Set up for local development

**Key terms:**

* **Cloud Functions for Firebase:** A service provided by Google that lets you run your webhook code in a managed environment on the cloud. You use the (command-line interface) tools to deploy your webhook to Google Cloud Functions.
* Navigate to **Fulfillment**.
* Disable **Inline Editor**.
* Click **Save.**

CLI is command line interface best used on Nodejs8

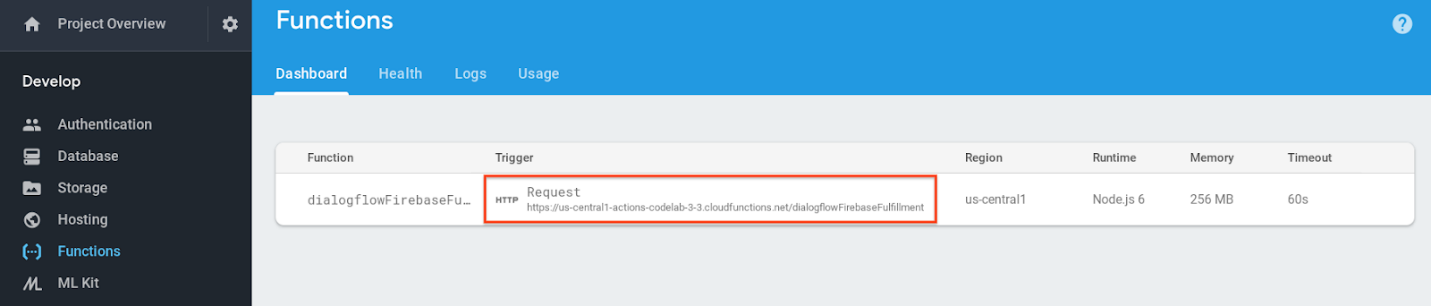
Firebase login

Firebase init to create new cloud in folder

Creates: Parent File firebase.json and Parent file/folder functions

Inside functions folder:

1. index.js
2. package.json
3. Then npm install inside folder FUNCTIONS
4. Cd to parent file
5. Or firebase deploy
6. Open the [Firebase Console](https://console.firebase.google.com/).
7. Select your Actions project from the list of options.
8. Navigate to **Develop > Functions** on the left navigation bar. If you're prompted to "Choose data sharing settings", you can ignore this option by clicking **Do this later**.
9. Under the **Dashboard** tab, you should see an entry for "dialogflowFirebaseFulfillment" with a URL under **Trigger**. Copy this URL.



### **Set the URL in Dialogflow**

Now you need to update your Dialogflow agent to use your webhook for fulfillment. To do so, follow these steps:

1. Open the [Dialogflow console](https://console.dialogflow.com/" \t "_blank) .
2. Navigate to **Fulfillment** on the left navigation.
3. Enable **Webhook**.
4. Paste the URL you copied from the Firebase dashboard if it doesn't already appear.
5. Click **Save**.
6. In the Dialogflow console left navigation, click on **Integrations > Google Assistant**.
7. Make sure **Auto-preview changes** is enabled and click **Test** to update your Actions project.
8. The Actions simulator loads your Actions project. To test your Action, type "Talk to my test app" into the **Input** field and hit enter.
9. You should see a response: "Welcome! What is your favorite color?"
10. Type "blue".
11. You should see a response: "Your lucky number is 4."
12. In the Dialogflow console left navigation, click on **Integrations > Google Assistant**.
13. Make sure **Auto-preview changes** is enabled and click **Test** to update your Actions project.
14. The Actions simulator loads your Actions project. To test your Action, type "Talk to my test app" into the **Input** field and hit enter.
15. You should see a response: "Welcome! What is your favorite color?"
16. Type "blue".
17. You should see a response: "Your lucky number is 4."

Add Intent

### **add intent for deep linking and implicit invocation**

To add your intent for deep linking and implicit invocation, do the following:

1. In the [Dialogflow console](https://console.dialogflow.com/" \t "_blank) left navigation, click on **Integrations**.
2. In the **Google Assistant** card, click **Integration Settings**.
3. Under **Discovery > Implicit Invocations**, click on **Add intent** followed by **favorite color**.

"Talk to my test app about blue" into the **Input** field and hit enter.

### **Define a custom fallback intent**

custom fallback intent is to handle invocation phrases that don't provide the parameters you are looking for. For example, if instead of saying a color, the user might say something unexpected like "Talk to my test app about bananas". The term "bananas" would not fit into any of our Dialogflow intents, so we'd need to build a catch-all intent. This can be used in different flows you design.

Since the Assistant now listens for any phrases which match the ‘favorite color' intent, you should provide a custom fallback intent specific for catching anything else.

To set up your custom fallback intent, do the following:

1. In the Dialogflow console, click on **Intents** in the left-navigation, then click **Create Intent**.
2. Name your intent ‘Unrecognized Deep Link' or equivalent. This intent name won't be referenced in your webhook so you can call it whatever you like.
3. Under **Contexts**, click **Add input context** and type "google\_assistant\_welcome**"**. By specifying that this intent uses the ‘google\_assistant\_welcome' input context, it can only be invoked at the start of the conversation. After you've entered your input context, "google\_assistant\_welcome" will appear as an output context as well. Click the **x** to remove that output context.
4. Under **Training phrases**, add "banana" (or any other noun) as a user expression.
5. We'll use the @sys.any entity to tell Dialogflow to generalize the expression to any grammar (not just "banana"). Double-click on "banana" and filter for or select @sys.any
6. A warning message will pop up not to use the @sys.any entity. You can safely ignore this for now. Click **OK**.
7. Under **Responses**, add "Sorry, I am not sure about $any . This is reminder favorite color fallback?" as a **Text response**
8. Click **Save**.

### **Test your custom fallback intent**

1. Talk to my test app about banana

## Personalize your responses with helper intents

**helper intents:** Your fulfillment calls a helper intent when it wants the Assistant to handle part of a conversation. Helper intents return frequently-requested information such as the user's name and location (actions\_intent\_PERMISSION), a user selection (actions\_intent\_OPTION), or a date and time (actions\_intent\_DATETIME).

use the actions\_intent\_PERMISSION helper intent to obtain the user's display name, with their permission. To use the permission helper intent:

1. In the [Dialogflow console](https://console.dialogflow.com/" \t "_blank), navigate to **Intents**.
2. Select the **Default Welcome Intent.**
3. Under **Fulfillment**, turn on **Enable webhook call for this intent**. Note that the response from the webhook will override any response you typed into **Text responses** above.
4. Click **Save**.
5. Navigate back to **Intents.**
6. Click **Create Intent**.
7. Name your intent "actions\_intent\_PERMISSION".
8. Under **Events**, click **Add event** and type "actions\_intent\_PERMISSION".
9. Under **Fulfillment**, turn on **Enable webhook call for this intent**.
10. Click **Save**.

Index.js

app.intent('Default Welcome Intent', (conv) => {

conv.ask(new Permission({

context: 'Hi there, to get to know you better',

permissions: 'NAME'

}));

});

### **Customize responses with user information**

Next, you'll need to update your webhook to handle the response. You'll use the user's information in your response if they granted permission and gracefully move the conversation forward regardless if permission was not granted.

To respond to the user:

1. Add the following code to index.js:

app.intent('actions\_intent\_PERMISSION', (conv, params, permissionGranted) => {

if (!permissionGranted) {

conv.ask(`Ok, no worries. What's your favorite color?`);

conv.ask(new Suggestions('Blue', 'Red', 'Green'));

} else {

conv.data.userName = conv.user.name.display;

conv.ask(`Thanks, ${conv.data.userName}. What's your favorite color?`);

conv.ask(new Suggestions('Blue', 'Red', 'Green'));

}

});

call the Suggestions() function to create suggestion chips that recommend some example colors. If the user is on a device with a screen, they can provide their input by tapping on a chip rather than by saying or typing their response.

Conv.data used for in dialog storage

app.intent('favorite color', (conv, {color}) => {

const luckyNumber = color.length;

if (conv.data.userName) {

conv.close(`${conv.data.userName}, your lucky number is ${luckyNumber}.`);

} else {

conv.close(`Your lucky number is ${luckyNumber}.`);

}

});

Save file, firebase deploy

'use strict';

const {dialogflow, Permission, Suggestions,}=require('actions-on-google');

const functions=require('firebase-functions');

const app=dialogflow({debug:true});

app.intent('Default Welcome Intent', (conv)=>{

    conv.ask(new Permission({

        context: "Hi there, to get to know you better,", //google permission kicks in

        permissions: "NAME"

    }));

});

app.intent('actions\_intent\_PERMISSION', (conv, params, permissionGranted)=>{

    const audioSound = 'https://actions.google.com/sounds/v1/cartoon/clang\_and\_wobble.ogg';

    if(!permissionGranted){

        conv.ask('Ok, permission for your name not granted, How can I help you?');

        conv.ask(new Suggestions('Checking', 'Savings', 'Credit Card'));

    }else{

        conv.data.userName=conv.user.name.display;

        conv.ask(`<speak>${conv.data.userName}, So what is your favorite color or how may I help you? ` +

        `<audio src="${audioSound}"></audio></speak>`);

        conv.ask(new Suggestions('red', 'Savings', 'Credit Card'));

    }

});

app.intent('favorite color', (conv, {color})=>{

    const luckyNumber=color.length;

    const audioSound = 'https://actions.google.com/sounds/v1/cartoon/clang\_and\_wobble.ogg';

    if(conv.data.userName){

        conv.add(`<speak>${conv.data.userName}, your lucky number is ` +

     `${luckyNumber}.<audio src="${audioSound}"></audio></speak>`);

    }else{

        conv.add(`Your fave color is your lucky number is` + luckyNumber);

    }

});

exports.dialogflowFirebaseFulfillment =functions.https.onRequest(app);

Here, you declare an audioSound variable containing the string URL for a statically hosted audio file on the web. You use the <speak> SSML tags around the strings for the user response, indicating to the Google Assistant that your response should be parsed as SSML.

The <audio> tag embedded in the string indicates that you want the Assistant to play some audio played at that point in the response. The src attribute of that tag indicates where the audio is hosted.

1. Save your file.
2. In the terminal, run the following command to deploy your webhook to Firebase.

firebase deploy

test

 To add follow-up intents to ‘favorite color', do the following:

1. In the [Dialogflow console](https://console.dialogflow.com/" \t "_blank) left navigation, click on **Intents**.
2. Hover your cursor over **favorite color**, then click **Add follow-up intent**. Do this twice, once selecting **yes** and again selecting **no**.
3. nder **favorite color - no**, do the following:

* Under **Responses**, add "Goodbye, see you next time!" as a **Text response**.
* Turn on **Set this intent as end of conversation**.
* Click **Save**.

1. Under **favorite color - yes**, do the following:

* Under **Responses**, add "Which color, indigo taco, pink unicorn, or blue grey coffee?" as a **Text response**.
* Click **Save**.

create a custom entity:

1. In the [Dialogflow console](https://console.dialogflow.com/" \t "_blank) left-navigation, click on **Entities**.
2. Click **Create entity** and call it "fakeColor".
3. Add the following entries and synonyms, then click **Save**:

* *indigo taco / the first one*
* *pink unicorn / the second one*
* *blue grey coffee / the third one*

1. lick on **Intents**.
2. Click **Create intent** and call it "favorite fake color"
3. Under **Training phrases**, type:

* "indigo taco",
* "tell me about pink unicorn",
* "i want to know about blue grey coffee".

You should see the "fakeColor" parameter show up under **Actions and parameters** now that Dialogflow recognizes your custom entity.

1. Under **Fulfillment**, turn on **Enable webhook call for this intent**.
2. Click **Save**.

When a user selects one of the fake colors you've defined, your webhook will respond basic cards that show each color.

const {

dialogflow,

Permission,

Suggestions,

BasicCard,

} = require('actions-on-google');

and

app.intent('favorite color', (conv, {color}) => {

const luckyNumber = color.length;

const audioSound = 'https://actions.google.com/sounds/v1/cartoon/clang\_and\_wobble.ogg';

if (conv.data.userName) {

*// If we collected user name previously, address them by name and use SSML*

*// to embed an audio snippet in the response.*

conv.ask(`<speak>${conv.data.userName}, your lucky number is ` +

`${luckyNumber}.<audio src="${audioSound}"></audio> ` +

`Would you like to hear some fake colors?</speak>`);

conv.ask(new Suggestions('Yes', 'No'));

} else {

conv.ask(`<speak>Your lucky number is ${luckyNumber}.` +

`<audio src="${audioSound}"></audio> ` +

`Would you like to hear some fake colors?</speak>`);

conv.ask(new Suggestions('Yes', 'No'));

}

});

*// Define a mapping of fake color strings to basic card objects.*

const colorMap = {

'indigo taco': {

title: 'Indigo Taco',

text: 'Indigo Taco is a subtle bluish tone.',

image: {

url: 'https://storage.googleapis.com/material-design/publish/material\_v\_12/assets/0BxFyKV4eeNjDN1JRbF9ZMHZsa1k/style-color-uiapplication-palette1.png',

accessibilityText: 'Indigo Taco Color',

},

display: 'WHITE',

},

'pink unicorn': {

title: 'Pink Unicorn',

text: 'Pink Unicorn is an imaginative reddish hue.',

image: {

url: 'https://storage.googleapis.com/material-design/publish/material\_v\_12/assets/0BxFyKV4eeNjDbFVfTXpoaEE5Vzg/style-color-uiapplication-palette2.png',

accessibilityText: 'Pink Unicorn Color',

},

display: 'WHITE',

},

'blue grey coffee': {

title: 'Blue Grey Coffee',

text: 'Calling out to rainy days, Blue Grey Coffee brings to mind your favorite coffee shop.',

image: {

url: 'https://storage.googleapis.com/material-design/publish/material\_v\_12/assets/0BxFyKV4eeNjDZUdpeURtaTUwLUk/style-color-colorsystem-gray-secondary-161116.png',

accessibilityText: 'Blue Grey Coffee Color',

},

display: 'WHITE',

},

};

*// Handle the Dialogflow intent named 'favorite fake color'.*

*// The intent collects a parameter named 'fakeColor'.*

app.intent('favorite fake color', (conv, {fakeColor}) => {

*// Present user with the corresponding basic card and end the conversation.*

conv.close(`Here's the color`, new BasicCard(colorMap[fakeColor]));

});

This new code performs two main tasks:

First, it sets up a mapping (colorMap) of color strings (e.g. "indigo taco", "pink unicorn", "blue grey coffee") to the content needed for BasicCard objects. BasicCard is a client library class for constructing visual responses

Firebase deploy test

1. pe "Talk to my test app" into the **Input** field and hit enter.
2. Type "yes".
3. Type "blue".
4. Type "sure".
5. Type "tell me about the first one".

Store data between conversations

app.intent('Default Welcome Intent', (conv) => {

const name = conv.user.storage.userName;

if (!name) {

*// Asks the user's permission to know their name, for personalization.*

conv.ask(new Permission({

context: 'Hi there, to get to know you better',

permissions: 'NAME',

}));

} else {

conv.ask(`Hi again, ${name}. What's your favorite color?`);

}

});

This is for returning customers

 If you want to clear the user storage during testing, you can do so by adding the following code to the first line of the function callback for the Default Welcome Intent:

conv.user.storage = {};

Next

* **No-input event:** Special event that Actions on Google sends to Dialogflow whenever a user doesn't provide input after a developer-defined number of reprompts. In Actions on Google, this is represented by the actions.intent.NO\_INPUT intent. If you are using Dialogflow, this is represented as an actions\_intent\_NO\_INPUT event.

### **Set up Dialogflow**

1. Set up a new intent to handle the no-input event. In the Dialogflow console, click the **+** button next to **Intents**in the left navigation to create a new intent.
2. You can name this new intent whatever you'd like. In our example, we've named it **actions\_intent\_NO\_INPUT**.
3. Under **Events**, add a new event called "actions\_intent\_NO\_INPUT".
4. Toggle on the webhook fulfillment switch and click **Save**
5. app.intent('actions\_intent\_NO\_INPUT', (conv) => {
6. *// Use the number of reprompts to vary response*
7. const repromptCount = parseInt(conv.arguments.get('REPROMPT\_COUNT'));
8. if (repromptCount === 0) {
9. conv.ask('Which color would you like to hear about?');
10. } else if (repromptCount === 1) {
11. conv.ask(`Please say the name of a color.`);
12. } else if (conv.arguments.get('IS\_FINAL\_REPROMPT')) {
13. conv.close(`Sorry we're having trouble. Let's ` +
14. `try this again later. Goodbye.`);
15. }
16. });

Keep going

## Add a custom exit

Your Action should allow users to quickly bow out of conversations, even if they haven't followed the conversation path all the way through. By default, Actions on Google exits the conversation and plays an earcon whenever the user utters one of these keywords: "exit", "cancel", "stop", "nevermind", and "goodbye".

You can customize this behavior by registering for the actions\_intent\_CANCEL event in Dialogflow and defining a custom response.

**Key terms:**

* **Cancel intent:** An intent that allows your Action to perform some code cleanup in fulfillment and send users a final response. In Dialogflow, this is triggered by the actions\_intent\_CANCEL event.
* **Final response:** The final message from your Action before it ends the conversation. Your final response must be a single [simple response](https://developers.google.com/assistant/conversational/responses#simple_responses) with a 60-character limit on its textToSpeech and displayText values.

In this section, you'll create a new cancel intent in Dialogflow and add a suitable final response message.

### **Set up Dialogflow**

1. Set up a new intent for handling the user exiting. In the Dialogflow console, click the **+** button next to **Intents**in the left navigation to create a new intent.
2. You can name this new intent whatever you'd like. In our example, we've named it actions\_intent\_CANCEL.
3. Under **Events**, add a new event called actions\_intent\_CANCEL.
4. Under **Responses**, add a **Text response** like "Let me know when you want to talk about colors again!".
5. Toggle on the **Set this intent as end of conversation** switch below **Add Responses**.
6. Click **Save.**
7. Test

### **Design the conversational experience**

It's important to design conversational experiences to be multimodal, which means that users can participate via voice and text as well as other interaction modes that their devices support (for example, touchscreen).

We always start with designing the conversation and writing sample dialogs for the voice-only experience. Then, we [design the multi-modal experience](https://designguidelines.withgoogle.com/conversation/conversation-design-process/scale-your-design.html), which involves adding visuals as enhancements where it makes sense.

1. click on **Intents**.
2. Click on the arrow next to the favorite colorintent and select favorite color - yes.
3. At the bottom of the page, under the **Fulfillment** section, toggle the **Enable webhook call for this intent** option
4. click on **Intents** and select the favorite fake color intent.
5. Under **Events**, add actions\_intent\_OPTION**.**Dialogflow will look for this specific event when a user selects an option from the carousel.

Load dependencies

const {

dialogflow,

BasicCard,

Permission,

Suggestions,

Carousel,

Image,

} = require('actions-on-google');

#### **Build the carousel**

Next, you'll define the fakeColorCarousel() function to build the carousel.

 In the index.js file, add a fakeColorCarousel() function with the following code:

### **index.js**

*// In the case the user is interacting with the Action on a screened device*

*// The Fake Color Carousel will display a carousel of color cards*

const fakeColorCarousel = () => {

const carousel = new Carousel({

items: {

'indigo taco': {

title: 'Indigo Taco',

synonyms: ['indigo', 'taco'],

image: new Image({

url: 'https://storage.googleapis.com/material-design/publish/material\_v\_12/assets/0BxFyKV4eeNjDN1JRbF9ZMHZsa1k/style-color-uiapplication-palette1.png',

alt: 'Indigo Taco Color',

}),

},

'pink unicorn': {

title: 'Pink Unicorn',

synonyms: ['pink', 'unicorn'],

image: new Image({

url: 'https://storage.googleapis.com/material-design/publish/material\_v\_12/assets/0BxFyKV4eeNjDbFVfTXpoaEE5Vzg/style-color-uiapplication-palette2.png',

alt: 'Pink Unicorn Color',

}),

},

'blue grey coffee': {

title: 'Blue Grey Coffee',

synonyms: ['blue', 'grey', 'coffee'],

image: new Image({

url: 'https://storage.googleapis.com/material-design/publish/material\_v\_12/assets/0BxFyKV4eeNjDZUdpeURtaTUwLUk/style-color-colorsystem-gray-secondary-161116.png',

alt: 'Blue Grey Coffee Color',

}),

},

}});

return carousel;

};

Notice that carousels are built using the Items object, which has several properties, including titles and Images. The Image type contains a url which opens when the user clicks on the selection, as well as alternative text for accessibility.

To identify which carousel card the user selected, you'll use the keys of the Items object; namely, "indigo taco", "pink unicorn", or "blue grey coffee".

#### **Add the intent handler for ‘favorite fake color - yes'**

Next, you'll need to add a handler for the favorite color - yes follow-up intent to check if the conv.screen property is true. If so, this indicates that the device has a screen. You can then send a response asking the user to select a fake color from the carousel by calling the ask() function with fakeColorCarousel(), which you pass as the argument.

 In the index.js file, add a check for a screen on the current surface by adding the following code to your fulfillment:

### **index.js**

*// Handle the Dialogflow intent named 'favorite color - yes'*

app.intent('favorite color - yes', (conv) => {

conv.ask('Which color, indigo taco, pink unicorn or blue grey coffee?');

*// If the user is using a screened device, display the carousel*

if (conv.screen) return conv.ask(fakeColorCarousel());

});

#### **Support non-screened devices**

If the surface capability check returned false, your user is interacting with your Action on a device that doesn't have a screen. You should support as many different users as possible with your Action, so you're now going to add an alternate response that reads the color's description instead of displaying a visual element.

 In the index.js file, add a screen capability check and fallback by replacing the following code:

### **index.js**

*// Handle the Dialogflow intent named 'favorite fake color'.*

*// The intent collects a parameter named 'fakeColor'.*

app.intent('favorite fake color', (conv, {fakeColor}) => {

*// Present user with the corresponding basic card and end the conversation.*

conv.close(`Here's the color`, new BasicCard(colorMap[fakeColor]));

});

with this:

### **index.js**

*// Handle the Dialogflow intent named 'favorite fake color'.*

*// The intent collects a parameter named 'fakeColor'.*

app.intent('favorite fake color', (conv, {fakeColor}) => {

fakeColor = conv.arguments.get('OPTION') || fakeColor;

*// Present user with the corresponding basic card and end the conversation.*

conv.ask(`Here's the color.`, new BasicCard(colorMap[fakeColor]));

if (!conv.screen) {

conv.ask(colorMap[fakeColor].text);

}

});

### **Test your carousel response**

In the terminal, run the following command to deploy your updated webhook code to Firebase:

firebase deploy

1. alk to my test app" into the **Input** field and press enter. If your Action doesn't remember your name, type "yes" and press enter.
2. Type "blue" and press enter.
3. Type yes and enter

Try speaker and smart screen

click on **Intents**.

1. Hover your cursor over **favorite fake color**, then click **Add follow-up intent**. Do this twice, once selecting **yes** and again selecting **no**.

**favorite fake color - no**intent, and do the following:

1. Under **Responses**, add "Goodbye, see you next time!" as a **Text response**.
2. Turn on **Set this intent as end of conversation**.
3. Click **Save**.

 click on the **favorite fake color - yes** intent. Then, do the following:

1. Under **Fulfillment**, turn on **Enable webhook call for this intent**.
2. Click **Save**.

### **Implement fulfillment**

Next, you'll need to add a handler for the favorite fake color - yes follow-up intent.

 In the index.js

app.intent('favorite color - yes', (conv) => {

conv.ask('Which color, indigo taco, pink unicorn or blue grey coffee?');

*// If the user is using a screened device, display the carousel*

if (conv.screen) return conv.ask(fakeColorCarousel());

});

with this:

### **index.js**

*// Handle the Dialogflow follow-up intents*

app.intent(['favorite color - yes', 'favorite fake color - yes'], (conv) => {

conv.ask('Which color, indigo taco, pink unicorn or blue grey coffee?');

*// If the user is using a screened device, display the carousel*

if (conv.screen) return conv.ask(fakeColorCarousel());

});

Lastly, you'll add suggestion chips to the favorite fake color intent handler that trigger your two new follow-up intents.

 In the index.js file, update the favorite fake color intent handler with suggestion chips by replacing the following code:

app.intent('favorite fake color', (conv, {fakeColor}) => {

fakeColor = conv.arguments.get('OPTION') || fakeColor;

*// Present user with the corresponding basic card and end the conversation.*

if (!conv.screen) {

conv.ask(colorMap[fakeColor].text);

} else {

conv.ask(`Here you go.`, new BasicCard(colorMap[fakeColor]));

}

conv.ask('Do you want to hear about another fake color?');

conv.ask(new Suggestions('Yes', 'No'));

});

Firebase deploy test, end

Telephony Transfer example:

Dialogflow Integration Telephone Transfer Example

Intent: Default Welcome Intent

Events: add Telephony Welcome

Training Phrases: Hi, hello

Action : input.welcome

Responses: Click + and Add Telephony

Play Audio: gs://dialogflow-v2/welcome.raw

SythesizeSpeech: <speak> Please begin by saying <break time="0.5s" /> test.</speak>

Intent: Test Telephony Intent

Training Phrases: Test, Test agent

Responses: + Add Telephony

Synthesize Speech: Thanks for calling would you like to transfer or terminate call?

Follow up Intent: Test Telephony Intent – Transfer

Training Phrases: Transfer, transfer call

Action: TestTelephonyIntent.TestTelephonyIntent-Transfer

Responses: + Add Telephony

SynthesizeSpeech: <speak>Transferring to 617</speak>

TransferCall: +1 6175044426

Follow Up Intent 2: Test Telephony Intent – Terminate

TrainingPhrases: Terminate Call, terminate

Action: TestTelephonyIntent.TestTelephonyIntent-Terminate

Responses: + Add Telephony

SynthesizeSpeech: Goodbye.

Set this intent as end of conversation, on.

end