot. There is a small catch however.


Alright! What kind of flowers would you like to add first?
6
Please add 5 red roses.
Alright! I put 5 red roses in your lovely bouquet. Should I add anything else?
11
No, thanks.
Super! You can purchase and get your wonderful bouquet at our office. Thank you for the order!

The problem is with the box marked number 6. It is actually a while loop, and using follow up intents, you cannot create a while loop where you don't know how many times you will be entering the loop.

In the next post, we will consider a use case which is almost perfect for the follow up intents.

Step by step guide to DialogFlow (API.AI) Part 18 – Dichotomous key bot

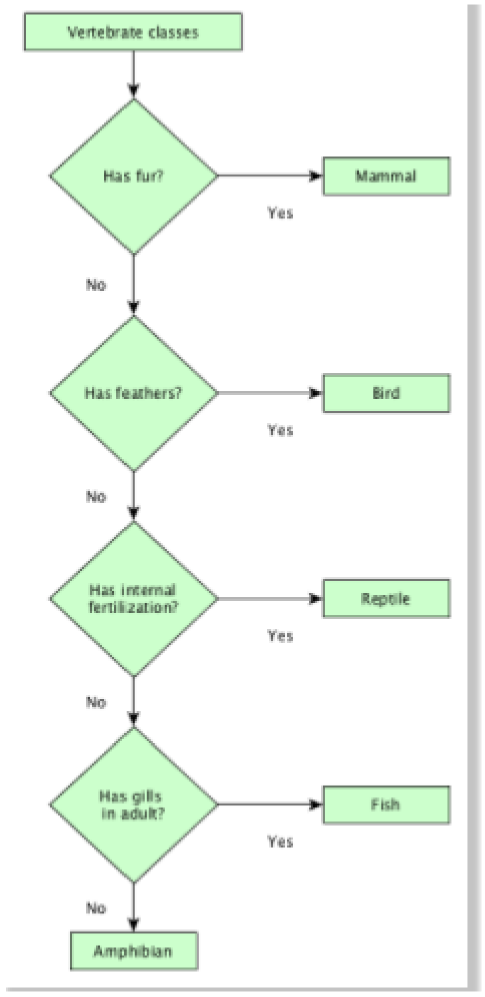
Table of Contents for Step by Step Guide

One of the best use cases which illustrates the use of follow up intents is the dichotomous key bot. Here is a definition of the Dichotomous Key:

A dichotomous key is a tool that allows the user to determine the identity of items in the natural world, such as trees, wildflowers, mammals, reptiles, rocks, and fish. Keys consist of a series of choices that lead the user to the correct name of a given item. "Dichotomous" means "divided into two parts". Therefore, dichotomous keys always give two choices in each step.

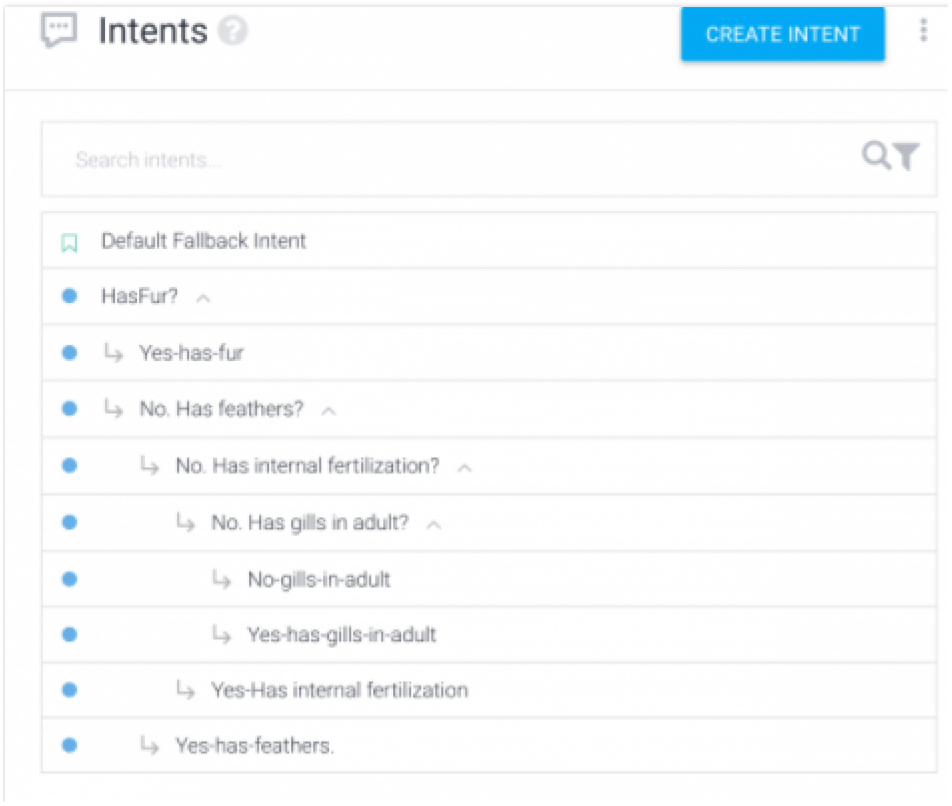
Dichotomous Key Bot

We will create a dichotomous key bot for identifying vertebrates. This is what a flowchart for the dichotomous key looks like (for a much better looking flowchart, refer to the source) :



Defining the Intents

Take a look at the Intents defined for this dichotomous key chatbot.



Default Fallback Intent
HasFur?
↳ Yes-has-fur
↳ No. Has feathers?
↳ No. Has internal fertilization?
↳ No. Has gills in adult?
↳ No-gills-in-adult
↳ Yes-has-gills-in-adult
↳ Yes-Has internal fertilization
↳ Yes-has-feathers.

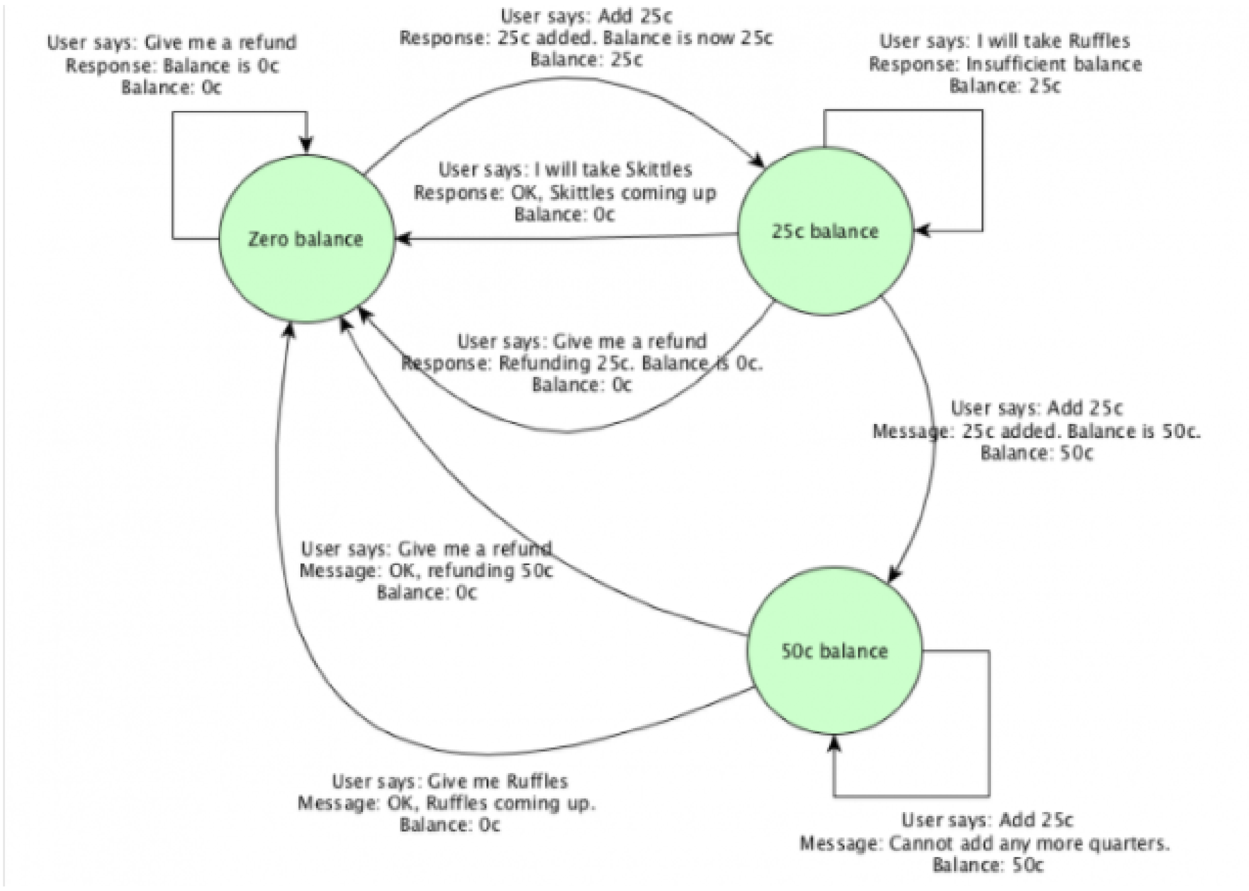
In other words, you can simply follow the tree like hierarchy and create corresponding intents for your bot.

Note: set the context lifespans to 1 for every autogenerated context. I have already written about it in a previous post, but I quite strongly believe that the optimal lifespan value of a context you create is 1, and you should take the responsibility of managing the context state when the conversation goes off track. In my demo bot, I am not handling the case where the user goes off track, but with a little practice you can see that it wouldn't be very hard to implement.

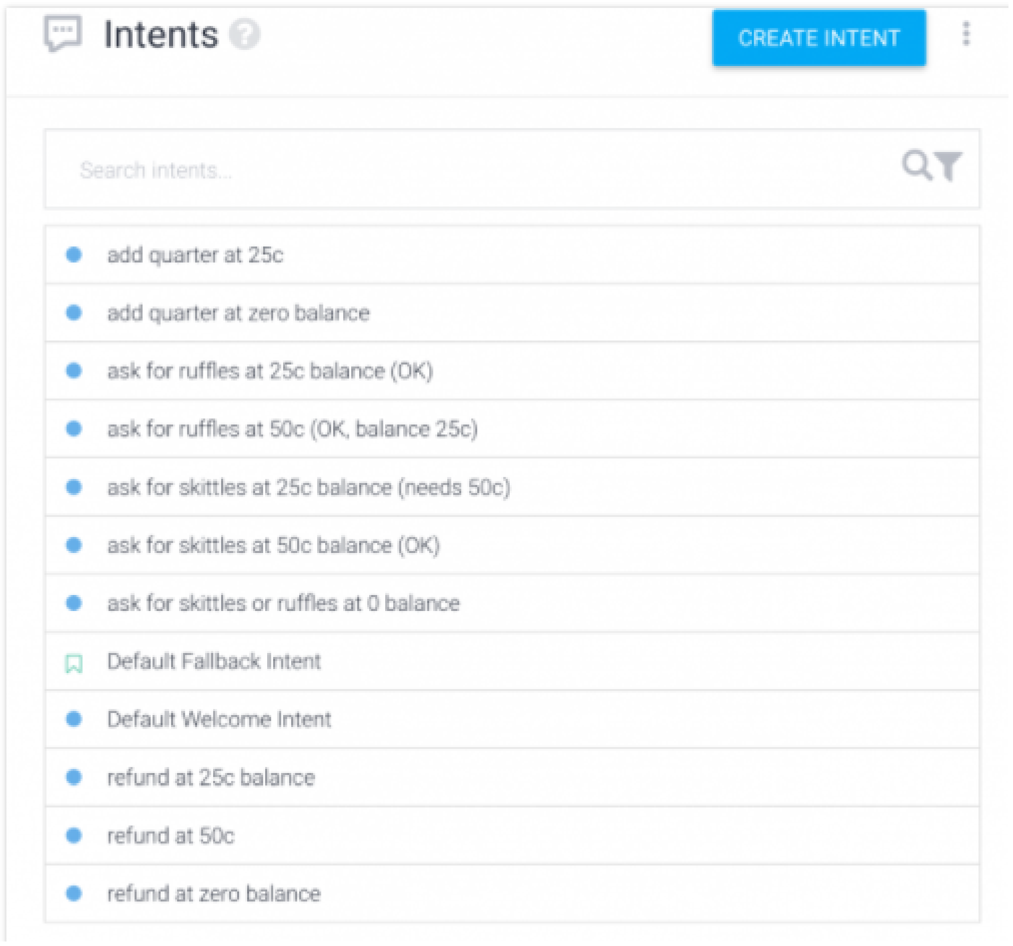
Step by step guide to DialogFlow (API.AI) Part 19 – Creating the vending machine bot with follow up intents

Table of Contents for Step by Step Guide

If you remember, this is what the state diagram for the vending machine bot looked like.


User says: Give me a refund
Response: Balance is 0c
Balance: 0c
User says: Add 25c
Response: 25c added. Balance is now 25c
Balance: 25c
User says: I will take Ruffles
Response: Insufficient balance
Balance: 25c
User says: I will take Skittles
Response: OK, Skittles coming up
Balance: 0c
Zero balance
25c balance
User says: Give me a refund
Response: Refunding 25c. Balance is 0c.
Balance: 0c
User says: Add 25c
Message: 25c added. Balance is 50c.
Balance: 50c
User says: Give me a refund
Message: OK, refunding 50c
Balance: 0c
50c balance
User says: Give me Ruffles
Message: OK, Ruffles coming up.
Balance: 0c
User says: Add 25c
Message: Cannot add any more quarters.
Balance: 50c

This is what the list of intents looks like for the old vending machine chatbot.



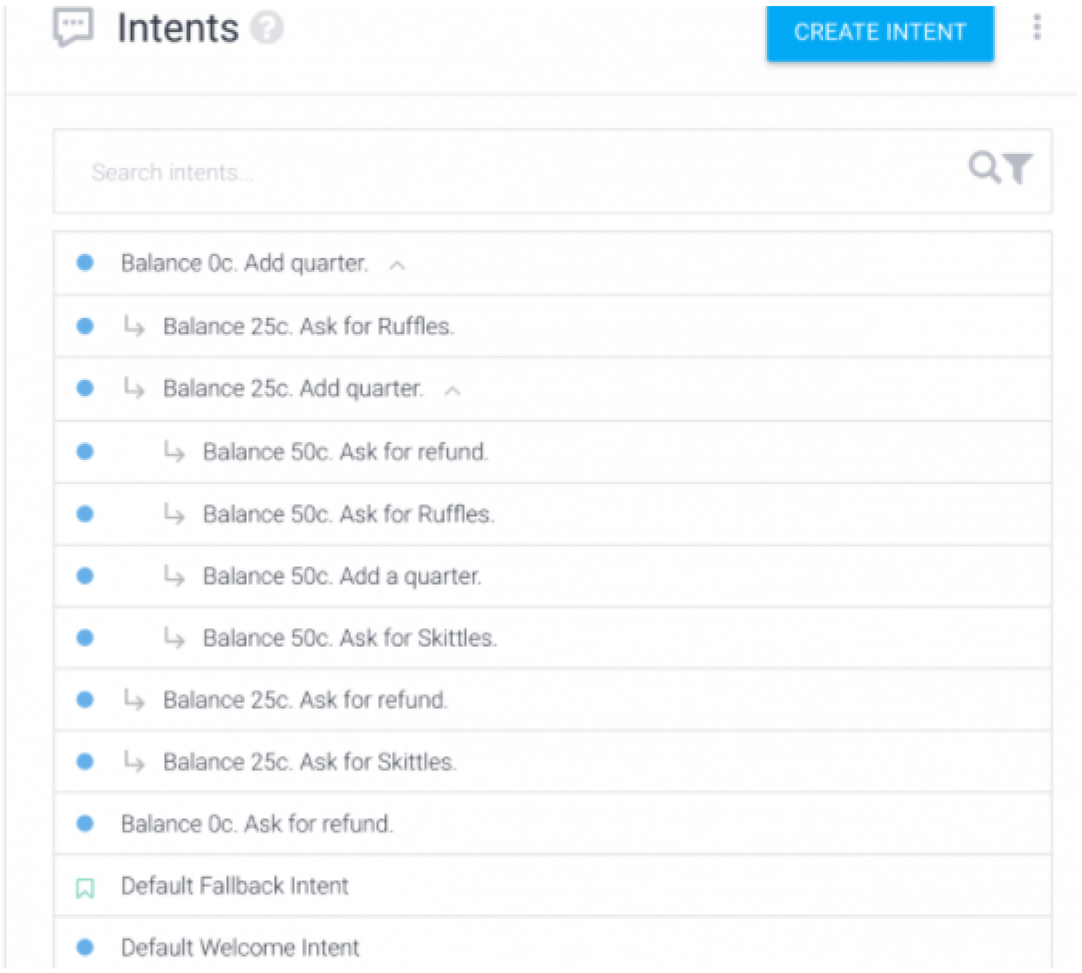
Intents

CREATE INTENT

Search intents...

- add quarter at 25c
- add quarter at zero balance
- ask for ruffles at 25c balance (OK)
- ask for ruffles at 50c (OK, balance 25c)
- ask for skittles at 25c balance (needs 50c)
- ask for skittles at 50c balance (OK)
- ask for skittles or ruffles at 0 balance
- Default Fallback Intent
- Default Welcome Intent
- refund at 25c balance
- refund at 50c
- refund at zero balance

After rewriting the vending machine chatbot, this is what the list of intents look like.



Balance 0c. Add quarter.
↳ Balance 25c. Ask for Ruffles.
↳ Balance 25c. Add quarter.
↳ Balance 50c. Ask for refund.
↳ Balance 50c. Ask for Ruffles.
↳ Balance 50c. Add a quarter.
↳ Balance 50c. Ask for Skittles.
↳ Balance 25c. Ask for refund.
↳ Balance 25c. Ask for Skittles.
Balance 0c. Ask for refund.
Default Fallback Intent
Default Welcome Intent

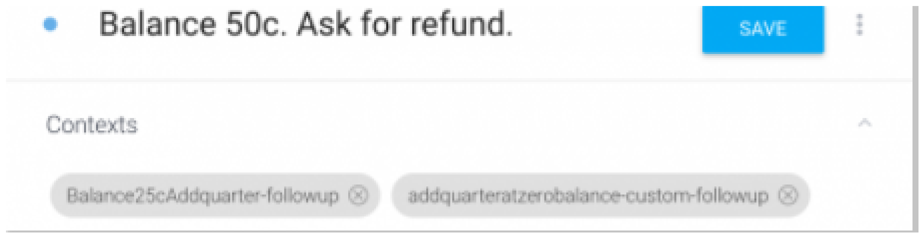
While the vending machine bot becomes easier to build using follow up intents, I have a couple of concerns with the way the follow up intents are implemented, under the hood

1 Contexts are used to create the hierarchy

Once you look under the hood, you will see that DialogFlow still uses contexts to create the hierarchical structure. There is one huge benefit when doing this – everything is completely transparent. On the other hand, this also means you are still going to have to track what your contexts are doing (for e.g. don't accidentally delete an autogenerated context, because that will remove the intent from the hierarchy).

2 The existing naming conventions for the automatically generated contexts is not very user friendly

Look at these autogenerated context names as an example:



Balance 50c. Ask for refund.

Balance25cAddquarter-followup

addquarteratzerobalance-custom-followup

On the other hand, you cannot rename it into something more user friendly without going back and updating every other context. Which is quite a pain. Maybe DialogFlow must provide a way to rename contexts in place and have it reflect everywhere automatically.