Chatbots that work

Conversational AI

Andrew R. Freed

MANNING

MEAP Edition
Manning Early Access Program

Conversational AI

Chatbots that work
Version 5

welcome

Dear reader,

Thank you for purchasing the MEAP for *Conversational AI*.

I've been privileged enough to read a lot of useful content in my career as a software engineer. In many ways I feel that I'm standing on the shoulders of giants. It's important to me to give back by producing content of my own, especially through this book. I hope you step on my shoulders as you learn from this book!

Before you start this book, you should be comfortable in reading and writing decision trees and process flows. You should be comfortable with branching control logic in the form of "if statements".

Virtual assistants use machine learning under the covers; but you do not need a PhD in mathematics to understand this book. I take great pains to make machine learning approachable.

It takes a dream team to build an effective virtual assistant. In this book you'll learn why it takes several different players to build a virtual assistant and you'll see what each member of the team needs to do. Even if you work in a "silo", you'll get a greater appreciation for what your teammates do.

I've had a lot of fun building virtual assistants in my career and I'm excited to share my experience with you. I encourage you to post any feedback or questions (good or bad!) in the liveBook Discussion forum. Your feedback will help me write the best possible book for you!

Andrew R. Freed

brief contents

2 Building your first virtual assistant

This chapter covers:

- **Identifying the intent and entity in a single user utterance**
- **Implementing a question-and-answer dialog for a recognized intent. Then, add contextual information to that answer when an entity is recognized.**
- **Implementing a multiple-question process flow to satisfy a user request.**

In the previous chapter, we learned about what virtual assistants are, why people use them, and some examples of virtual assistants that we encounter every day. In this chapter, you will learn how they work, and you will build an assistant from the ground up. We will work through a case study for a retail company looking to build their first virtual assistant.

The first thing assistants need to do is recognize what user's utterances actually mean. A conversation starts with a user stating an intent and any related entities. The intent will help the assistant understand what to do next – whether to respond with an answer or to start a new dialog flow. We will first see in detail how the assistant breaks down a user utterance into intents and entities.

The simplest response is for the assistant to respond to each intent with a single "answer" response. We will configure an assistant to recognize a variety of different intents and to give a unique response for each. We will further explore how to provide additional information in a response based on the entities detected in the user's utterance.

Some user intents require a multi-step process flow that includes one or more follow-up questions. We will also explore methods for building these process flows in a virtual assistant. We will also implement a "two strikes" rule so that if the assistant consistently doesn't understand the user, the user will be given a different resolution option.

By the end of this chapter, you will be able to build your first virtual assistant. You will be able to implement simple question and answer responses. You will be able to tailor a response based on contextual details in the user's utterance. You will also be able to implement multi-step process flows that guide a user through a series of steps, ending with a resolution for the user. Armed with this knowledge you will be ready to design the process flows for your next virtual assistant.

2.1 Building a virtual assistant for FICTITIOUS INC

FICTITIOUS INC is a growing retail chain looking to reduce the burden on their support representatives. They want to build a self-service virtual assistant to handle their most frequent customer questions. They primarily receive questions on store locations and store hours, scheduling appointments, and job applications. In this chapter, we will start building their assistant.

FICTITIOUS INC's use case is a typical retail scenario. Customer service departments are often overloaded with routine inquiries that can easily be automated. FICTITIOUS INC will offer a virtual assistant to their users as a fast way to get answers, without needing to wait for a human to answer questions. When their virtual assistant cannot answer the user's questions, users will be redirected to the existing human-based customer service processes.

FICTITIOUS INC prefers to leverage pre-trained content to reduce their virtual assistant development time, though this is not a hard requirement. They have made it clear that they will use any technology that works, giving you free range to pick the virtual assistant platform of your choice. They only require that the platform includes a "try before you buy" option. In this chapter, I will demonstrate the FICTITIOUS INC scenario using IBM Watson Assistant.

2.1.1 Creating the assistant

FICTITIOUS INC first needs to get some virtual assistant software. For this demo I registered an account on the IBM Cloud[1], and created a personal (free) instance of Watson Assistant by importing it from the IBM Cloud Catalog[2]. I'm asked to configure a service instance name (any name will do) and select a geographic region (the closest to your users, the best).

If you are using a different cloud-based platform you will encounter a similar workflow. You generally need to sign up (possibly with a credit card) and can find a free/lite/trial option. If you are not using a cloud-based platform you will need to download and install the virtual assistant software and any dependencies it requires (be sure to check the documentation).

After this virtual assistant software is configured for your use you need to create a specific virtual assistant that uses this software. Figure 1 demonstrates the relationship between the former and latter steps.

[1]https://cloud.ibm.com
[2]https://cloud.ibm.com/catalog

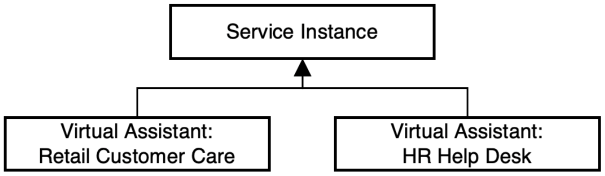

Service Instance
Virtual Assistant: Retail Customer Care
Virtual Assistant: HR Help Desk

Figure 1 After configuring the Service Instance (or synonymous item) for your platform, you'll need to create a virtual assistant for your specific use case. You can create separate virtual assistants using the same service instance.

The next step is to create a virtual assistant specific to your use case. The specific way you do this varies greatly depending on your platform of choice.

Terminology alert!

You want to create a virtual assistant, but what does your platform call it? They may call it an Assistant, a Bot, an Agent, or something else entirely! Sometimes the Assistant/Bot/Agent has everything you need and sometimes you couple it with a Skill, another service, or raw code.

The different naming schemes are sometimes just cosmetic and sometimes indicative of different architectural patterns. Many platforms use a layered architecture allowing you to connect to a variety of front-end user interfaces and back-end service integrations. For the FICTITIOUS INC demo, we will not worry about integrating with any other systems.

The one exception to this rule is if your virtual assistant platform separates the "virtual assistant" service from the classifier or "natural language understanding" service, you will need to integrate them to build your assistant. Check your platform's documentation when we get to the intent training section.

Since I am working with Watson Assistant I have two more components to create. (Your platform will likely use different terminology. Use Figure 2 to figure out what your virtual assistant platform requires.) The smallest possible configuration in Watson Assistant's v2 API uses these two parts:

1. **Assistant**[3] – The interface integration layer.
2. **Skill**[4] - The dialog engine.

[3]https://cloud.ibm.com/docs/assistant?topic=assistant-assistants
[4]https://cloud.ibm.com/docs/assistant?topic=assistant-skill-dialog-add

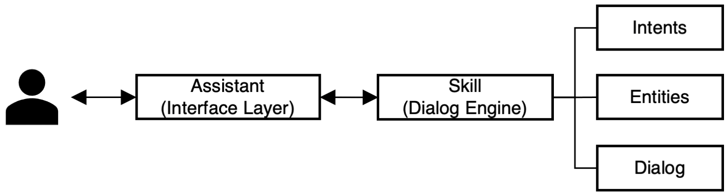
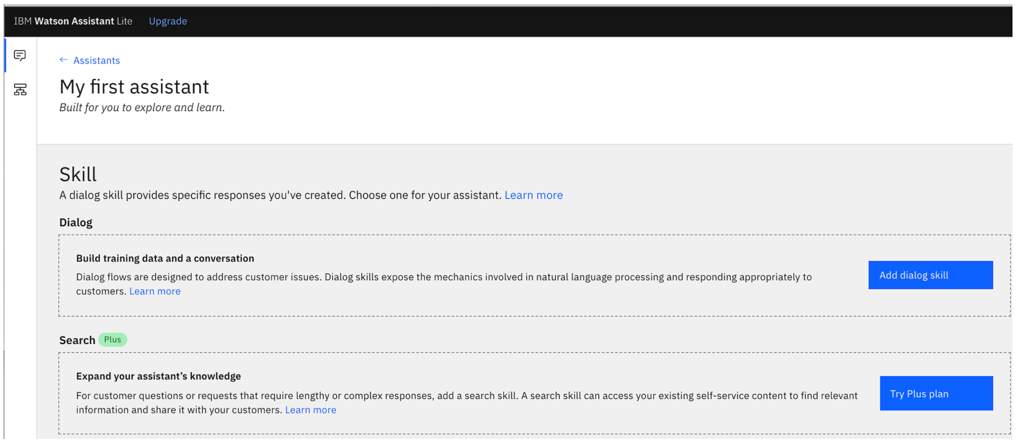


Figure 2 The Watson Assistant components used for FICTITIOUS INC. The terms Assistant and Skill may vary in your platform, but you can expect to find intents, entities, and dialog.

When I created a new Watson Assistant instance it came pre-loaded with an Assistant called "My first assistant". There was no other default configuration provided with the assistant, so I added a Dialog Skill to implement FICTITIOUS INC'S scenario. The "Add dialog skill" button creates a new, empty Dialog Skill as shown in Figure 3.



My first assistant

Built for you to explore and learn.

Skill

A dialog skill provides specific responses you've created. Choose one for your assistant. Learn more

Dialog

Build training data and a conversation

Dialog flows are designed to address customer issues. Dialog skills expose the mechanics involved in natural language processing and responding appropriately to customers. Learn more

Search Plus

Expand your assistant's knowledge

For customer questions or requests that require lengthy or complex responses, add a search skill. A search skill can access your existing self-service content to find relevant information and share it with your customers. Learn more

Figure 3 Adding a dialog skill to the assistant. Your platform likely has choices that differ in name but are similar in function.

When creating a Skill, you will be prompted to provide a name, description, and language. (Your platform will collect similar data.) Be sure to provide a useful name. The name and description are made available for your benefit. I created the FICTITIOUS INC Skill with the name "Customer Service Skill", description "Answer most common customer service questions", and the English language. After clicking "Create dialog skill" the Skill is linked to the assistant.

If you are using another virtual assistant platform you will have similar options. You may be prompted to pick a specific language that the assistant will converse in. Some of the virtual assistant platforms I surveyed allow you to create an assistant based on a sample provided by the platform. For following along in this chapter, I suggest creating a new, empty assistant.

With the basic virtual assistant infrastructure (Assistant and Skill in Watson Assistant, a Bot or Assistant or Agent in other platforms) in place, it's time to start implementing FICTITIOUS INC's use case. We configure three specific components inside the virtual assistant:

- Intents – this will show us all the training data made available to the assistant. This will be empty until we manually provide some intents or import them from the content catalog.
- Entities – we will use this to identify several FICTITIOUS INC store locations
- Dialog – this is where we will encode the dialog flows used in FICTITIOUS INC's assistant.

Our new assistant is almost completely empty. Let's start by adding some intents, so that the assistant can understand what the user means.

2.2 What's the user's intent?

A conversation in a virtual assistant generally starts with the following pattern:

1. The user speaks/types something. This is called the utterance.
2. The assistant determines what the utterance means, or what the user is trying to accomplish. This is called the intent.
3. The assistant formulates a response and sends it to the user. This response may be an answer or a request for additional information.

The pattern is displayed in Figure 4.

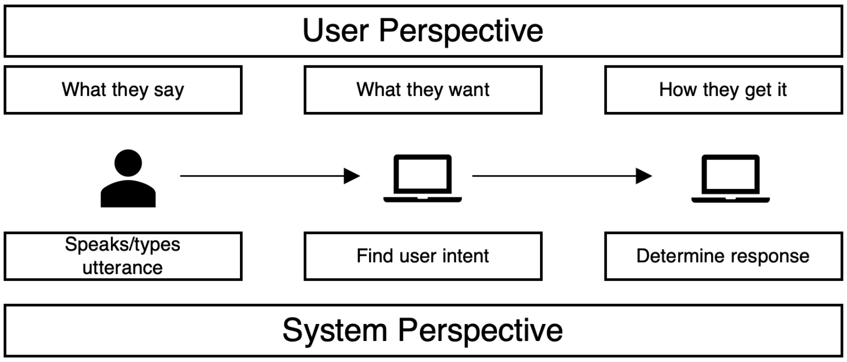


Figure 4 One turn of a conversation, from the user and system perspectives.

Terminology alert!

Every virtual assistant platform I surveyed at the time of this writing used the term "intent" in the same way. Hooray for consistency! Some platforms are starting to hide the "intent" terminology (are they afraid it scares users?), even though they use intents under the covers.

FICTITIOUS INC's users will use a nearly infinite number of utterances and these will map to a finite set of intents. For FICTITIOUS INC's assistant, we will need to create a set of intents and map dialog responses to those intents. Let's explore each of these more deeply.

2.2.1 What's an utterance?

The user primarily interacts with assistants through natural language. The user's input is called an "utterance" and the utterance is literally just what the user says (in a telephone assistant) or types (in a web/chat assistant).

Terminology alert!

Nearly every virtual assistant platform I surveyed used the term "utterance". A few platforms referred to them as "examples" or "training phrases". Search your platform's documentation for the section on training intents and you'll quickly find how they refer to "utterances".

Example utterances include:

- "Where are you located?"
- "What time do you open?"
- "How can I reset my password?"
- "Tell me a joke"

Can users provide input to a virtual assistant besides text?

Yes! There are other ways for users to provide input. In a web channel, assistants the user may also provide input by clicking on a button. In a voice channel, assistants the user may enter numeric values on their keyboard with Dual-Tone Multi-Frequency (DTMF) signals. Virtual assistants still generally treat this input as textual.

2.2.2 What's a response?

The end goal for a virtual assistant is to respond appropriately to a user's utterance. A *response* is simply whatever the assistant returns to the user. A set of example utterances that FICTITIOUS INC's users might say, along with likely FICTITIOUS INC responses, are shown in Table 1.

Table 1 Example dialog utterances and responses

Example user utterance	Example system response
“Where are you located?”	“We have two locations near you: 123 Maple St and 456 Elm Drive.”
"What time do you open?"	"Our hours are 9am to 8pm on weekdays and 10am to 10pm on weekends."
"How can I reset my password?"	"What's your user ID?"
"Tell me a joke"	"I'm not trained to handle that. I can help you with several other common IT requests like password resets."

Terminology alert!

Nearly every virtual assistant platform I surveyed used the term “response”. I also saw “dialog response”, “action”, “message”, “prompt”, and “statement”.

If you consider the paradigm “if this, then that”, the “that” is the response.

Table 1 shows what the user sees, but internally the assistant does a little more work to determine that response. The assistant uses natural language understanding (NLU) to extract an intent from the user’s utterance and uses that to further drive the conversation. Let’s look at how an NLU module does that.

Can virtual assistants respond with output other than text?

Yes! There are multiple ways to provide output in a virtual assistant. Web assistants can use buttons, images, hyperlinks, and other rich media in their responses. Voice assistants can add music or other audio in a response.

2.2.3 How does the assistant understand what the user means?

The assistant makes use of a natural language understanding engine that finds meaning in the user's utterance. The natural language understanding component has two parts:

- Intent classifier: Finds meaning that applies to the entire utterance. The meaning is generally expressed as a (verb-based) intent and is used extracted via machine learning.
- Entity extractor: Finds zero or more subsets of the utterance with a specific meaning, such as dates, locations, or other objects. The entities are usually noun-based and may be extracted by machine learning, hardcoded rules, or both.

The natural language understanding component in your platform may offer separate training options for the classifier and the entity extractor. Table 2 summarizes when you use intents and entities.

Table 2 Key differentiation between intents and entities. Both are extracted via a natural language understanding component.

Intent	Entity
Verb-based	Noun-based
Considers the whole utterance	Extracted from a word or segment in the utterance
Usually uses machine learning	Uses rules and/or machine learning

WHY MACHINE LEARNING?

A classifier uses machine learning because learning to understand text by example is a more scalable approach than implementing rules. Utterances come in natural language which has nearly infinite variations. It is difficult to code enough rules to handle all possible variations in text, but a machine learning classifier can scale quickly. Table 3 shows several possible ways that a user may state they need their password reset.

Table 3 Example utterances from a user whose intent is to reset their password. Set A is handled easily with rules, but Set B is more easily handled with a machine learning classifier.

Set A	Set B
• Reset my password • Please help me reset my account password • My password needs to be reset	• I forgot my password • Reset my login information • I can't sign into the intranet • I'm locked out of my email account

A first and naive approach to intent detection would be to use keywords to apply structure to unstructured text. Table 3's Set A is easily handled with a rule: any utterance containing "reset" and "password" should be treated as a password reset request. Table 3's Set B shows why a *rules-based* approach grows unwieldy quickly. There are nearly infinite ways to express most intents in natural language and it is too hard to write rules to cover them all.

The classifier uses machine learning techniques at its heart. We'll see how the classifier learns to identify a user's request later, so for now we can assume we have one trained and ready to use. The classifier attempts to derive intent from every input phrase and passes this intent to the dialog engine. The dialog engine is then able to use structured rules to determine the next dialog step.

WHAT'S AN INTENT?

An intent is a normalization of what the user means by their utterance, or what they want to do. An intent is generally centered around a key verb. Note this verb may not be explicitly included in the utterance! By convention, I will reference intents with their name preceded by a pound (#) sign, such as `#reset_password`. This is the convention used by most of the popular assistant platforms. Table 4 shows sample utterances from FICTITIOUS INC's users along with the identified intent, and the verb(s) at the center of that intent.

Table 4 Sample utterances with associated intent and implied verb

Utterance	Intent	Implied Verb(s) of the Intent
"Where are you located?"	`#Store_Location`	`Locate`
"What time do you open?"	`#Store_Hours`	`Operate/Open/Close`
"How can I reset my password?"	`#Reset_password`	`Reset/Unlock`
"Tell me a joke"	`#Chitchat`	`n/a`

An intent represents a logical grouping of similar utterances that all have the same user meaning, i.e. user intent. As in the previous section "I need to reset my password" and "I'm locked out of my account" both indicate locking/resetting problems. A classifier is more powerful than a simple keyword matcher and uses contextual clues in the natural language to derive the user's intent.

Classifiers are trained with a series of examples and their corresponding intents.

It is possible that a user utterance will not map to any of the intents the classifier is trained on. In this case, the classifier will respond with some indication that the utterance is poorly understood, either by not returning an intent or returning an error condition. In Watson Assistant's development interface this scenario is flagged as `#Irrelevant` and in the API no intents are returned. Your virtual assistant platform of choice will do something similar when a user utterance is not well understood.

A classifier needs careful training for optimal performance. This training will be covered more thoroughly in the following chapters. For now, just consider that intents are trained with a variety of example utterances with similar meaning and that intents should be clearly differentiated from one another. In the end, the classifier will evaluate the user utterance for an intent, and send that intent back to the dialog engine.

WHAT'S AN ENTITY?

Entities are noun-based terms or phrases. An intent applies to the entire user utterance, but an entity applies to part of the utterance, often just a single noun or noun phrase. By convention virtual assistant platforms reference entities with an "at" (@) sign preceding their name. Table 5 shows some example entities FICTITIOUS INC can use in their assistant.

Table 5 Example entities for FICTITIOUS INC.

Entity Type	Entity Value (canonical form)	Entity synonyms (surface forms)
`@store`	Elm	Elm Dr, Elm Drive
`@store`	Maple	Maple St, Maple Street
`@contact_method`	phone	mobile, SMS, smartphone
`@contact_method`	email	webmail

The simplest entity pattern is to use a data structure called a dictionary - an enumerated list of all possible words and phrases you want to treat the same way.

Terminology alert!

Most virtual assistant platforms use the term “entity”. Occasionally I saw “parameter” or “slot type”.

Entities are typically defined in one of three ways:

1. A pre-defined, static list (sometimes called a “dictionary” or “lookup table”)
2. Regular expression patterns
3. Trained by example

Virtual assistants often come with built-in “system entities” to detect things like dates and countries.

If all of the entity variations are known ahead of time the dictionary-based approach works very well. FICTITIOUS INC’s virtual assistant will use dictionary-based entities for their customer service use case.

What’s a dictionary?

The traditional usage of a dictionary is a book, or another repository, with a list of words and information about those words: definition, pronunciation, part of speech, and sometimes other features like synonyms. In this context, a synonym for a given word is meant to be a similar but different word.

In the context of virtual assistants, a dictionary is simply a list of known words, each word having an associated type. Synonyms in this kind of dictionary are completely interchangeable.

In Table 5 `@store` was a type. “Elm”, “Elm Drive”, and “Elm Dr” can all treated interchangeably. Many virtual assistants have spelling correction, or a feature like “fuzzy matching”, which allows misspellings like “Eml Drvie” to be treated as if they too were in the dictionary.

Combining intents and entities

Virtual assistant platforms use entities to provide additional contextual information, besides just the intent, that can be used to provide a response. When utterances share an intent but differ only by the nouns, entities are the best way to determine a response. Table 6 contrasts two assistants, one that does not use entities and another that does.

Table 6 Contrasting two virtual assistants, one that does not use entities and one that does. The assistant without entities will be much harder to train.

Utterance	Without entities	With entities
Where are your stores?	`#store_location_all`	`#store_location`
Where is your Elm store?	`#store_location_elm`	`#store_location, @store:Elm`
Where is your Maple store?	`#store_location_maple`	`#store_location, @store:Maple`

You might try to treat "Where is your Elm store" and "Where is your Maple store" as different intents. But machine learning has difficulty distinguishing multiple intents that use nearly identical utterances. We will explore the reasoning in Chapter 7. For now, we can assume that using entities will help us build a more accurate virtual assistant.

Use entities when multiple questions have the same intent but need different responses.

2.2.4 Adding intents to FICTITIOUS INC's assistant

FICTITIOUS INC's new assistant is almost completely empty. Critically, it does not include any intents, therefore it will not understand any user responses. If you are using Watson Assistant you can import intents from the Watson Assistant Content Catalog[5] – this will provide some initial training data for the general customer service intents FICTITIOUS INC is interested in.

Not using Watson Assistant? No problem!

If your platform does not have sample intents for a retail scenario, you can create your own training data. In this chapter, we will use the following intents, and you can create a few example utterances for each such as noted in this sidebar. Add a few utterances of your own to increase the training quality.

`Customer_Care_Store_Location`: "Where are you" and "How do I find your store"

`Customer_Care_Store_Hours`: "What times are you open" and "When do you close tonight"

`Customer_Care_Employment_Inquiry`: "I need a job application" and "Can I work for you"

`Customer_Care_Appointments`: "I need to schedule an appointment" and "When can I see a manager"

After clicking the Content Catalog menu item, I selected the Customer Care category and clicked "Add to skill". The assistant is now populated with 18 different customer service intents, and each intent has 20 different examples. This is a robust set of training data to start FICTITIOUS INC's assistant. For example, the assistant will be able to recognize store location questions like "Where is your store?" via `#Customer_Care_Store_Location` intent.

As soon as the Customer Care content is loaded into the assistant you need to train the natural language understanding component. Your platform may train automatically (Watson

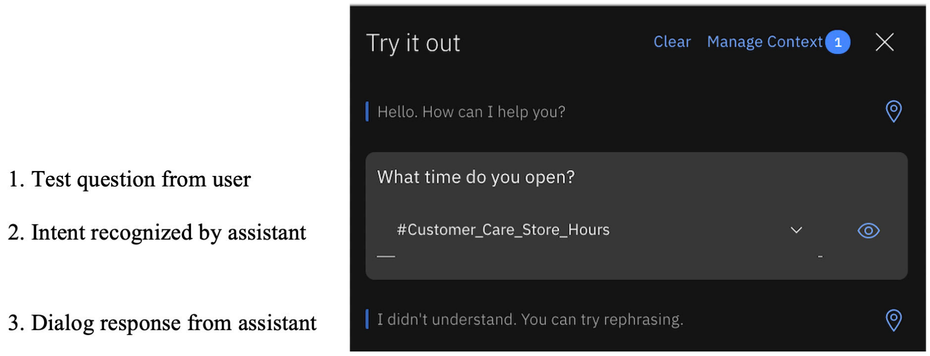
[5]https://cloud.ibm.com/docs/assistant?topic=assistant-catalog. Your virtual assistant platform may also offer sample training data; check your documentation.

Assistant is one such platform) or you may need to explicitly invoke a training action (check your platform's documentation). Regardless of platform, the training process should only take a few moments.

Virtual assistant platforms generally include a testing interface. In Watson Assistant the interface is called "Try it out". I've also seen the test interface referred to as a simulator or developer console. Some platforms do not provide a graphical testing interface but expose APIs you can use for testing. The testing interface will usually include information that helps you debug your assistant but that you won't expose to users (such as the name and confidence of any intents found). After your assistant is finished training, you can explore how well it recognizes intents by using your platform's testing interface.

Test your assistant with questions that were not included in your training data. (Use your imagination!) I asked my assistant "What time do you open?" and "Where is the nearest store?". As shown in Figure 5, the assistant recognizes the intent behind these statements (`#Customer_Care_Store_Hours` and `#Customer_Care_Store_Location`, respectively) but responds to the user as if the assistant didn't actually understand.

Testing tip: See how your assistant responds to phrases you did not explicitly train on!


1. Test question from user
2. Intent recognized by assistant
3. Dialog response from assistant
Try it out
Clear
Manage Context 1

Hello. How can I help you?

What time do you open?

#Customer_Care_Store_Hours

I didn't understand. You can try rephrasing.

Figure 5 A testing interface shows some information that you won't expose to users (like the name of the intent). Here I can see that the new assistant recognizes my intent. I can now infer that the "I didn't understand" response is because no dialog condition is associated with that intent.

The assistant understands several different intents but does not know how to respond to any of them. Figure 6 summarizes what our assistant can (and can't!) do.

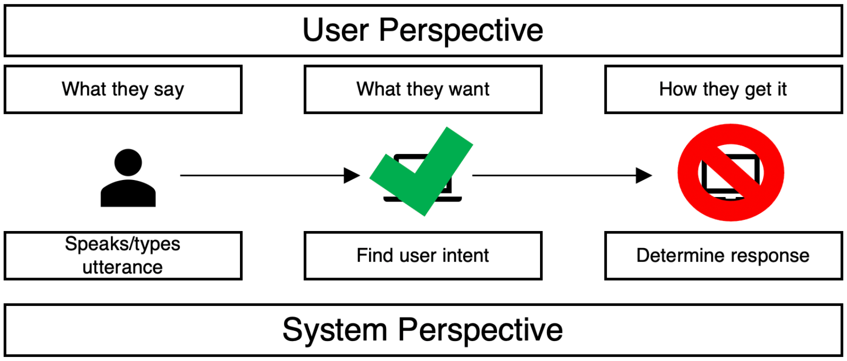


Figure 6 When you add intent training data, but do not configure dialog responses, you are here.

We've made nice progress but we're only halfway there. Let's teach the assistant how to respond to the user in a meaningful way.

Why are intents and responses separated?

In many virtual assistant platforms finding the intent is only one part of generating a response. Training an intent classifier and building dialog rules to generate responses are two separate skills.

Nonetheless, there is a growing trend among virtual assistant providers to combine these steps. You may be using a platform that abstracts away intents. The training interface may ask you to just provide the "this" (utterance) and "that" (response) in an "if *this*, then *that*" pattern.

2.3 Responding to the user

Just like a brand-new assistant needs to be loaded with intents, the assistant also needs to be populated with dialog rules. It's not sufficient to recognize the user intent – you need to respond appropriately to that intent. Virtual assistants use conditional logic to determine which response to give to the user. The simplest conditional logic is "if the user's intent is *this*, respond with *that*".

Terminology alert!

Dear reader, dialog logic is where there is the most variation between virtual assistant vendors! They all use some form of conditional logic, but the specifics of this logic vary.

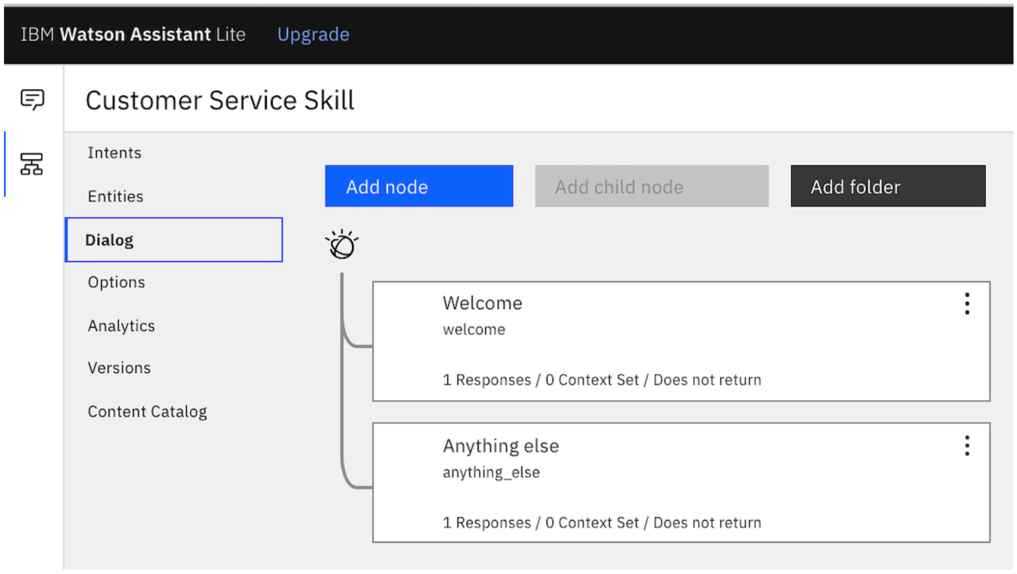
One major difference is the interface paradigm: some use a graphical interface, and some require you to write code using their libraries, to implement dialog logic.

Most virtual assistants have an event loop at their core. Most interactions with the assistant follow an "if this, then that" pattern over and over until a conversation ends. Virtual assistant platforms have very similar conditional logic primitives even if they use different names for them.

In this section, I will use lots of screenshots since "a picture is worth a thousand words". I am confident you can map each of the examples in this section to your virtual assistant platform of choice.

Your virtual assistant platform provides some method to implement dialog logic, either through a graphical editor or through pure code. A common way to reason about dialog logic is to think of the dialog as a decision tree. (This harkens to call centers that train employees on scripted "call trees".) In this methodology, every decision point in the dialog could be called a "dialog node". You'll be creating several dialog nodes as you implement all of the conversational paths supported by your assistant.

Before creating new dialog response nodes for FICTITIOUS INC let's explore the dialog nodes they got by default. Virtual assistant platforms will generally include a few dialog nodes, or dialog rules, to help you get started in your assistant. Figure 7 shows an example set of default dialog nodes.



Welcome
welcome
1 Responses / 0 Context Set / Does not return

Anything else
anything_else
1 Responses / 0 Context Set / Does not return

Figure 7 Dialog nodes created automatically by Watson Assistant. Your virtual assistant platform of choice will come with some default conditions and/or dialog nodes.

Each of the dialog nodes in Figure 7 includes a special condition that allows it to execute. The first is the "welcome" node – this is the node that contributed the "Hello. How can I help you?" greeting. The second is the "anything else" node which provided all the other dialog responses like "I didn't understand. You can try rephrasing."

The two nodes and their conditions are described below in Table 7.

Table 7 Pre-created dialog nodes and associated conditions in Watson Assistant. Other platforms have similar conditions, intents, events, and/or dialog nodes that handle these very common situations!

Dialog node title	Node condition	Description
Welcome	welcome	This represents your bot's greeting to users. The welcome condition is only true on a user's very first interaction with the assistant. In most virtual assistant instances, the assistant "speaks first" as a response to the user opening a chat window.
Anything else	anything_else	This represents a fallback or catch-all condition. The anything_else condition is always true and thus can be used to catch utterances that the bot does not understand.

Way back in Figure 5 we saw an assistant that recognized intents, but it responded with "I didn't understand". Now we know why. An assistant must be configured both to recognize intents and respond to conditions. If the assistant is not configured for both, it will fall back to some default behavior. Our newly created assistant has only a message that it uses to start a conversation (welcome) and a message it uses for every other response (anything else).

Terminology alert!

Your virtual assistant platform comes with some built-in conditions, events, or triggers that you can use.

"welcome" may be expressed as a "conversation start event" or a "welcome intent" – it just means a new conversation has started

"anything else" may be referred to as the "fallback intent", "default intent", "default condition", or "catch-all".

These two conditions are so prevalent in virtual assistants that your platform's tutorial is sure to include them!

FICTITIOUS INC needs rules coded into the assistant, telling it how to respond when more interesting conditions are met, such as when the `#Customer_Care_Store_Hours` intent is recognized in a user question. The assistant should respond to any `#Customer_Care_Store_Hours` question with some sort of answer about when stores are open!

2.3.1 Simple question and answer responses

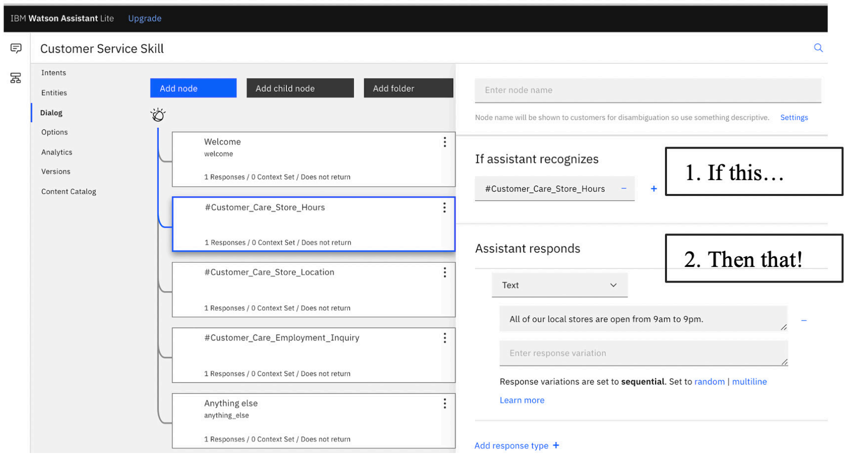
The simplest possible dialog response is to provide a single, static answer based on the recognized intent. FICTITIOUS INC most commonly receives questions about store hours, store locations, and how to apply for a job. Sample responses for these conditions will be coded into FICTITIOUS INC's assistant as shown in Table 8.

Table 8 The responses we'll code into the FICTITIOUS INC assistant when certain intents are recognized

When the assistant recognizes this intent…	It will give the following response
`#Customer_Care_Store_Hours`	**All of our local stores are open from 9am to 9pm.**
`#Customer_Care_Store_Location`	**We have two locations near you: 123 Maple St and 456 Elm Drive.**
`#Customer_Care_Employment_Inquiry`	**We accept job applications at FictitiousInc.com/jobs. Feel free to apply today!**

After coding these responses like Table 8 into the assistant, anyone who asks a question to the assistant will get a more useful response!

Figure 8 shows how to configure the first dialog rule for FICTITIOUS INC. If the assistant recognizes the user input as `#Customer_Care_Store_Hours`, it will respond with "All of our local stores are open from 9am to 9pm" and then wait for the user to provide additional input.



Welcome
welcome
1 Responses / 0 Context Set / Does not return

#Customer_Care_Store_Hours
1 Responses / 0 Context Set / Does not return

#Customer_Care_Store_Location
1 Responses / 0 Context Set / Does not return

#Customer_Care_Employment_Inquiry
1 Responses / 0 Context Set / Does not return

Anything else
anything_else
1 Responses / 0 Context Set / Does not return

Node name will be shown to customers for disambiguation so use something descriptive.

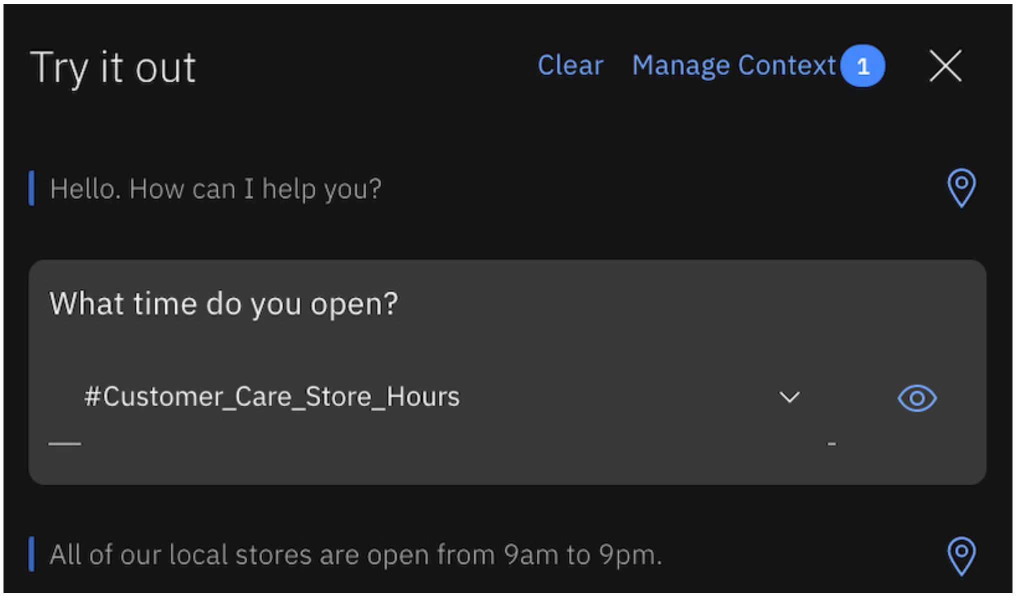
If assistant recognizes
#Customer_Care_Store_Hours

Assistant responds
All of our local stores are open from 9am to 9pm.

Response variations are set to **sequential**. Set to random | multiline

Figure 8 A simple dialog rule takes the form "if this *condition*, then give this *response*". Your virtual assistant platform may have a different interface, but you will surely be able to connect conditions to responses.

Figure 9 shows a test of the dialog configured as per Table 8. Note that we have only provided answers to three of the intents - for the other remaining Customer Care intents, even if the intent is recognized the assistant will use the catch-all "anything else" dialog node to respond. Each intent-specific response needs to be coded into the dialog as shown in Figure 8.



Hello. How can I help you?

What time do you open?

#Customer_Care_Store_Hours

All of our local stores are open from 9am to 9pm.

Figure 9 Testing after we have associated intent conditions with dialog responses. The testing interface now shows both the correct intent and the correct dialog response. A successful test!

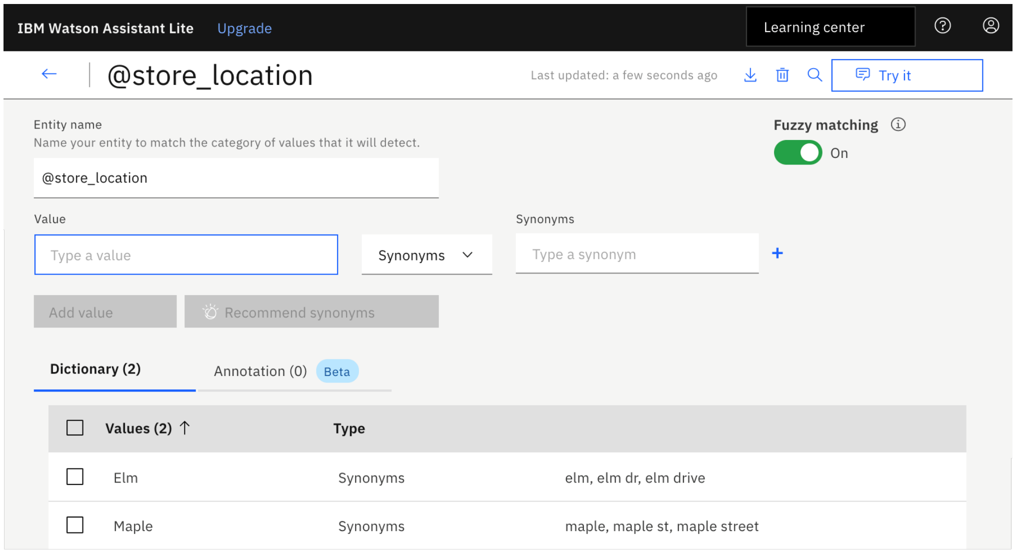
The assistant is now more useful to users, at least when it comes to three question types (store hours, store locations, and employment inquiries). However, FICTITIOUS INC has two retail locations, one at Maple St and one at Elm Dr. If users ask about a specific store they should get a store-specific response. The current implementation gives the same response for an intent no matter how detailed the user’s question gets. The assistant needs to understand more detailed context from the conversation

2.3.2 Contextualizing a response by using entities

Giving a store-specific answer has two parts: first detecting that a specific location was mentioned, and second replying with a specific condition. The specific location will be detected by training the assistant to recognize entities; the entity will be used to give a specific response.

Detecting an entity

Since the number of FICTITIOUS INC stores is both limited and well-defined, the simplest solution to use a static dictionary-based entity. We open the Entities tab and create a new entity called `@store_location` with two possible values: "elm" and "maple". We'll include a couple of variations with the street/drive qualifier as well, though these are not strictly required. We will also enable "Fuzzy matching" so that our entity dictionary will be robust in the case of user typos and misspellings.



Entity name
Name your entity to match the category of values that it will detect.
@store_location

Fuzzy matching: On

Values (2)	Type	
Elm	Synonyms	elm, elm dr, elm drive
Maple	Synonyms	maple, maple st, maple street

Figure 10 Example interface for defining entities. FICTITIOUS INC uses a @store_location entity with values for their two stores: Elm and Maple.

Terminology alert!

"Fuzzy matching" is a pretty common term! Each of the virtual assistant platforms I surveyed had fuzzy matching. Use fuzzy matching so that your assistant is resilient against typos and misspellings.

USING AN ENTITY TO GIVE A SPECIFIC RESPONSE

Next, we have to update the dialog conditions to check for a mention of one of our `@store_location` values in the user's utterance. We will need to update the `#Customer_Care_Store_Location` node to check for a `@store_location` and give a targeted response if present, and a generic response if not present.

Follow these steps to update the dialog:

1. Find the `#Customer_Care_Store_Location` condition.
2. Within that conditional block, create child conditions for `@store_location:Elm`, `@store_location:Maple`, and another fallback or "anything else" node (for when `#Customer_Care_Store_Location` is detected but no `@store_location` is present)
3. Associate a dialog response to each of the new conditions.

1. First dialog node detects the intent.

2. Defer a response until the child nodes are evaluated for entities.

3. If the user asked about Elm, tell them about the Elm location.

4. If the user asked about Maple, tell them about the Maple location.

5. Otherwise, give the generic #Customer_Care_Store_Location response

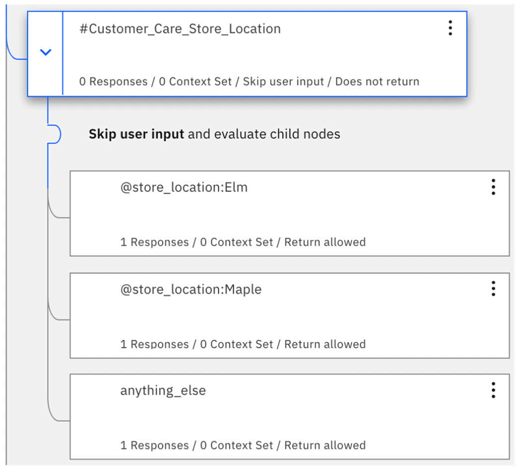

#Customer_Care_Store_Location
0 Responses / 0 Context Set / Skip user input / Does not return
Skip user input and evaluate child nodes
@store_location:Elm
1 Responses / 0 Context Set / Return allowed
@store_location:Maple
1 Responses / 0 Context Set / Return allowed
anything_else
1 Responses / 0 Context Set / Return allowed

Figure 11 Dialog rules implemented for detecting a specific store location. Three different answers will be given to a "store location" question based on the specific store, or lack or specific store, in the user's utterance. You can also implement this by putting multiple conditions in each dialog, joined by an "and"

This may look different in your virtual assistant platform, but you can still do it!

"Skip user input and evaluate child nodes" is my personal favorite way to implement this pattern. I like having one condition per dialog node which makes each node easier to read. The last three nodes in Figure 11 effectively have two conditions by being children of the first node.

You could alternately write three nodes with these conditions:

```
#Customer_Care_Store_Location and @store_location:Elm
#Customer_Care_Store_Location and @store_location:Maple
#Customer_Care_Store_Location
```

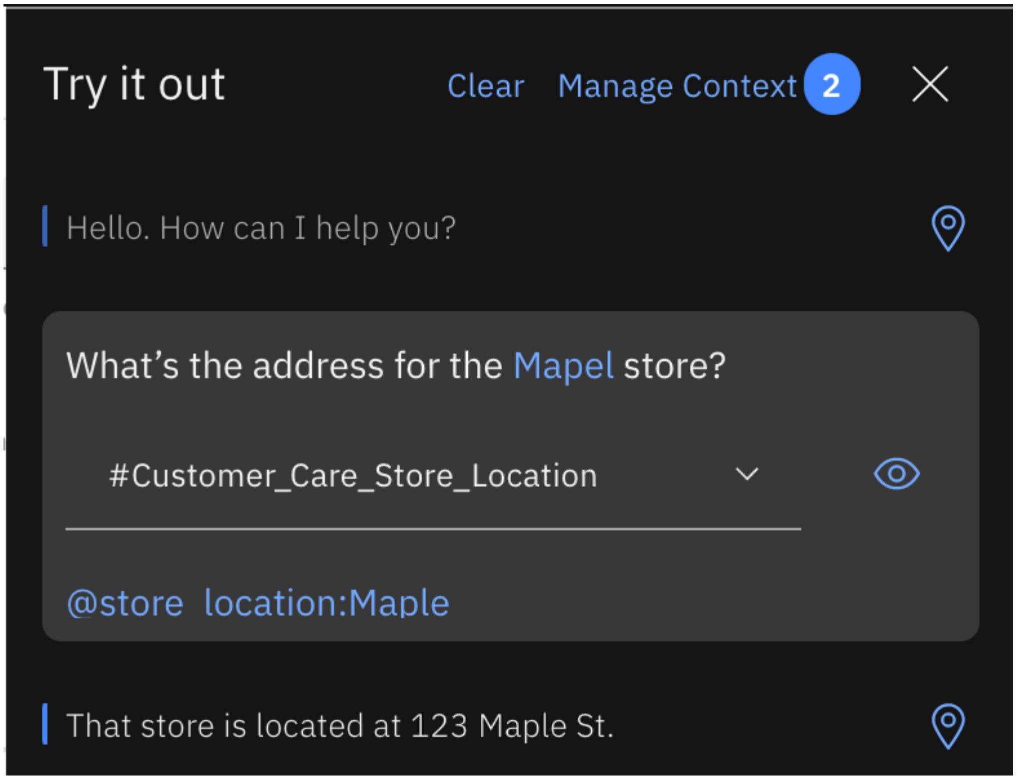
It's a matter of what your platform supports and your personal preference!

In Table 9 you can see that the assistant detects the three separate variations of the store location intent based on the contextual clues.

Table 9 How the assistant interprets three different "store location" utterances when entities are used

User utterance	Assistant detects	Assistant responds
Where is the Elm store?	`#Customer_Care_Store_Location` `@store_location:Elm`	That store is located at 456 Elm Drive.
What's the address for the Mapel store?	`#Customer_Care_Store_Location` `@store_location:Maple`	That store is located at 123 Maple St.
Where are your local stores?	`#Customer_Care_Store_Location`	We have two locations near you: 123 Maple St and 456 Elm Drive.

Figure 12 shows one variation of the store location questions in Watson's "Try it out" interface.



Hello. How can I help you?

What's the address for the Mapel store?

#Customer_Care_Store_Location

@store_location:Maple

That store is located at 123 Maple St.

Figure 12 The assistant has been trained to detect specific stores via entities and gives targeted responses. The assistant is even adaptive to misspellings ("Mapel") thanks to fuzzy matching.

The ability to tailor a specific response based on one or more contextual clues is a powerful way to satisfy your end users!

This may look different in your virtual assistant platform, but you can still do it!

For the virtual assistant platforms I surveyed, each provided some way of inspecting entities by telling you the entity type (`@store_location`), value ("Maple"), and "covered text" ("Mapel") in the testing interface.

2.3.3 An alternate way to provide contextual responses

FICTITIOUS INC provided contextual responses to `#Customer_Care_Store_Location` questions through the use of four dialog nodes. The pattern used requires *n + 2* dialog nodes, where *n* is the number of unique entity values. FICTITIOUS INC will also provide contextual responses to the `#Customer_Care_Store_Hours` intent based on store location variations. However, for this intent, they use a different technique called "Multiple conditioned responses"[6].

Multiple conditioned responses allow you to put conditional response logic in a single dialog node. To update this dialog with multiple conditioned responses:

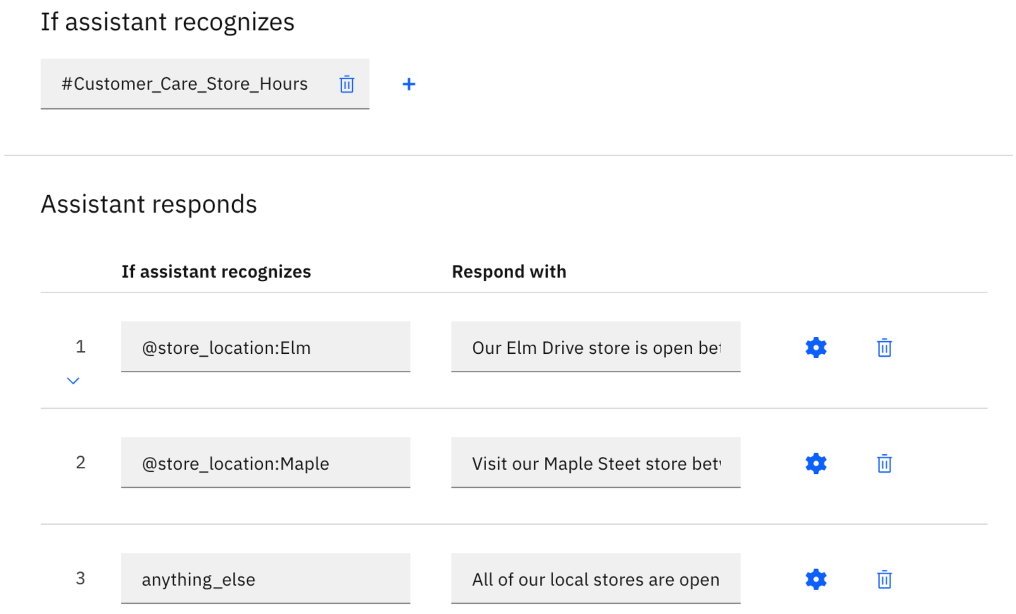
1. Enable multiple conditioned responses on the dialog node using the settings (triple dot) icon
2. Create conditioned responses for `@store_location:Elm`, `@store_location:Maple`, and an "anything else" node (for when no `@store_location` is present)

Coding conventions and style

Your platform may support multiple ways to implement complex conditions. Methods like multiple conditioned responses have the advantage and disadvantage of being more compact. These responses use less screen real estate, allowing you to see more of your dialog nodes on the screen at a time.

Consistency is important in software development. Consistency makes software easier to read and understand. However you choose to implement complex conditions, try to do it consistently!

[6]https://cloud.ibm.com/docs/assistant?topic=assistant-dialog-overview#dialog-overview-multiple



If assistant recognizes

#Customer_Care_Store_Hours

Assistant responds

	If assistant recognizes	Respond with
1	@store_location:Elm	Our Elm Drive store is open bet
2	@store_location:Maple	Visit our Maple Steet store bet
3	anything_else	All of our local stores are open

Figure 13 Giving a targeted "store hours" response based on which store (if any) the user asked about. A single dialog node is configured with three different possible responses. This is more compact on the screen but can be harder to read.

We've now seen two possible methods to improve the assistant's responses using contextual information. Either method will satisfy your end users and make them feel understood. When the user asks a specific form of a question ("where's the **Elm** store?") it's important to give a specific response ("The **Elm** store is at…")

Dictionaries seem pretty static – would I really use them to list my stores?

The preceding examples using dictionary-based entities are intended as a simplistic example. A national chain would prompt for zip code and use a store locator service called from an orchestrator to select a store, or get the zip code from the user's profile. The chain would not code hundreds or thousands of entities for each specific store.

A dictionary-based entity works best on a relatively small and finite list of possibilities.

2.3.4 Responding with a process flow

Many self-service interactions require a process flow, or some way of collecting all of the information necessary to complete a task. FICTITIOUS INC will implement a simple process flow for setting up an appointment.

When FICTITIOUS INC first configured the dialog for their assistant, they only created static responses. The response for `#Customer_Care_Employment_Inquiry` was a redirection to a website (“Apply on FictitiousInc.com/jobs now!). Instead of doing a similar redirection for `#Customer_Care_Appointments`, the assistant should actually book an appointment. Booking an appointment has a process flow – an appointment consists of a date, time, and specific store location.

FICTITIOUS INC can implement this process flow in one dialog node using a capability called "Slots". They will also use built-in entities (called System Entities) to gather date and time of the appointment.

Terminology alert!

Each virtual assistant platform I surveyed at the time of this writing had “Slots” capability though some gave it a different name like “parameters and actions”. Most virtual assistant platforms fill slots with “entities” though some call them “slot resolution”

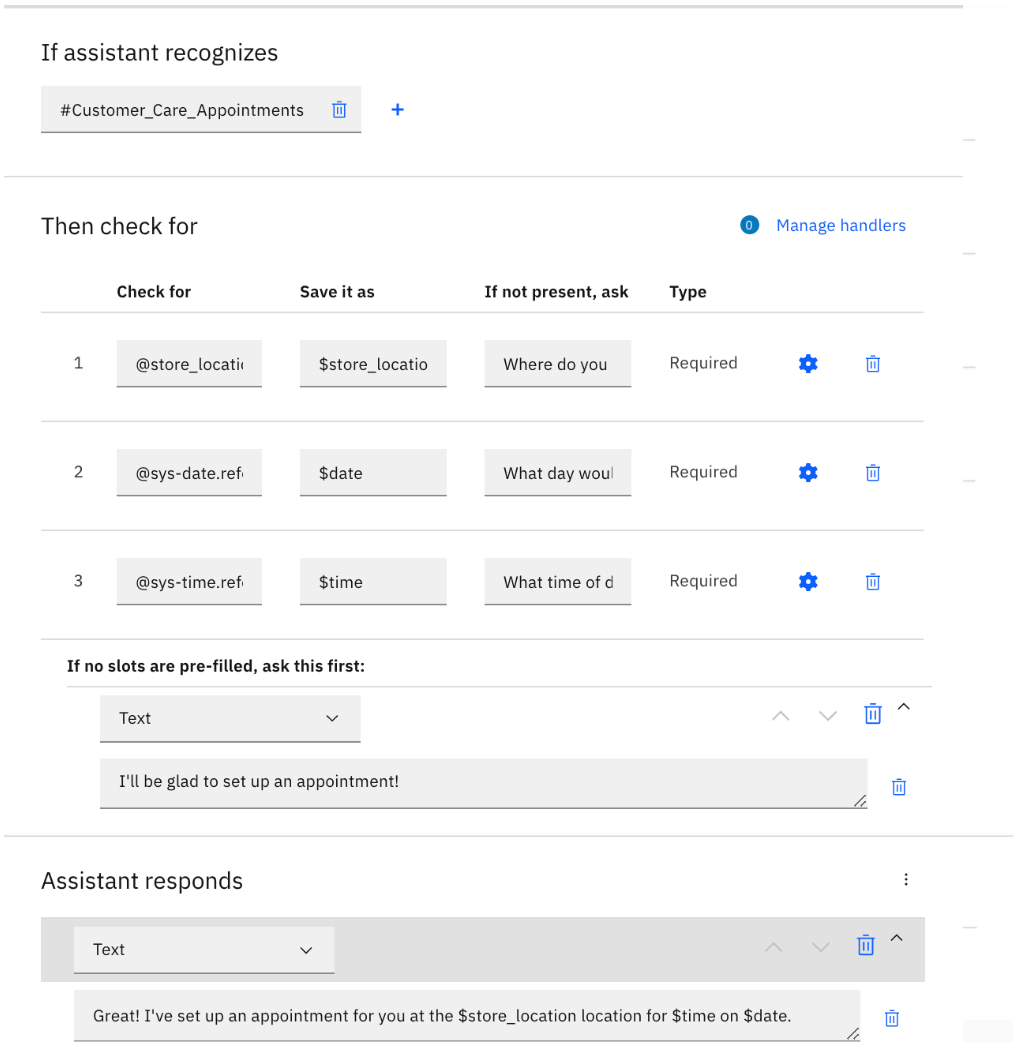
Think of “slots” as something that must be filled before a request can be worked on.
Setting an alarm has one slot (the time of the alarm)
Setting an appointment may have three slots (the date, time, and location of the appointment)

Virtual assistant platforms also usually come with “system” or “default” entities to cover basic scenarios like detecting dates. (Building your own date detection is fiendishly difficult, be glad your platform does it for you!)

Check your virtual assistant platform’s documentation for how slots are implemented. Slots are a cool feature and platform providers can’t help but brag about it!

Figure 14 shows how the appointment flow can look in a virtual assistant platform that uses slots. While the specific steps may vary slightly in other platforms, the Watson Assistant steps are listed here:

1. Enable Slots on the dialog node using the settings (triple dot) icon
2. Enable System Entities for sys-date and sys-time
3. Create conditions for each of the slot variables: how to detect them, where to store them, and how to prompt for them.
4. Create a text response when no slot variables are present
5. Create a response when all variables are filled in



If assistant recognizes

#Customer_Care_Appointments

Then check for

0 Manage handlers

	Check for	Save it as	If not present, ask	Type
1	@store_locati	$store_locatio	Where do you	Required
2	@sys-date.ref	$date	What day woul	Required
3	@sys-time.ref	$time	What time of d	Required

If no slots are pre-filled, ask this first:

Text

I'll be glad to set up an appointment!

Assistant responds

Text

Great! I've set up an appointment for you at the $store_location location for $time on $date.

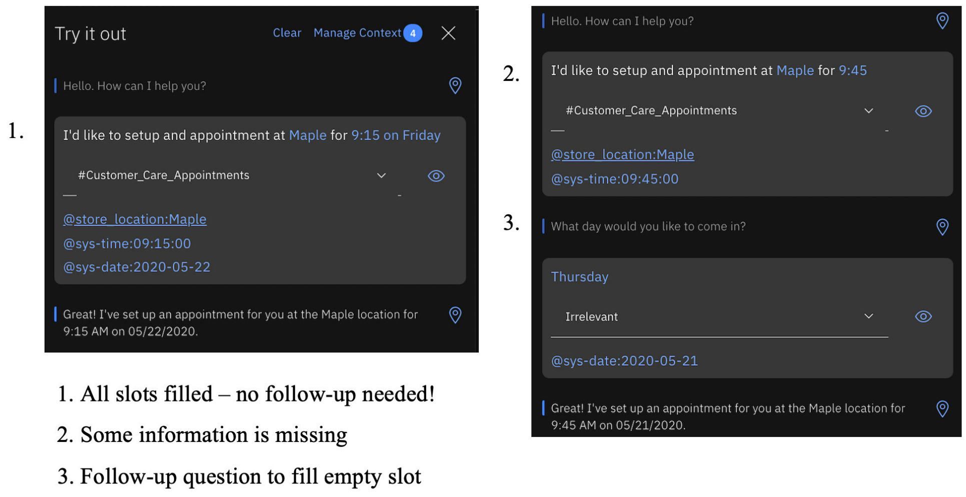
Figure 14 Configuring dialog to set up an appointment using slots. The dialog flow has slots for collecting the appointment location, date, and time. The user will be asked follow-up questions until all the slots are filled since an appointment cannot be scheduled without all three pieces of information. Your virtual assistant platform may have a different interface but probably still has slots!

Now that the dialog engine is coded with these conditions it can respond with a message reflecting all of the appointment details, such as: "Great! I've set up an appointment for you at the `$store_location` location for `$time` on `$date`." Your assistant will fill in the details dynamically as shown in the "Try it out" panel in Figure 14.

Technique alert!

Your platform will let you use variables in your responses, just check the syntax from your provider's documentation. Common patterns are $ plus a variable name (`$time`) or to put the variable name in braces (`{name}`). Search for your platform's documentation for "variables" or "parameters" and you should find what you need.

Figure 15 shows two different examples of conversational flows using slots. In the first example, the user supplies all three pieces of information in a single statement: "I'd like to set up an appointment at **Maple** for **9:15** on **Friday**". Since all information has been provided the system responds that the appointment has been set up. In the second the user's statement "I'd like to setup an appointment at **Maple** for **9:45**" does not include a date, so the system prompts the user "What day would you like to come in?".



Hello. How can I help you?

I'd like to setup and appointment at Maple for 9:15 on Friday

#Customer_Care_Appointments

@store_location:Maple
@sys-time:09:15:00
@sys-date:2020-05-22

Great! I've set up an appointment for you at the Maple location for 9:15 AM on 05/22/2020.

Hello. How can I help you?

I'd like to setup and appointment at Maple for 9:45

#Customer_Care_Appointments

@store_location:Maple
@sys-time:09:45:00

What day would you like to come in?

Thursday

Irrelevant

@sys-date:2020-05-21

Great! I've set up an appointment for you at the Maple location for 9:45 AM on 05/21/2020.

1. All slots filled – no follow-up needed!
2. Some information is missing
3. Follow-up question to fill empty slot

Figure 15 Two conversational examples with slots. In the left-hand example, all information is provided in the initial statement. In the right-hand example, the assistant asks follow-up questions to fill all of the required slots. This capability is common across major virtual assistant platforms!

The example ends with a message printed out to the user: "Great! I've set up an appointment for you at the $store_location location for $time on $date." I used Spring Expression Language (SpEL) expressions to reformat the date and time to a more user-friendly representation, rather than the internal date format.

Platform variation!

Watson Assistant uses SpEL[7] for “low-code” manipulation within the virtual assistant dialog response. Your platform may require you to use “no-code”, “low-code”, or just plain code in your responses. Check your platform documentation on how to customize dialog responses with system variables.

Listing 1: The SpEL logic used in the Watson Assistant date formatting example

```
{
  "context": {
    "date": "<? @sys-date.reformatDateTime(\"MM/dd/yyyy\") ?>"
  }
}
```

Listing 2: The SpEL logic used in the Watson Assistant time formatting example

```
{
  "context": {
    "time": "<? @sys-time.reformatDateTime(\"h:mm a\") ?>"
  }
}
```

This appointment flow ends with a dialog response to the user, telling them about their appointment, but it does not actually book the appointment. Booking an appointment would require an API call to an external booking system, coordinated by the orchestration layer. For the sake of brevity, this was not demonstrated.

Booking an appointment can be slightly more complex!

We have walked through an illustrative example of appointment booking. A more robust example would include calling an appointment service API which would check availability, book the appointment, and return a success or failure response. Your platform may enable this – look for “webhooks”, “orchestration”, “connectors”, “integration”, or anywhere you can write code!

Slots are a useful way to implement a process flow that needs to collect multiple pieces of information before proceeding. Slots provide a syntax shortcut for gathering multiple input variables in a coordinated way. You can choose to implement process flows with or without slots. Anything you could implement in Slots could also be implemented in regular dialog nodes.

[7]https://cloud.ibm.com/docs/assistant?topic=assistant-dialog-methods

Further discussion on conventions and style

As in the entities example, there are multiple ways of implementing process flows. You may wish to gather "slot" variables one-by-one and do an external validation. For instance, when authenticating with two pieces of information like user ID and date of birth, you may want to validate the user ID exists in the system before prompting for date of birth, and you'll want to execute a resolution flow if the user ID is not correct.

2.4 Other useful responses

FICTITIOUS INC's assistant now has the following capabilities:

- Greeting the user
- Providing a static, non-contextual answer for employment inquiries
- Providing static, contextual answers for store location and store hour requests
- Executing a process flow to set up appointments
- Responding with "I didn't understand" to any other utterances

Before FICTITIOUS INC accepts this demonstration they need two more capabilities:

- Determining when the assistant is unsure about the user's intent
- Handing the user off to another channel when the assistant repeatedly misunderstands the user

Three strikes and you're out!

Virtual assistants often use a metaphor from baseball. When a baseball batter swings and misses it's called a "strike". If the batter accrues three strikes before hitting the ball, they're out!

Similarly, if a virtual assistant misunderstands the user, we can call that a strike. If the assistant misunderstands the user three times, the assistant should hand the user off to get their resolution elsewhere.

Many virtual assistant implementers use a "two strikes" rule with similar logic.

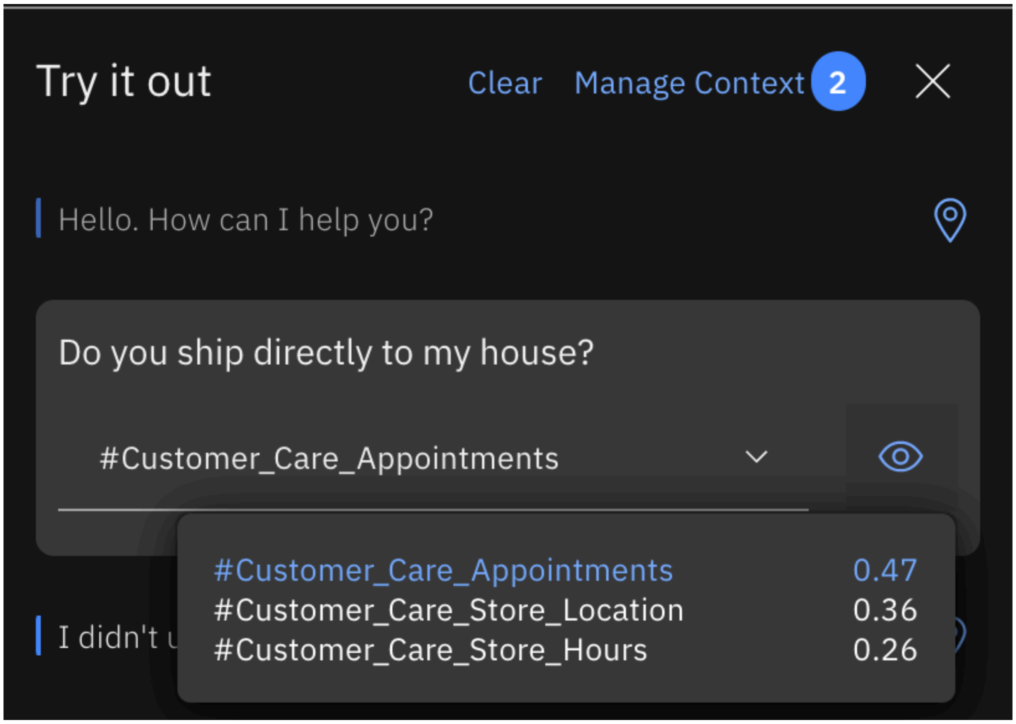
Let's update the assistant with a two-strikes rule. We'll implement two pieces of logic in the dialog nodes:

1. Detecting low confidence conditions from the classifier and routing them to the "Anything Else" node.
2. Counting how many times the "Anything Else" node is visited.

2.4.1 Detecting low confidence

Most virtual assistant classifiers not only predict an intent but also tell you the confidence in that prediction. By default, they only return an intent if the best prediction has confidence over some threshold (for example, this threshold is 0.2 in Watson Assistant). Intent predictions with low confidence values are frequently incorrect and you may need to tune this threshold for your particular assistant.

In Chapter 7 we will see how the confidence value is derived. For now, just consider that the lower the confidence value is, the more likely the predicted intent is wrong. Figure 16 shows how you can inspect the confidence values in a virtual assistant testing interface.



Hello. How can I help you?

Do you ship directly to my house?

#Customer_Care_Appointments

```
#Customer_Care_Appointments    0.47
#Customer_Care_Store_Location  0.36
#Customer_Care_Store_Hours     0.26
```

I didn't u

Figure 16 The most likely intents and a confidence value associated with each for an input utterance. Note that low confidence results are often wrong. The utterance "do you ship directly to my house" does not match ANY of the intents coded into this assistant.

The confidence values can be used in conditional logic in dialog nodes.

Listing 3: Conditional logic for low-confidence intent selection in Watson Assistant

```
intents.size() > 0 && intents.get(0).confidence < 0.5
```

Terminology alert

Each platform I surveyed had a "confidence" term. Hooray again for common terminology!

The platforms differed in how they referred to the threshold and how you configure it. The threshold is also called "confidence cutoff", "intent threshold", or "classification threshold", though is still described as a threshold.

Most virtual assistant platforms have a single setting for this classification threshold to apply it globally to your assistant. Some platforms let you use a different (higher) threshold in specific dialog logic.

2.4.2 Counting misunderstandings

When the dialog reaches the catch-all "Anything else" node, this implies the user was not understood, and their satisfaction with the assistant will decrease. Therefore, it's a good idea to count how many times this dialog node has been visited. The second time the catch-all node is visited we should respond by offering the user an alternate channel before they get really frustrated.

Most virtual assistant platforms come with a way to store arbitrary information in variables during a conversation. These are often called "context variables". We'll use a context variable to count how many times the catch-all node is visited.

Some platforms automatically count the number of visits to the catch-all node

Be sure to check your platform's documentation to see if it has this feature – it will save you some time!

The two-strikes rule is thus implemented by storing the number of visits to the catch-all "Anything else" node in a context variable. Two strikes (two visits to this node) and the assistant should send the user to another channel such as a customer service phone line, website, or another human-backed process.

FICTITIOUS INC wants to be conservative with their first assistant. They only want the assistant to respond to a user's intent if that intent is detected with high confidence. Thus, their assistant should treat low-confidence predictions as misunderstandings. Low confidence predictions should be sent to the "Anything else" node. Let's see how to implement these two features in the assistant.

2.4.3 Implementing confidence detection and the two-strikes rule

This functionality is implemented via the following steps:

1. Initialize a context variable `misunderstood_count` to 0 in the `Welcome` node.
2. Create a new "Misunderstanding" node at the bottom of the dialog with the fallback condition.
3. Increment `misunderstood_count` inside the "Misunderstanding" node
4. Add a child node to "Misunderstanding" checking the misunderstood count, and if `$misunderstood_count >= 2` deflect the user with the message "I'm sorry I'm not able to understand your question. Please try our phone support at 1-800-555-5555."

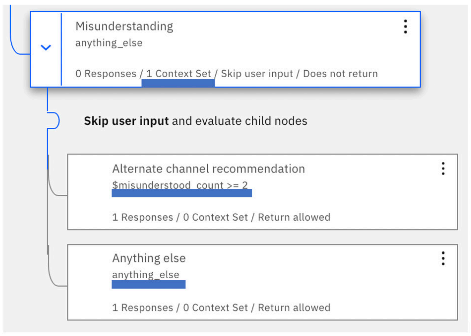
5. Moving the existing "Anything Else" node as the last child of the "Misunderstanding" node.
6. Create a "Low Confidence Prediction" node before all the other intent nodes with the "low confidence" conditional logic `(intents.size() > 0 && intents.get(0).confidence < 0.5)`. When this node fires, it should "go to" and evaluate the "Misunderstanding" node conditions.

The code is visualized in Figure 17. The code listings are also shown below the figure.

1. "Catch-all" node for any misunderstandings. Increments a <u>context variable</u> on each visit.

2. If this is the <u>second strike</u>, deflect the user!

3. <u>Otherwise</u>, ask the user to restate their utterance.



Misunderstanding
anything_else
0 Responses / 1 Context Set / Skip user input / Does not return

Skip user input and evaluate child nodes

Alternate channel recommendation
$misunderstood_count >= 2
1 Responses / 0 Context Set / Return allowed

Anything else
anything_else
1 Responses / 0 Context Set / Return allowed

Figure 17 Visualization of two-strikes logic. This will look different in your virtual assistant platform of choice but should still be implementable.

Listing 4: Watson Assistant code for the Welcome node: initialize the misunderstanding counter

```
{
  "output": {
    "generic": [
      {
        "values": [
          {
            "text": "Hello. How can I help you?"
          }
        ],
        "response_type": "text",
        "selection_policy": "sequential"
      }
    ]
  },
  "context": {
    "misunderstood_count": 0
  }
}
```

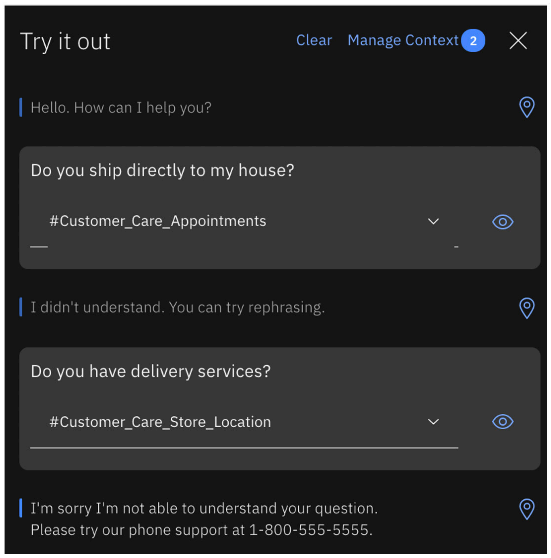
Listing 5: Watson Assistant code for the catch-all node: increment the misunderstanding counter

```
{
  "output": {
    "generic": []
  },
  "context": {
    "misunderstood_count": "<? $misunderstood_count + 1 ?>"
  }
}
```

A demonstration of the "two strikes" rule in action is found in Figure 18.

1. An intent was recognized but with low confidence. Strike one!

2. Another low confidence intent. Strike two, and the user redirected.



Hello. How can I help you?

Do you ship directly to my house?

#Customer_Care_Appointments

I didn't understand. You can try rephrasing.

Do you have delivery services?

#Customer_Care_Store_Location

I'm sorry I'm not able to understand your question.
Please try our phone support at 1-800-555-5555.

Figure 18 The assistant is asked two questions. For each one, it detects an intent with low confidence and visits the catch-all node. After two misunderstandings, the user is deflected to call customer service.

In Figure 18, the assistant selected an intent for each user utterance. The first utterance, "Do you ship directly to my house" was classified as `#Customer_Care_Appointments` but with low confidence. The second utterance "Do you have delivery services" was classified as `#Customer_Care_Store_Hours` but again with low confidence. Because of the Low Confidence Response dialog node, the low confidence condition fired before any of the intent-based nodes, and thus the system does not respond based on a (possibly wrong) intent selection.

How can you know the intents were recognized with low confidence?

Watson Assistant will show you the confidence behind an intent when you hover over the “eye” icon next to the identified intent. The algorithms used by virtual assistant providers are constantly changing, but if you try out this example you should see low confidence values, probably between 0.4 and 0.5.

Your virtual assistant platform will provide a way to see the intent confidence either through a testing interface or API.

2.5 Try it out!

The principles demonstrated in this chapter will work across most virtual assistant platforms. I have provided screenshots, terminology guides, and the reasoning behind each concept and example so that you can implement them in your virtual assistant platform of choice. Some of the low-level details are specific to Watson Assistant but I am confident you can implement FICTITIOUS INC’s assistant in any virtual assistant platform.

The Watson Assistant source code is available in this book’s GitHub site at https://github.com/andrewrfreed/CreatingVirtualAssistants. The file is `skill-Customer-Service-Skill-Ch2.json` and you can import it into your Watson Assistant instance. The JSON format used is not human-friendly and is best read from the Watson Assistant interface.

Try expanding the capabilities of the assistant on your own! For a first exercise try responding to `#Customer_Care_Contact_Us` questions by giving out a phone number and an email address. Next work on a multi-step response for `#Customer_Care_Open_Account` inquiries. What other capabilities would you like to add to the assistant?

2.6 Summary

- User input to a virtual assistant is called an utterance. An utterance contains an intent (centered around a verb) and/or entities (centered around nouns).
- A virtual assistant responds to a user by first extracting meaning from the user’s utterance and then mapping that meaning to a response.
- The simplest virtual assistant response is to provide an answer to the user’s utterance and wait for the user to send another utterance.
- A virtual assistant can execute a process flow based on a user’s utterance. That process flow will dictate additional questions or steps in the conversation that needs to take place before an answer can be provided to the user.