The Dialogflow Manual

Written at 2am on 8th of June, 2019
Slight revision on 1st Apr, 2020

Author's note:
This instruction manual aims to briefly cover the major concepts of Dialogflow - how to create intents, build conversation flow and more advanced chatbot features *in a light-hearted manner*. The target audience is assumed to have no programming background, although only for concepts related to Events, Fulfillment, backup and some troubleshooting, knowledge of NodeJS and JSON file is required.

Viet respectfully borrowed certain materials from Dialogflow documentation, although most of the concepts here are derived from his own experience of working on Lyon 1.0
. Nonetheless, it is generally encouraged that readers should refer to the Google Documentation for more details. This is just a quick User Guide that do two things:

1. A brief walkthrough of the system
2. Major possible exploits to turn the impossible possible
3. In the revised version, there are external links to possible help sources.

Google's documentation are good, but they are everywhere and are technical most of the time, so this guide aims to serve as the backbone for your learning and provides the link to more technical stuff.

Now...Let's get to business

Content Structure

Part I. Dialogflow Core Concepts

1. Intents
2. Entities
3. Contexts
4. Events
5. Fulfillment and other features

Part II. Dialogflow Implementation - Lyona

1. Configuration
2. Creating simple intents
3. Intent naming convention
4. Creating entities
5. Creating a simple conversation flow
6. Creating complex conversation flows
7. Optimize intents with shared intents and context
8. Webhooks
9. Training

Part III. Troubleshooting and Hacks
Part IV. Conclusion

Part I. Dialogflow Concepts

1. **Intents**

   Intent is the smallest building block of any Dialogflow chatbot. It is the Questions - Answer pair defined in the bot. Questions are the phrases (utterance[1]) that the user says to the bot, also known as training phrases. Answers are the, well, answers given in response to that question, also known as response. When the bot recognizes a phrase by the user, it will give the corresponding answer. This is called intent **matched.**

   Having more training phrases will allow the bot to respond to a wider range of question phrasings (and human's creativity). Having more answers will make the bot appear more natural in responding to queries.

   Under each intent, there can be different ways to ask the same questions (we called them different "training phrases") as well as different ways to answer (or "variant answers"). Most of the phrases have to be defined so the answers are usually hard-coded, although as you will see below, there are ways to give flexible answers.

   There will be multiple ways people can refer to these 3 concepts of Intent-Questions-Answers. For those doing UI/UX and content management, it is best to be called Intent, Q and A. For those who are doing the Dialogflow backend engine and the chatbot trainers, it is better to be called intent, training phrases and responses. There are other names from Google, such as user sayings, user utterances - you might see them in Google's official documentation.

   For the less experienced user, maybe now is the right time to use the "Step by Step Guide" document instead.

   Useful link:
   https://cloud.google.com/dialogflow/docs/intents-overview

   Now assume instead of putting a simple text in the response field, we want to send back an image or a clickable button that opens up an external link to a website. How do we do this?

   Currently there are two ways to do this:
   1. Dialogflow custom payloads (our own code)
   2. Google Assistant payload (Google's code)

[1] Google likes to make things complicated

There will be a lot of performance and maintenance issues related to this (refer to part 2, section 7, point B). But for now, all you need to know that it is through these custom payloads that we can deliver a richer experience to our users, by providing them with multimedia responses, responses with prompts, etc … The possibilities are endless.

Useful links:
https://cloud.google.com/dialogflow/docs/intents-rich-messages (RAW)
https://developers.google.com/assistant/conversational/rich-responses (GA Payload)

2. Entities

These are the parameters we, the trainers and developers, want to capture in what the user says. For example, if the user says "Hi I'm Xavier", we would like to answer "Hi Xavier how are you"/"Hi there Xavier". We want to capture the name "Xavier" and use it in the bot's answer. By using the entity "given-name", we can pass the name "Xavier" in the user's saying to the bot.

Another use of entity is to define synonyms. Instead of giving 6 different training phrases that only differ in one word, by using synonyms (created by entity) we can cut down the number of training phrases.

There will be other usage of entities for much more complex features later on, but for now it suffices to know entities do the following:

1. Capture users' info if we wish
2. Create synonyms

As of April 2020, Viet still holds the view that you should **NOT** use entities at all if possible for these following reasons:

1. Once entities are used, they must be removed from all intents they appeared in before the declaration can be deleted.
2. The behaviours of entities are very unpredictable.

You had my warnings.

Useful links:
https://cloud.google.com/dialogflow/docs/entities-overview
From the last time I saw it, there is now regex matching (https://cloud.google.com/dialogflow/docs/entities-regexp) and fuzzy matching (https://cloud.google.com/dialogflow/docs/entities-fuzzy) (those who took CZ 3005 Art Intel should know this). These are *very* powerful mechanisms, and if used wisely can solve a lot of problems in one line. That also means when there is a bug, things are a lot harder to debug as bugs are going to be very obscure.

3. **Contexts**

Just like in everyday conversation, different words means different things in different contexts, Dialogflow's contexts allow different training phrases to be interpreted differently.

There are two type of contexts: Input & Output

When an intent has an input context, in order for that intent to be matched, its input context must be set AND the user phrase must match the training phrases.

On the other hand, after the bot gives a response, it can set an output context. Kind of to control how the conversation will continue afterwards, because the next time the bot responds, it has to match BOTH the context set earlier on and what the user said.

By now you probably have no idea what context is. This will get clearer when we go to conversation flow, hopefully… Just keep it in view for now.

Useful links:
Beginner:
https://chatbotslife.com/how-to-handle-context-with-dialogflow-part-1-knock-knock-jokes-4659b346d83b
A detailed working mechanism can be found at : https://cloud.google.com/dialogflow/docs/contexts-input-output

4. **Events**

Recap: Intents can be matched by matching training phrases, and if that intent has input context, the context must also be matched.

Now, let's consider an intent named A with training phrase Q_a and response A_a. The only way to invoke answer A_a is if the user says the phrase Q_a (and the context must match, of course).

What if….we want to invoke answer A_a if certain *conditions* are met, such as "user idle for 10 minutes" ? There is no saying from the user whatsoever, so how can we trigger the response, aka make the bot say A_a in such a case?

That is where Events come in. An Event is a condition (it is just a json entry in the request) that can directly trigger the answer A_a, or any other response without the need for entry matching. This will be extremely helpful later on with complex conversation flows, and it will also cause a lot of problems.

Useful Links:
https://cloud.google.com/dialogflow/docs/events-overview

Events are often used in conjunction with Fulfillments. You will see it later.

5. **Fulfillment and other features**

I haven't mentioned that Dialogflow is powered by Natural Language Processing (NLP) and Google's Machine Learning (ML) Algorithm. Thanks to that, it is superior to conditional logic chatbot (FlowXO, manychat) where bot is built by if [user says something] then [bot says something]. In those cases, when the user says something slightly different, the bot will not be able to answer. NLP and ML greatly expand the bot capacity to better understand human utterance. Combining that with the different ways to ask the same question, Dialogflow provides a powerful brain for our bot in a very cost-effective and efficient manner.

But that's only the brain. Anything not related to the Question-Answer logic will not be a focus of dialogflow. So what if we want to customize the response or create more complex use cases, beyond what Dialogflow was designed for?

For response, we can diversify (include buttons, suggestion chips, images) and beautify it via third party integration like Kommunicate or Dialogflow supported platforms (see their docs) - together with custom payloads discussed in point 1. For complex or customised logic, we have to create this logic in a server (Firebase) and pull the logic into Dialogflow (more on this later).

All of this require Dialogflow to actually communicate with other servers via some API calls, and they communicate via *webhooks*. Just imagine Dialogflow send a request to the other server, and that server will give it some response, which Dialogflow gonna use. The format of the request and response is dictated by the communication protocol between Dialogflow and the server (known as the Application Programming Interface). And the server is Firebase.

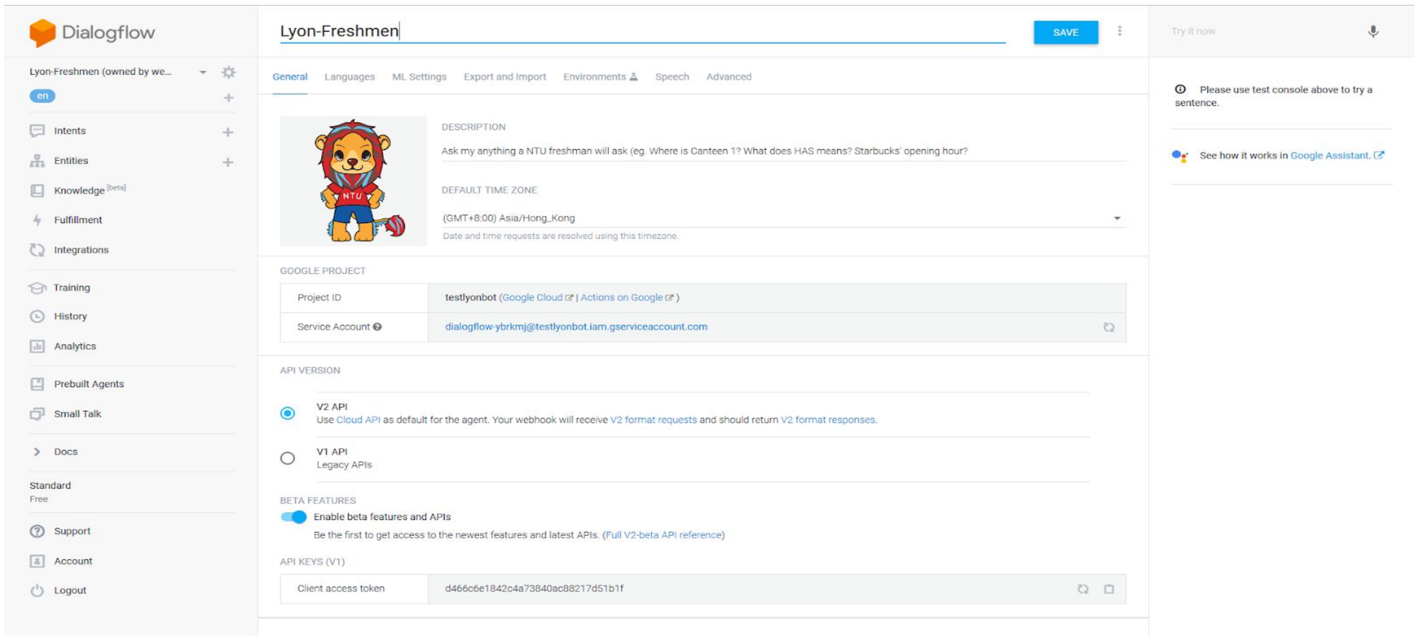
The interesting part is we can choose to enable webhooks for each intent. If we do not enable it, the normal process of keyword matching goes as usual. But if we do, we can bypass the response and do many more funny things, *if we know how to.* Being able to expand and develop more functionalities on top of Dialogflow built-in capabilities will allow us to enhance our bot a lot further.

There is another feature called "Knowledge Base". We will not use them here (yet) so please visit the official documentation on Dialogflow if you wish to know more about them. One word of caution: the ML machine works terribly awful with KB (again, those who took CZ3005 Art Intel will know).

Part II. Dialogflow Implementations - Lyona

The readers are assumed to know the interface of Dialogflow console. I will only include illustration when absolutely necessary to save space.

0. **Configuration**
   Set up a Google Cloud Project and then create an agent. You will be brought here



Lyon-Freshmen

DESCRIPTION
Ask my anything a NTU freshman will ask (eg. Where is Canteen 1? What does HAS means? Starbucks' opening hour?

DEFAULT TIME ZONE
(GMT+8:00) Asia/Hong_Kong
Date and time requests are resolved using this timezone.

GOOGLE PROJECT

Project ID	testlyonbot (Google Cloud | Actions on Google)
Service Account	dialogflow-ybrkmj@testlyonbot.iam.gserviceaccount.com

API VERSION
[✓ V2 API
Use Cloud API as default for the agent. Your webhook will receive V2 format requests and should return V2 format responses.]
V1 API
Legacy APIs

BETA FEATURES
[✓ Enable beta features and APIs]
Be the first to get access to the newest features and latest APIs. (Full V2-beta API reference)

API KEYS (V1)

Client access token	d466c6e1842c4a73840ac88217d51b1f

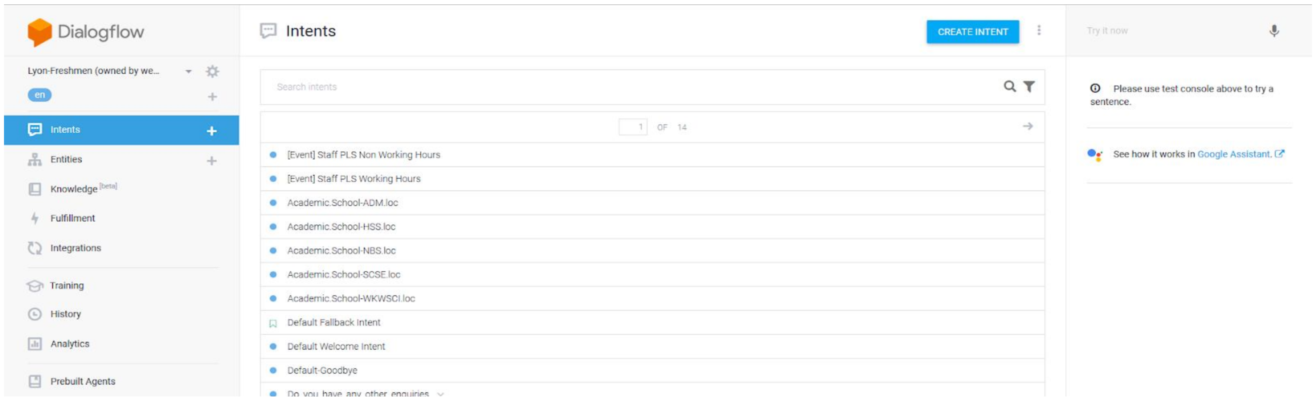
Please use test console above to try a sentence.

See how it works in Google Assistant.

Give your agent a name and a profile pic (please avoid test_abcxyz as it's hard to change later on). Choose V2 API. The default settings will do fine for now, so let's jump to creating intents.

Note that I will refer to the left-hand side panel very often and the cogwheel next to the agent's name will be called "settings"

1. **Creating simple intents**
   "Intents" is the first and foremost tab on the panel, and for the right reason. It is the main building blocks of an agent. Click on the "Intents" to see all the existing intents, click on the "+" next to it or the "create intents on top right corner" to create more intents

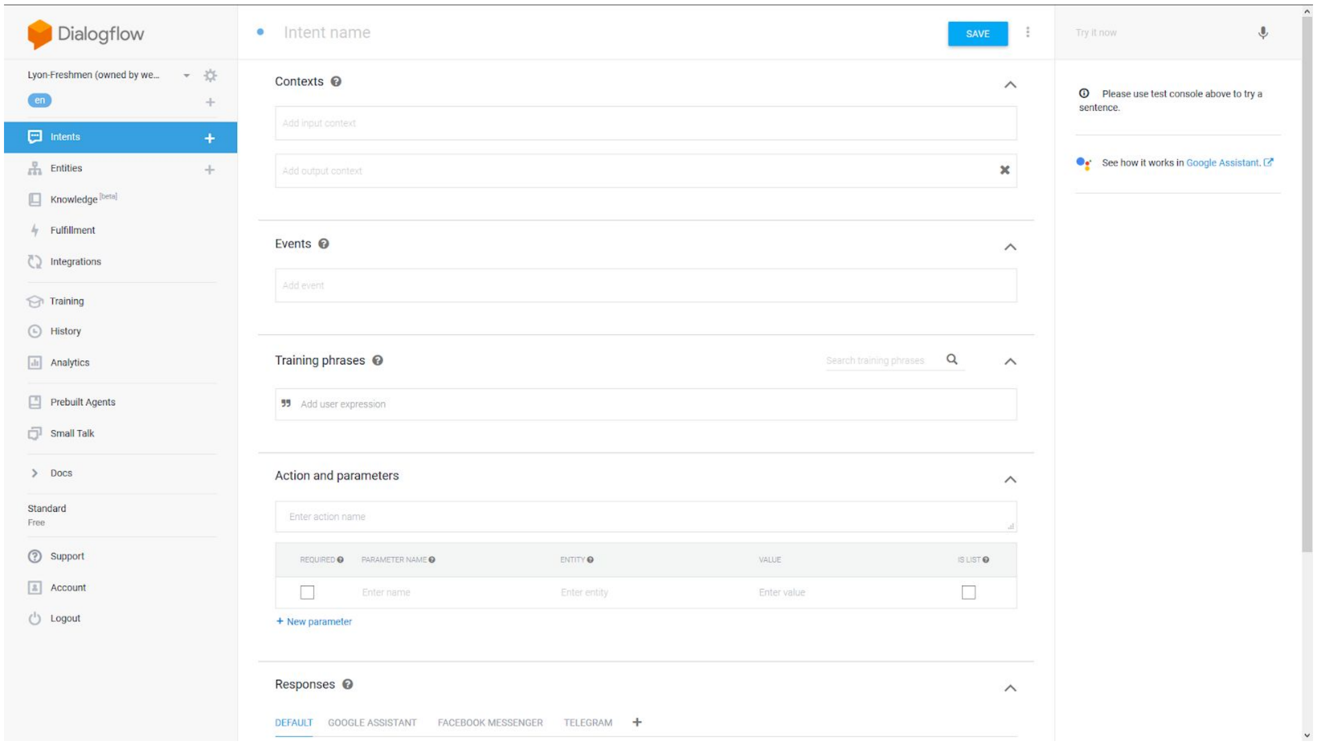


[Event] Staff PLS Non Working Hours
[Event] Staff PLS Working Hours
Academic.School-ADM.loc
Academic.School-HSS.loc
Academic.School-NBS.loc
Academic.School-SCSE.loc
Academic.School-WKWSCI.loc
Default Fallback Intent
Default Welcome Intent
Default-Goodbye
Do you have any other enquiries

Please use test console above to try a sentence.

See how it works in Google Assistant.

The new intent-creating page would look like this



Please use test console above to try a sentence.

See how it works in Google Assistant.

There are:

a. Intent name
b. Context
c. Events
d. Training Phrase
e. Action and Parameters
f. Responses
g. Fulfillment

We will deal with Training Phrases and Responses first. Intent name, Context, Action & Params, Events and Fulfillment will be dealt with later, in that order.

Training Phrases are *what you think the user would say*. (I would like to highlight this as to differentiate training phrases from users' phrases later on in Training section). For every intent, try to have at least 10 different training phrases so that the bot can better understand the question from the user and hence respond accurately. Google standard is 23 phrases for every intent, but let's not get worked up to that level. The more varied and creative, colloquial the training phrases are, the better.

However, please keep the training phrases of each intent unique. If different intents have the same training phrases, funny and nasty things can happen.

The responses are the answer the bot gives. You can give variant answers so that the bot can pick from a list of answers to answer the user, making the conversation more flexible and natural.
Once you are done, give the intent a name (more on this later) and click save. There will be a green tab said "Intent saved" appearing at the bottom right. Wait for "Agent training started" and "Agent training completed" before you move on to another page/tab.

2. **Intents naming Convention**

As the agent grows with intents, thing will start to get messy. "Oh the question abc is in which intent again?" "Where is the intent that contains the answer xyz?" etc. So it is absolutely important to start out on the right note by naming things properly.

Intent names should avoid special characters like ? ! @ & % $ ( ) as well as whitespace (for programming reasons, explained in Fulfillment section). Use full stop . and underscore _ or hyphen to delineate separation. What kind of separation you may wonder?

Hierarchy

As intent grows to 200+ or so, it may be 10 pages of intents to scroll through so usually it is sensible to group them up. And then as intents grow further to 500 + we start to need to group these groups together. And so on, so we need a way to group them and name them. An industry standard for naming these groups is provided below. It might not be the best for every case, but it provides a reasonable balance between length and clarity.

[Overall Group].[Main Category].[Sub category].[Intent name]

For e.g in Lyona: Freshmen.Lifestyle.Facts.Canteen14

Overall Group should usually be the main target audience of the bot

Main category and subcategory is up to discussion
All names should be capitalized or at least camelCase for ease of reading.

You may wish to use hyphen or underscore but keep it consistent. Not only will it standardize, it is easier for other people who are unfortunate enough to have to maintain your work later on.

3. **Creating entities**
   There is nothing much to say for entities for 2 reasons:
   1. I don't use entities a lot
   2. The built-in entities (denoted by @sys.[entity name]) are quite sufficient

   So before you create any entities, please check Dialogflow documentations if there is already a built in entity that suits your needs. But if there is none, feel free to create one. As I said before, entities are mainly used for synonyms. After you have created the entities, you have to annotate them in the training phrases.

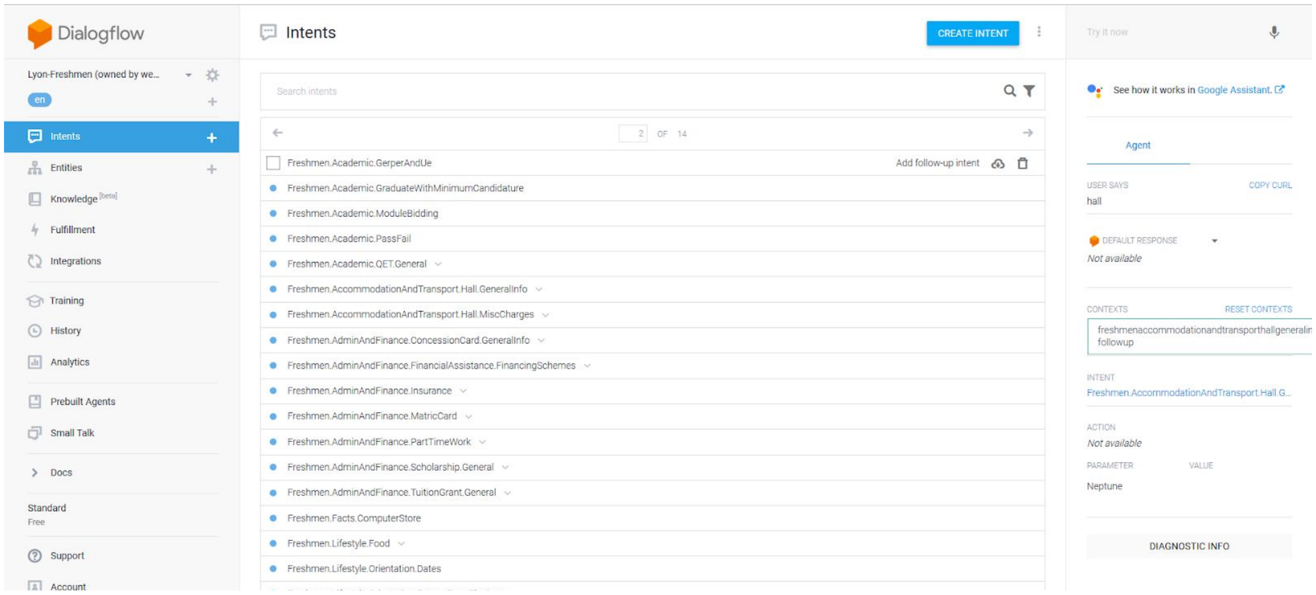
   Go back to an intent whose training phrases contain these words. Highlight the word and you should see a small popup open up. It has a list of entities to choose from. Click on the one you desire and you should see the word is now highlighted. The details in Action and Parameters will automatically be filled in. If you annotate it by accident, just click on the training phrase and click on the trash bin icon in the dropdown.

   More usage of entities are given in Fulfillment section as well as part III, use case 4.

4. **Creating a simple conversational flow**
   You will be tempted to give a very long answer to each question, as ahem, many of us did. This is not the way for an user-friendly chatbot as users don't want to be flooded with info in one go. So the trick here is to feed them one bite by one bite, giving some info and ask them if they want to know more, slowly leading them into a conversation.

   E.g: User wants to know about the hall. Instead of sprouting all 24 pages about NTU halls in one go, let's break it down into this:
   User: Any Hall in NTU?
   Lyona: Yes there are 24 halls in ntu. What would you like to know?
   Lyona: *Shows suggestion chips for different categories of info about hall*

   This is clearly a much user friendly way of doing things. And to do this we need to use follow up intent. In the intent tab hover on any intent it will show up on the right side


Dialogflow

Lyon-Freshmen (owned by we...

en

Intents | Entities | Knowledge [beta] | Fulfillment | Integrations | Training | History | Analytics | Prebuilt Agents | Small Talk | Docs | Standard Free | Support | Account

Intents — CREATE INTENT

Search intents

2 OF 14

- Freshmen.Academic.GerperAndUe — Add follow-up intent
- Freshmen.Academic.GraduateWithMinimumCandidature
- Freshmen.Academic.ModuleBidding
- Freshmen.Academic.PassFail
- Freshmen.Academic.QET.General
- Freshmen.AccommodationAndTransport.Hall.GeneralInfo
- Freshmen.AccommodationAndTransport.Hall.MiscCharges
- Freshmen.AdminAndFinance.ConcessionCard.GeneralInfo
- Freshmen.AdminAndFinance.FinancialAssistance.FinancingSchemes
- Freshmen.AdminAndFinance.Insurance
- Freshmen.AdminAndFinance.MatricCard
- Freshmen.AdminAndFinance.PartTimeWork
- Freshmen.AdminAndFinance.Scholarship.General
- Freshmen.AdminAndFinance.TuitionGrant.General
- Freshmen.Facts.ComputerStore
- Freshmen.Lifestyle.Food
- Freshmen.Lifestyle.Orientation.Dates

Try it now

See how it works in Google Assistant.

Agent

USER SAYS: hall — COPY CURL

DEFAULT RESPONSE: Not available

CONTEXTS — RESET CONTEXTS: freshmenaccommodationandtransporthallgeneralin followup

INTENT: Freshmen.AccommodationAndTransport.Hall.G...

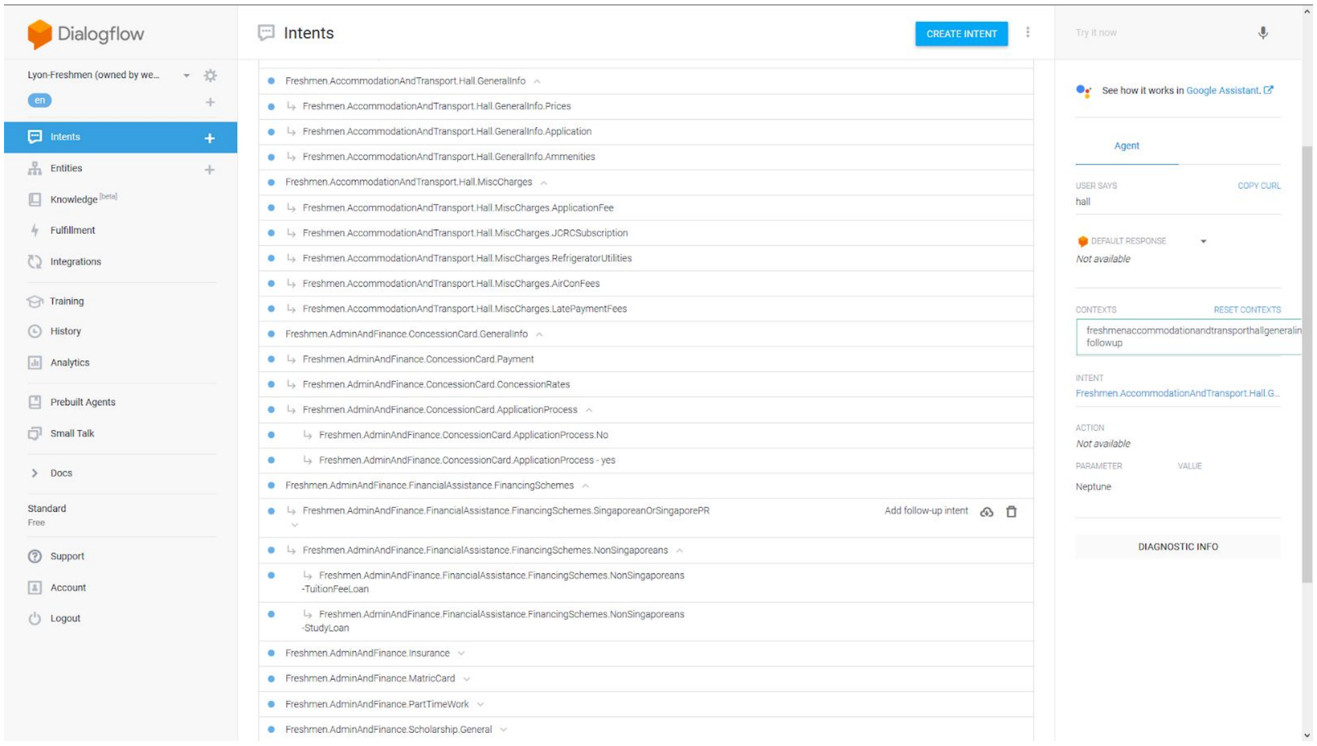
ACTION: Not available

PARAMETER: Neptune — VALUE

DIAGNOSTIC INFO

So for our example earlier, we would have an intent called Freshmen.Accomodation.Hall.GeneralInfo, under which there is 3 follow-up intents: Price, Amenities and Application process. Under each of these follow up intents (or subintents) you can branch out even more follow up intents.

In Lyona, we created many flows of different conversations, as shown below


Dialogflow

Lyon-Freshmen (owned by we...

en

Intents | Entities | Knowledge [beta] | Fulfillment | Integrations | Training | History | Analytics | Prebuilt Agents | Small Talk | Docs | Standard Free | Support | Account | Logout

Intents — CREATE INTENT

- Freshmen.AccommodationAndTransport.Hall.GeneralInfo
  - Freshmen.AccommodationAndTransport.Hall.GeneralInfo.Prices
  - Freshmen.AccommodationAndTransport.Hall.GeneralInfo.Application
  - Freshmen.AccommodationAndTransport.Hall.GeneralInfo.Ammenities
- Freshmen.AccommodationAndTransport.Hall.MiscCharges
  - Freshmen.AccommodationAndTransport.Hall.MiscCharges.ApplicationFee
  - Freshmen.AccommodationAndTransport.Hall.MiscCharges.JCRCSubscription
  - Freshmen.AccommodationAndTransport.Hall.MiscCharges.RefrigeratorUtilities
  - Freshmen.AccommodationAndTransport.Hall.MiscCharges.AirConFees
  - Freshmen.AccommodationAndTransport.Hall.MiscCharges.LatePaymentFees
- Freshmen.AdminAndFinance.ConcessionCard.GeneralInfo
  - Freshmen.AdminAndFinance.ConcessionCard.Payment
  - Freshmen.AdminAndFinance.ConcessionCard.ConcessionRates
  - Freshmen.AdminAndFinance.ConcessionCard.ApplicationProcess
    - Freshmen.AdminAndFinance.ConcessionCard.ApplicationProcess.No
    - Freshmen.AdminAndFinance.ConcessionCard.ApplicationProcess - yes
- Freshmen.AdminAndFinance.FinancialAssistance.FinancingSchemes
  - Freshmen.AdminAndFinance.FinancialAssistance.FinancingSchemes.SingaporeanOrSingaporePR — Add follow-up intent
  - Freshmen.AdminAndFinance.FinancialAssistance.FinancingSchemes.NonSingaporeans
    - Freshmen.AdminAndFinance.FinancialAssistance.FinancingSchemes.NonSingaporeans-TuitionFeeLoan
    - Freshmen.AdminAndFinance.FinancialAssistance.FinancingSchemes.NonSingaporeans-StudyLoan
- Freshmen.AdminAndFinance.Insurance
- Freshmen.AdminAndFinance.MatricCard
- Freshmen.AdminAndFinance.PartTimeWork
- Freshmen.AdminAndFinance.Scholarship.General

Try it now

See how it works in Google Assistant.

Agent

USER SAYS: hall — COPY CURL

DEFAULT RESPONSE: Not available

CONTEXTS — RESET CONTEXTS: freshmenaccommodationandtransporthallgeneralin followup

INTENT: Freshmen.AccommodationAndTransport.Hall.G...

ACTION: Not available

PARAMETER: Neptune — VALUE

DIAGNOSTIC INFO

The bottom line is to feed users with only a little bit of info each time to make our bot's answer conversational rather than an info dump.

5. **Create complex conversation flows**
   By now if you have not noticed, by introducing flows into our conversation, our dialogue with Lyona bot will no longer be piecemeal. Imagine if all intents are given as individual intents, no follow up. User asks one, Lyona answers one, conversation stops. Pretty boring, isn't it?

   Another issue is that when the answer is too long, the answer spans longer than the screen of most our phones can handle. As such it is a ***good conversational design practice*** to put at most 30 words in each answer.

   The practice we adopted for Lyon 1.0 is as followed:

   Assume the user asks question about Hall system in NTU, and the answer composes of 3 parts A (type of halls), B (location) and C (price)

   The approach is to ***break down*** the answer into A, and prompt if the user wants to know more about **B** or **C,** rather than give all ABC in one go.

   So it will look something like:
   User: I want to know more about NTU Halls.
   Bot: There are 24 types of halls in NTU [a short explanation]. Do you want to know more about hall locations? Or Perhaps Price?
   User: I want to know about rental fees
   Bot: Each hall has different fees that are bound to change every year. For this year's price, you can visit this link <link>. ***Do you have any other questions?***
   User: Yes I also want to know about the hall's locations
   Bot: The halls are located across NTU campus. There are x halls in the north side and y halls in the south side of the campus. This link <link> will give you more details if you are interested.

   As you can see, now rather than a simple QnA, our d*ialogue flows.* This should be the highest aim of us developers - to make the bot as hospitable and human like in its interaction as possible. This is what Google aims to provide at Dialogflow . A more sophisticated bot. (Just a joke, originally it was API.AI - just being complicated. Now it becomes Dialogflow, which is a bit more human)

   A few features of a sophisticated bot:
   1. Buttons/Suggestion Chips

Instead of letting users type, we can let the bot show buttons for users to click on (known as "suggestion chips") and generate a reply. Nuancedly integrated buttons with follow-up intents will create a seamless conversational flow.

The idea is that if the bot shows a button to click on, by clicking on the button, there will be a reply automatically sent back to the bot instead of the user having to type at all. If this reply matches the training phrase of the follow-up intents, Dialogflow will match the intents and carry on the conversation as we have designed.

Therefore, we have to design the button such that the replies created by the buttons match the training phrases of the follow-up intents.

2. Do you have any other questions?

We often forget that the bot oftens ends a conversation abruptly. This gives a very mechanical feels to our UX, and thus a bad practice to follow. As a suggestion, you can always prompt the user by asking ***Do you have any other questions?*** (this is covered in greater details in the next section about optimized context). But I believe the bot has the potential to be even more flexible on how it handles this issue.

Notes:
So far follow-up intents seem like a good thing, which it is. But it can also cause a lot of problems. The most major one is about contexts, which I will explain below:

Let's assume we have an intent A1 and its follow-up intent is intent A2. How Dialogflow "follow-ups" happens like this: A1 has an output context, let's say A1-followup. At the same time, A2 being the follow-up intent of A1, will have an input context of A1-followup. Assume the user says the correct things then it should go like this

User: [A1 training phrase]
Lyona: [Answer to A1] + Set context to A1-followup
User: [A2 training phrase]
Lyona: [Answer to A2] (*)

At (*), as context is set properly and the training phrase is matched, the A2 intent will be properly matched. Problems arise when the training phrase **does not** match properly. Imagine if we have another intent A3 which is more similar to what the user says than intent A2's training phrases. Lyona will jump to A3 after A1 and hence breaks the conversation flows.

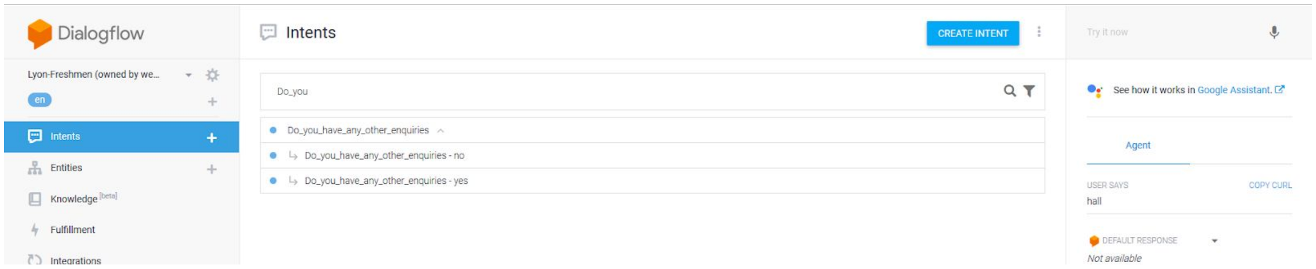
As such, we utilize buttons to control the response from the users. The reply from the button is pre-programmed by us, and it will match perfectly with the follow-up intents. However we also try to not overuse it. It is a chatbot after all, not a clickbot.

There are also problems with breaking into a flow i.e matching a follow-up intent without going through the designed flow. This is a complex problem with an equally complex solution proposed in part III, point 5.

6. **Optimize intents with shared intents and context**

As we branch out into more complex, lengthy flows, we will eventually bump into the classic problem: how can we reuse some of the intents. The most profound one would be at the end of every flow, we would like to ask the user “Do you have any other questions?”. We wouldn’t want to create a DoYouHaveAnyOtherQn follow up intent for every end cases of every flow - that would be horrendous (and a programming nightmare as well). Luckily enough by tweaking the use of contexts we can achieve this quite easily.

First, create a intent like this



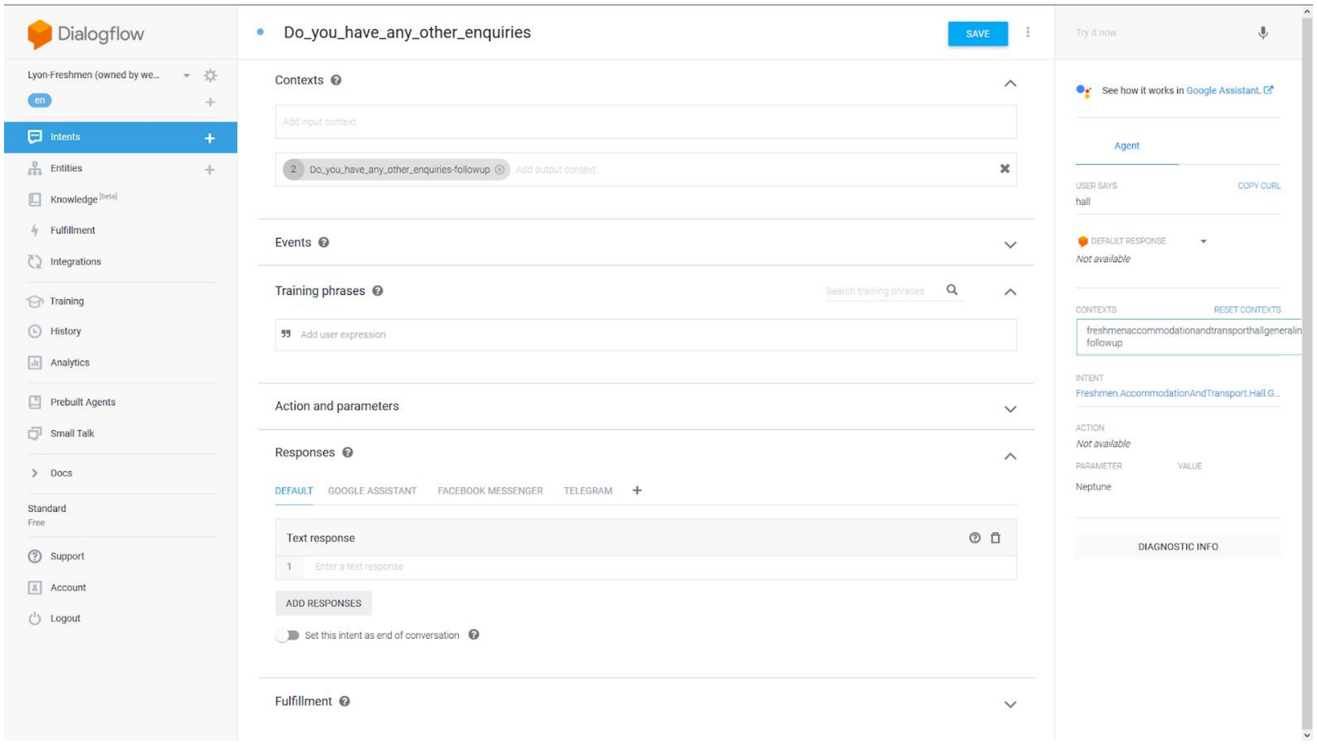
Do_you

Do_you_have_any_other_enquiries
↳ Do_you_have_any_other_enquiries - no
↳ Do_you_have_any_other_enquiries - yes

hall

Not available

The parent Intent is empty, what we actually wants is the auto context generated for us



Do_you_have_any_other_enquiries

Contexts

Add input context

2 Do_you_have_any_other_enquiries-followup

Add output context

USER SAYS: hall

DEFAULT RESPONSE: Not available

CONTEXTS: freshmenaccommodationandtransporthallgeneralin followup

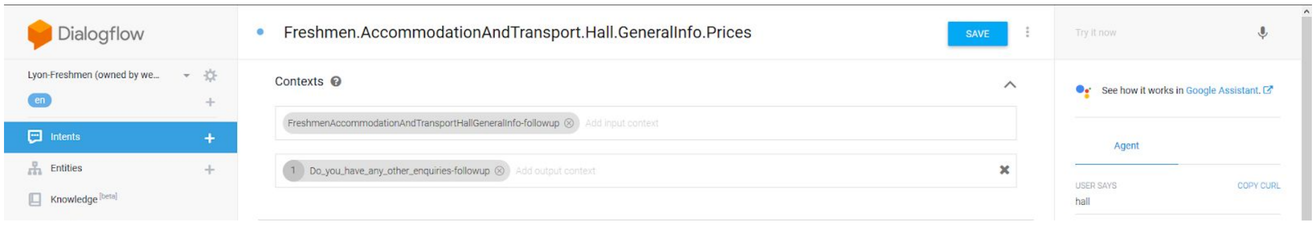
INTENT: Freshmen.AccommodationAndTransport.Hall.G...

ACTION: Not available

PARAMETER	VALUE
Neptune	

See the output context there? Copy it somewhere because we gonna reuse it soon.

Now, goes to the end of a flow and add that context to the output context field, like this:



Freshmen.AccommodationAndTransport.Hall.GeneralInfo.Prices

Contexts

FreshmenAccommodationAndTransportHallGeneralInfo-followup

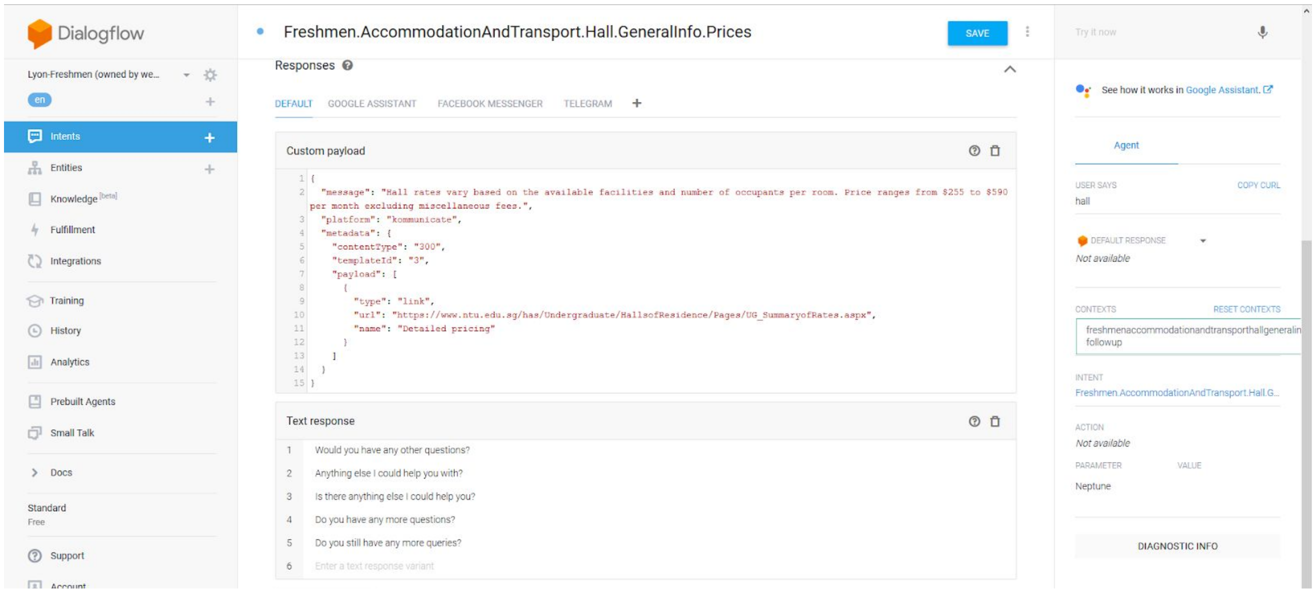
Add input context

1 Do_you_have_any_other_enquiries-followup

Add output context

USER SAYS: hall

And add an additional response **(***Not a variant phrase***)** like this



Freshmen.AccommodationAndTransport.Hall.GeneralInfo.Prices

Responses

DEFAULT | GOOGLE ASSISTANT | FACEBOOK MESSENGER | TELEGRAM

Custom payload

```
{
  "message": "Hall rates vary based on the available facilities and number of occupants per room. Price ranges from $255 to $590 per month excluding miscellaneous fees.",
  "platform": "kommunicate",
  "metadata": {
    "contentType": "300",
    "templateId": "3",
    "payload": [
      {
        "type": "link",
        "url": "https://www.ntu.edu.sg/has/Undergraduate/HallsofResidence/Pages/UG_SummaryofRates.aspx",
        "name": "Detailed pricing"
      }
    ]
  }
}
```

Text response

1 Would you have any other questions?
2 Anything else I could help you with?
3 Is there anything else I could help you?
4 Do you have any more questions?
5 Do you still have any more queries?
6 Enter a text response variant

USER SAYS: hall
DEFAULT RESPONSE: Not available
CONTEXTS: freshmenaccommodationandtransporthallgeneralin followup
INTENT: Freshmen.AccommodationAndTransport.Hall.G...
ACTION: Not available
PARAMETER / VALUE: Neptune

See how the two responses are separated into two separate tabs? That's additional, not variant.

Now what will happen?

By the time that end-of-flow intent gives an answer, it will also ends it with "Do you have any other question" and set the context to "Do_you_have_any_other_enquiries-followup". If the user says yes or no, it matches the training phrase of DYHAOE intent as well as the context, bringing them to the respective response defined as follow up earlier.

TL;DR: if you want to reuse an intent, give it an input context A-followup. Then at the intent before that set the output context to A-followup. The context matching will take care of the rest. An intent can set multiple output context and an intent can also require multiple input contexts.

A word of caution:
If you mess around with the contexts of a flow, it will throw the follow up intents out of the flow. This is because context IS the link between follow-ups and their parent intents. Fixing this is difficult but possible (see part III)

Also, if you rename the parent, the follow-up intent name WILL NOT be changed accordingly. And hence it will get throw out of the flow.

7. Webhooks, Integration and 5 reasons why Viet gonna get called back to CITS to fix issues

So far we have not talked about the platform Lyona is on - Kommunicate, as well as the different platforms that she *can* be on (Facebook, Telegram, etc.) We also have not discussed the custom logic we want to implement, for e.g give different emergency contact based on time of request, says after working hours contact and during working hours contact. I have delayed this so far because all of this has to do with something called Fulfillment, and it requires a heavy amount of Programming as well as understanding of how different software/platforms communicate with each other. So if you are unfamiliar with what I have just said, you may wish to skip this section.

A. About Google-supported platforms

We need to give Lyona a platform to appear in. There are a few platforms supported by Dialogflow - Google Assistant, Facebook, Telegram, Twillio etc These platforms do not only allow Dialogflow to appear and interact with out users but also allow “rich messages” - images, suggestion chips and so on to be implemented. Do note that all these interactive non-text thing are **platform-specific.** What does that mean?

It means if you want to put Dialogflow on 2 platforms you better design the payload that suit specifically to each platform. The response in Dialogflow has to follow their API request - mostly JSON based. Which major player has not used JS front end nowadays ever since Facebook created ReactJS…

In short, design the response in each intent to fit the platform chosen. You can see this by going to Intent, under “Response” click “Add response”, choose “custom payload”.

B. About Kommunicate

Our bots need to make calls , send emails, give suggestion chips and what not. And most importantly it needs to appear on a website if not how can we use it? Now the thing is as I mentioned earlier anything not related to the logic is conveniently ignored by Dialogflow so the onus is on us to find a suitable platform to host this. We can always build it ourselves - there is a NodeJS library and REST API provided. However due to, management’s concerns about sustainability, we cannot write our codes and hence we need to use a third party plugins chat widget that also support dialogflow.

The ONLY viable solution is Kommunicate. Trust me on this. You don’t want to convert our dear Dialogflow NLP into conditional logic chatbot.

Now...with every solution come new problems. Ok so now we can do almost identical stuff at Dialogflow supported platforms. But that also means all our interactive stuff - anything that involves buttons, phone number, link buttons, suggested replies (suggestion chips) has to be redesigned to fit Kommunicate payload template.

And it also means if we want to switch platforms we have to redesign everything again. Jeez

To design the rich message (Google's term)/ actionable messages (Kommunicate 's term)/interactive stuff (My term), just google Kommunicate actionable message, copy paste that template into the custom payload (Intent>under "Response" click Add response>choose custom payload) and put in the corresponding values.

C. About Fulfillment, Events and the server we have on Firebase

If you understand what a webhook is, you can skip this paragraph. Webhook is how a web-based application hook up another application, send a request to that app and gets a response through the application program interface (API).

As said in part I earlier on, if we enable webhook for an intent, it will send the Raw response to a server. That server is on Firebase, and the logic is hosted in 2 files: index.js and package.js. The logic is obviously index,js, package.js only handles the versioning and metadata stuff. So Dialogflow provides a way for us to put in our custom logic to do our own stuff.

There are a lot of things you can do with fulfillment as it is just programming. My advice is therefore general and I will leave the exploration to you:

a. Before Enabling webhook, check "Diagnostic Info" tab and verify the json request that Dialogflow is going to send. Make sure your code input abides by this format.
b. Understand what the agent can do. Check it out in here https://dialogflow.com/docs/reference/fulfillment-library/webhook-client
c. The only few parts that need to change is adding in functions and map them to the intent at the end. And may be some import libraries. Most important is still the mapping. It basically tells Firebase to tell Dialogflow that when this intent is matched, instead of giving the answer, do what the function in Firebase tell you to do. Most of the time I will trigger an Event so it will trigger the response of another intent. Neat. Plus firebase doesn't allow me to send Kommunicate styled payload so I have to put that payload in an intent, and trigger Event to that Intent.
d. Remember to enable webhook for the intents

So create Events, give them a logic routing in Fulfillment and remember to Enable Webhooks. You are good to go to control this bot.

8. Training

So you have come a long way and now you think that you created a wonderful bot. You bring it to the test, and …… unfortunately (expectedly) it doesn’t perform that well. How come?

The answer is I don’t know why. But I know how to partially fixed it.

When the bot gives a wrong answer to a question you thought you have an intent for, goes to “Training” tab on the left hand side panel. It will give a list of sessions of chat that was given to Dialogflow.

Click on the session you wants
Find that particular question that the bot fails to answer
Assign to the correct intent
Click “Approve”
Wait for “Agent training completed”

Now it will answer correctly.

Part III. Troubleshooting, Hacks and Documentation of problems

1. **Broken Flow**

1. Export Agent out as zip (Export and import tab, access via the cogwheel next to the agent name)
2. Extract the folders out, you will find 2 folders (entities and intents) and 2 files - package.js and index.js .
3. Open the “intents” folder. Find all the intents (both [intent_name].json and [intent_name]_usersays_en.json files) under the broken flow and copy them all into another folder (in some other directory)
4. Name this folder “intents”.
5. Fix the “parentId” and “rootParentId” of the [intent_name].json files that you want to be back in the flow. “parentId” is the immediate predecessor intent id, “rootParentId” is the one all the way at the beginning of the flow. Add those 2 keys in if you want to include an intent into an existing flow. Every followup intent must have these 2 fields.
6. Copy entities folder and the other 2 files into the same dir as the new “intents” folder
7. Zip these 4 up and import them back into the bot using “import from zip” option

2. **Different response at different time of the day**

This one is easy. First we need to get the time of the day. This can done by simply use the JS Date object and getHours() & get Minutes() method. Convert this to proper format (I convert to hours) and adjust for the GMT offset (Singapore is Singapore +8)

Also get the day (Mon Tue Wed Thu Fri Sat Sun) but I dont know how to do that so I just leav it there.

Use a simple if else check and carry out the corresponding intent and we are done

3. **Phone number and mail handler**
The url “mailto:someone@email.com” and “tel:12345678” will invoke respective handler in the device. This is purely html protocol.

4. **Entities and trapping user until they give the correct input**

Let’s suppose we create an entity called “HallGeneralInfo” and inside it we have a bunch of synonyms for Price : Fees, Rates, Cost etc. How do we use this entity for correct input?

We need to understand how Actions and Parameters work in Entities. Now, Actions and Parameters are the way Dialogflow sees Entities. Let’s address Parameters first.

Parameter name is the entity name. Parameter value is the leader of the synonyms (refer to previous example, we define synonym for Price as Fees, Rates, Cost. Whenever these words are said by the user, the parameter value given to Dialogflow is Price)

For programmers, parameter name:value is the key:value pair inside each intent json request. If you pop the diagnostic info button to read the full request, you will see the parameter there.

Actions can be either 1. Required params and 2. Define Prompts. If a parameter is required, click on the is required column. Prompts are the prompts given if the training phrases matches what user says but Dialogflow cannot capture a value for the parameter defined.

Says we want the user to ask about hall fees and they keep sprouting rubbish. Enabling “is required” and define a prompt, every time the user says something that does not contain price/rates/fees/cost, prompt will be shown until he gives the correct input.

5. Break into a flow

So what if ...we want to break into the middle of a flow? If we have built a flow so far on follow-up Intents, all the respective contexts are automatically set. Meaning if we only give the training phrase out of nowhere, the input context will not be matched. If we remove the input context, the flow will be broken and our intents will spill all over the place, especially for highly dense flow (with lots of follow-up intents). So how…?

The good news is yes there is a way. The bad news is the way consists of entities, fulfillment, events and webhooks, plus all these must coordinate precisely with one another.

For simplicity, we will consider the flow Hall.GeneralInfo with 3 sub intents Price, Amenities and Application.

First step is to create an entity called HallGeneralInfo (avoid special characters), and define the synonym for keywords of each subintents. So under this entity there are 3 corresponding category:
Price (price, cost, rate, fees, etc)
Amenities (amenities,facilities,courts, etc)
Application (application, apply, etc).

Next is under each subintent, create an event to trigger them. For simplicity let's call the event 'hallPrice', 'hallApp', 'hallAmen'.

Then go to Hall.GeneralInfo intent, put all the training phrases of children to the parent and annotate the phrases with the defined synonym. For safety reason, disable webhook and save intents, check if the entity is capturing the necessary value. Then enable webhook

Finally, go to Fulfillment. Create a function that identifies the entity keywords and trigger the corresponding events.

Sample:

```
function hallBreak(agent) {
    switch (agent.parameters.HallGeneralInfo) {
        case 'Application': agent.setFollowupEvent('hallApp');break;
        case 'Amenities': agent.setFollowupEvent('hallAmen');break;
        case 'Price': agent.setFollowupEvent('hallPrice'');break;
        }
 }
```

Then at the end of index.js map the parent intent to the function via intent.map

Deploy and we are done.

For more complex flows, (e.g if you want to break into the 3rd level, you have to set its context first before trigger the event)

6. Import from zip

The agent.json and package.json can create some problems, like activating input.unknown in the bot. Input.unknown was an event in default fallback intent. This will cause a problem with kommunicate: when it receive that event, kommunicate will assign the conversation to a human, effectively disable the bot

7. The Ghost Intents

So far we do know that if we create a followup intent, the parent intent's output context is automatically set and will be the same as the child intent's input context. And so if we delete this context from one of the places - be it from the parent or the child, then the link is disrupted and hence break the parent-child connection, hence surfacing the child intent as an individual intent.

We also know that the actual json file of each intent has a ‘parentId’ and ‘rootParentId’ keys for followup intents. Via this ID, Dialogflow knows who is the parent and who is the child.

What if.... We sustain this ID linkage but remove all the values for context in json, namely “affected Context” for parent and “required contexts” for child (something along that line, the name is not exact as I pull it from memory). Then import these into Dialogflow. Tada we will have a series of intents that ***looks like*** follow-up intents but are actually individual intents.

Check the smalltalk.agent.developers for an example

8. Fallback of the fallback

To make fallback more nuanced in terms of response, let’s just consider this situation

User: [says some gibberish]
Bot: Sorry could you say that one more time?
User: [says some other gibberish]
Bot: Sorry, what you said is beyond my knowledge. …

We can implement this using a fallback followup on the fallback intent itself. How does it work, I will let you figure it out yourself as if you are reading till here already, it shouldn’t be an issue.

9. Mega agents and future developments

As of Jan 2020, Google has introduced the new Mega Agent, which is just a bypass of the 2000 intents quota. Please refer to the other technical manual if you have not read on the mega agent yet. The point is, it is just a facade (CZ2006: Software Engineering). If Mega agent receives a query, it will forward this query to all its subagents, see which answer has the highest match, and then return. Although this will reduce the configuration and a lot of work, there are two potential pitfalls i foresee with improper implementation:

1. If one bot has many vague training phrases, it will adversely affect the whole system.
2. The content segregation issue will be amplified manifold.

Due to the gravity of this issue, it has been discussed in a separate document (Technical Guide to implement multibot system).

Nonetheless, as the old adage goes: if it can’t do the little things well, it probably can’t handle big things well either. The most appropriate answer should lie in the Razor of Occam.

Part IV. Conclusion

Thank you for reading until here. This manual was created at Lyon 1.0 and at this revision it has become less personal a script and focus on covering the different issues surrounding the chatbot, and what are the better practices to adopt when building one. There will be a lot more hurdles that need to be crossed, and a lot of complications when we wish to scale up what the bot can answer. But the fundamentals remain: Understand the purpose of the bot, improve conversational design and exploit Dialogflow to create such UX. This should serve you well on your journey with building even better bot than the one I started out with.

I wish you all the best for your chatbot development journey.