User Guide | PUBLIC
SAP Conversational AI
Document Version: 1.0 – 2021-08-24

Concepts of SAP Conversational AI

© 2021 SAP SE or an SAP affiliate company. All rights reserved.

THE BEST RUN SAP

Content

1 Overview. 5
1.1 Concepts. 5
1.2 Platform Overview. 6
1.3 Browser Support. 7

2 Getting Started with the Bot Builder. 9
2.1 Create Your Chatbot. 9
2.2 Train Your Chatbot. 13
2.3 Build Your Conversation. 22

3 Natural Language Processing (NLP) Lexicon. 27
3.1 Intents. 27
3.2 Expressions. 29
3.3 Entities. 31
Importing Entity Values. 36
Enrichments for Custom Entities. 41
Enrichments for Gold Entities. 49
List of Gold Entities. 55
References Between Entities. 72
3.4 Sentiments. 77
3.5 Sentence Acts. 77
3.6 Sentence Types. 77
3.7 Languages. 80
3.8 Creating a Good Dataset. 82

4 Bot Builder. 86
4.1 Introduction. 86
4.2 Skills. 87
Disambiguation Skill. 90
Initialize Skill. 94
4.3 Conditions. 96
4.4 Triggers. 99
4.5 Requirements. 100
4.6 Actions. 108
4.7 Memory Management. 111
4.8 Messages. 114
Message Types. 120
4.9 Conversation State. 132

4.10 User Context. . . . 133
4.11 Single Sign-On with SAP Product Integration. . . . 135
4.12 Connect to External Service. . . . 136
System Alias Configuration. . . . 140
Authentication Configuration. . . . 142
Header Configuration. . . . 144
Body Configuration. . . . 144
API Response Configuration. . . . 146
Getting Response Using Webhook. . . . 152
4.13 Scripting with Variables. . . . 157
Scripting Syntax. . . . 159
Runtime Data Accessible. . . . 179

5 Bot Connector. . . . 181
5.1 Messaging Channels. . . . 181
Amazon Alexa. . . . 182
Facebook Messenger. . . . 183
LINE. . . . 184
Microsoft (Skype, Teams). . . . 184
SAP Conversational AI Web Client. . . . 185
SAP CoPilot. . . . 201
SAP Jam Collaboration. . . . 202
Slack. . . . 203
Telegram. . . . 203
Twilio. . . . 204
Twitter. . . . 205
Fallback Channels. . . . 205
Webchat. . . . 206
5.2 Getting Started with the Bot Connector. . . . 211
5.3 Receive Messages. . . . 214
5.4 Send Rich Messages. . . . 215

6 Monitoring and Analytics. . . . 224
6.1 Log Feed. . . . 224
6.2 Usage Metrics. . . . 225
6.3 Training Analytics. . . . 226
6.4 Conversation Logs. . . . 234

7 Collaboration. . . . 237
7.1 Organizations. . . . 237
7.2 Permissions at Organization Level. . . . 241
7.3 Permissions at Bot Level. . . . 242

8 Bot Management. 248
8.1 Versions and Environments. 248
8.2 Forking Bots, Skills, Intents, and Entities. 255
8.3 Authentication. 262
8.4 Transporting Bots From User Interface. 266
Overview. 266
Requirements. 267
Export and Import your Bots across Tenants. 268

9 Getting Started with FAQ Bot. 271
9.1 Introduction and Prerequisites. 271
9.2 Create an FAQ Bot. 271
9.3 Upload Your FAQ Document. 272
9.4 Edit your FAQ Document. 275
9.5 Upload a Document with Alternative Questions 276
9.6 Start Chatting with Your Bot. 277
Skills. 278
Custom Normalization. 286
9.7 Connect Your Bot to a Channel 289
9.8 Monitor Your Bot Conversations. 289
Monitor Conversations Using Log Feed 290
View Conversation Logs. 290
Assign Users' Questions to an Answer 291
Measure FAQ Bot Accuracy 292
9.9 Export and Import your Bots 294

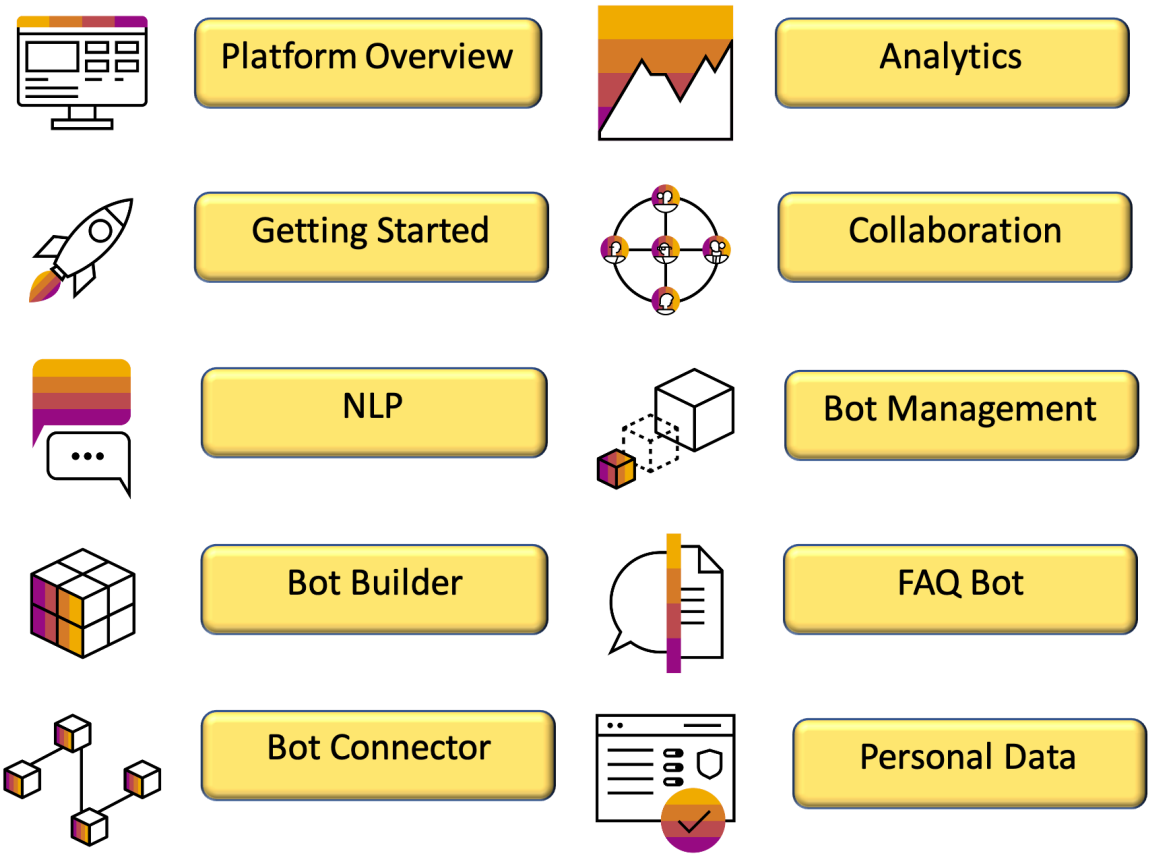
10 Personal Data. 296
10.1 Update or Delete Your Personal Data. 296
10.2 Access or Delete a Bot User's Personal Data. 296

1 Overview

1.1 Concepts

SAP Conversational AI is a bot building platform that gives you the ability to build and deploy a conversational agent in your application.

The following graphic shows the main concepts that help you build a conversational agent and deploy it in your application.



- Platform Overview [page 6]
- Create Your Chatbot [page 9]
- Intents [page 27]
- Introduction [page 86]
- Log Feed [page 224]

- Organizations [page 237]
- Versions and Environments [page 248]
- Introduction and Prerequisites [page 271]
- Messaging Channels [page 181]
- Update or Delete Your Personal Data [page 296]

1.2 Platform Overview

SAP Conversational AI provides a web user interface that serves as a platform to create, build, and test chatbots for your business and individual needs.

In most of the scenarios, you can use the SAP Conversational AI platform to build chatbots. But in advanced scenarios, you can also use the API reference to build bots.

Log in to the Platform

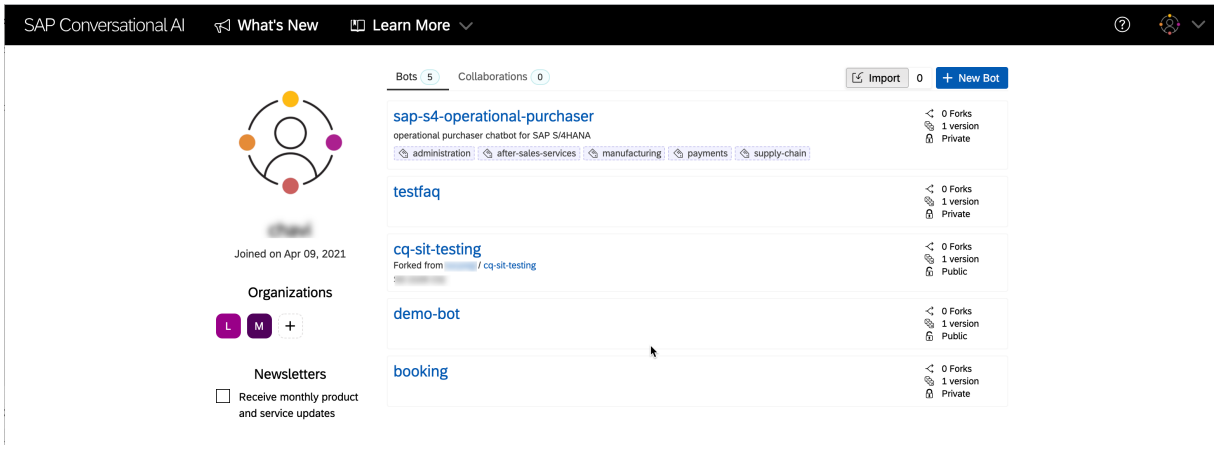
Go to https://cai.tools.sap/ . Before you use the platform for the first time, you need to sign up and create your user account.

- Sign up with SAP Conversational AI
  1. Click *Sign Up* on the top right corner or click the first option *Create*.
  2. Provide your first name, last name and email id (personal or corporate). The first name will be used to identify you publicly on the platform. If there are multiple users with the same first name, a number is appended to make it unique.
  3. Secure your account with a password.
  4. Accept SAP Conversational AI's terms and conditions and click *Register*.
     You will receive an email to verify your account. Validate your account and log in with your email and the password that you provided.
     You will be logged in to your new SAP Conversational AI account.

     > **i Note**
     >
     > If you already have an SAP ID but you use another email to sign into SAP Conversational AI, then you should select the third option to link your SAP ID with your existing sign in credentials.
- Log in to SAP Conversational AI (New users)
  1. Once you access the platform, there are three options. Select the second option and click *Sign In*.
  2. Provide your credentials to log in to your account.

Platform Design

The information in your profile page is visible in three main panels.



Header bar

In the top right corner, you have the option to

- click the help icon and navigate to SAP Conversational AI Community portal
- click your profile icon and choose *Account Settings*. Here you can perform the following actions:
  - You can change your nickname (username) in the *Profile* tab, however, the email ID for your account can not be changed.
  - Select your time zone in the *Preferences* tab
  - Change your profile visiblity (public or private), import bots from other SAP Conversational AI tenants or delete your account in the *Danger Zone* tab

Left panel

On the left side of your profile page, your user name is visible. To add your profile picture, first create an account in `https://gravatar.com/` with the same email ID that is associated with your SAP Conversational AI account. Follow the instructions as provided in `https://gravatar.com/support/`. For more information on your profile settings, see Update or Delete Your Personal Data [page 296].

Under *Organizations*, you can create a new organization and add members or teams of people to collaborate on several bots at the same time. For more information, see Organizations [page 237].

Main content (Right panel)

All the bots that you create are visible in the *Bots* tab.

All the bots for which you have been added as a collaborator by another bot developer, are visible in the *Collaborations* tab.

1.3 Browser Support

For the user interface of SAP Conversational AI platform, SAP Conversational AI Web Client and Wechat channels, the following browsers are supported on Microsoft Windows and on Mac OS:

Browser	Version	Supported By
Microsoft Internet Explorer	11	SAP Conversational AI platform
Microsoft Edge	Latest version	SAP Conversational AI platform, SAP Conversational AI Web Client and Wechat
Mozilla Firefox	Extended Support Release (ESR) and latest version	SAP Conversational AI platform, SAP Conversational AI Web Client and Wechat
Google Chrome	Latest version	SAP Conversational AI platform, SAP Conversational AI Web Client and Wechat
Safari	Latest 2 versions (for macOS only)	SAP Conversational AI platform, SAP Conversational AI Web Client and Wechat

Related Information

Recommendations and Constraints

2 Getting Started with the Bot Builder

2.1 Create Your Chatbot

Introduction

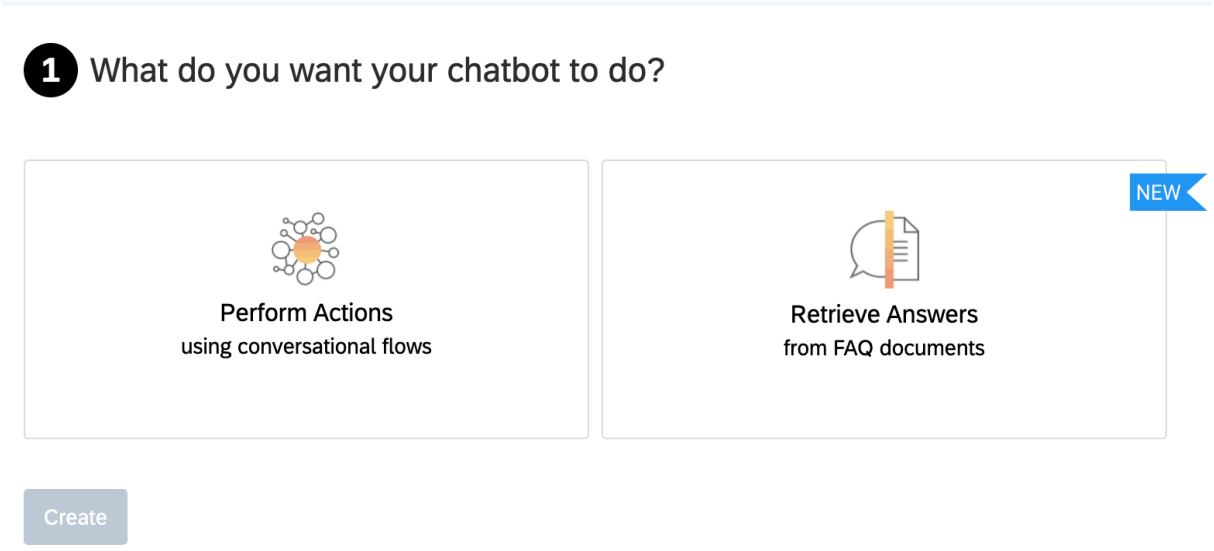
Let's start by understanding the core concepts of the SAP Conversational AI platform. You'll then be able to build a chatbot that can manage an entire conversation with a user.

An SAP Conversational AI chatbot comprises two main elements:

- Skills
  A skill is a block of conversation that has a clear purpose and that your bot can execute to achieve a goal. You need to configure these skills to build the scope of your bot.
- Training dataset
  A training dataset is composed of many sentences organized into intents [page 27] that represent what users say to your chatbot. The training dataset is used to train the bot to understand the user's needs and to trigger the right piece of conversation, to reply correctly, and to have a smooth conversation.

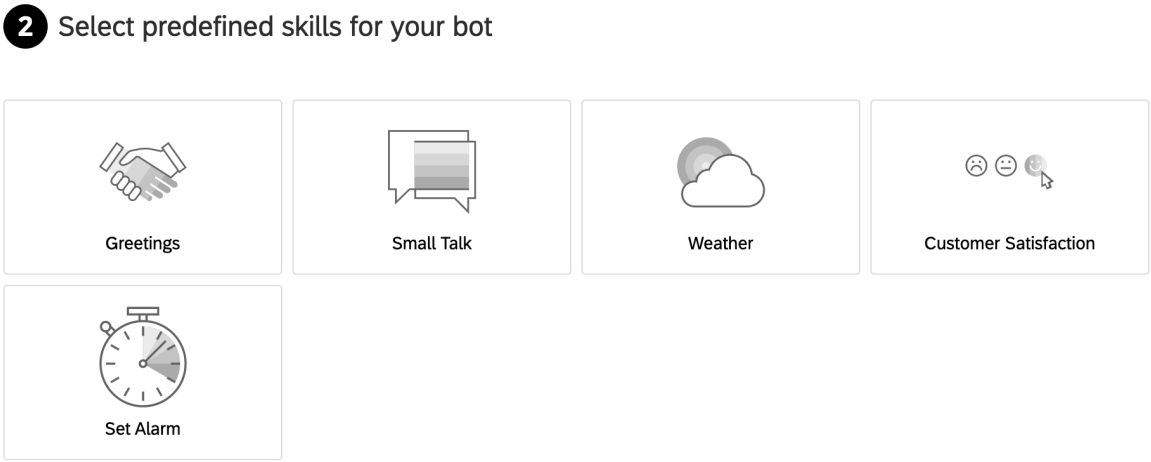
Ready? Click *+ New Bot* at the top right of the page in SAP Conversational AI and let's create your first chatbot.

Create Your Bot



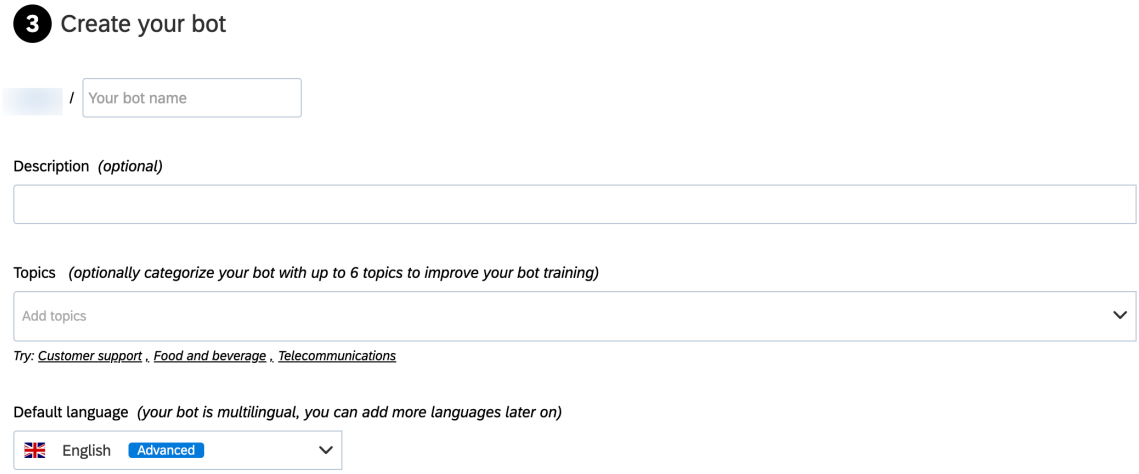
- Create Your Chatbot [page 9]
- Create Your Chatbot [page 9]

1. Choose if you wish to create a bot that can perform actions, or retrieve answers to users' questions that are related to a policy or any other static content. For more information on creating an FAQ bot, see Getting Started with FAQ Bot [page 271].
2. Choose one or several predefined skills to use as a starting point. Let's select *Greetings*. You're free to modify them if you don't like them as such, or even delete them once you're ready to make your own. If you want to fork the skills later on, they're available at https://cai.tools.sap/scaffolder/starter-skills .



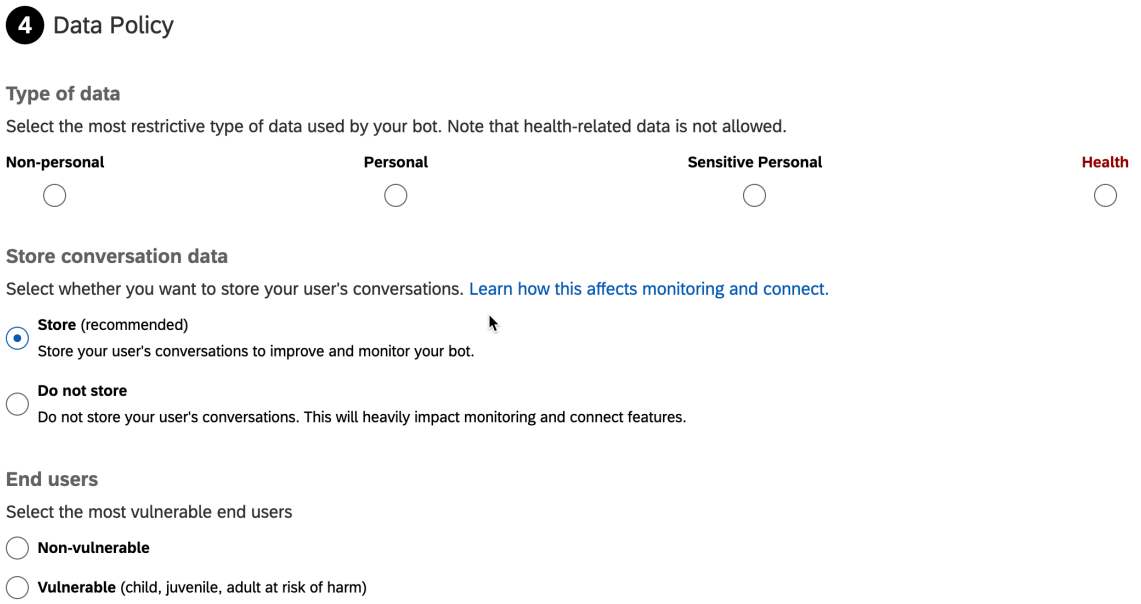
> **i Note**
>
> The available predefined skills change based on the language selected under *Create your bot*, hence it is recommended that you select the language first and then select the predefined skills.

3. Create your bot:
   1. Enter a name and, if desired, a description for your bot.
   2. (Optional) Add up to six topics to your bot (for example, *Customer Support*, *HR*, *Payments*, and so on). By categorizing your bot in this way, we can suggest more appropriate training data to improve it later on.
   3. Set the default language. You can add more languages later.



4. To support compliance with General Data Protection Regulation (GDPR) requirements, the bot can be tagged for the type of data processed by your bot (non-personal, personal, sensitive personal, or health), and for the type of end users (non-vulnerable or vulnerable). Your selection for the type of data will impact the bot visibility and maximum retention period of the conversation logs only; **the setting is not evaluated by the platform and no additional processing is done by SAP Conversational AI**. For more information, see Bot Data Policy Settings.

> **i Note**
>
> Since SAP Conversational AI provides a bot building platform only, compliance with GDPR requirements for the end-to-end scenario is the responsibility of the bot developer.



Data Policy	Bot visibility	Data retention period
Non-personal	You can choose your bot visibility to be public or private. With your URL slug (user name), your public bot is accessible to everyone. For example, *https://cai.tools.sap/tester* If your bot is private, it is accessible only by you and the developers you decide to share it with.	unlimited
Personal	The bot visibility is automatically set to private. You can not create a public bot with this data policy.	3 years

Data Policy	Bot visibility	Data retention period
Sensitive personal	The bot visibility is automatically set to private. You can not create a public bot with this data policy.	1 year
Health	SAP Conversational AI does not support health data	none

5. Specify whether your bot is public or private. Your bot visibility is decided based on the Bot Data Policy Settings you have chosen.

> **i Note**
>
> After the bot is created, you have the option to switch your bot visibility under your bot *Settings* > *Danger Zone*.

6. Choose if you want to store your conversation data or not. This is applicable only if you are using SAP Conversational AI Web Client and Webchat channels. If you do not wish to store the conversation data, the data will not be visible under *Log Feed*, *Usage Metrics* (entity values, enrichment values, user utterances), and *Conversation Logs*.
   For existing bots, you need to select the appropriate option under your bot *Settings* > *Data policy*.
7. Click *CREATE A BOT*.

Discover Your First Intents and Skills

Train

If you selected the skill *Greetings*, you'll see two intents on the *Train* tab: *greetings* and *goodbye*.

An intent is a collection of sentences that all have the same meaning, even though they can be very different from one another. When a user sends a message to your bot, our algorithm predicts the intents to which it's close enough and decides what the intention of the message is. Here are three examples of sentences with the same meaning:

- Are you a bot?
- You reply so fast, I'm sure you must be some kind of robot.
- Am I speaking to a human or not?

They're all different, but they all ask the same question that we can sum up as *Are you a bot?* Well, that would make a great intent! If your bot is able to recognize this question, you can prepare a smart reaction, like *I'm a robot and I'm proud of it.*

Build

On the *Build* tab, you'll find two skills: *greetings* and *fallback*. Click *greetings*. You'll see that a skill has four parts:

- README.md
  Where you explain the purpose of the skill.
- Triggers
  Where you define why this skill should be activated after a user message.

- Requirements
  What information this skill has to collect, and what questions need to be asked to fulfill the requirements.
- Actions
  What to do once the requirements are fulfilled.

If you navigate through the tabs, you'll see that this skill is structured as follows:

It is triggered if the intention *greetings* or *goodbye* is matched. It has no requirements because it does not need to collect additional information. This means that it will execute actions directly after a trigger. It has two possible actions. If the intention matched is *greetings*, it sends a random welcoming message chosen from a list. If the intention is *goodbye*, it does the same thing, but picks the message from a different list.

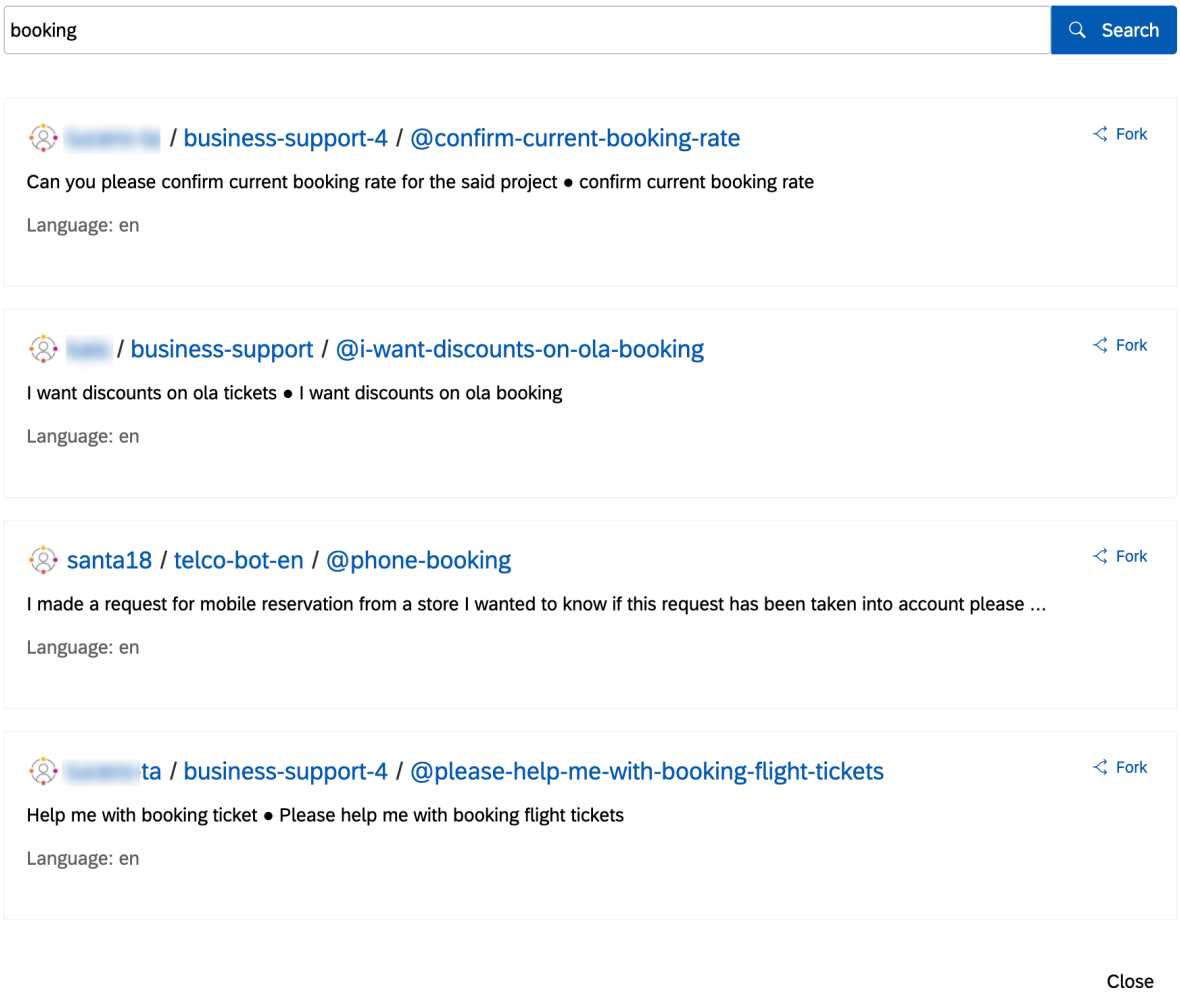
2.2 Train Your Chatbot

Create an Intent

Everything your chatbot understands is in the intents. Each intent corresponds to an action that your user wants to perform. For example, the intent *greetings* enables your bot to understand when a user says *Hello*.

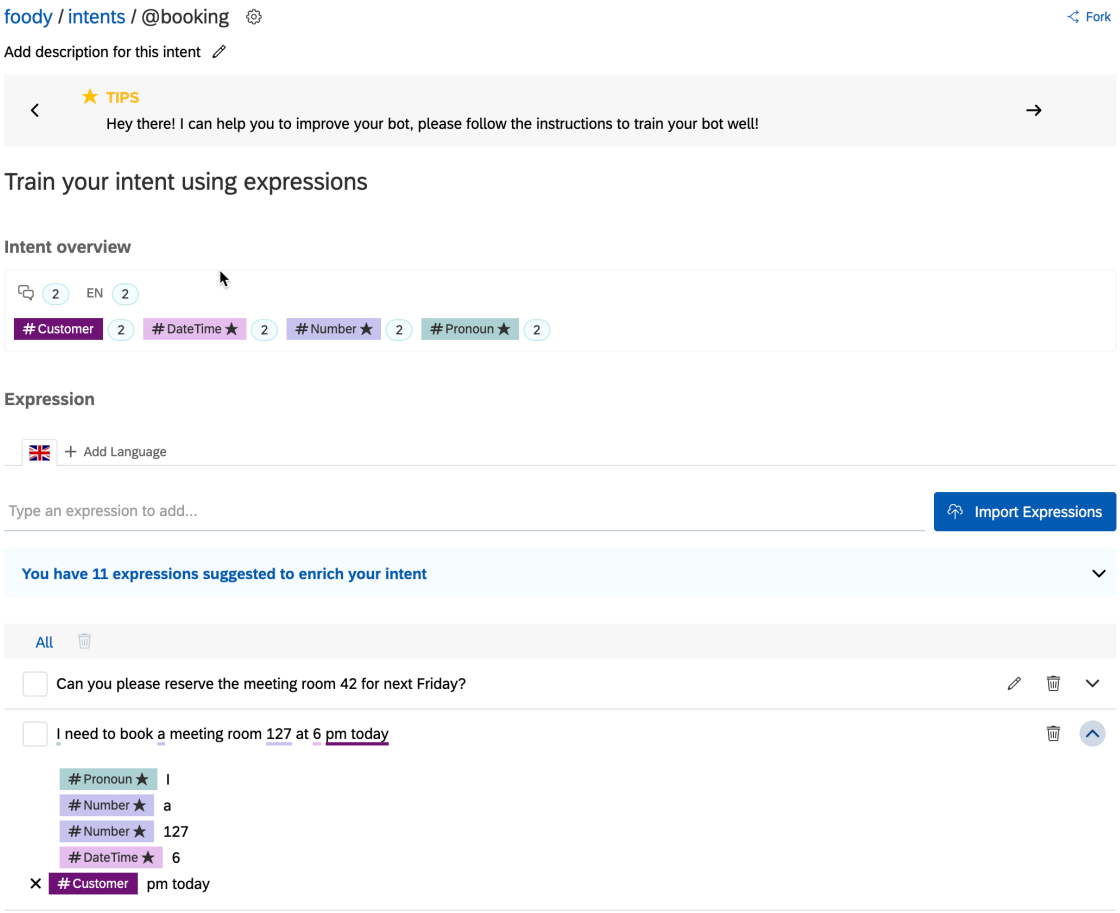
Explore each intent by clicking the name of the intent (for example, *greetings*), and you'll see the expressions inside that train your bot to understand the user's intent.

Let's add a new capability to our chatbot to book a meeting room. Add a new intent called *booking* to understand when users ask your chatbot to book a room. In the *Search* field, type **`booking`** and click *Search* to search and fork this intent from the community. Since the platform is collaborative, many intents have already been created. Select one of the first results and click *Fork*.



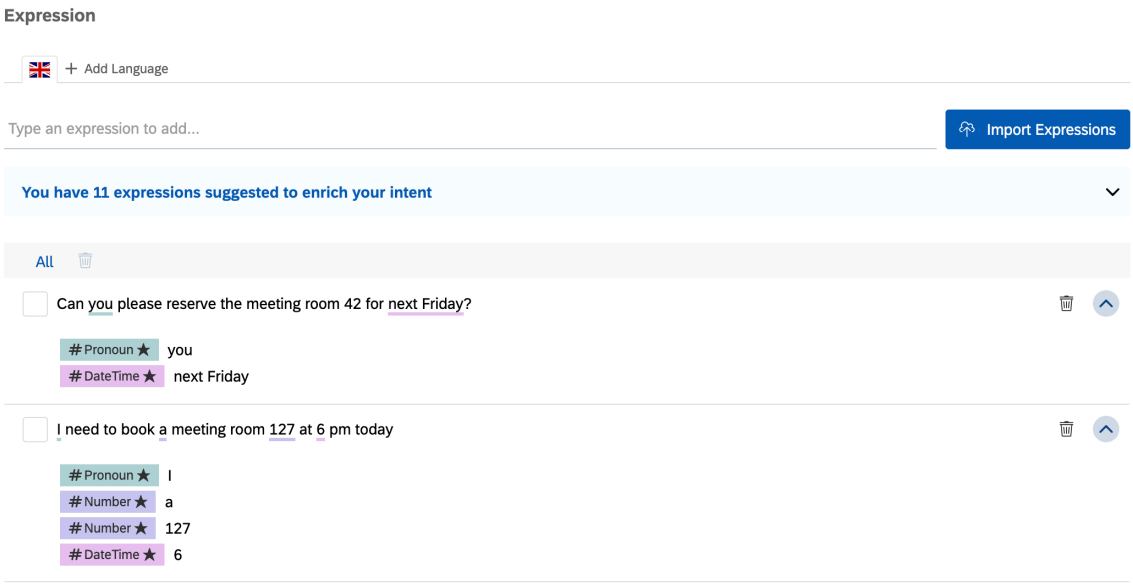
Add Expressions

Click the intent *booking* that you've just added to your bot. The optimal setting for an intent is to contain around twenty paraphrased expressions. Identify some expressions that your users are likely to say and add each expression to the intent by entering it in the *Type an expression to add...* field and pressing *Enter*. Based on the expression that you added, the system suggests more such expressions that help you to enrich your intent.

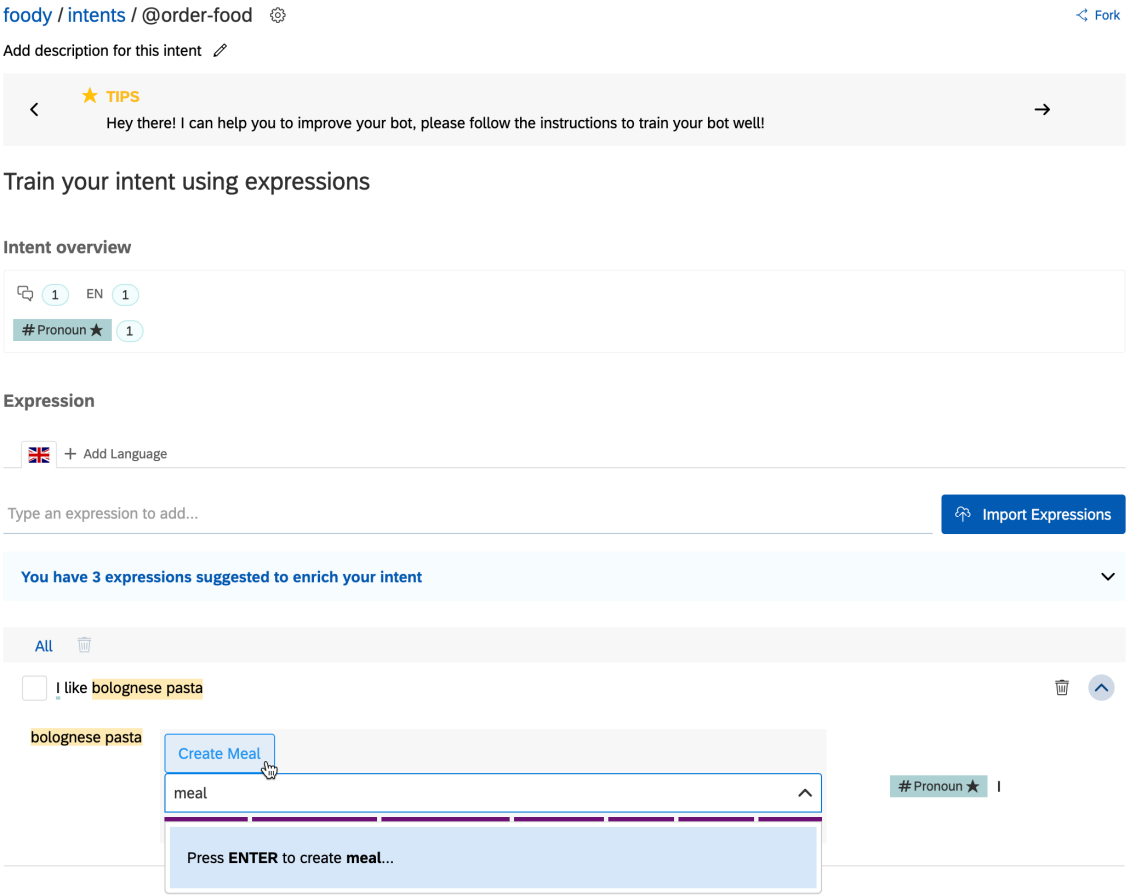


Use Entities

Go to the intent *booking*. If you click one of the expressions, you'll see highlighted words with tags. These are entities. Entities are keywords detected in expressions that are important to you in order to automate a task.



We automatically detect 28 different entities [page 55] such as *Datetime*, *Location*, *Person*, and so on. We call them gold entities. If you need another entity – for example, a custom entity like a meal for a cooking bot – just select what you want to tag as your new entity and type a name. The more examples you provide, the better the detection will be.



Training Mode

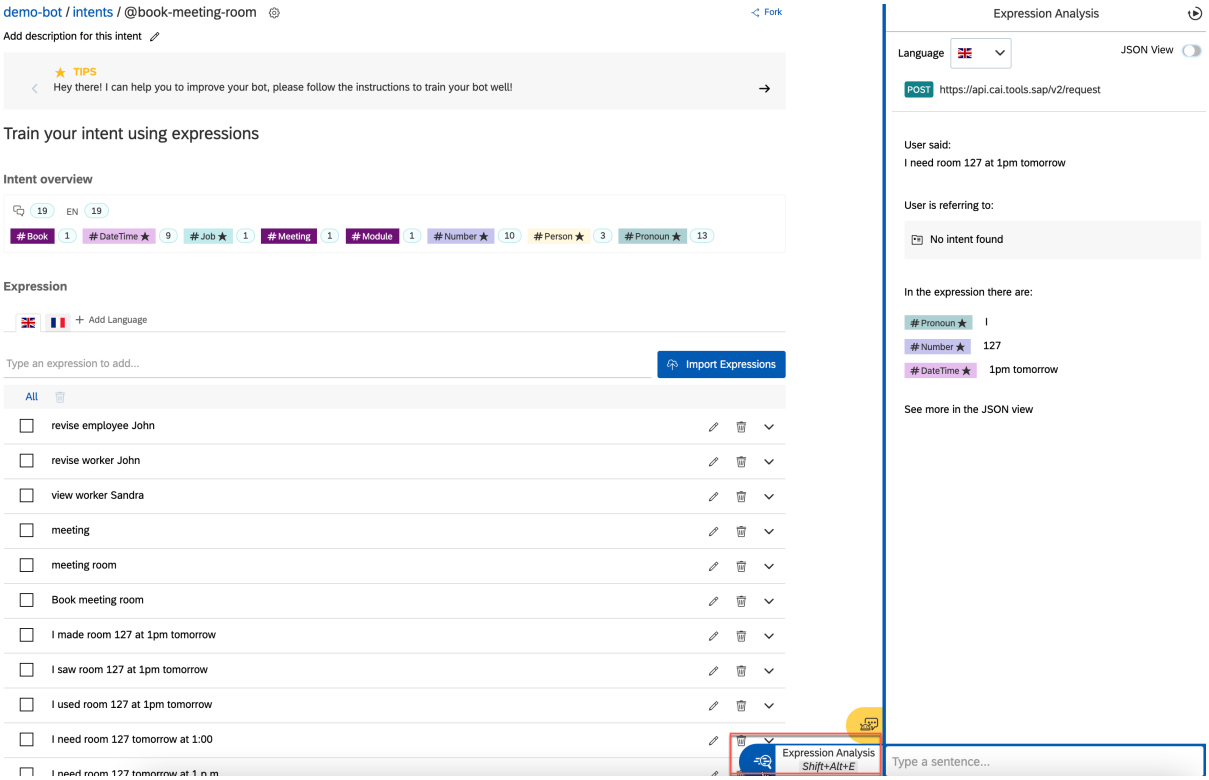
Unless your bot is a big bot, the default setting for the training mode is *Manual*. This means that you need to manually enforce training everytime you make changes to your dataset. If you wish, you can change the training mode to Automatic in your bot *Settings*. This lets you decide for yourself when you want to update the bot's training mode.

For big bots (that is, bots with more than 10,000 expressions or more than 15 custom entities), the training mode is always set to *Manual*. It cannot be changed to *Automatic*.

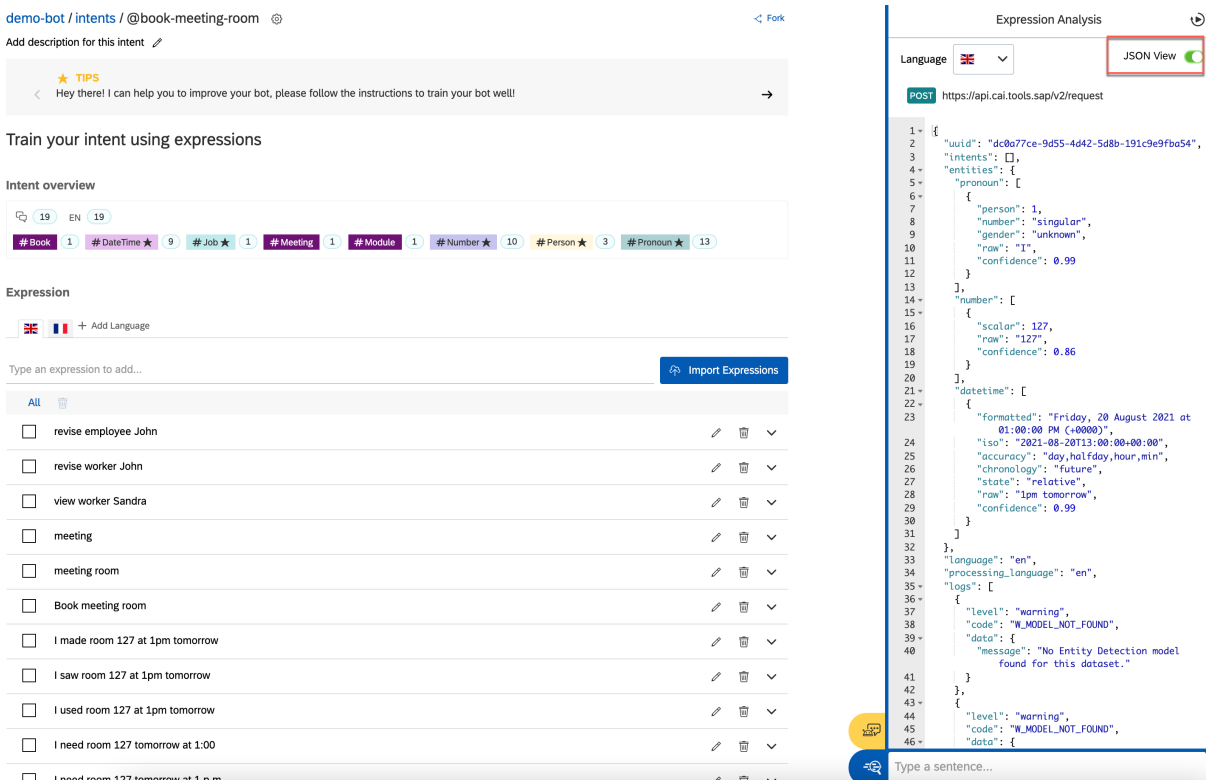
If you are using APIs to maintain your dataset, at the end of your script or program, you need to add an HTTP request to the /train endpoint in order to update the machine learning model.

Inspect (Test) with the Expression Analysis Console

Once you've created new intents, you can test them with the console. To display the console, click *Expression Analysis* at the bottom right of the page. To test whether your bot is well-trained, try typing `I want room 2 for tomorrow`.

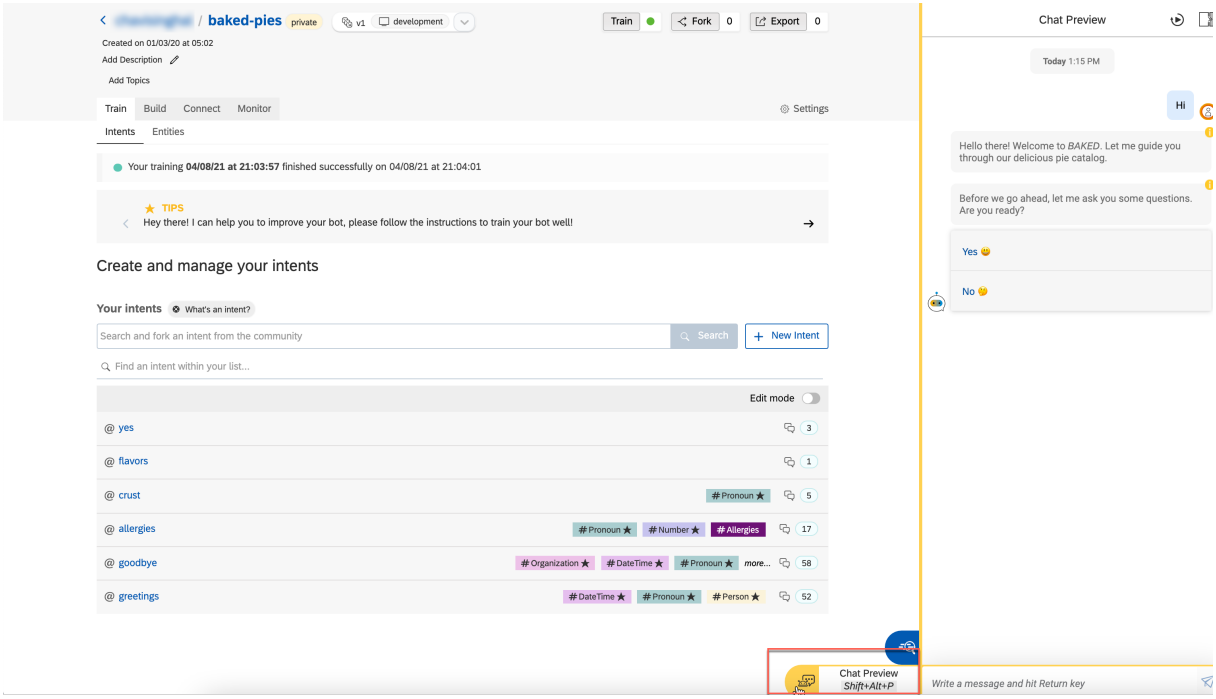


You can see which intent is detected and which entities are extracted. Switch on the *JSON View*. The JSON contains a lot of useful information about the message you've sent, such as all the enrichments we can provide for the gold entities.

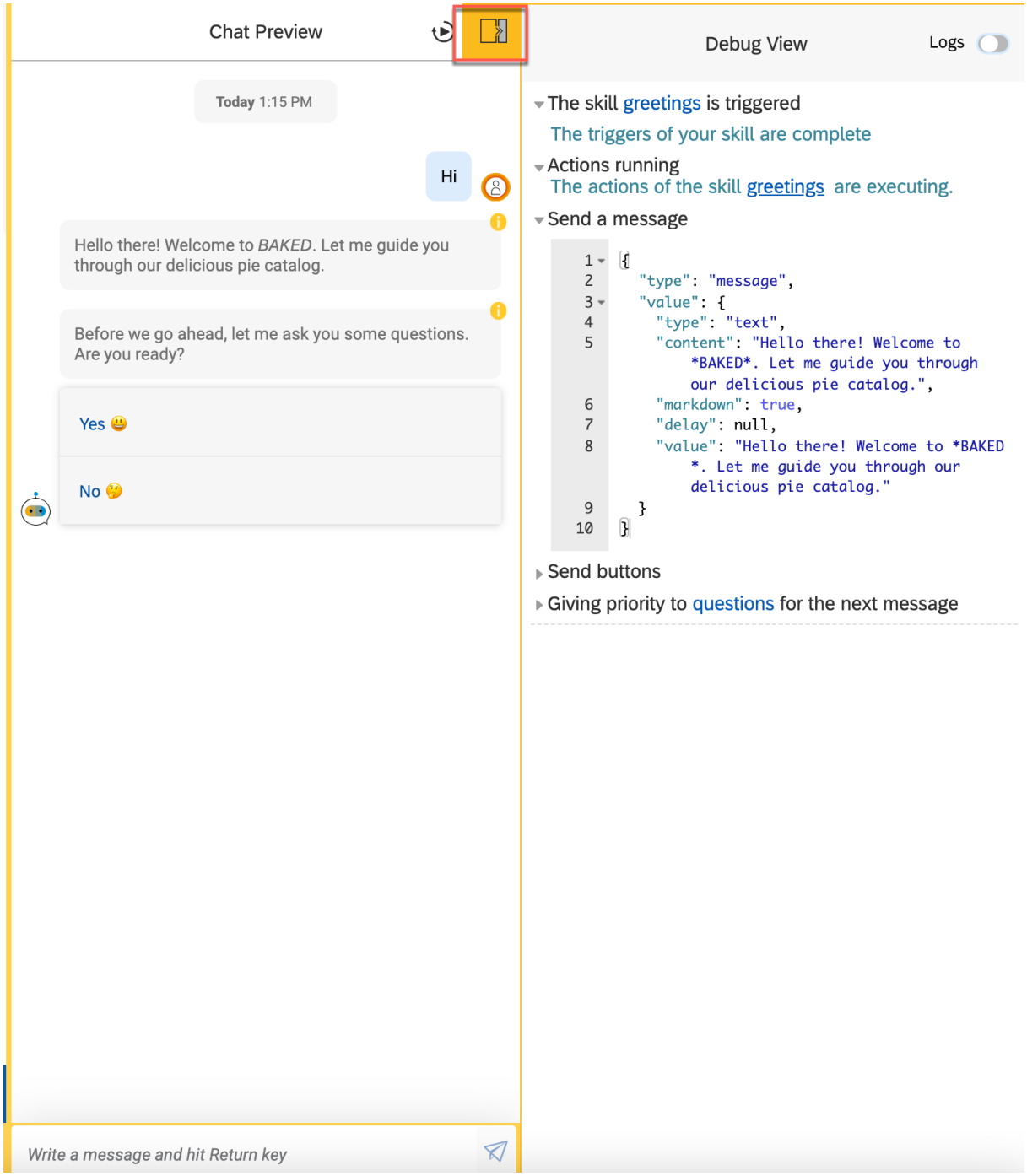


Chat with Your Bot

Once you have trained your bot and defined the skills, you can check if the bot responds as intended. Click *Chat Preview* at the bottom right and type a message to initate the conversation. Make sure that the message is included in your training dataset for your bot to respond as expected.



Open the *Debug View*. You can see which skill has been triggerd, the actions that are running, details of the message that you have defined and so on. This helps you to understand what went wrong if the bot didn't understand the question or a fallback skill was triggered.



Click the **i** icon next to a response. This opens the JSON view that shows all the details of the message such as -

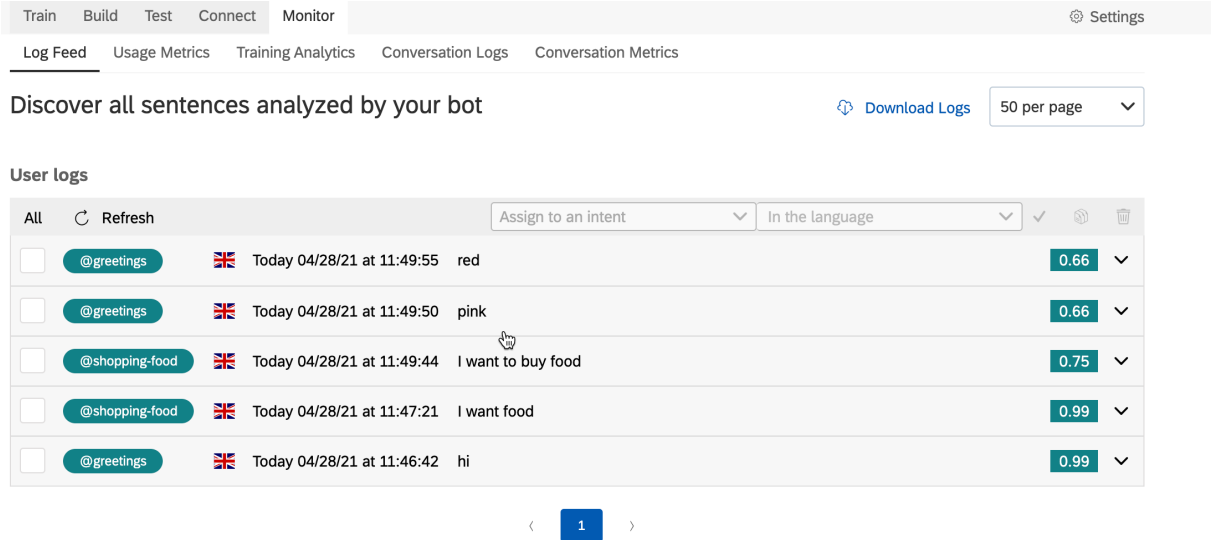
- the intents and entitites detected by the NLP
- details of the message that you have defined
- detected language and the processing language for the conversation
- memory that includes data stored in the conversation

JSON View


```
1   {
2     "nlp": {
3       "uuid": "cec1bb08-5010-4318-46f7
            -368496811f1f",
4       "source": "Hi",
5       "intents": [
6         {
7           "slug": "greetings",
8           "confidence": 0.99,
9           "description": "Says hello"
10        }
11      ],
12      "entities": {},
13      "act": null,
14      "sentiment": "vpositive",
15      "language": "un",
16      "processing_language": "en",
17      "version": "2108.0.0",
18      "timestamp": "2021-08-19T07:42:25.886877+00
            :00",
19      "status": 200,
20      "type": null,
21      "logs": []
22    },
23    "messages": [...],
66    "conversation": {
67      "id": "test-1629358920626",
68      "language": "en",
69      "memory": {},
70      "skill": "greetings",
71      "skill_occurences": 1,
72      "skill_id": "2cb000e6-8090-42b7-90cd
            -a3dd6ac4b33e",
73      "client_info": null
74    },
75    "logs": {...},
165   "qna": null,
166   "fallback": null,
167   "hasDelay": true,
168   "hasNextMessage": false
169 }
```

Train Your Bot

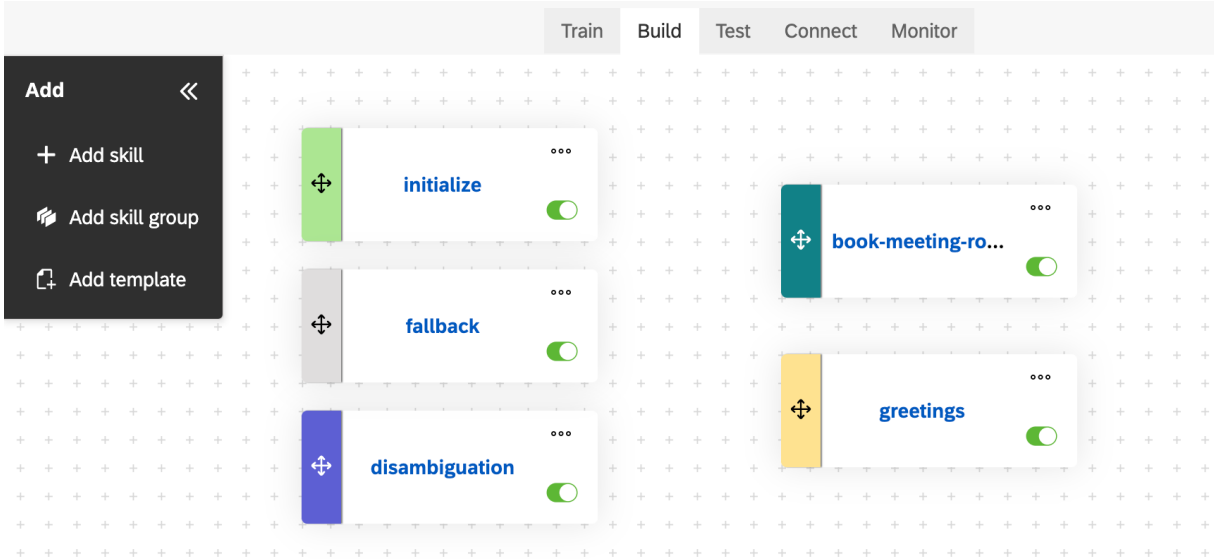
If your message doesn't match an intent, you need to train your bot. On the *Monitor* tab, click the *Log Feed* option to show the logs for your bot. Select the expressions that didn't match an intent and redirect them to the correct intent. Then check that your custom entities have been automatically tagged. If not, tag them – and remember: *Bot trained, mommy approved!* For more information, see Log Feed [page 224].



2.3 Build Your Conversation

Build

To find the predefined skills that you selected when creating your bot, click the *Build* tab. The gray panel on the left is your command panel. It lets you add new skills.

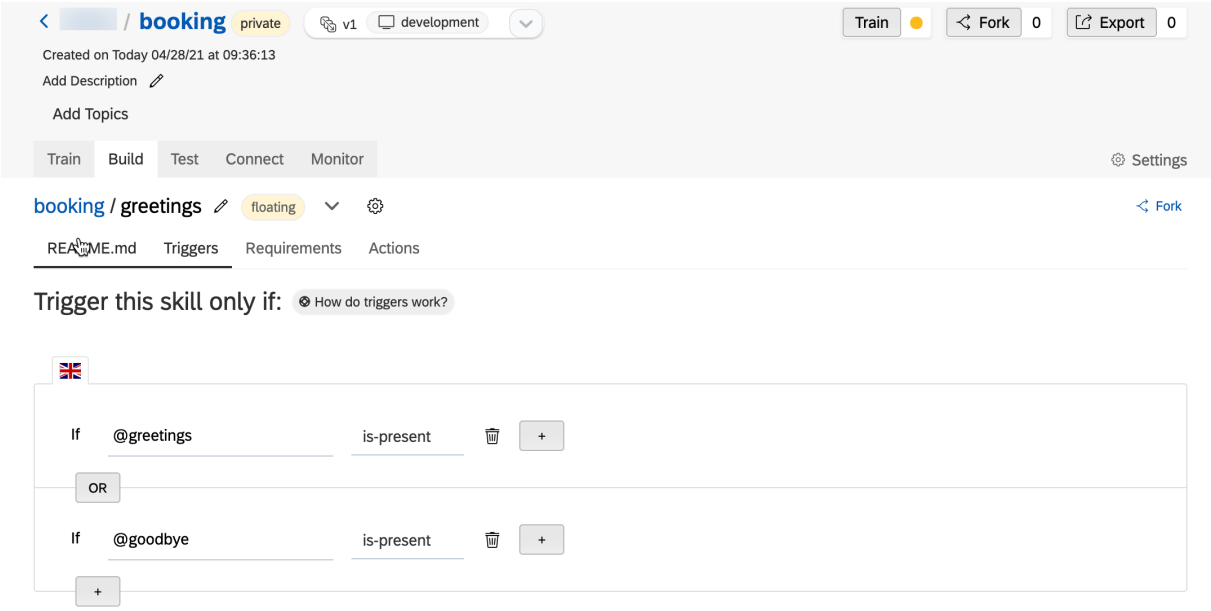


To explain what your skill does, which APIs it calls (if any), or even link a Git repository if your skill requires code, click the skill. A README.md opens, where you can enter this information.

Triggers

Triggers are the conditions that need to be completed for your skill to be initiated. You can define a wide range of conditions:

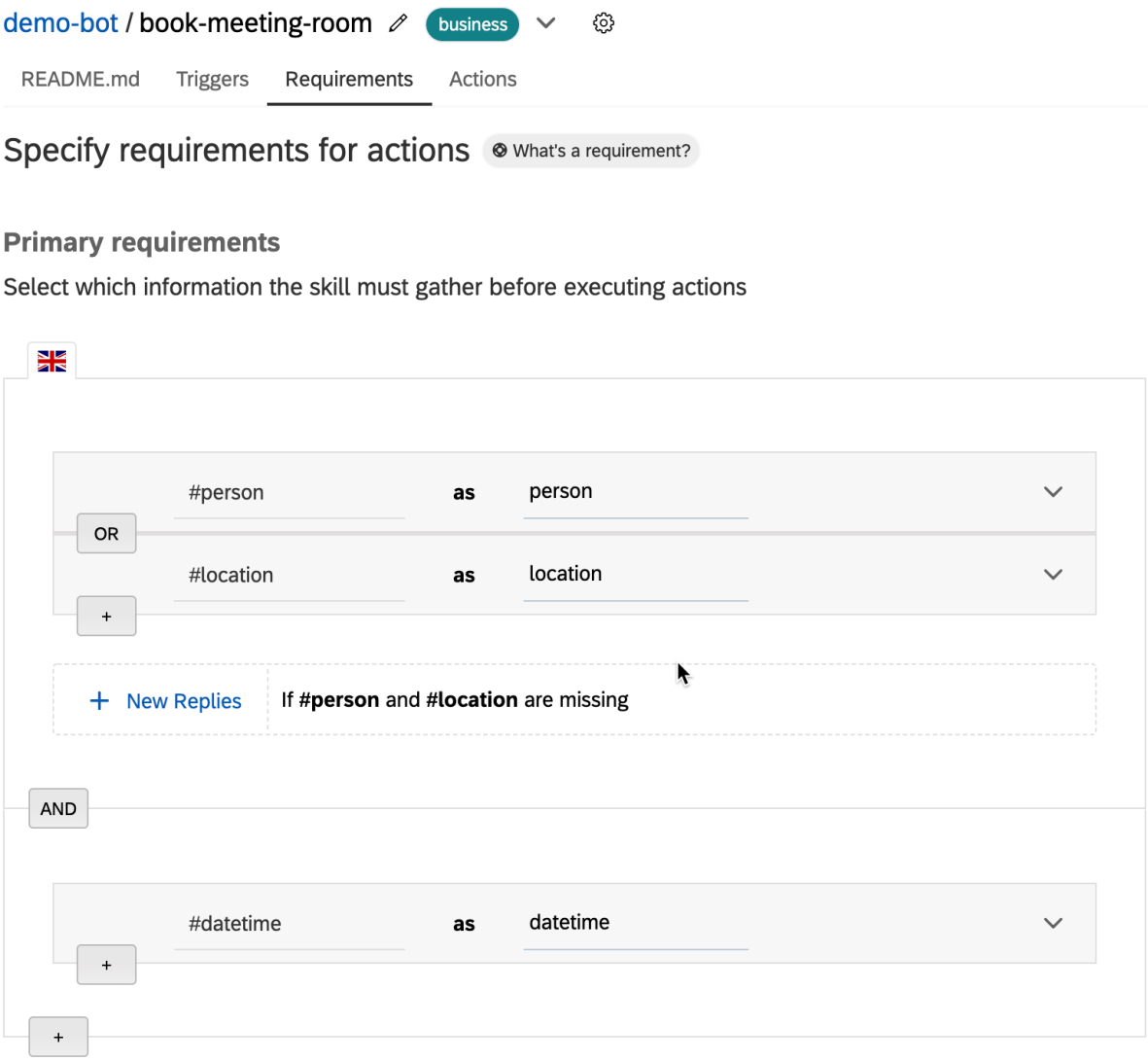
- Check which intent has been detected
- Check if a specific entity has been detected
- Check if the sentiment of the user sentence is positive or negative
- Check that a *#location* entity has been detected, and that its value is, for example, *San Francisco*, *Paris*, or *Singapore*.



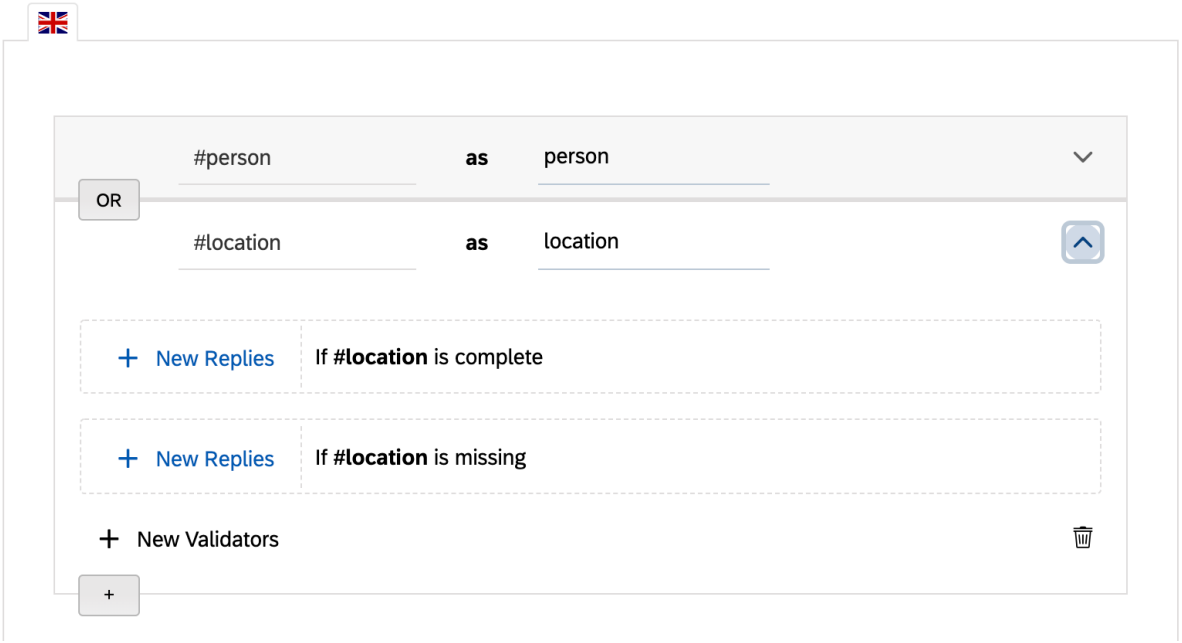
You can create up to two levels of conditions, and switch between an AND and an OR condition. An AND condition is true if every element of the condition is true; an OR condition is true if at least one of its elements is true.

Requirements

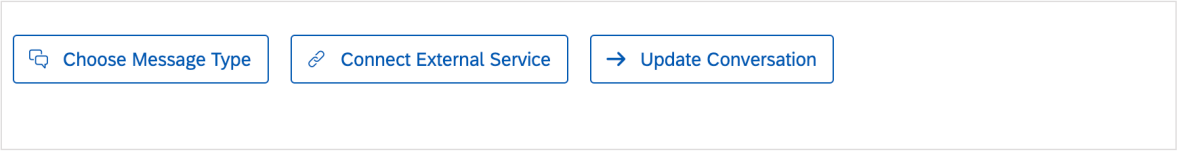
A requirement is a piece of information that your bot needs to have detected and saved in its memory before continuing the conversation. This section is executed once the triggers have been executed. You can require entities and intents. The second half of the requirement, after the *as*, is the alias of your requirement. It is under this name that you'll find it in your bot's memory.



To define how your bot asks for and then validates the requirement, click the expand icon to the right of the requirement. A settings panel opens.



If the requirement is missing (for example, *If #location is missing*), click the adjacent *+ NEW REPLIES* to define the actions that you want your bot to execute. You can send a message, call a webhook, or update the conversation: for example, by going to another skill.



Actions

Actions are things that your bot does at certain points when executing a skill. They can be the following:

- Message to send back to the user
- Connect to an external service (webhook or API)
- Fallback to a human agent
- Execution of another skill
- Update the memory of the current conversation
- Change the language
- Reset the conversation

Related Information

Skills [page 87]

Conditions [page 96]
Triggers [page 99]
Requirements [page 100]
Actions [page 108]

3 Natural Language Processing (NLP) Lexicon

3.1 Intents

Definition

An intent is a set of expressions that mean the same thing, but are constructed in different ways. Intents are central to your bot's understanding. Each one of your intents represents an idea that your bot is able to understand. You can add as many intents to your bot as you wish.

Tips

Make sure your intents are **distinct** enough to avoid misunderstandings and unnecessary confusion.

Balance your intents: Try to have the same number of expressions in each intent.

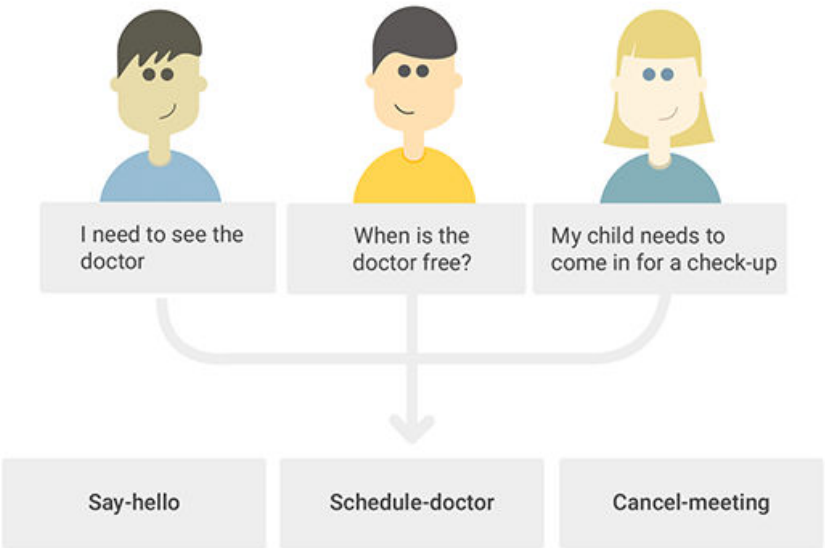
Diversify your intents: Use as many different grammatical structures as you can.

Make sure the expressions that you add under different intents are distinct enough to avoid intent confusion

Example

You want your bot to understand when someone asks for help. Just create an intent called *help* and fill it with every expression that a user might say when asking for guidance.

- Can you help me?
- I need some assistance.
- Need help
- What can you do?



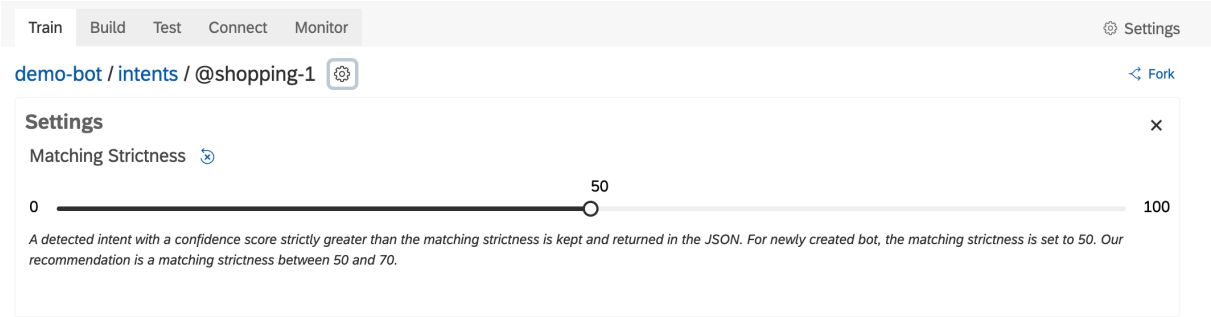
Intent Matching Strictness

You can define a strictness parameter for all intents collectively or individually.

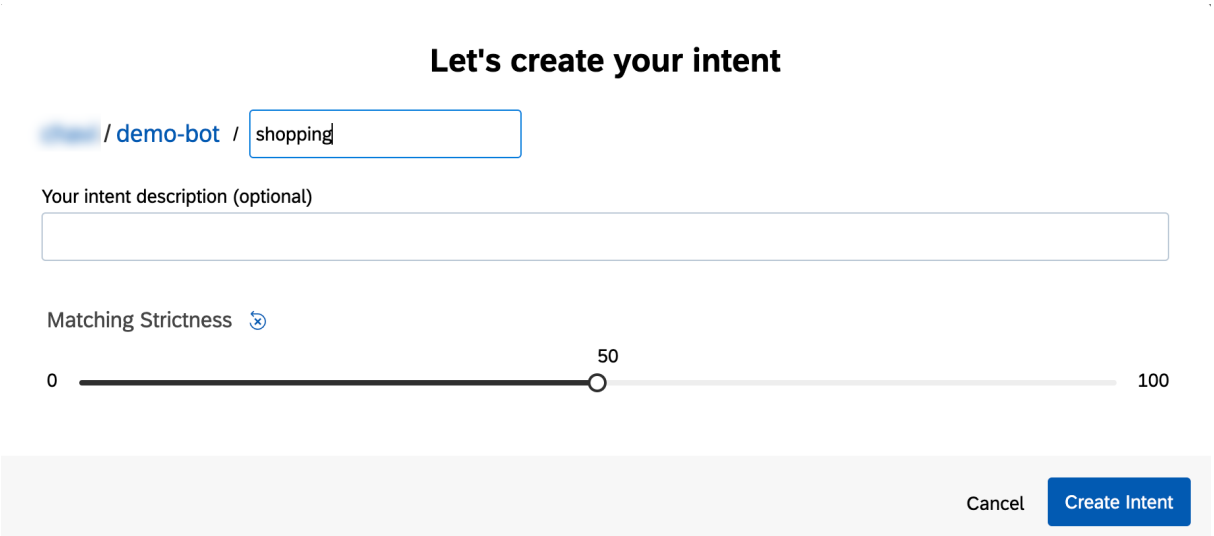
With the matching strictness of 80, an intent detected with confidence score 0.8 is filtered out and an intent detected with 0.81 confidence or higher is considered. With a strictness of 100, an expression within an intent must exactly match with the user's utterance to be detected as such. The default strictness is set to 50.

The matching strictness that you define in your bot *Settings* is applied to all the intents that have the default setting.

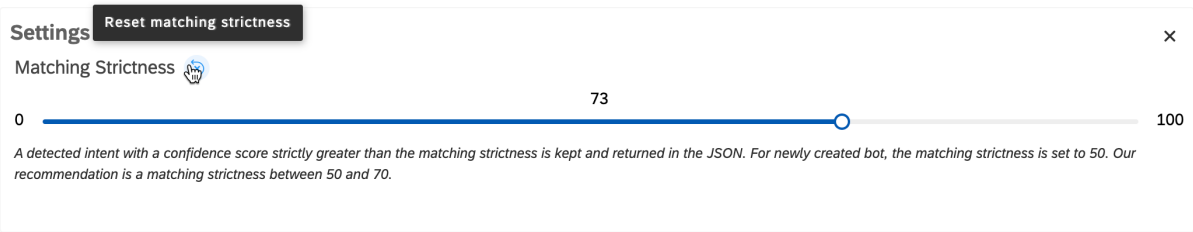
If you reset the matching strictness, the default value defined in the bot *Settings* is applied to all intents (with default or specific values).



When you create an intent, you have the option to define the matching strictness specific to this intent. By default, it is the value that is defined in your bot *Settings*.



Click the cog wheel icon, you can see the value of the matching strictness. You can change the matching strictness or reset the matching strictness to the value defined in your bot *Settings*.



3.2 Expressions

Definition

An expression is a sentence that your bot can understand – it's basically something that a user might say to your bot. Expressions are organized into intents and constitute the entire knowledge of your bot. The more expressions you have, the more precisely your bot can understand its users.

If you've added the languages English, French, German, or Spanish to an intent, and enter a new expression for the intent in any of those languages, SAP Conversational AI automatically suggests additional expressions in those languages. You can then easily add the suggested expressions to the intent and quickly build up the training dataset for your bot.

In an expression, you can annotate custom entities to train your bot to recognize key elements in your sentences. For more information, see Entities [page 31].

Tips

Put yourself in your users' shoes; imagine what they might ask your bot.

Keep your expressions to a reasonable length.

Have at least 30 expressions in an intent.

Train your bot with diversified expressions. In the following example, note how the expressions are structured differently. They try to anticipate the different ways that your user might ask for something. If all the expressions were structured the same way, for example, *I'd like a pizza*, *I'd like a hamburger*, *I'd like a salad*, your bot will have less success understanding the user.

Example

If the intent is `order-food`, some good expressions could be:

- I'd like to order a pizza.
- Can you get me some pasta?
- How about a salad?
- A veggie burger and fries would do nicely!

Importing Expressions with a CSV File

By importing expressions, you can speed up the bot development process.

Key	Required	Value	Description
expression	Yes	String	A sentence or word group
language	Yes	String	The ISO code for the language

Please format the CSV file as follows:

Sample Code

```
"expression";"language"
"I want to travel to NYC";"en"
"Let's travel to New York!";"en"
```

expression	language
I want to travel to NYC	en
Let's travel to New York!	en

When importing expressions, please note the following:

- You can import up to 10,000 expressions at the same time.
- Be sure not to exceed the file size limit of 1 MB.
- You cannot add the same expression multiple times to the same intent.

3.3 Entities

Definition

An entity is a keyword that is extracted from an expression. We automatically detect 28 different entities [page 55] such as *Datetime*, *Location*, *Person*, and so on. We call them gold entities. However, you're not limited to these gold entities. You can also tag your own custom entities to detect keywords depending on your bot's context: for example, subway stations if you're building a transport assistant.

Gold Entities

All gold entities are detected automatically. This means that you can't deactivate them, however, you can train them. To provide a precise bot response, you can enrich each gold entities with desired values. For example, when the gold entity *tomorrow* is detected in a sentence, a formatted version of the *datetime* that you can use as a reply is returned.

> **Note**
>
> It is not possible to turn off gold entities detection. To avoid gold entity detection, try to improve your training dataset so that a custom entity is detected instead of a gold one.

> **Sample Code**

```
{
  "formatted": "Thursday, 06 October 2018 at 09:00:00 AM",
  "iso": "2018-10-06T09:00:00Z",
  "accuracy": "day",
  "chronology": "future",
  "raw": "tomorrow",
  "confidence": 0.92
}
```

See all gold entities [page 55] and their enrichment.

Custom Entities

You don't have to tag everything in your expressions. Just annotate what really **needs** to be extracted. You can use custom entities for three different reasons:

- You want to detect all possible occurrences of something in a sentence. For example, you're building a transport bot and you want to detect all subway stations.
- You want to understand if something is present or not in a sentence.
- Entities have an influence on intent detection. If you still can't get a good intent classification after you have followed our guidance of how to create a good dataset, you could create a custom entity unique to an intent to facilitate this intent's detection.

In the *Train* tab, go to *Entities* and click *CREATE*. Provide a suitable name for the entity, choose the type and click *CREATE*. Custom entities can be **free** or **restricted**.

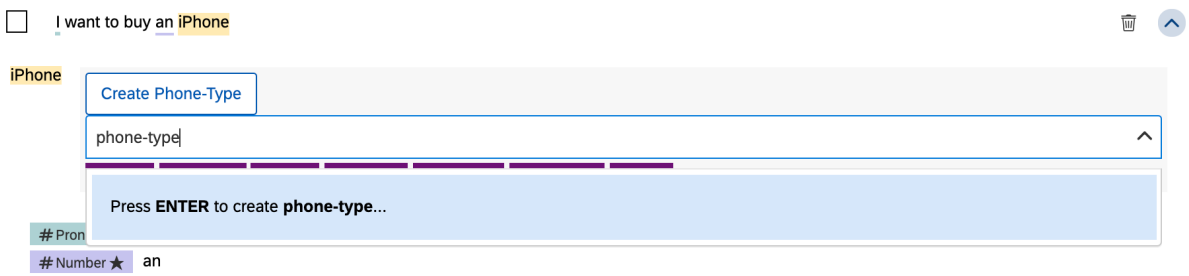
Free Custom Entities

You use a free custom entity if you don't have a strict list of values and you want machine learning to detect all possible values. For example, you want to detect book titles.

These entities are detected through machine learning. This means that you need to provide examples of the characteristics to train the detection, that is, provide possible values and the way the entity is used in a sentence.

To Train a Free Custom Entity

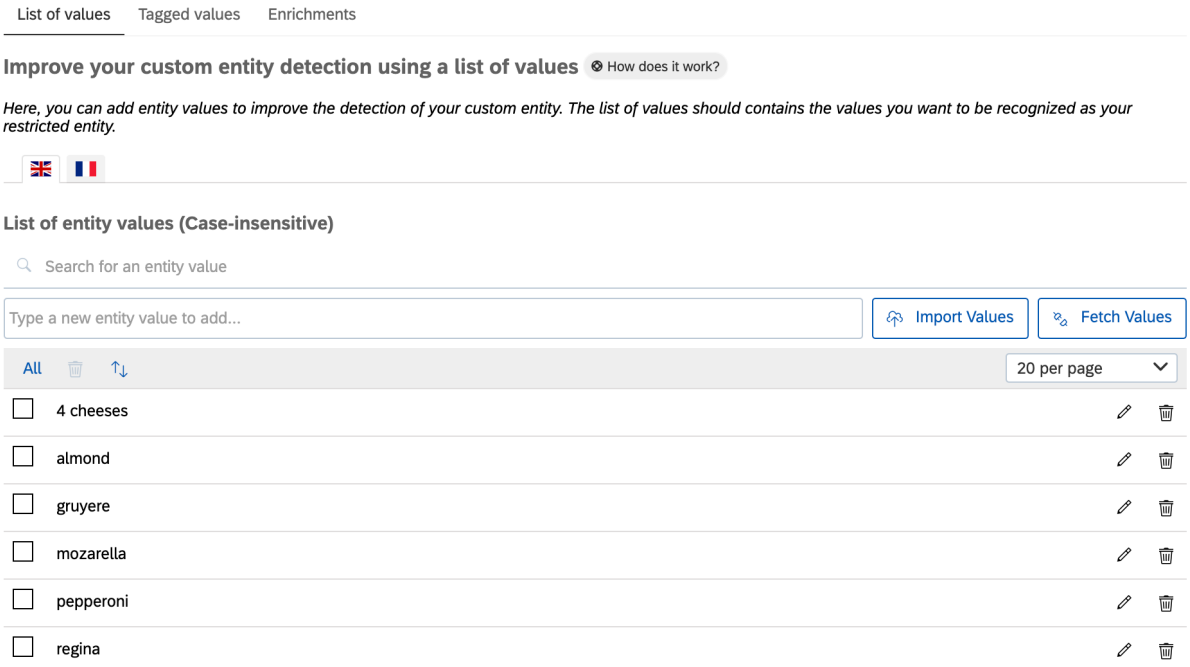
In your intent, tag the appropriate words (by highlighting a word or group of words, and adding the entity label). Annotate it in each expression and continue to add expressions until your entity is detected automatically.



Restricted Custom Entities

You use a restricted custom entity if you have a strict list of words to detect and don't need automatic detection of the entity. No word can be recognized as an entity if it doesn't appear in a closed list of values. For example, you build a bot to help your customers order pizza. You want to detect all pizza names that your restaurant offers.

Create a restricted custom entity and then add values for this entity. You can also upload a CSV file or use the */gazette* endpoint of the API to quickly create a large list of values.



You still need to tag these values in the dataset, but if you tag values other than those in *List of values*, it will not help with the entity or intent detection.

Regular Expression (regex) Entities

A regular expression (regex) entity extracts entity values based on a pattern that needs to be detected. By using regex entity, you don't need to create a strict list of all words, which can be inconvenient in case of huge list of values, nor need to train the entity to be detected by machine learning.

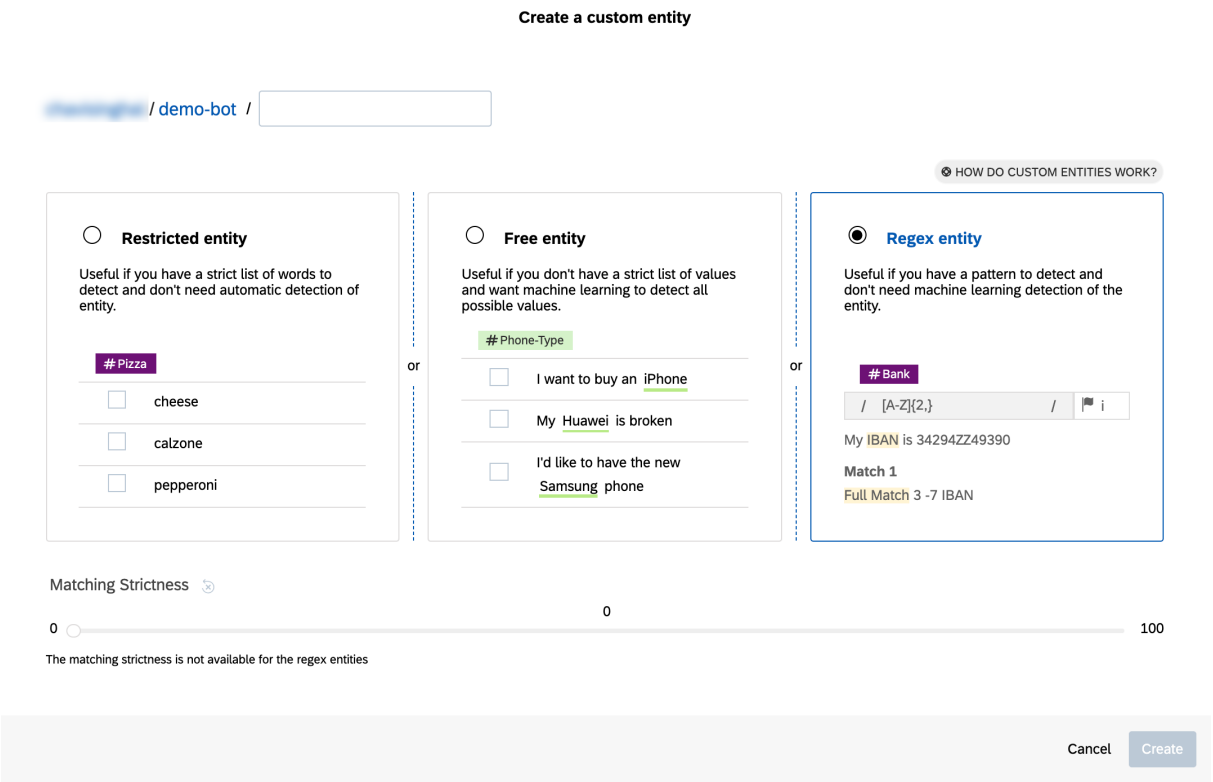
For example, you want to detect product-ids that matches 3 letters followed by 4 digits. This would be `[A-Z]{3}[0-9]{4}`.

These entities are automatically detected based on the defined pattern, therefore it is not possible to tag regex entity in the dataset and no list of values are needed.

> **i Note**
>
> It is not possible to change a regex entity to another custom entity type (free or restricted), and vice-versa.

Create a Regex Pattern

When you create a new entity, choose *Regex Entity* option as shown in the following figure:



Once created, you can define your regex pattern and use the supported python flags.

A default pattern to match an entire token value has been already applied to your regex. This is important as during entity detection, regex entities have higher priority than other types of entities (gold, free, restricted).

For more information about regex operations, see `https://docs.python.org/3/library/re.html`.

Once the pattern is saved, you can test your pattern and check if it works as expected. In case the regex matching takes more than 100 ms during runtime, this entity detection will be aborted, and an error is visible in JSON logs and also in the Logfeed.

Recommendations

- Make sure that your regex entity pattern is unique and doesn't match with another regex entity pattern. In case a token matches several regex entities, then the longest match is returned. But in case of equal length, the entity that has the oldest creation date is returned.
- In case of several matches, regex entity overrides gold, free and restricted entities.

Mapping Enrichments

For regex entity values, you can configure static enrichments or fetch them using an API.

The entity values defined in groups are automatically created and there is no need to for you define the list of values manually. If an entity value created within a group doesn't match the regex pattern, then an error notification is displayed.

In case you modify the regex pattern and the defined entity values don't match the updated pattern anymore, a red dot is displayed in the *Enrichments* tab. This means that you need to update your entity values in the *Enrichments* tab.

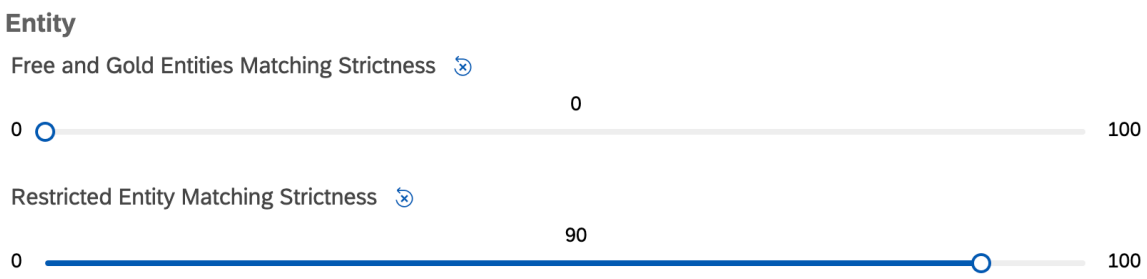
Entity Matching Strictness

You can define a strictness parameter for all types of entities (free, gold and restricted) that determines if a word matches a given value in your list. For example, if you have the restricted entity #PIZZA with values like *margherita* and *pepperoni*, your user may type **`margarita`** or **`peperoni`**. By adjusting the matching strictness, you can define if *margarita* and *peperoni* should be considered as #PIZZA or not.

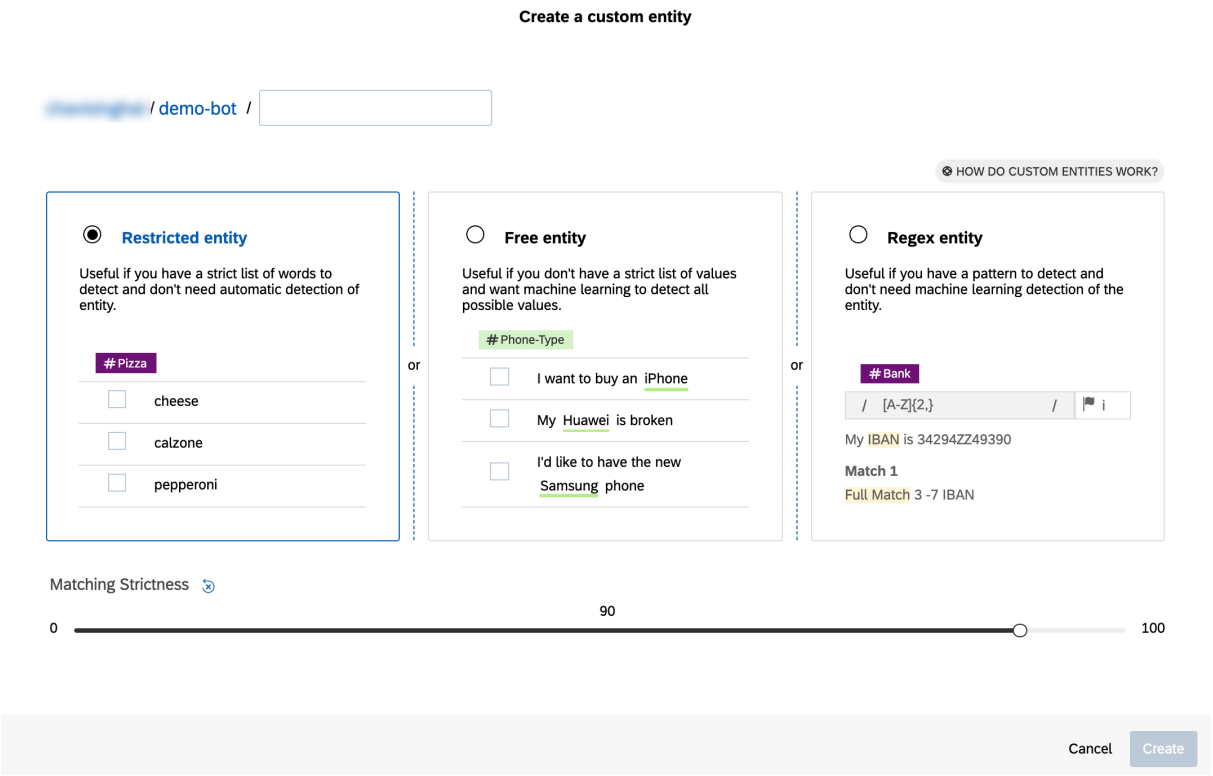
With the matching strictness of 80, an entity detected with confidence score 0.8 is filtered out and an entity detected with 0.81 confidence or higher is considered. With a strictness of 100, a word must exactly match an entry in the list to be detected as such. The default strictness for a restricted entity is set to 90.

The value defined in your bot *Settings* under NLP, corresponds to a default value that is applied to all entities. The value that is defined in the entity settings of a particular entity, corresponds to a specific value that overrides the default one.

If you reset the matching strictness, the default value defined in the bot *Settings* is applied to all entities (with default or specific values).



When you create an entity, you have the option to define the matching strictness specific to this entity. By default, it is the value that is defined in your bot *Settings*.



Click the cog wheel icon to see the matching strictness for the entity. You can change the value of the matching strictness or reset the matching strictness to the value defined in your bot *Settings*.

3.3.1 Importing Entity Values

Importing Entity Values

Depending on your business need, you may have a large number of entity values with which you want to train your custom entity. If you have a list of entity values saved in a CSV file or stored in an external database, you can import these entity values using a CSV file or a service API.

Import Values

To import entity values, you need to specify the actual entity value as well as the ISO code for the language of the value.

Key	Required	Value	Description
value	Yes	String	The entity value
language	Yes	String	The ISO code for the language

Please format the CSV file as follows:

Sample Code

```
"value";"language"
"The Big Apple";"en"
"NYC";"en"
"New York";"en"
"New York City";"en"
"la grande pomme";"fr"
"nou yorke";"fr"
```

value	language
NYC	en
The Big Apple	en
la grande pomme	fr

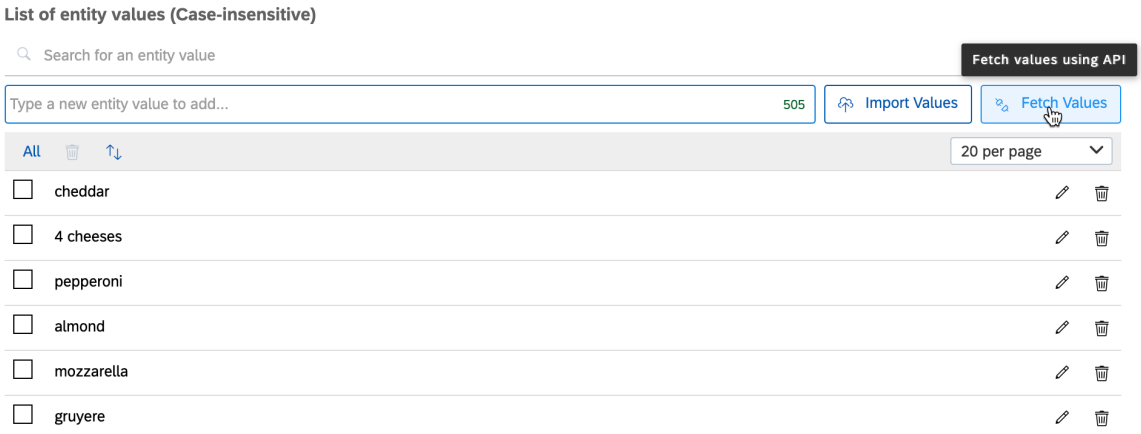
When importing entity values, please note the following:

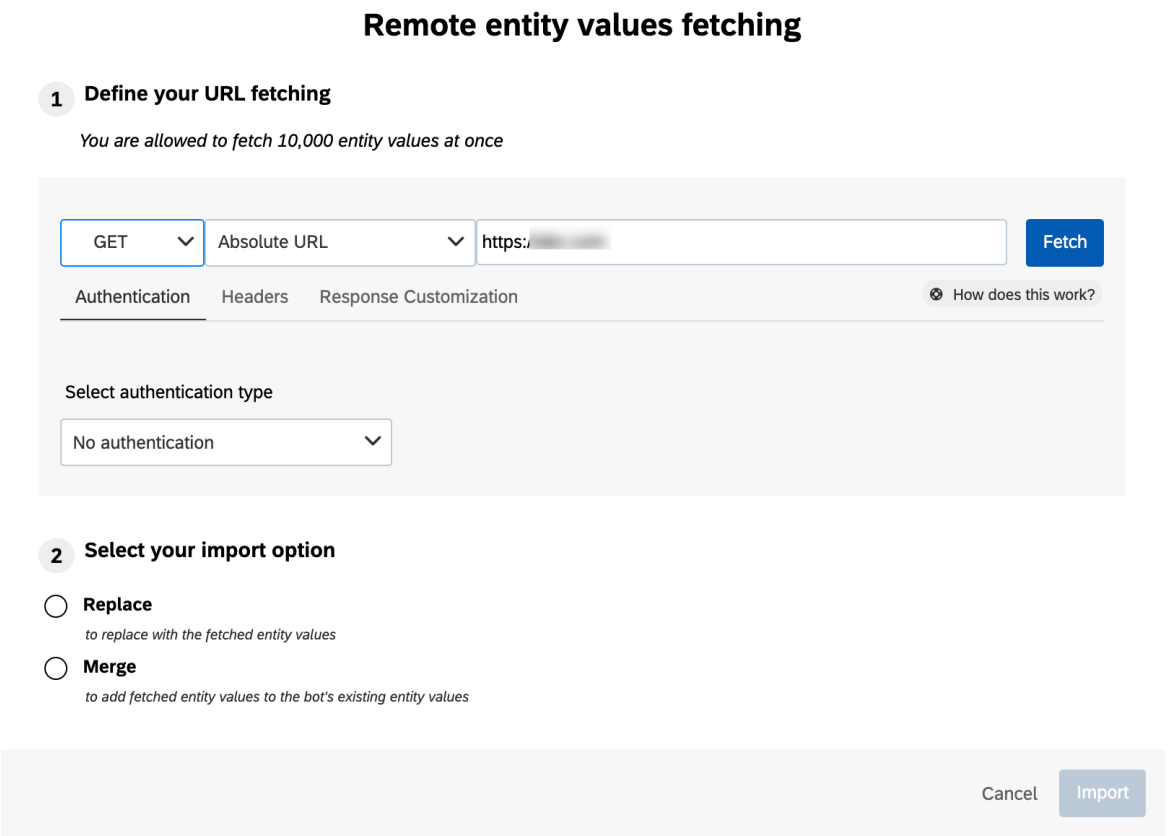
- You can import up to 10,000 entity values at the same time.
- Be sure not to exceed the file size limit of 1 MB.
- The import process using the merge option is not executed if the value of the synonym already exists.

Fetch Values

To fetch entity values, you need to configure your service API.

1. Click *Fetch Values* in the List of values tab. For more information on how to configure a service API, see Connect to External Service [page 136] and to know how to configure the response for fetching entity values, see Configuring Response for Fetching Entity Values [page 149].
   When using your entity in multiple languages, you can optionally use the variable {{`language`}} in the URL, headers or body in order to pass the currently selected language (for example, “en”) to the external service.

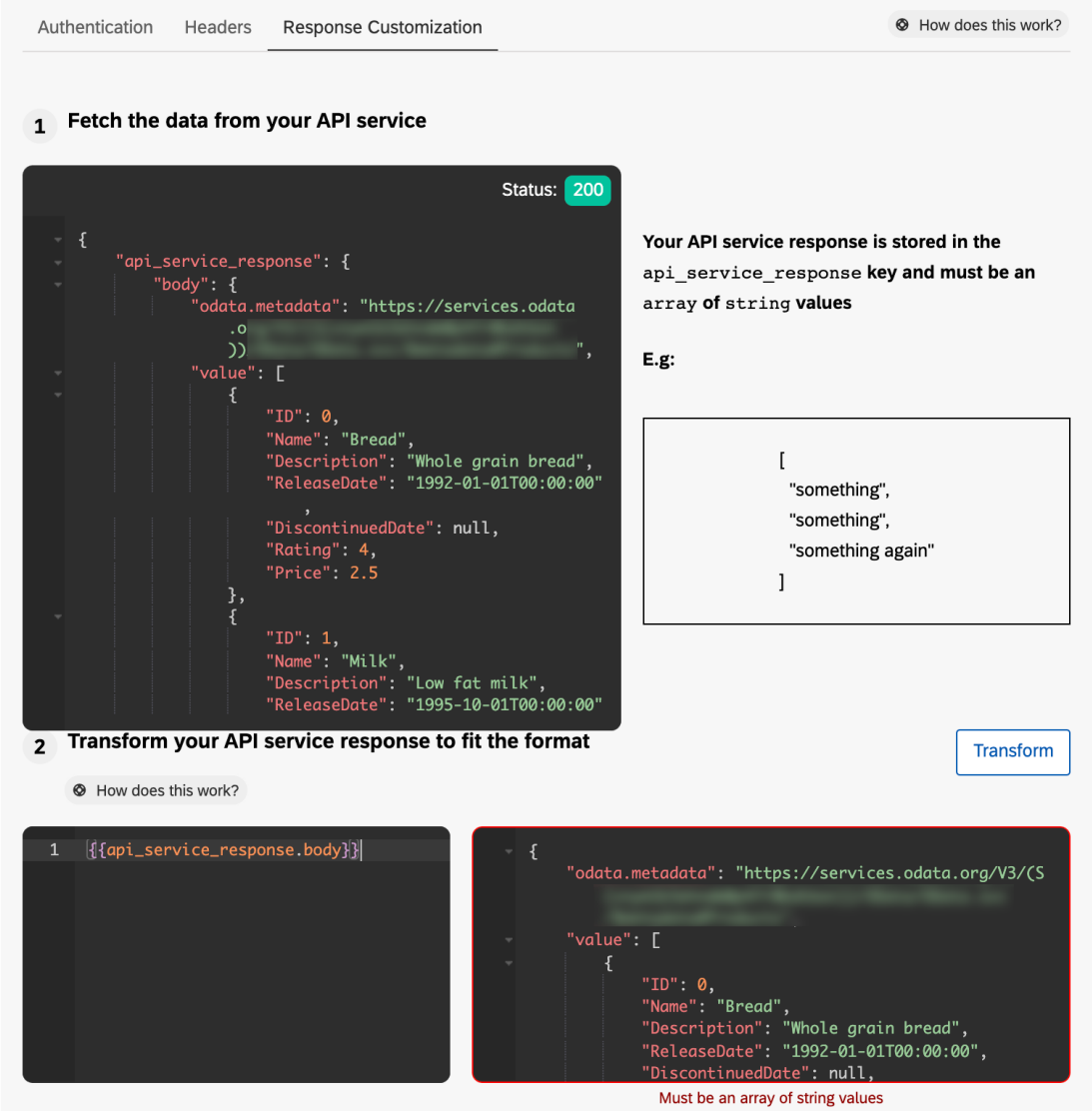




> **i Note**
>
> You can configure a destination only if you are using the Enterprise edition of SAP Conversational AI. This is not supported in the Community edition. For more information see Configuring the Enterprise Edition.

2. Configure the authentication, headers, and body of the API and click *Fetch*.
   The entity values are fetched using a technical user that is authorized for this even if the current business user is not allowed to see them.
   Under the *Response Customization* tab, you can see the API service response (including errors).

The result of the object from the connection needs to be an array of strings.

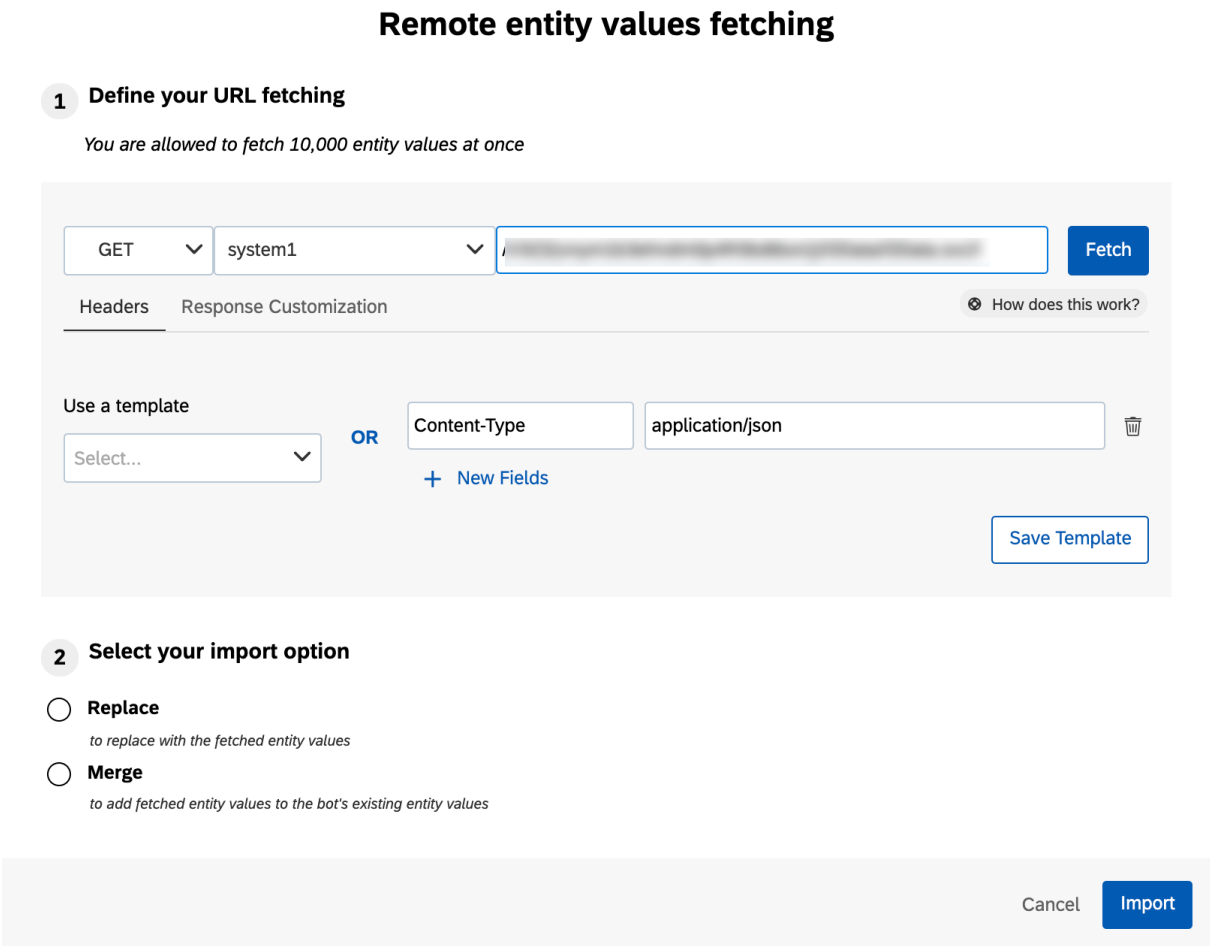


You can edit the script to make sure that the final response is an array of strings and click *Transform*.

3. Once you are satisfied with the final transformed API service response, choose if you want to replace the existing entity values with the latest ones or merge them to the existing list of values.
4. Click *Import*.
   All the entity values are imported into the platform.

Using System Aliases

If you have configured system aliases, you need to select the appropriate system that gets prepended to the service API. For more information, see System Alias Configuration [page 140].



When importing entity values, please note the following:

- You can import up to 10,000 entity values at the same time.
- The import process using the merge option is not executed if the value of the entity already exists.
- If there are more than 10,000 entity values, only the first 10,000 will be imported

> **Note**
>
> When you fork an entity, the values that have been added using the service API, are also forked.

3.3.2 Enrichments for Custom Entities

Enrichments for Custom Entities

Whenever an entity is detected, the JSON returned by the NLP API is enriched with additional information about the entity. For example, the following JSON is for a `datetime`, which is a gold entity.

> **Sample Code**

```
{
  "formatted": "Thursday, 06 October 2018 at 09:00:00 AM",
  "iso": "2018-10-06T09:00:00Z",
  "accuracy": "day",
  "chronology": "future",
  "raw": "tomorrow",
  "confidence": 0.92
}
```

You can configure additional enrichments for custom entities and gold entities. For example, you create the custom entity `#CHEESE` for your shopping assistant. When *Cheddar* is detected in a sentence, you could have this JSON:

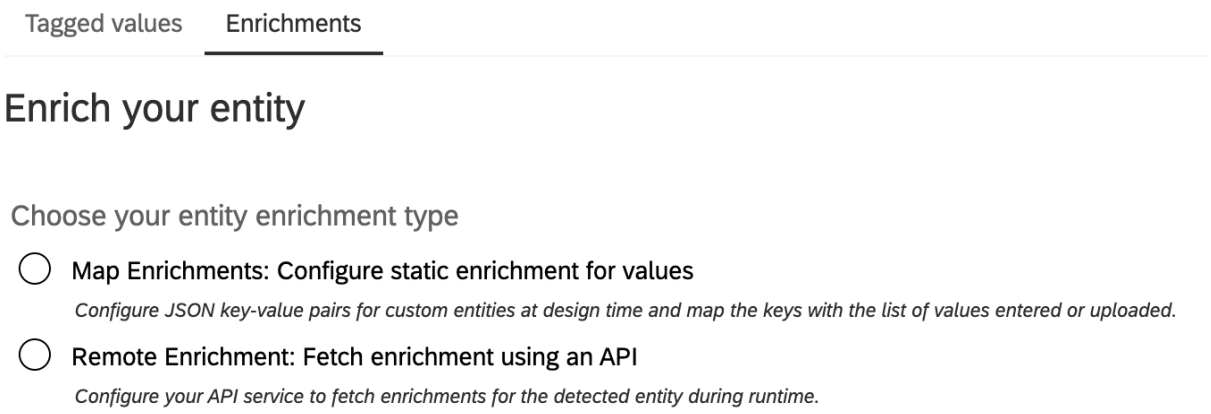
> **Sample Code**

```
{
  "value": "cheddar",
  "raw": "cheddar",
  "origin": "USA, Wisconsin",
  "price": "$1.30",
  "confidence": 0.92
}
```

Fetching Entity Value Enrichments

A custom entity can have a lot of different values and enrichments. You can choose to add static enrichments for values by manually entering them. If you have a lot of enrichments saved in an external system, you can configure service API to fetch them. During a conversation, when an entity is detected, an API call is made to the corresponding business service and enrichments for the given entity value are retrieved.

Based on your requirement, choose the appropriate option. The option needs to be chosen for each entity and is applied for all languages.



Configure Static Enrichments

Using this configuration, you can:

- Add enrichments for gold entities
- Add enrichments for custom entities

Add Enrichments for Gold Entities

You can add gold enrichments to custom entities to create more specific entities. For example, you can define a custom entity `#DISCOUNT` and use the `#PERCENT` enrichment to extract the scalar or add `#LOCATION` enrichment for a city.

Choose a gold entity

Location ★

Key	Value
" country " :	value retrieved during runtime
" state " :	value retrieved during runtime
" city " :	value retrieved during runtime
" postal_code " :	value retrieved during runtime
" street_name " :	value retrieved during runtime
" street_number " :	value retrieved during runtime
" place " :	value retrieved during runtime
" type " :	value retrieved during runtime
" lng " :	value retrieved during runtime
" lat " :	value retrieved during runtime
" formatted " :	value retrieved during runtime

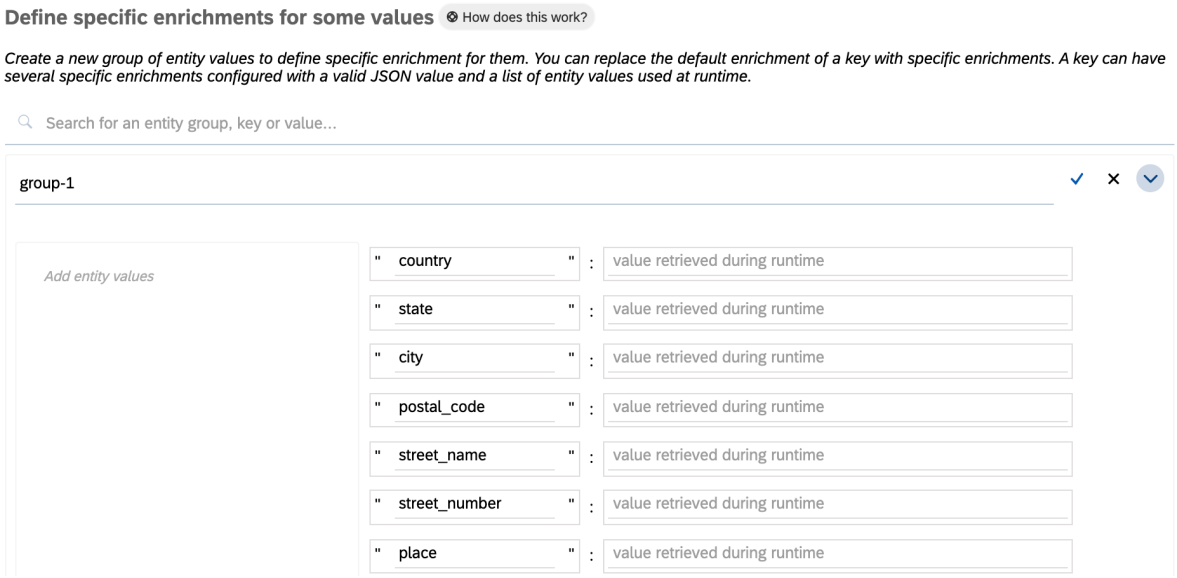
! Restriction

You can add gold enrichments along with static enrichments only and not in addition to enrichments fetched by an API service.

You can add enrichments from one gold entity for a given custom entity.

If you add a new group for your custom entities, all gold enrichment keys are added to this group. You can set a specific value for each enrichment key.

When deleting a gold enrichment, the corresponding keys are moved from gold to custom enrichment keys. If not needed anymore, you need to manually delete each key individually.



For a free entity, the list of entity values is free and created manually. Additionally, you can configure a matching strictness for a free entity.

You can refer to this blog for more information.

Add Enrichments for Custom Entities

You do this configuration in two steps:

1. Define new JSON {key, default value} pairs (like `origin` and `price` in this example).
2. Create group of entity values and define enrichments for these keys (for example, the desired price). Note that the custom entity enrichments are applied to the list of entity values defined in a group.

{key, default value} pair

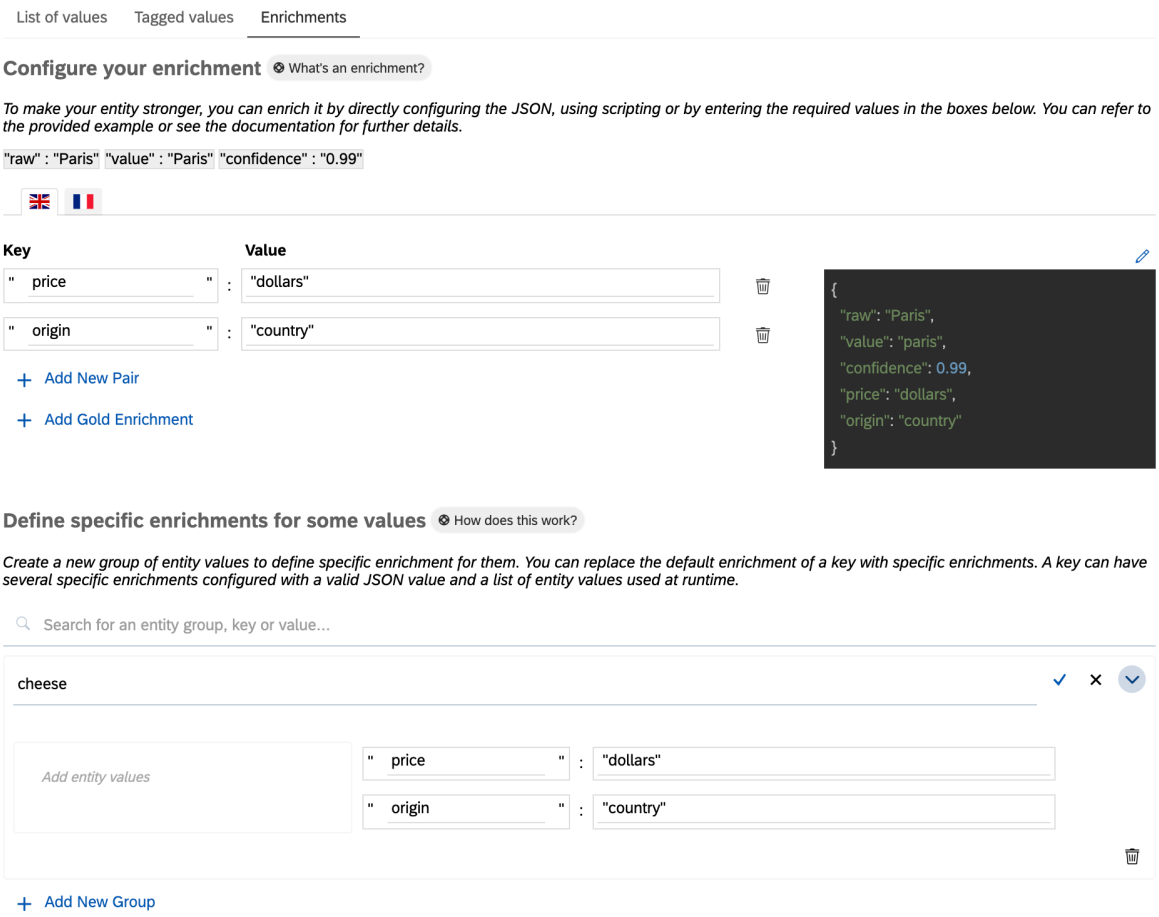
You can create new JSON {key, default value} pairs by providing a name and a default enrichment.

An enrichment value must be a valid JSON value .

Keys are language-independent, while enrichments are language-dependent. For example, if you create the key `price`, it will always be present in your JSON in all languages. If you don't define an enrichment for this key, null will be sent, for example, `{ "price": null }`.

Create and Enrich Groups of Entities

You can create a group of entity values by providing a name for the group and adding entity values (at least one value is needed).



Once the group is created, all the {key, default value} pairs created in the first step are assigned with the default values.

The default enrichment for a key can be overridden with the defined enrichments. A single key can have several enrichments, but an entity value cannot belong to several groups.

An enrichment is configured with:

- A valid JSON value
- A list of entity values

The list of entity values is used at runtime. When a custom entity is detected, the corresponding value is compared to this list of entity values to decide which enrichment should be applied. For example, in the case of our entity `#CHEESE` and its enrichments, if the value *Mozzarella* is detected in a sentence, the enriched JSON is as follows:

Sample Code

```
{
  "raw": "mozzarella",
  "value": "mozzarella",
  "deliciousness": -10,
  "confidence": 0.92
}
```

For a restricted entity, the list of entity values is a subset of the entity values.

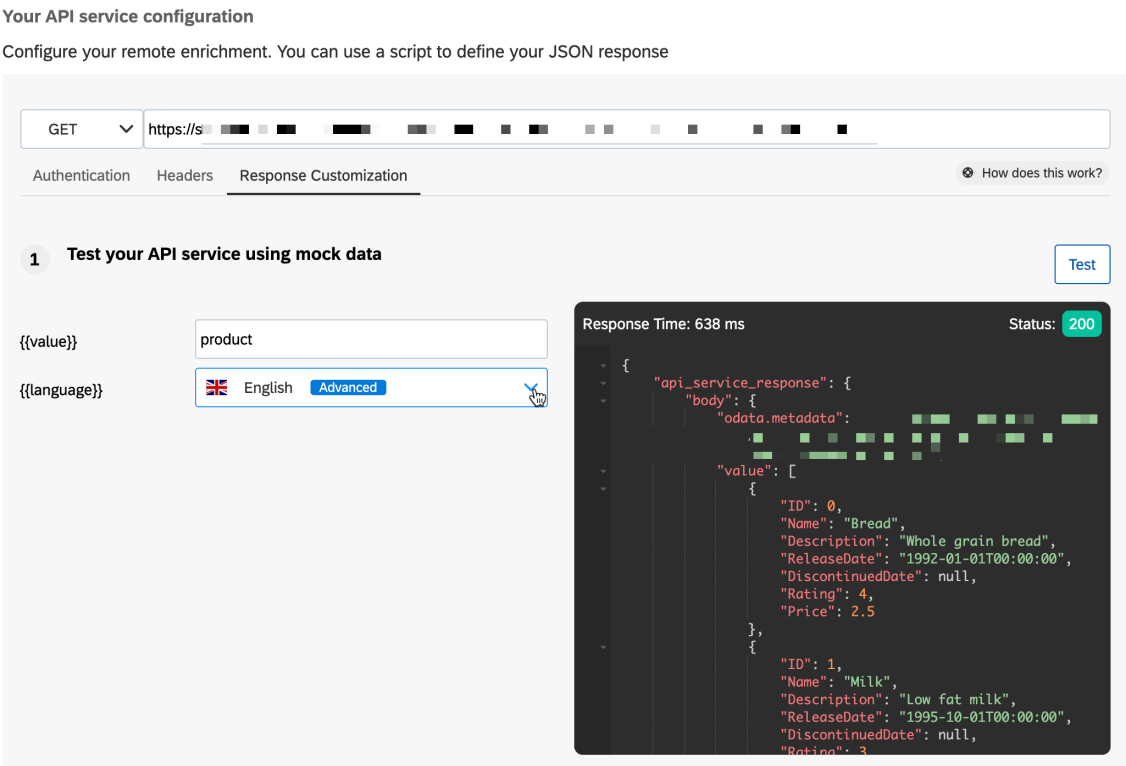
Fetch Enrichments Using Service API

To fetch entity value enrichments, you need to configure your service API.

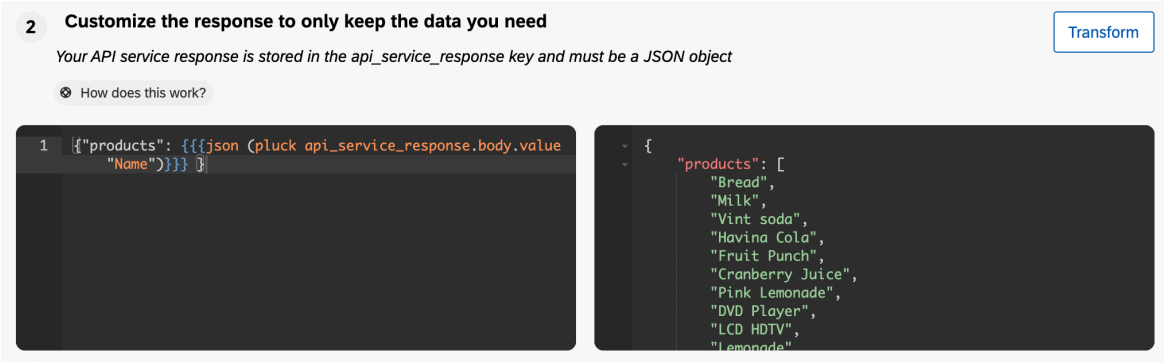
1. Provide the URL of your service API and choose *SAVE*. For more information on how to configure a service API, see Connect to External Service [page 136].

   > **i Note**
   >
   > You can configure a destination only if you are using the Enterprise edition of SAP Conversational AI. This is not supported in the Community edition. For more information see Configuring the Enterprise Edition.

   If you have configured system aliases, you need to select the appropriate system that gets appended to the service API. For more information, see System Alias Configuration [page 140].
2. Configure the authentication and headers. Enter an entity value and choose a language for which you want to test your API response. Click *TEST*.



   Under the *Response Customization* tab, you can see the API service response (including errors).
3. You can customize the API response with the help of scripting and click *TRANSFORM*.



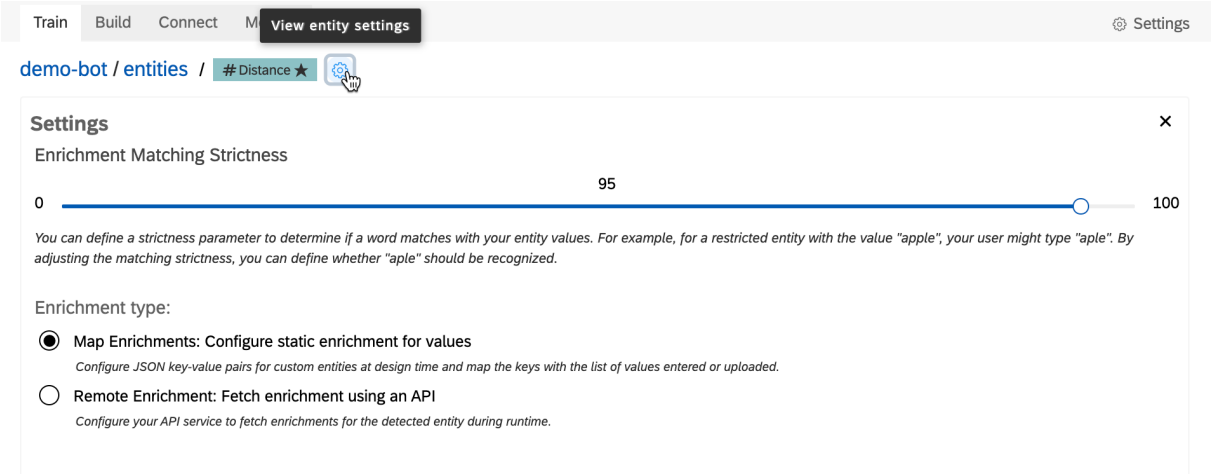
Check the response in the *Expression Analysis* console. You can see the enrichments for the value in the Json.

```
{
 "uuid": "56f63082-747d-429c-4e8f-9af71201d311",
 "intents": [
  {
   "slug": "show-products",
   "confidence": 0.99,
   "description": null
  }
 ],
 "entities": {
  "food": [
   {
    "value": "products",
    "raw": "products",
    "confidence": 1,
    "products": [
     "Bread",
     "Milk",
     "Vint soda",
     "Havina Cola",
     "Fruit Punch",
     "Cranberry Juice",
     "Pink Lemonade",
     "DVD Player",
     "LCD HDTV",
     "Lemonade",
     "Coffee"
    ]
   }
  ]
 },
 "language": "en",
 "processing_language": "en",
 "logs": [
  {
   "level": "info",
   "data": {
    "http_status": "200",
    "url":
"https://services.odata.org/V3/(S(vnym1b3ehndm0p4fr0bdtbon))/OData
ta/OData.svc/Products/?$format=json",
    "info": "External service successfully called. Service responded
with HTTP status 200 OK"
   },
```


> **i Note**
>
> When you fork an entity, then the entity and its enrichment configuration is forked, except for the authorization credentials.
>
> Under the forked entity, you can see that the API service enrichment is disabled because the credentials are not forked.

Switch Enrichment Type

You can switch enrichment type under your entity settings (cog wheel icon). Choose between *Map Enrichments* and *Remote Enrichment*.

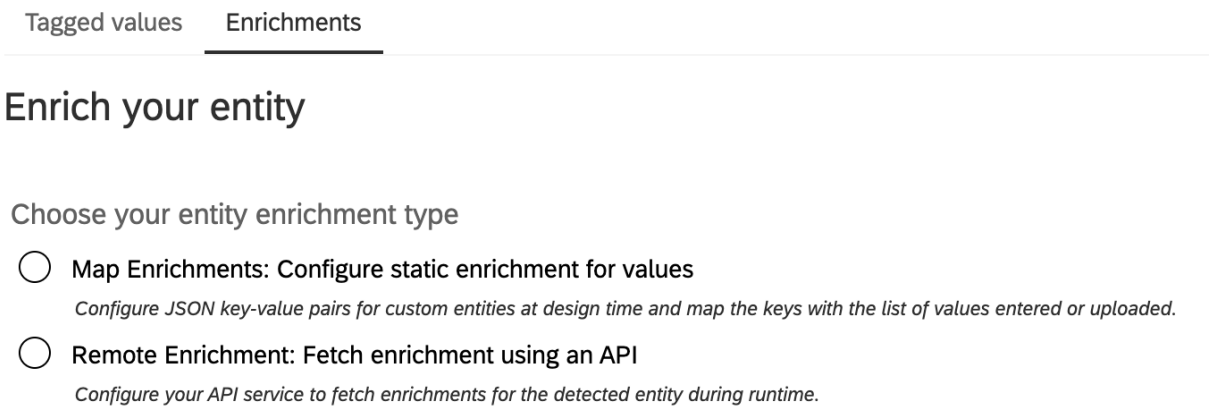


3.3.3 Enrichments for Gold Entities

Gold entities are automatically extracted from an expression. These gold entities are already trained, that is, they have data already added to them to make your bots better. However, you can add additional enrichments to these gold entities.

Fetching Gold Entity Value Enrichments

A gold entity can have different enrichments besides the default ones. You can either choose to add static enrichments for values by manually entering them or if you have a lot of enrichments saved in an external system, you can configure service API to fetch them. During a conversation, when an entity is detected, an API call is made to the corresponding business service and enrichments for the given entity value are retrieved.



Based on your requirement, choose the appropriate option. The option needs to be chosen for each gold entity and is applied for all languages.

Configure Static Enrichments

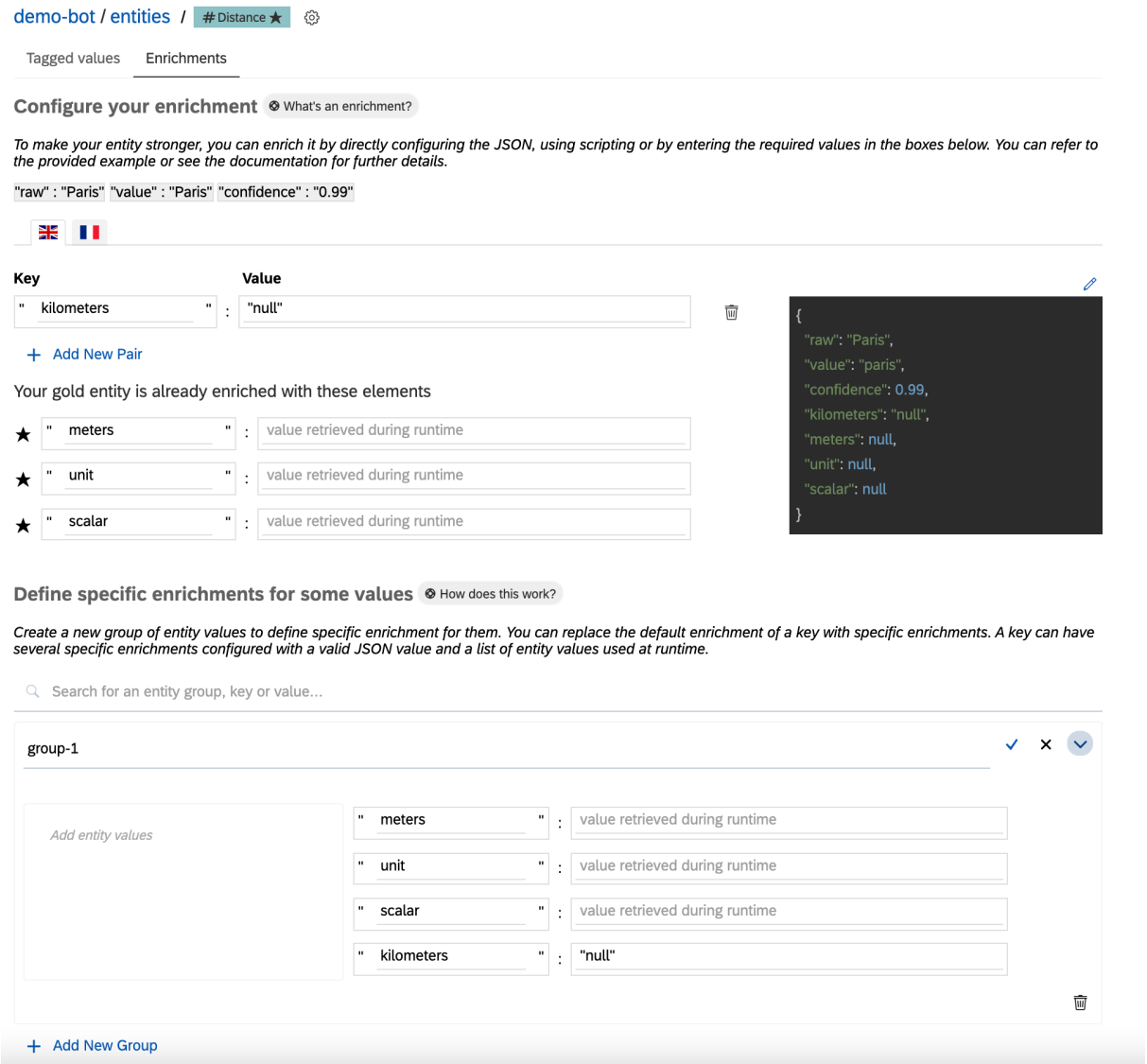
You do this configuration in two steps:

1. Besides the default key value pairs, you can add new JSON {key, default value} pairs.
2. Create group of entity values and define enrichments for these keys (for example, the required *Distance*).

> **i Note**
>
> The gold entity enrichments are applied to the list of entity values defined in a group.

{key, default value} pair

The gold enrichment already includes the default key-value pairs that cannot be deleted or modified. You can create new JSON `{key, default value}` pairs by providing a name and a default enrichment value.



An enrichment value must be a valid JSON value.

Keys are language-independent, while enrichments are language-dependent. For example, if you create the key kilometers, it will always be present in your JSON in all languages. If you don't define an enrichment for this key, null will be sent, for example, { "kilometers": null }.

Create and Enrich Groups of Entities

You can create a group of entity values by providing a name for the group and adding entity values (at least one value is needed).

Once the group is created, all the {key, default value} pairs created in the first step are assigned with the default values.

The default enrichment for a key can be overridden with the defined enrichments. A single key can have several enrichments, but an entity value cannot belong to several groups.

An enrichment is configured with:

- A valid JSON value

- A list of entity values

The list of entity values is used at runtime. When a gold entity is detected, the corresponding value is compared to this list of entity values to decide which enrichment should be applied. For example, in the case of our entity #DISTANCE and its enrichments, if the value Paris is detected in a sentence, the enriched JSON is as follows:

```
{
  "raw": "Paris",
  "value": "paris",
  "confidence": 0.99,
  "kilometers": "null",
  "meters": null,
  "unit": null,
  "scalar": null
}
```

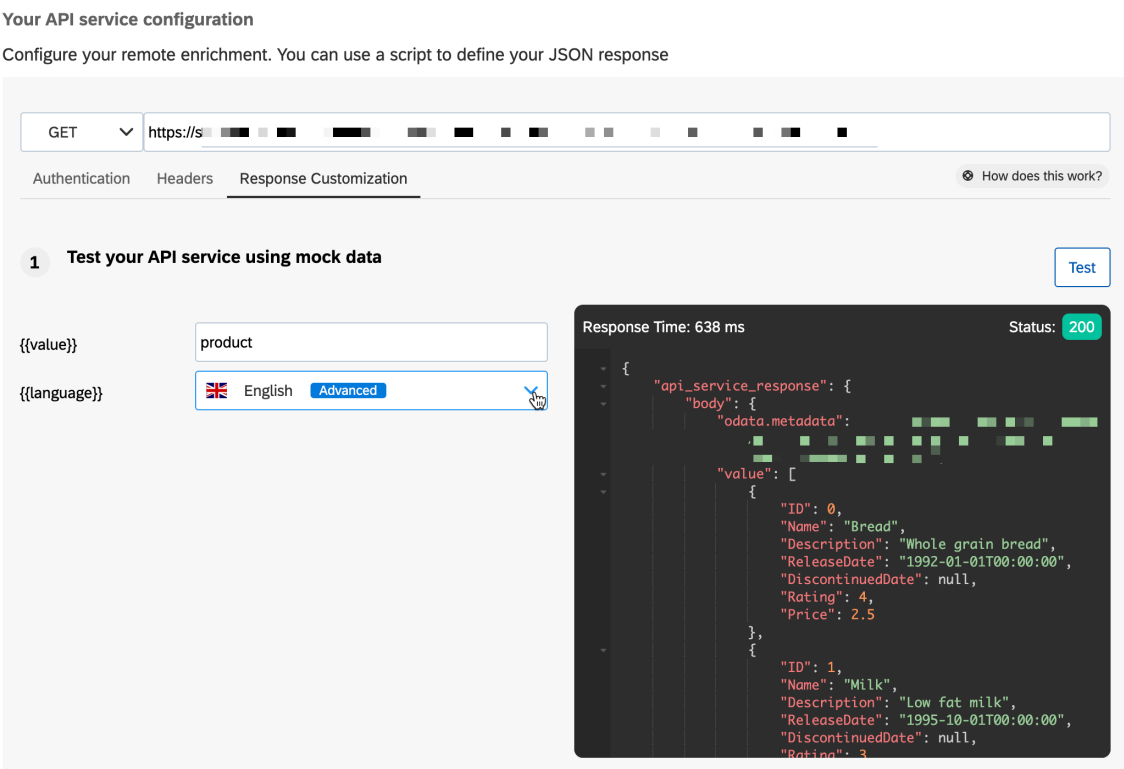
Fetch Enrichments Using Service API

To fetch entity value enrichments, you need to configure your service API.

1. Provide the URL of your service API and choose *Save*. For more information on how to configure a service API, see Connect to External Service [page 136].

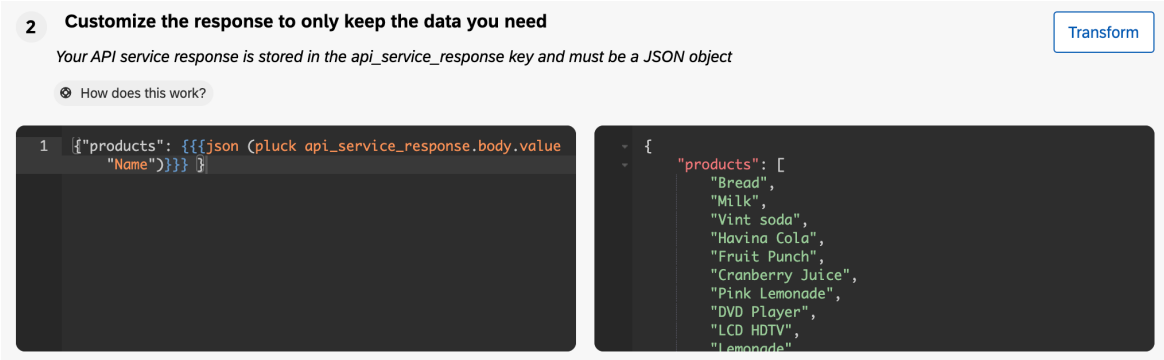
   > **i Note**
   >
   > You can configure a destination only if you are using the Enterprise edition of SAP Conversational AI. This is not supported in the Community edition. For more information see Configuring the Enterprise Edition.

   If you have configured system aliases, you need to select the appropriate system that gets appended to the service API. For more information, see System Alias Configuration [page 140].
2. Configure the authentication and headers. Enter an entity value and choose a language for which you want to test your API response. Click *TEST*.



Under the *Response Customization* tab, you can see the API service response (including errors).

3. You can customize the API response with the help of scripting and click *Transform*.



Check the response in the *Expression Analysis* console. You can see the enrichments for the value in the JSON.

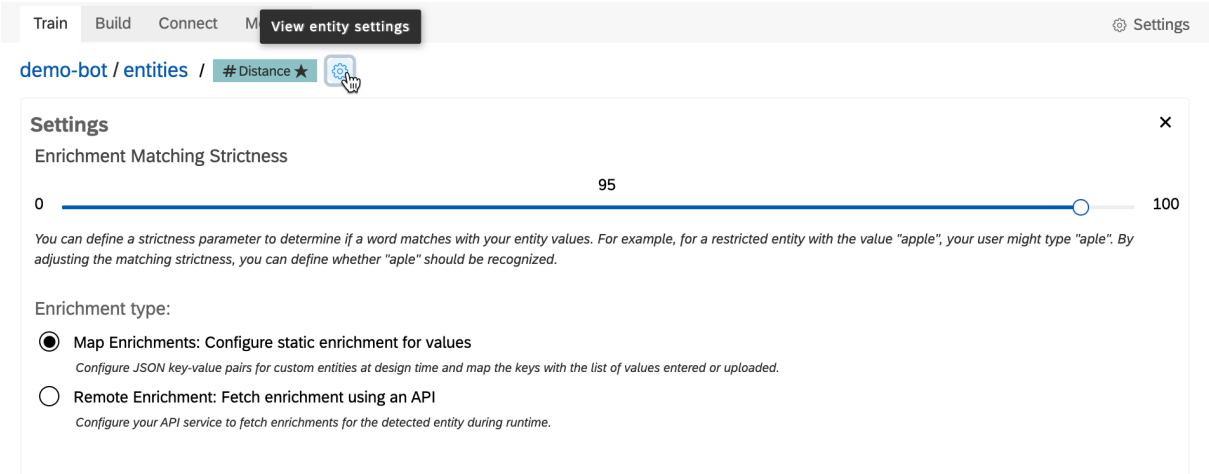
```
{
 "uuid": "56f63082-747d-429c-4e8f-9af71201d311",
 "intents": [
  {
   "slug": "show-products",
   "confidence": 0.99,
   "description": null
  }
 ],
 "entities": {
  "food": [
   {
    "value": "products",
    "raw": "products",
    "confidence": 1,
    "products": [
     "Bread",
     "Milk",
     "Vint soda",
     "Havina Cola",
     "Fruit Punch",
     "Cranberry Juice",
     "Pink Lemonade",
     "DVD Player",
     "LCD HDTV",
     "Lemonade",
     "Coffee"
    ]
   }
  ]
 },
 "language": "en",
 "processing_language": "en",
 "logs": [
  {
   "level": "info",
   "data": {
    "http_status": "200",
    "url":
"https://services.odata.org/V3/(S(vnym1b3ehndm0p4fr0bdtbon))/OData
ta/OData.svc/Products/?$format=json",
    "info": "External service successfully called. Service responded
with HTTP status 200 OK"
   },
```


It is possible to override a gold enrichment key with custom enrichment. If the gold enrichment is not retrieved, the custom enrichment values are fetched and displayed to the user. However, if the custom enrichment value is not retrieved, *null* is displayed in the JSON response.

> **i Note**
>
> When you fork an entity, then the entity and its enrichment configuration is forked, (for both static and remote enrichments). However, for remote enrichments, the authorization credentials are not forked into the target bot.

Switch Enrichment Type

You can switch enrichment type under your entity settings (cog wheel icon). Choose between *Map Enrichments* and *Remote Enrichment*.



3.3.4 List of Gold Entities

Overview

This is the list of the 28 gold entities that we currently detect, with examples and formatted information for each. Keep an eye on it, as we're always improving the detection for current entities, adding new entities, and improving the information we extract from them.

a – d	e – j	h – n	n – p	p – s	t – x
cardinal	email	language	number	percent	temperature
color	emoji	location	ordinal	person	url
datetime	ip	mass	organization	set	volume
distance	interval	money	phone	sort	
duration	job	nationality	pronoun	speed	

Cardinal

Sample Code

```
{
  "bearing": 45.0,
  "raw": "northeast",
  "confidence": 0.99
```

```
}
```

Entity	Examples
cardinal	north, southeast, north-west, south south east

Key	Comments
bearing	Float: The cardinal point bearing in degrees
raw	The raw value extracted from the sentence
confidence	The confidence score between 0 and 1 for the detection

Color

Sample Code

```
{
  "rgb": "rgb(0,0,255)",
  "hex": "#0000ff",
  "raw": "blue",
  "confidence": 0.99
}
```

Entity	Examples
color	blue, red, orange, dark blue, light green

Key	Comments
rgb	String: The RGB code of the color
hex	String: The hexadecimal value of the color
raw	The raw value extracted from the sentence
confidence	The confidence score between 0 and 1 for the detection

Datetime

Sample Code

```
{
  "formatted": "Thursday, 06 October 2018 at 09:00:00 AM",
  "iso": "2018-10-06T09:00:00Z",
  "accuracy": "day",
  "chronology": "future",
  "state": "relative",
  "raw": "next Thursday",
```

```
  "confidence": 0.92
}
```

Entity	Examples
datetime	next Friday, today, September 7 2018, 12/12/1992, this evening, mid-November, eoy

Key	Comments
formatted	String: The written format of the datetime
iso	String: The ISO-8601 standard of the datetime in UTC
accuracy	String: The accuracy of the explicitly given datetime Can be composed of one or more of year, month, week, day, halfday, hour, min, sec, now separated by a comma (,)
chronology	String: The point in time referenced by the datetime Can be past, present, or future
state	String: The type of the datetime Can be relative or absolute
raw	The raw value extracted from the sentence
confidence	The confidence score between 0 and 1 for the detection

Distance

Sample Code

```
{
  "scalar": 24.0,
  "unit": "mi",
  "meters": 38624.159999999996,
  "raw": "twenty-four miles",
  "confidence": 0.97
}
```

Entity	Examples
distance	20 meters, seven miles, ten km, 156 centimeters, 0.8 feet

Key	Comments
scalar	Float: The countable

Key	Comments
`unit`	String: The quantifier Can be `km` (kilometers), `m` (meters), `mi` (miles), `ft` (feet), `in` (inches), and so on
`meters`	Float: The distance in meters
`raw`	The raw value extracted from the sentence
`confidence`	The confidence score between 0 and 1 for the detection

Duration

Sample Code

```
{
  "chrono": "02:00:00:00",
  "years": 0.005478757133798352,
  "months": 0.06575342465753424,
  "days": 2.0,
  "hours": 48.0,
  "minutes": 2880.0,
  "seconds": 172800.0,
  "raw": "two days",
  "confidence": 0.99
}
```

Entity	Examples
`duration`	five days, one year, 27 seconds, two days and 3 hours, 72 weeks

Key	Comments
`chrono`	String: A formatted representation of the duration in the form of `:xx:xx:`
`years`	Float: The number of years in this duration
`months`	Float: The number of months in this duration
`days`	Float: The number of days in this duration
`hours`	Float: The number of hours in this duration
`minutes`	Float: The number of minutes in this duration
`seconds`	Float: The number of seconds in this duration
`raw`	The raw value extracted from the sentence
`confidence`	The confidence score between 0 and 1 for the detection

Email

Sample Code

```
{
  "local": "paul",
  "tag": null,
  "domain": "sap.com",
  "raw": "paul@sap.com",
  "confidence": 0.99
}
```

Entity	Examples
email	helloconversationalai@sap.com

Key	Comments
local	String: The local part of the email
tag	String: The tag part of the email
domain	String: The domain of the email
raw	The raw value extracted from the sentence
confidence	The confidence score between 0 and 1 for the detection

Emoji

Sample Code

```
{
    "formatted": "happy",
    "feeling": "happy",
    "tags": [
        "eye",
        "face",
        "mouth",
        "open",
        "smile"
    ],
    "unicode": "U+1F604",
    "description": "smiling face with open mouth & smiling eyes",
    "raw": ":)",
    "confidence": 0.99
}
```

Entity	Examples
emoji	:), :heart:

Key	Comments
`formatted`	String: The localized feeling of the emoji
`feeling`	String: The expressed sentiment of the emoji
`tags`	Array of string: A list of words related to the emoji
`unicode`	String: The Unicode codepoint of the emoji
`description`	String: A fully-written sentence describing the emoji
`raw`	The raw value extracted from the sentence
`confidence`	The confidence score between 0 and 1 for the detection

IP

Sample Code

```
{
  "formatted": "Fontenay-sous-Bois, Île-de-France, FR",
  "lat": 48.8544,
  "lng": 2.4827,
  "raw": "82.121.114.213",
  "confidence": 0.99
}
```

Entity	Examples
`ip`	127.0.0.1, 192.157.0.54, 153.34.43.0

Key	Comments
`formatted`	String: The full denomination of the IP's location
`lat`	Float: The latitude of the IP's location
`lng`	Float: The longitude of the IP's location
`raw`	The raw value extracted from the sentence
`confidence`	The confidence score between 0 and 1 for the detection

Interval

Sample Code

```
{
  "begin": "2018-10-31T09:00:00Z",
  "end": "2018-11-06T09:00:00Z",
  "begin_accuracy": "day",
  "end_accuracy": "day",
```

```
  "begin_chronology": "future",
  "end_chronology": "future",
  "timespan": 518400.0,
  "raw": "from monday to sunday",
  "confidence": 0.96
}
```

Entity	Examples
`interval`	last week, last 3 months, between today and tomorrow, from now to next week, Wednesday the 3rd between 2pm and 3pm, starting Sunday ending Monday

Key	Comments
`begin`	String: The ISO-8601 standard of the start point in UTC
`end`	String: The ISO-8601 standard of the end point in UTC
`begin_chronology`	String: Comma-separated points in time referenced by the begin field Can be `past`, `present`, or `future`
`end_chronology`	String: Comma-separated points in time referenced by the end field Can be `past`, `present`, or `future`
`begin_accuracy`	String: The accuracy of the explicitly given begin datetime Can be composed of one or more of `year`, `month`, `week`, `day`, `halfday`, `hour`, `min`, `sec`, `now` separated by a comma (,)
`end_accuracy`	String: The accuracy of the explicitly given end datetime Can be composed of one or more of `year`, `month`, `week`, `day`, `halfday`, `hour`, `min`, `sec`, `now` separated by a comma (,)
`timespan`	Float: The duration of the interval
`raw`	The raw value extracted from the sentence
`confidence`	The confidence score between 0 and 1 for the detection

Job

Sample Code

```
{
    "raw": "web designer",
  "confidence": 0.85
}
```

Entity	Examples
job	CTO, farmer, financial accountant, chief operator, actress

Key	Comments
raw	The raw value extracted from the sentence
confidence	The confidence score between 0 and 1 for the detection

Language

Sample Code

```
{
  "short": "NL",
  "long": "NLD",
  "raw": "Dutch",
  "confidence": 0.76
}
```

Entity	Examples
language	Chinese, English, Spanish

Key	Comments
short	String: The ISO 639-1 standard language code
long	String: The ISO 639-2 standard language code
raw	The raw value extracted from the sentence
confidence	The confidence score between 0 and 1 for the detection

Location

Sample Code

```
{
  "formatted": "3410 Hillview Avenue, Palo Alto, CA 94304, United States",
  "lat": 37.399169,
  "lng": -122.146475,
  "type": "establishment",
  "place": "ChIJEyiYgJ66j4ARVZ9bxK83pSQ",
  "street_number": "3410",
  "street_name": "Hillview Avenue",
  "postal_code": "94304",
  "city": "Palo Alto",
  "state": "CA",
  "country": "us",
```

```
    "raw": "3410 hillview avenue",
    "confidence": 0.99
  }
```

Entity	Examples
location	San Francisco, Paris, East London, 123 Abbey Road

Key	Comments
formatted	String: The full denomination of the location
lat	Float: The latitude of the location
lng	Float: The longitude of the location
type	Float: The precision type of the location Can be one of country, locality, sublocality, postal_code, route, intersection, political, neighborhood, premise, airport, park, ...
place	String: The Google Places ID of the location
street_number	String, the street number of the location
street_name	String, the street name of the location
postal_code	String, the ZIP or postal code of the location
city	String, the city of the location
state	String, the state or province of the location
country	String: The ISO 3166-2 code for the country of the location
raw	The raw value extracted from the sentence
confidence	The confidence score between 0 and 1 for the detection

Mass

Sample Code

```
"mass": [{"confidence":0.99, "raw": "28 lbs",
"scalar": 28, "unit": "lbs", "grams": 12700.576}]
```

Entity	Examples
mass	45 pounds, twenty-one grams, thirty seven kgs, 0.98 mg, 23 kilograms

Key	Comments
`scalar`	Float: The countable
`unit`	String: The quantifier Can be `lbs` (pounds), `kg` (kilograms), `g` (grams), `oz` (ounces), and so on
`grams`	Float: The mass in grams
`raw`	The raw value extracted from the sentence
`confidence`	The confidence score between 0 and 1 for the detection

Money

Sample Code

```
{
  "amount": 16.0,
  "currency": "EUR",
  "dollars": 17.92,
  "raw": "sixteen euros",
  "confidence": 0.98
}
```

Entity	Examples
`money`	3.14 euros, eight millions dollars, $6, 56, seventy-eight zlotys

Key	Comments
`amount`	Float: The countable
`currency`	String: The ISO 4217 standard currency code
`dollars`	Float: The amount of money in dollars
`raw`	The raw value extracted from the sentence
`confidence`	The confidence score between 0 and 1 for the detection

Nationality

Sample Code

```
{
  "short": "PT",
  "long": "PRT",
  "country": "Portugal",
  "raw": "Portuguese",
```

```
  "confidence": 0.97
}
```

Entity	Examples
nationality	French, Spanish, Australian

Key	Comments
short	String: The ISO 3166-1 alpha2 standard country code
long	String: The ISO 3166-1 alpha3 standard country code
country	String: The name of the country to which the nationality refers
raw	The raw value extracted from the sentence
confidence	The confidence score between 0 and 1 for the detection

Number

Sample Code

```
{
  "scalar": 27000,
  "raw": "twenty-seven thousand",
  "confidence": 0.83
}
```

Entity	Examples
number	one thousand, 3, 9,000, seven million

Key	Comments
scalar	Integer: The number
raw	The raw value extracted from the sentence
confidence	The confidence score between 0 and 1 for the detection

Ordinal

Sample Code

```
{
  "rank": -1,
  "raw": "last",
  "confidence": 0.98
```

```
}
```

Entity	Examples
`ordinal`	3rd, 158th, last, seventh

Key	Comments
`rank`	Integer: The number behind the ordinal
`raw`	The raw value extracted from the sentence
`confidence`	The confidence score between 0 and 1 for the detection

Organization

Sample Code

```
{
  "raw": "Apple",
  "confidence": 0.99
}
```

Entity	Examples
`organization`	Lehman Brothers, NASA, Apple

Key	Comments
`raw`	The raw value extracted from the sentence
`confidence`	The confidence score between 0 and 1 for the detection

Percent

Sample Code

```
{
  "scalar": 86.0,
  "unit": "%",
  "percent": 86.0,
  "raw": "86 percent",
  "confidence": 0.99
}
```

Entity	Examples
`percent`	99%, 2 percent, seventy-seven percent, 12 permyriad

Key	Comments
`scalar`	Float: The countable
`unit`	String: The quantifier Can be % (percent), ‰ (permil), ‱ (permyriad), `ppb` (part per billion), and so on
`raw`	The raw value extracted from the sentence
`confidence`	The confidence score between 0 and 1 for the detection

Person

Sample Code

```
{
  "fullname": "Dave Pitterson",
  "raw": "Dave Pitterson",
  "confidence": 0.97
}
```

Entity	Examples
`person`	Michael Adams, Julie D. Armstrong, John

Key	Comments
`fullname`	String: The full name of the person
`raw`	The raw value extracted from the sentence
`confidence`	The confidence score between 0 and 1 for the detection

Phone

Sample Code

```
{
  "number": "3612374040",
  "raw": "(361) 237 4040",
  "confidence": 0.88
}
```

Entity	Examples
phone	+91-22-265 9000, 64 4 437-4746, 0682753582, (123) 123 1234

Key	Comments
number	String: The normalized phone extracted
raw	The raw value extracted from the sentence
confidence	The confidence score between 0 and 1 for the detection

Pronoun

Sample Code

```
{
  "person": 1,
  "number": "singular",
  "gender": "unknown",
  "raw": "I",
  "confidence": 0.99
}
```

Entity	Examples
pronoun	I, we, it, you, us

Key	Comments
person	Integer: The person of the pronoun Can be 1, 2, or 3
number	String: The number of the pronoun Can be singular or plural
gender	String: The gender of the pronoun Can be unknown, neutral, male, or female
raw	The raw value extracted from the sentence
confidence	The confidence score between 0 and 1 for the detection

Set

Sample Code

```
{
```

```
  "next": "2018-12-02T18:18:02Z",
  "frequency": "monthly",
  "interval": 2,
  "rrule": "RRULE:FREQ=MONTHLY;INTERVAL=2",
  "raw": "every two months",
  "confidence": 0.99
}
```

Entity	Examples
set	every Sunday, each day, monthly, every 2 weeks

Key	Comments
next	String: The ISO-8601 representation of the next occurrence in UTC
frequency	String: The frequency this event is repeating Can be yearly, monthly, weekly, daily, hourly, minutely, secondly
interval	Integer: The interval between two occurrences relative to the frequency
rrule	String: The RFC 5545 compliant recurrence rule
raw	The raw value extracted from the sentence
confidence	The confidence score between 0 and 1 for the detection

Sort

Sample Code

```
{
  "order": "DESC",
  "criterion": "expensive",
  "raw": "least expensive",
  "confidence": 0.96
}
```

Entity	Examples
sort	most valuable, best, least affordable, cheapest

Key	Comments
order	String: The order to sort (MySQL inspired)
criterion	String: The criterion to sort
raw	The raw value extracted from the sentence

Key	Comments
`confidence`	The confidence score between 0 and 1 for the detection

Speed

Sample Code

```
{
  "scalar": 37.0,
  "unit": "km/h",
  "mps": 10.277777777777779,
  "raw": "thirty-seven kilometers per hour",
  "confidence": 0.57
}
```

Entity	Examples
`speed`	7 mph, 10 km/h, seven meters per second

Key	Comments
`scalar`	Float: The countable
`unit`	String: The quantifier Can be `km/h` (kilometer per hour), `mi/s` (miles per second), `kt` (knots), and so on
`mps`	Float: The speed in meters per second
`raw`	The raw value extracted from the sentence
`confidence`	The confidence score between 0 and 1 for the detection

Temperature

Sample Code

```
{
  "scalar": 9.0,
  "unit": "F",
  "celsius": -12.777777777777777,
  "raw": "9 degrees Fahrenheit",
  "confidence": 0.97
}
```

Entity	Examples
`temperature`	25 degrees Celsius, 70°F, seven degC, 5 rankines

Key	Comments
`scalar`	Float: The countable
`unit`	String: The quantifier Can be `C` (Celsius), `K` (Kelvin), `F` (Fahrenheit), `R` (Rankine), and so on
`celsius`	Float: The temperature in Celsius
`raw`	The raw value extracted from the sentence
`confidence`	The confidence score between 0 and 1 for the detection

URL

Sample Code

```
{
  "scheme": "https",
  "host": "pokebot.cai.tools.sap",
  "path": "/register",
  "params": null,
  "query": null,
  "fragment": null,
  "raw": "https://pokebot.cai.tools.sap/register",
  "confidence": 0.99
}
```

Entity	Examples
`url`	https://cai.tools.sap, localhost:9000, api.cai.tools.sap/v2/request

Key	Comments
`scheme`	String: The URL scheme Can be `http`, `https`, `mailto`, `ssh`, `git`, and so on
`host`	String: The host of the URL
`path`	String: The URL path
`params`	String: The parameters of the URL
`query`	String: The query parameters of the URL
`fragment`	String: The anchor of the URL

Key	Comments
`raw`	The raw value extracted from the sentence
`confidence`	The confidence score between 0 and 1 for the detection

Volume

Sample Code

```
{
  "scalar": 90.0,
  "unit": "hl",
  "liters": 9000.0,
  "raw": "90 hectoliters",
  "confidence": 0.96
}
```

Entity	Examples
`volume`	30 liters, two barrels, 1/2 tbsp

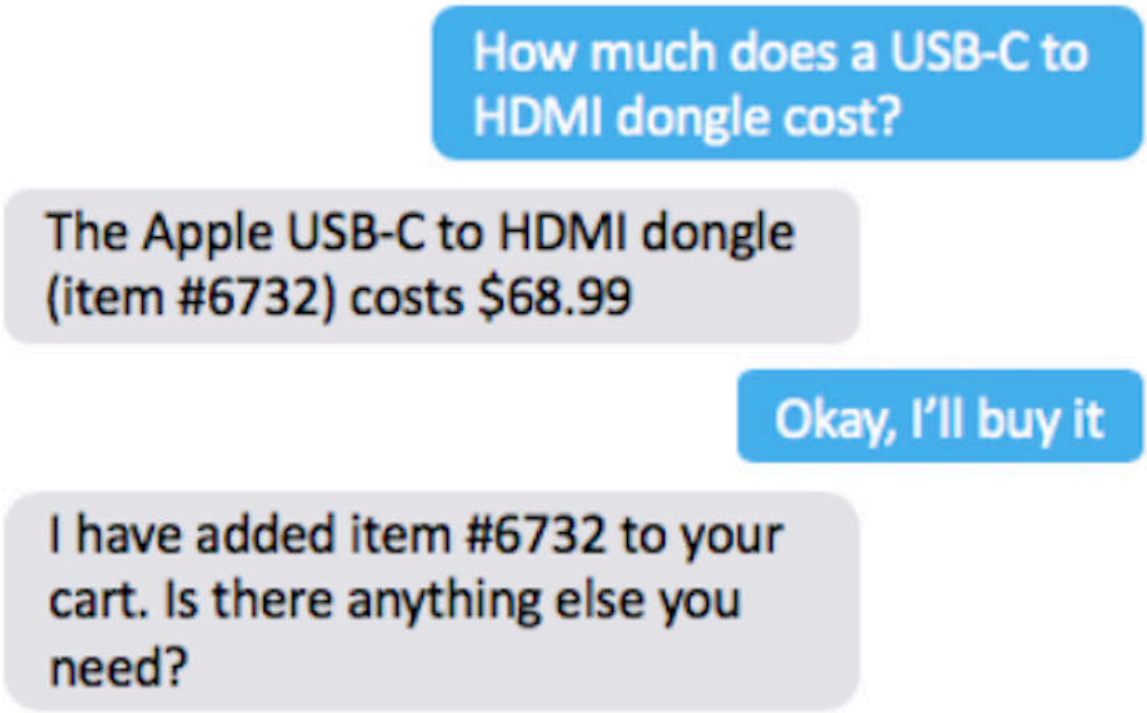
Key	Comments
`scalar`	Float: The countable
`unit`	String: The quantifier Can be `l` (liters), `tsp` (teaspoons), `pt` (pints), and so on
`liters`	Float: The volume in liters
`raw`	The raw value extracted from the sentence
`confidence`	The confidence score between 0 and 1 for the detection

3.3.5 References Between Entities

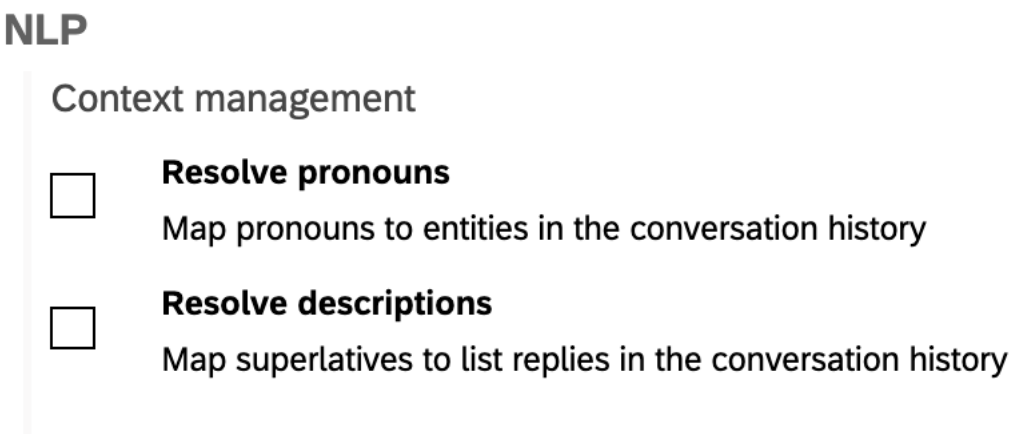
References Between Entities

Resolve Pronouns

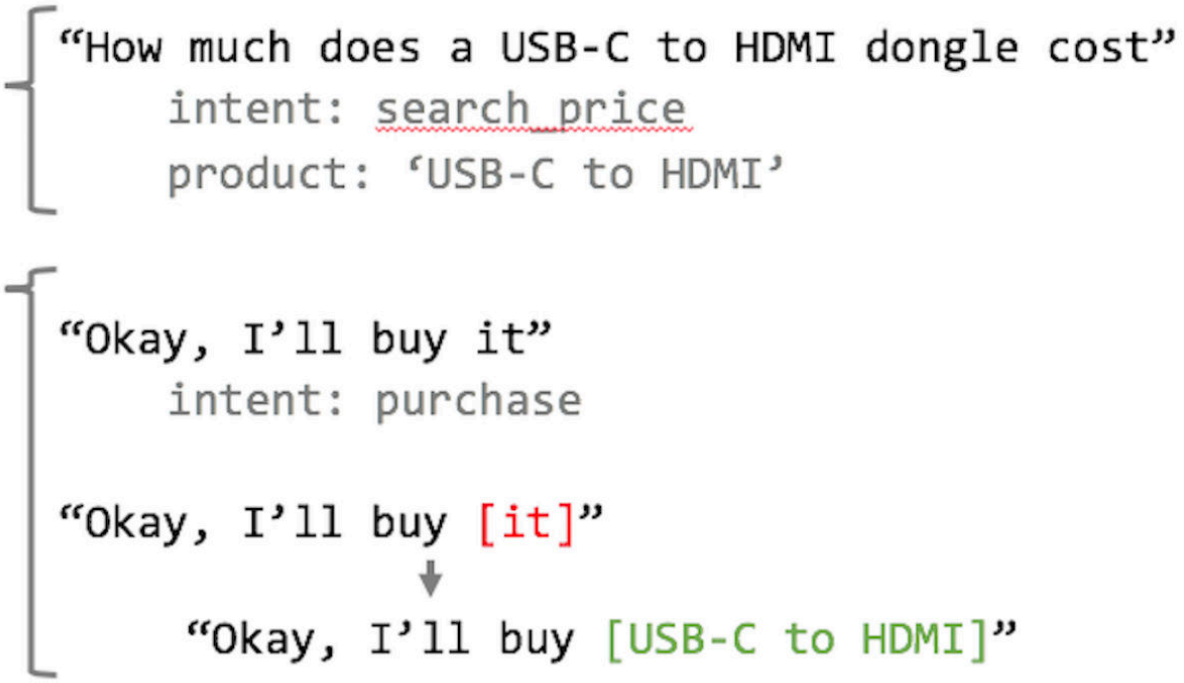
For users to meaningfully converse with your bot using natural language, your bot needs to be able to recognize pronouns (like *it* or *that*) and map them to entities previously mentioned in the conversation. In the following example, the pronoun *it* refers to the entity *Apple USB-C to HDMI dongle*.



For your bot to resolve pronouns, you must first go to the *Settings* page for your bot, choose *Versions*, click the arrow next to a version and select the *Resolve pronouns* checkbox. (The default setting is not selected.) Selecting this checkbox enables your bot to resolve the following pronouns: *she*, *he*, *it*, *we*, *they*, *her*, *him*, *us*, *them*, *his*, *this*, *that*.



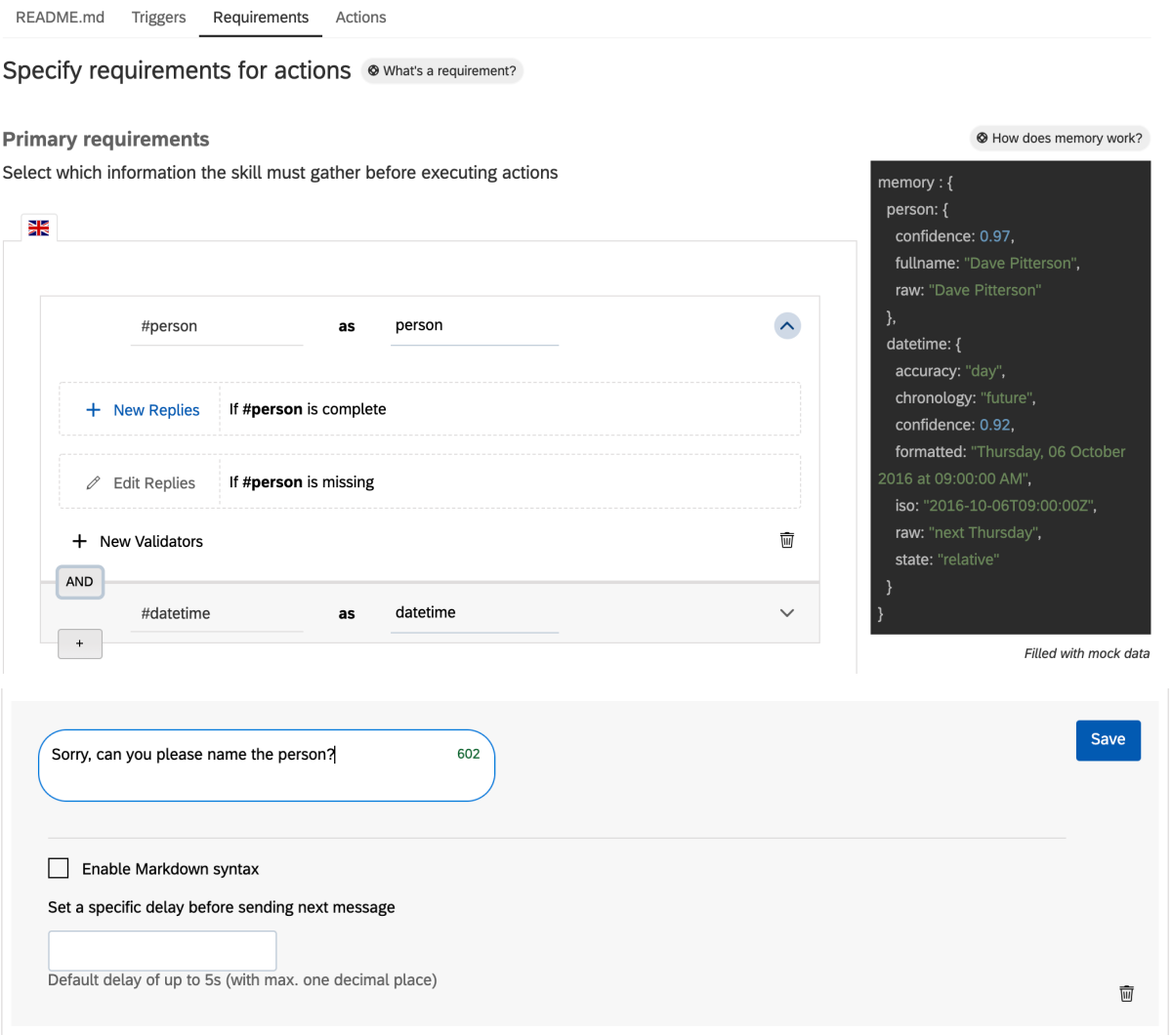
With this checkbox selected, the bot now successfully maps the pronoun *it* to the entity *Apple USB-C to HDMI dongle*.



The following are not supported:

- Split antecedents
  This is where you have more than one entity (for example, *Check whether* **Harry** *and* **Sally** *are available*) before a pronoun is used that encompasses these multiple entities (for example, *Set up a meeting with* **them**).
- Cataphora
  This is the use of a pronoun that refers to or stands for a subsequent entity (for example, *When* **she** *arrives, let* **Sally** *know I'll be waiting in the conference room*).

Remember to set a message that your bot can use if it is unable to map the pronoun to an entity. For example, if your bot is unable to map the pronoun *her* to a person, you might want to set the message *Sorry, can you please name the person?* To do this, first open the skill. Under *Requirements*, click *Edit Replies* next to *If #person is missing* and enter the message.



Resolve Descriptions

When a user is conversing with your bot in English, French, or Spanish, and your bot replies with a list, carousel, quick replies, or buttons, the user can refer to an item in the message using a superlative like *cheapest* or *most expensive* or using an ordinal like *first* or *second*. For example, if the bot displays a list of flights, the user can tell the bot to book the cheapest or shortest flight, or to book the first or last flight.

For your bot to map superlatives or ordinals to items in the message, you must select the *Resolve descriptions* checkbox on the *Settings* page for your bot under *Options*. Remember that only English, French, and Spanish are currently supported.

When mapping superlatives or ordinals to items in a message, the bot will always use the most recent list, carousel, quick replies, or buttons in the conversation history (if the conversation history contains more than one of these).

Certain superlatives can describe different types of entities. For example, *longest* can refer to duration and distance. If the message contains more than one of these entity types, the bot will always choose the first entity type that the superlative can refer to. For example, if flight CAI 001 is listed as 3 hours and 1,000 miles, the bot will interpret *longest* as referring to the duration of 3 hours.

Remember to set a message that your bot can use if it is unable to map the description to an entity. You can also use the information provided by superlatives in a webhook. The detected superlatives can be enriched with this information in the NLP JSON.

How a Sentence is Analyzed?

The following steps explain the way a sentence is analyzed by SAP Conversational AI.

Let us consider an example sentence:

`My name is John and I am a project manager.`

1. First the entities are tagged, then the sentence is analyzed by replacing the words with the entities.

#PERSON ★	John
#PRONOUN ★	I
#NUMBER ★	a
#JOB ★	project manager

   The sentence becomes **`My name is #PERSON and #PRONOUN am #NUMBER #JOB`**

2. This sentence is classified to the appropriate intent as per your dataset.

If there are two intents both having at least one expression with the same structure, this may lead to intent confusion.

For example, the two intents *@view-module-activity* and *@revise-module-activity* have an expression with the same structure **`#JOB #MODULE #PERSON`**. Both intents might be detected with the same (or mostly the same) confidence resulting in intent confusion.

Retrieving Your Bot's Entities with an API Call

You can fetch the entities for a specific bot with an API call. For more information, see Indexing a Dataset's Entities in the API Reference.

3.4 Sentiments

Sentiment detection is an important part of analyzing a user's input. We decided to follow guidelines suggesting a higher granularity of sentiments than you may be used to. This allows you to treat different levels of positive and negative inputs.

Polarity	Examples
`vpositive`	That was awesome to see that man eat a peach.
`positive`	The man ate a peach.
`neutral`	peach
`negative`	Sadly, the man did not eat a peach.
`vnegative`	That was awful to see that man eat a peach.

3.5 Sentence Acts

We currently detect 4 acts of a sentence, as defined in section 8.7 of *Natural Language Understanding* by James Allen (1995). Those 4 categories are defined as surface speech acts, which indicate how the proposition described is intended to be used to update the discourse situation.

Type	Example	Description
assert	The man ate a peach.	The proposition is being asserted
command	Eat a peach.	The proposition describes an action to perform
wh-query	What did the man eat?	The proposition describes an object to be identified
yn-query	Did the man eat a peach?	The proposition is being queried

> **i Note**
>
> `act` detection is no longer supported.

3.6 Sentence Types

The type of sentence comes from the task of question classification in the domain of question answering systems. Detecting the type of a question helps you to define what the answer to your user's request needs to be. Below is a list of the classes that we detect, together with a description and an example of each type of sentence.

Class	Subclass	Description	Example
ABBR	abb	abbreviation	What is the acronym for the Federal Bureau of Investigation?
	exp	expression abbreviated	What does BMW stand for?
DESC	def	definition of something	Define the cosmology.
	desc	description of something	What are the differences between 1980 and 1990?
	manner	manner of an action	How can I find a list of celebrities?
	reason	reasons	Explain why she said you were late.
ENTY	animal	animals	A corgi is a kind of what?
	body	organs of body	What is the longest bone in the human body?
	color	colors	What color are crickets?
	cremat	inventions, books, and other creative pieces	In which films was Jude Law an actor?
	currency	currency names	What money do Italians use?
	dis.med.	diseases and medicine	What are the 10 plagues of Egypt?
	event	events	In what war was the first submarine used?
	food	food	What are the top vegetable crops in the world?
	instru	musical instrument	What kind of trumpet is the loudest?
	lang	languages	Name a Gaelic language.
	letter	letters like a–z	Name a consonant.
	other	other entities	To what does Microsoft's Windows 3 owe its success?
	plant	plants	What is the state tree of Nebraska?
	product	products	Germany is the largest producer of what?
	religion	religions	A cardinal is a rank in which religion?
	sport	sports	Garry Kasparov played what game?
	substance	elements and substances	What are cigarettes made of?
	symbol	symbols and signs	What is the name for signs made in crops?

Class	Subclass	Description	Example
	techmeth	techniques and methods	What are common methods used to regulate monopolies?
	termeq	equivalent terms	What was another name for East Germany?
	veh	vehicles	Name a French car.
	word	words with a special property	Give a synonym for *alphabet*.
HUM	desc	description of a person	Can you tell me who she is?
	gr	a group or organization of persons	What are Google employees called?
	ind	an individual	CNN is owned by whom?
	title	title of a person	What is her profession?
LOC	city	cities	Give me the name of Paraguay's capital.
	country	countries	In which state would you find the Catskill Mountains?
	mount	mountains	What is the name of the highest peak of Africa?
	other	other locations	Name a civil war battlefield.
NUM	code	postcodes or other codes	Give me the country code of France.
	count	number of something	About how many soldiers died in World War II?
	date	dates	CNN began broadcasting in what year?
	dist	linear measures	What is the wingspan of a condor?
	money	prices	How much do drugs to treat tuberculosis cost?
	ord	ranks	Tell me my final ranking!
	other	other numbers	How loud is thunder?
	period	the duration of something	For how long is an elephant pregnant?
	perc	fractions	At what percentage are you right now?
	speed	speed	How fast can cheetahs run?
	temp	temperature	What is the temperature at the center of the earth?
	volsize	size, area, and volume	What is the size of Argentina?

Class	Subclass	Description	Example
	weight	weight	How much did a knight's armor weigh?

> **i Note**
>
> `type` detection is no longer supported.

3.7 Languages

Definition

Bots are multilingual, meaning that you can speak several languages with the same bot. SAP Conversational AI currently supports all languages with different levels of functionality: advanced, standard, and basic.

To enable users to speak different languages with your bot, add the desired languages for each intent on the *Train* tab and create expressions in those languages. (For advanced level languages, remember that SAP Conversational AI suggests additional expressions for each expression you add, so you can add expressions quickly and easily to an intent.)

SAP Conversational AI automatically detects the input language. For advanced and standard level languages, this lets you adapt your answers.

After SAP Conversational AI detects the language, the following rules apply:

- If you have expressions in that language, it is used for processing.
- If you don't have any expressions in that language, your default bot language is used for processing.

Advanced Level Languages

Language	Intent classification	Custom entities	Gold entities	Sentiment analysis	Language detection
English	Yes	Yes	Yes	Yes	Yes
French	Yes	Yes	Yes	Yes	Yes
German	Yes	Yes	Yes	Yes	Yes
Spanish	Yes	Yes	Yes	Yes	Yes

> **i Note**
>
> The FAQ bot is currently supported for advanced languages only. For more information, see Getting Started with FAQ Bot [page 271]

Standard Level Languages

Language	Intent classification	Custom entities	Gold entities	Sentiment analysis	Language detection
Arabic	Yes	Yes	No	No	Yes
Catalan	Yes	Yes	No	No	Yes
Chinese	Yes	Yes	No	No	Yes
Danish	Yes	Yes	No	No	Yes
Dutch	Yes	Yes	No	No	Yes
Finnish	Yes	Yes	No	No	Yes
Hindi	Yes	Yes	No	No	Yes
Italian	Yes	Yes	No	No	Yes
Japanese	Yes	Yes	No	No	Yes
Korean	Yes	Yes	No	No	Yes
Norwegian	Yes	Yes	No	No	Yes
Polish	Yes	Yes	No	No	Yes
Portuguese	Yes	Yes	No	No	Yes
Russian	Yes	Yes	No	No	Yes
Swedish	Yes	Yes	No	No	Yes

Basic Level Languages

Tips

If you use a single language, pass your language as a request parameter to avoid the language detection step when you want to analyze text or use the Bot Builder API .

Consider using a translation service when you start constructing an intent in a new language. It'll make the operation faster.

Don't forget to set up all your intents in the new language.

Example of How the Language is Detected

You have intents in French and English, but none in Spanish, and your bot's default language is French.

- You receive a user utterance that SAP Conversational AI detects as Spanish.
  SAP Conversational AI uses French as the processing language because your bot doesn't handle Spanish, and returns a JSON containing `fr` in the `processing_language` field and `es` in the `language` field.

  > **i Note**
  >
  > This happens at the first utterance itself. At this point, the language gets locked and no language detection happens further in the conversation. The bot continues to respond in the processing language.

- You send a user utterance to SAP Conversational AI and tell it that the user utterance is English.
  SAP Conversational AI uses English as the processing language and returns a JSON containing `en` in the `processing_language` field and `en` in the `language` field.
- You receive a user utterance that SAP Conversational AI detects as French.
  SAP Conversational AI uses French as the processing language and returns a JSON containing `fr` in the `processing_language` field and `fr` in the `language` field.
- You send a user utterance to SAP Conversational AI and tell it that the user utterance is Spanish.
  SAP Conversational AI uses French as the processing language because your bot doesn't handle Spanish, and returns a JSON containing `fr` in the `processing_language` field and `es` in the `language` field.

How the Language is Handled in a Conversation

In the Bot Builder, the first sentence sent in a new conversation is analyzed by the natural language processing (NLP) API, and the language is detected. SAP Conversational AI sets the `conversation_language` to the `processing_language` detected.

All subsequent messages are processed with the `conversation_language` that was detected in the first sentence of the conversation. This is to avoid changing the language when processing ambiguous international expressions like *OK*, *Cool*, and so on.

If you want to change the `conversation_language`, you can use a Change language action [page 108].

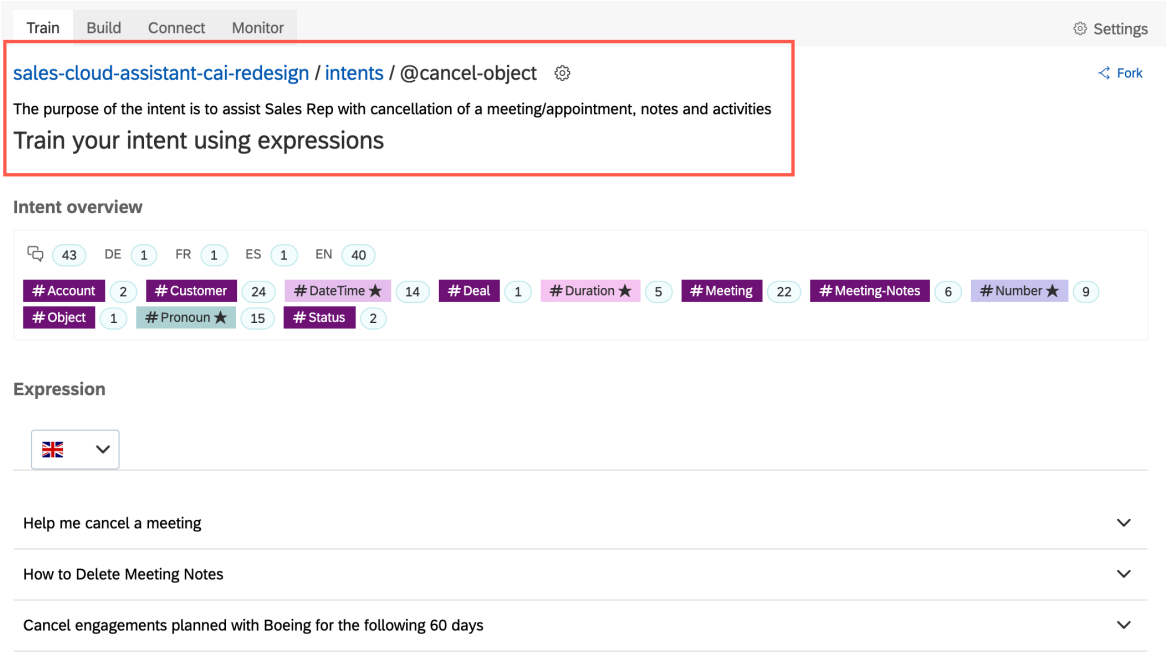
3.8 Creating a Good Dataset

Tips to Create a Good Data Set

Intents and utterances

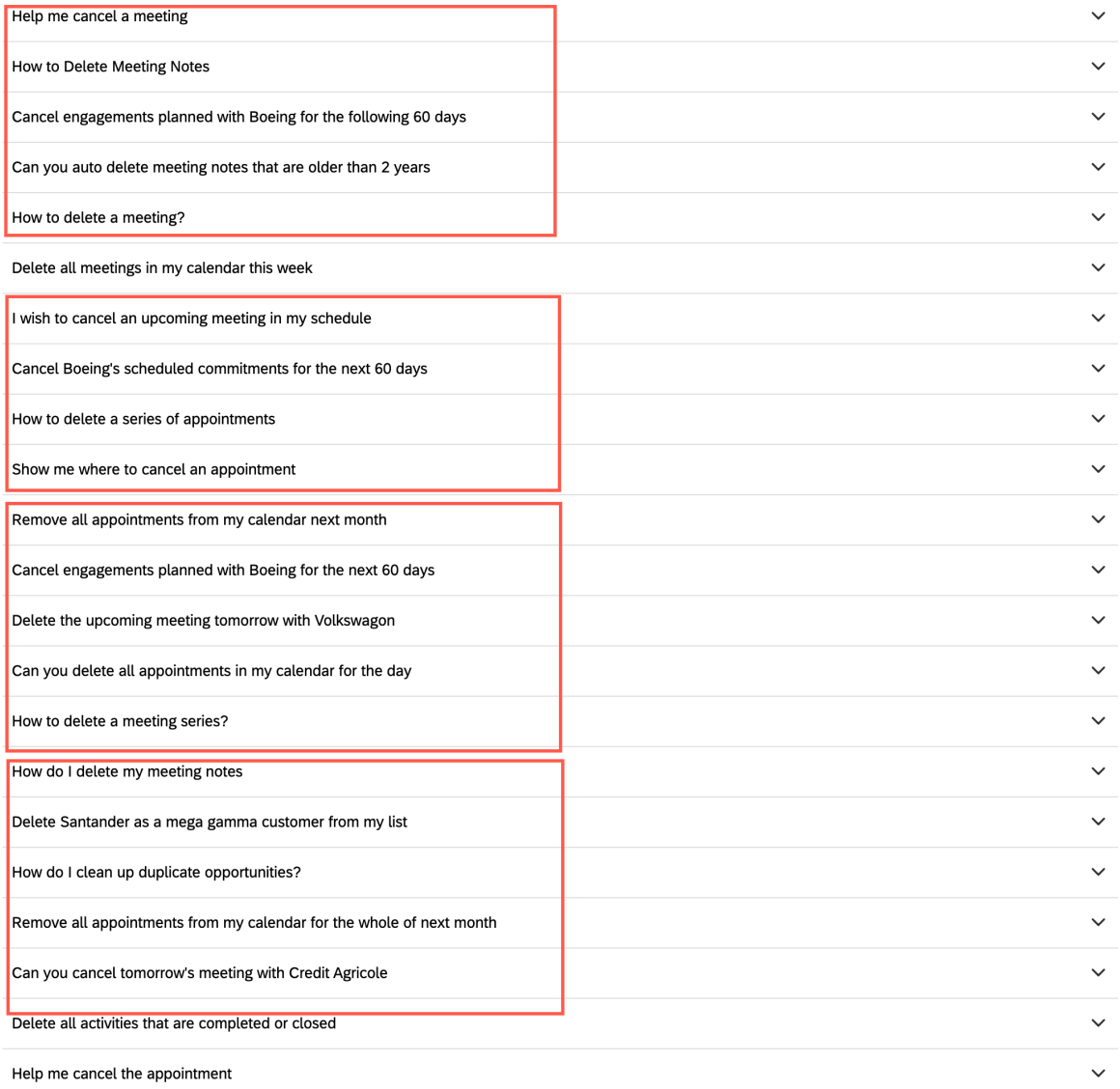
- It is recommended that you gather 10,000 sentences. This could be obtained either through logs or crowdsourcing survey administered using tools such as Qualtrics.

- Cluster the sentences into groups based on the intention of the sentence, in a way that the clusters are homogeneous within and heterogeneous across.
- Add a description for each intent, as shown in the following figure:



- Annotate sentences to the right intent

- Ensure each intent has at least 50 sentences; add sentences with different sentence structures to train the bot better, as shown in the following figure:

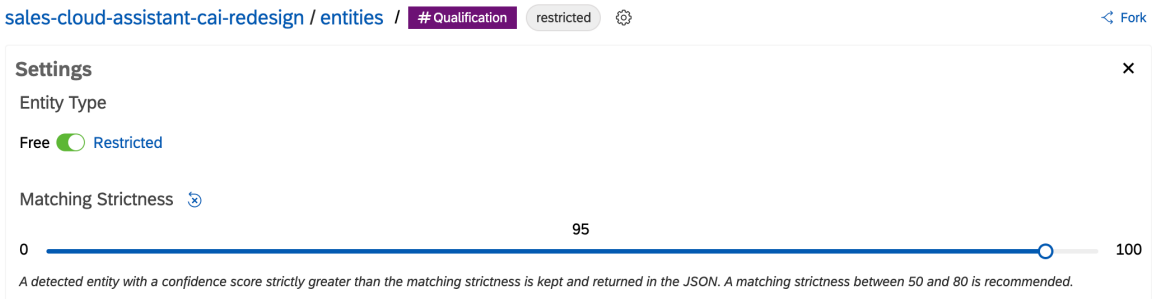


- Split a complex sentence into multiple simple sentences as illustrated in the following example:
  `In Services Sales, can a service credit be put partially against a deal, and the other part billed separately, or does it need to be the full amount?`
  This complex sentence can be split into simple sentences like:
  - In Services Sales, does service credit for a deal need to be the full amount?
  - In Services Sales, can service credit be applied partially against a deal?
- Ensure there is no repetition of names or terms such as person name, organization name, city name, product name and so on. Train the bot with varied real names (that is, value of the entity)
- Avoid generic terms like ABC or XYZ; replace with examples of person names, organization name, product name and so on, depending on the context.
- In case of small talk intents, review their relevance and need; retain only those that are necessary.

Entities

- Ensure that the all the values are tagged in relevant sentences. Add sentences, tag and train, if necessary.

- Free versus restricted entities:
  - Although *free* entities are relatively complex, but they are scalable. It is recommended to create free entities and add more examples to train, unless there is a compelling reason to have the entity as *restricted*.
  - For each entity, understand 'why do we have it', 'which flows use it' and 'why to have it as *free* or *restricted*. Validate values for restricted entities with LoB or customer.
- Set appropriate matching strictness for restricted entities. The percentage depends on the length of words (values). The longer the words, the higher should be the percentage. The minimum recommended matching strictness is 95% as shown in the following example.



- Gold entities
  - Remove or edit incorrect tagging wherever possible by rephrasing the sentence or tagging appropriate custom entities (particularly *Location/Geo* and *Organization* gold entities).
  - If the gold entities are not tagged in sentences (expressions), reprocess the sentences by editing them to ensure that the gold entity gets tagged.
  - If a gold entity is not tagged in the right way, contact support at https://launchpad.support.sap.com/.
- Note that the same word can have different meanings.
  Example: *Show me my lead* versus *Lead me to my customer location*. In the latter example, *lead* is a verb and not an entity.
- It is important to add sufficient examples for same words with different meanings (of the latter example type) for machine to learn better; this would help the machine to learn when to tag *lead* as an entity and when not to.

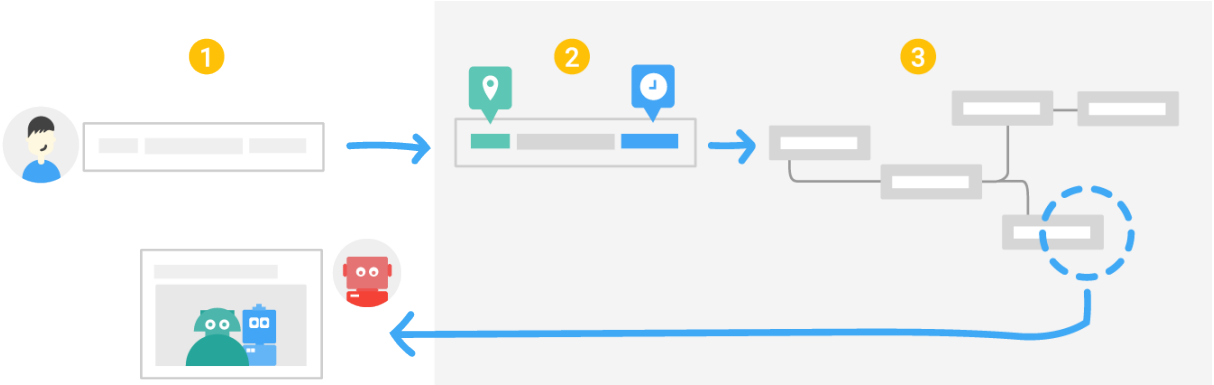
4 Bot Builder

4.1 Introduction

This introduction explains how the Bot Builder interacts with the other services of the platform.

The Bot Builder process is split into three distinct parts:

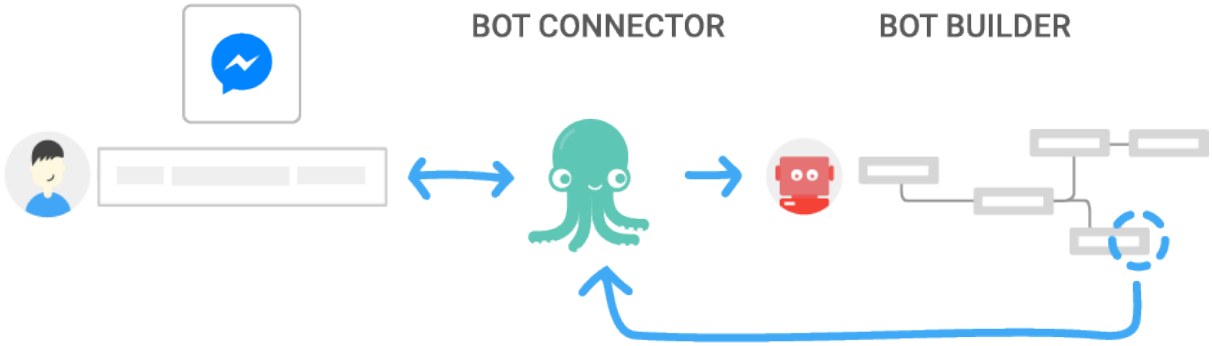
1. Get the user's input through a messaging channel.
   This can be done by the Bot Connector, meaning that when the Bot Connector receives a message, it dispatches the message to your Bot Builder. You can also collect the user's input by your own way and send it to the Bot Builder API directly.
2. Understand the user's input using natural language processing (NLP).
   Once the Bot Builder receives an input, it posts the message to our NLP API to extract structured information from this sentence.
3. Manage the conversation and context.
   This consists of using the JSON, which was returned from the NLP API, to manage the conversation using skills and conditions.



Bot Builder with Bot Connector

If you're using the Bot Builder with Bot Connector, all incoming messages are sent by the Bot Connector to the Bot Builder API (as described above). By default, every single message received by the Bot Connector is sent to the `https://api.cai.tools.sap/build/v1/dialog endpoint`, which is the endpoint of your Bot Builder.

The Bot Builder then replies to the Bot Connector with messages formatted as described in Send Rich Messages [page 215].



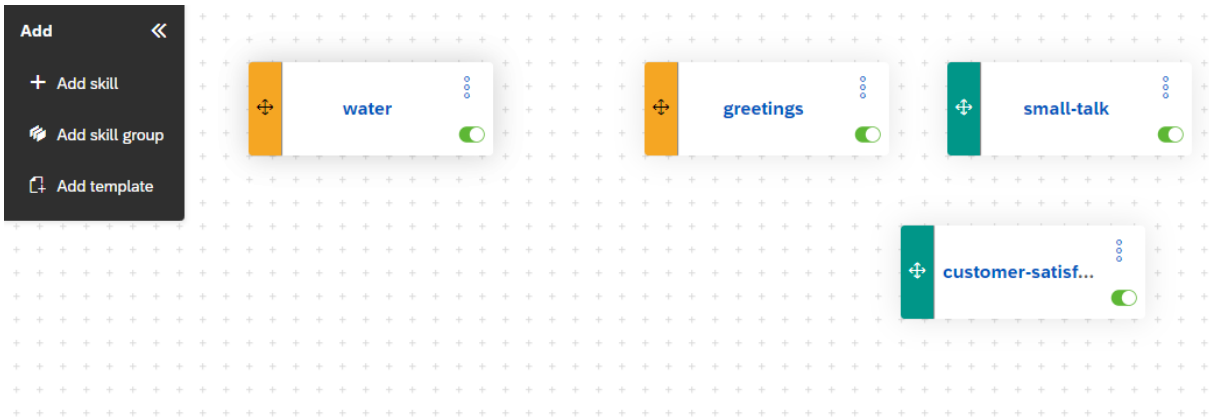
Bot Builder Without Bot Connector

You need to retrieve the user input by your own way, for example, through a channel that you've implemented. You then directly request the Bot Builder API and follow the Dialog endpoints documentation in the API Reference to create a new conversation.

4.2 Skills

A skill is a block of conversation that has a clear purpose and that your bot can execute to achieve a goal. It can be as simple as the ability to greet someone, but it can also be more complex, like giving movie suggestions based on information provided by the user.

You can add a skill to your bot on the *Build* tab by clicking *Add skill* in the command panel on the left. You can add as many skills to your bot as you wish.



A skill is not limited to one exchange with the user. In the movie suggestion example, the skill runs through multiple exchanges. It starts by asking the type of the movie, then the year the movie was released, and then the language of the movie before making the actual suggestion.

You can link your skills together to create more complex conversations. For this you must define an action to indicate which skill should be executed next. For more information, see *Go to another skill* in Actions [page 108].

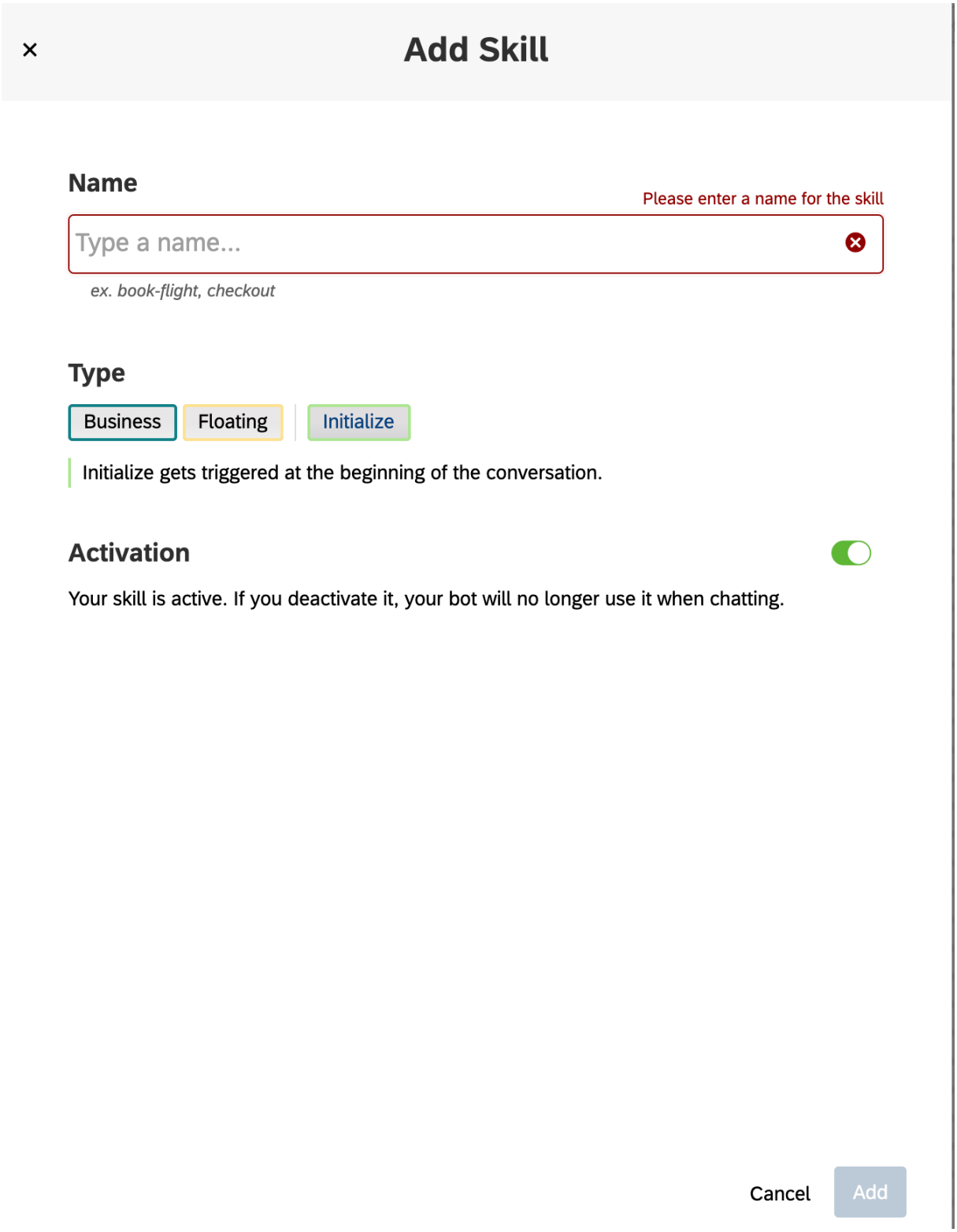
Skill Types

The following skills are created by default with your bot:

Skill type	Description
Fallback	This is a predefined skill that is created by default with your bot. It is triggered if no other skill is triggered. Your bot can only have one fallback skill. So when you add a skill to your bot, the skill type Fallback is offered only if your bot doesn't already have a fallback skill.
Disambiguation	This is a predefined skill that is created by default with your bot. This is triggered during a conversation in case multiple skills are resolved. If the disambiguation skill is deactivated the fallback skill is triggered. See Disambiguation Skill [page 90].

You can create the following types of skills:

Skill type	Description
Business	Skills that are closely linked to the core purpose of your bot.
Floating	Small-talk skills, that is, topics that are not closely related to the core purpose of your bot.
Initialize	Skill that gets triggered at the beginning of the conversation and can be used to initialize memory fields or the conversation language. See Initialize Skill [page 94].

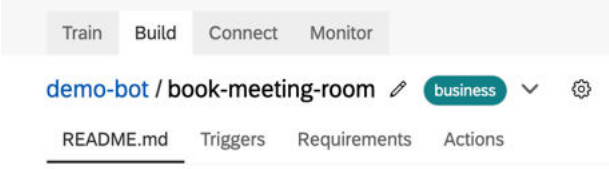


> **i Note**
>
> It is important to define a skill title as this is displayed (as option title) while disambiguating during a chat. If the skill title is not defined, by default, the skill slug is displayed.

Composition of a Skill

A skill is made up of three distinct parts:

- Triggers [page 99]
  Triggers are conditions that determine whether the skill should be activated.
- Requirements [page 100]
  Requirements determine the information that the bot needs to retrieve from the user and how to retrieve it.
- Actions [page 108]
  Actions are performed by the bot (for example, send a message) when all requirements are complete.



Skill Groups

If you have a lot of skills, you can organize them into skill groups for better housekeeping:

1. In the gray command panel on the right, switch to *List view*.
2. In the gray command panel on the left, click *Add skill group*.
3. Enter a name for the skill group and click *CREATE GROUP*.
4. Select the skills you want to add to the skill group, click the *Select group to move* dropdown, select the group, and click the checkmark.
   The skill group is then activated.

Next Step

Conditions [page 96]

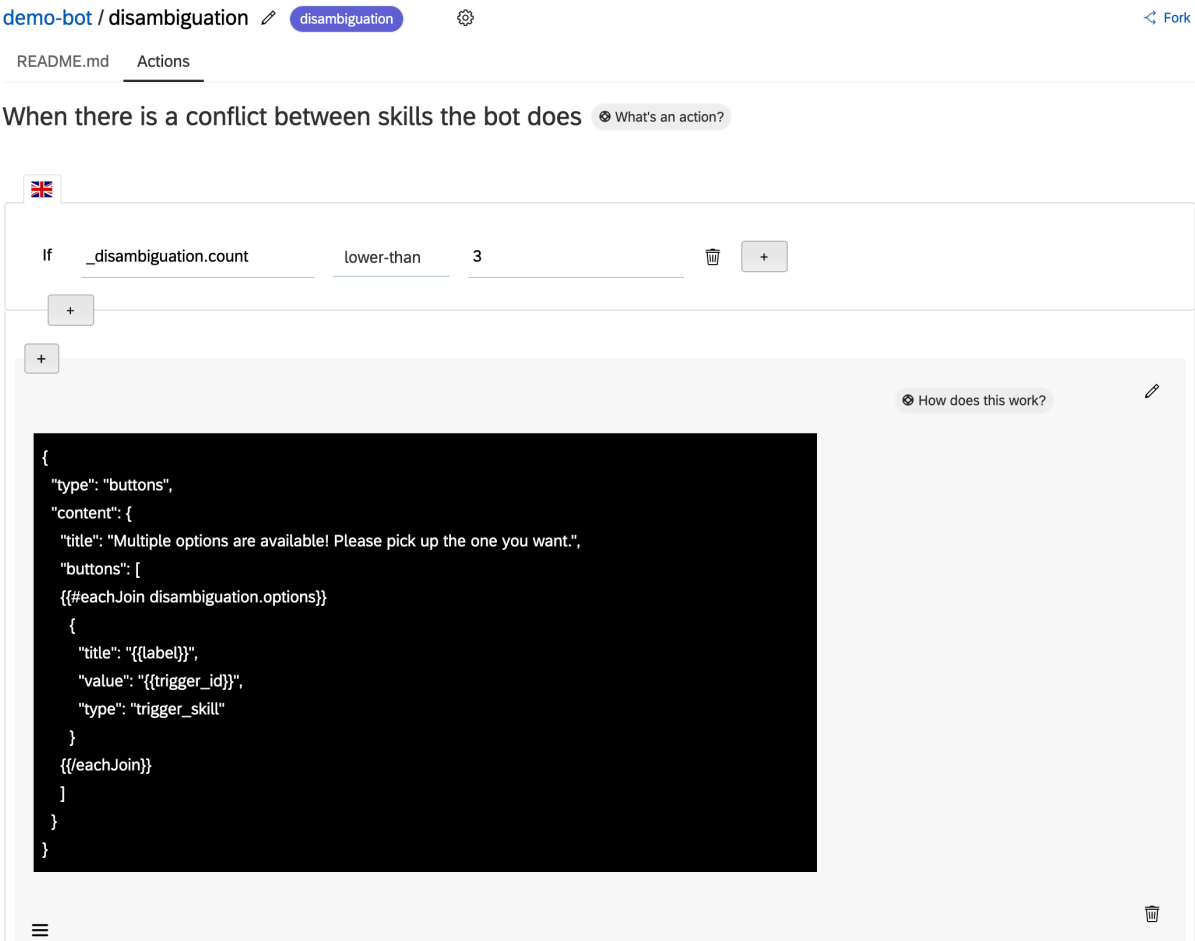
4.2.1 Disambiguation Skill

Disambiguation skill allows the bot to request clarification from the user. In scenarios where user's utterance leads to resolution of multiple skills, the bot triggers disambiguation skill and provides the user with options to choose a skill.

The *Disambiguation* skill is available by default when a bot is created, just like the *Fallback* skill. It is automatically triggered if a user's utterance resolves multiple skills. The user can choose the required option from the list of skills that are displayed in the chat.

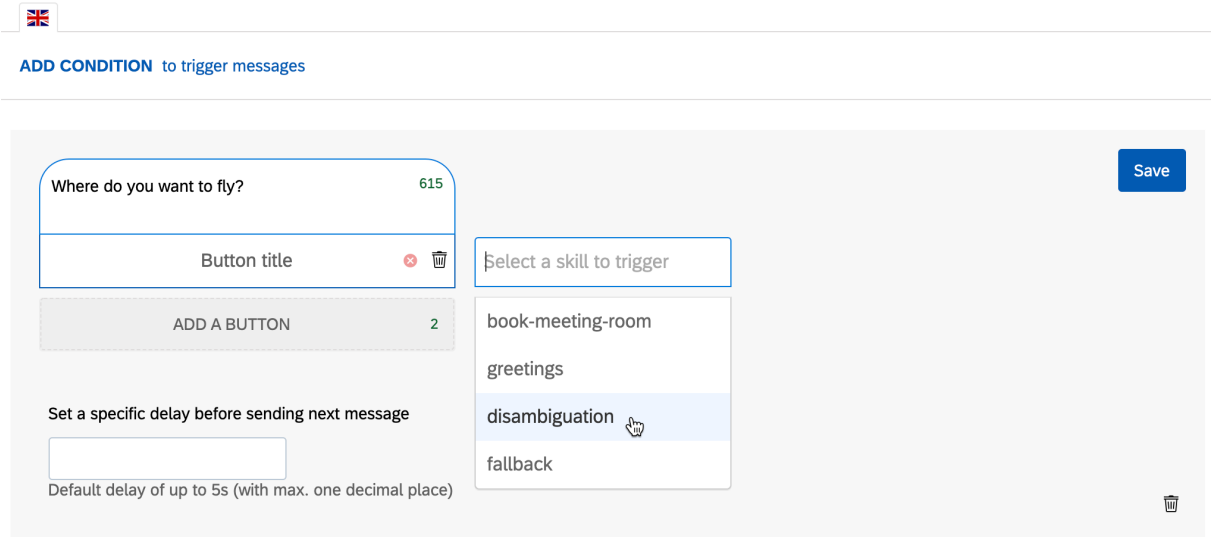
Using the right operand [page 98] *_disambiguation* you can define conditions for messages in the *Actions* tab, to determine if a message group should be executed or not.

You can customize the pre-defined message as per your need.



It is recommended to retain the disambiguation skill or add it (using *Add skill* option), if the bot doesn't have it yet. However, if you don't need the disambiguation skill, you can either deactivate it or delete it.

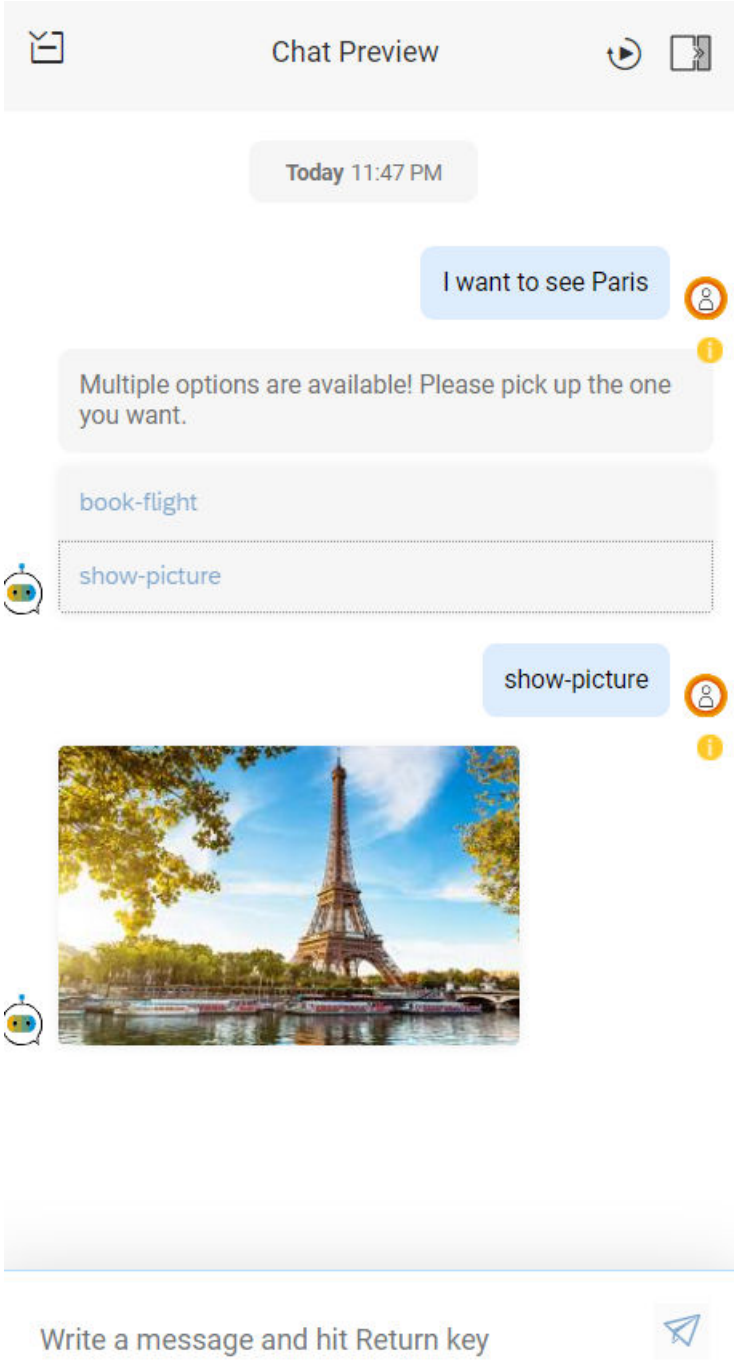
> **i Note**
>
> You can monitor disambiguation skill occurrences under the conversation logs. This will help you train your bot better.



To understand the disambiguation skill better, let us consider a scenario of booking flight tickets. The book-flight bot has the *show-picture* and *book-flight* skills.

If the user enters `I want to see Paris`, the bot determines that the user utterance identifies two skills (*show-picture* and *book-flight*), and to resolve this, it triggers the disambiguation skill instead. This lets the user select a skill of choice and receive the required response.

> **i Note**
>
> By default, the slug of the skill is displayed as the title for each option. You can define a suitable title for each skill that will override the slug. For more information, see Skills [page 87].



Constraints

- If the disambiguation skill is deactivated or deleted, the fallback skill is triggered.
- You can have one disambiguation skill per bot version.
- You cannot switch the disambiguation skill type to business or floating.
- You only need to define the Actions for the disambiguation skill. *Triggers* and *Requirements* tabs are not enabled.

- The disambiguation skill is available in all four advanced languages and can be forked into new bots. It has no language-dependent skill titles (like the fallback). If you create a disambiguation skill, it will be created in the main language of the bot or an empty skill is created for non-advanced languages.
- The disambiguation skill is currently supported by SAP Conversational AI Web Client and Webchat only as the button *Skill Trigger* is not available for other channels.

4.2.2 Initialize Skill

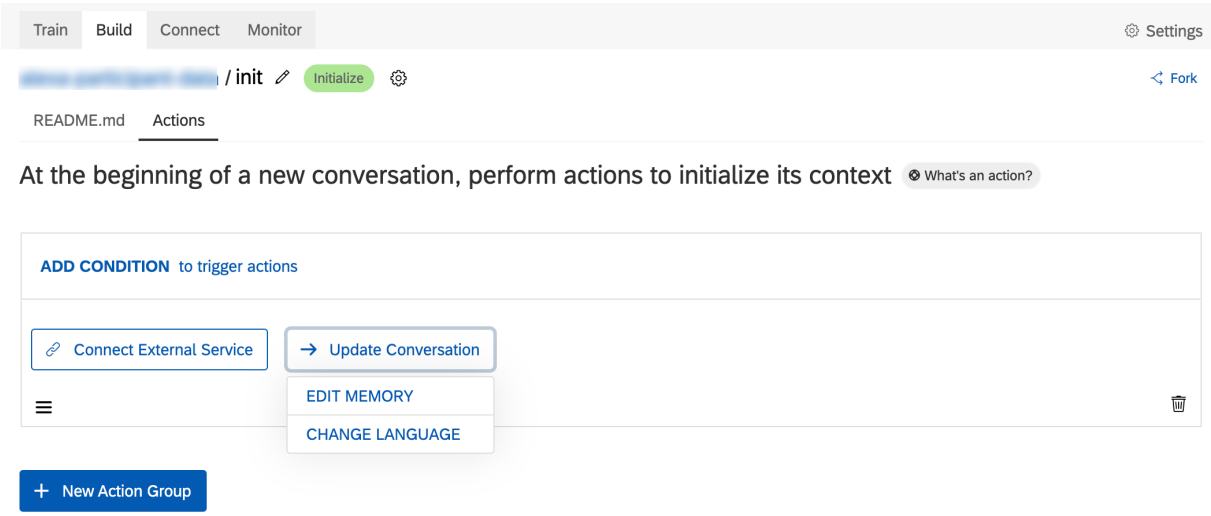
Initialize skill enables you to define some actions that are required to initialize a conversation with your chatbot. This skill is automatically executed at the beginning of the conversation and can be used to initialize the memory or the language of the conversation.

As the *Initialize* skill is executed before the natural language processing is performed, the language and other NLP related information (like intents and entities) are not yet present in the conversation context. All actions and condition operands in the Initialize skill are therefore language independent and the runtime data, accessible as condition operands and through scripting syntax, are limited. Messages cannot be sent to the user in the Initialize skill.

Supported Actions

The following actions are supported in the Initialize skill. For more information, see Actions [page 108].

- Call Webhook (*replies* returned by the webhook are ignored, only memory and language are considered)
- Consume API Service
- Edit Memory
- Change Language



Runtime Data

The following runtime data are supported in the Initialize skill. For more information, see Runtime Data Accessible [page 179].

- `{{conversation_id}}`
- `{{participant_data}}`

- `{{memory}}`
- `{{client_id}}`
- `{{current_message}}`

You can create the Initialize skill using the *Add Skill* option under the *Build* tab. It is recommended to create it only when it is needed. Actions added in the Initialize skill (for example, API calls to external systems) are executed when the user enters the first expression in the conversation, which means that the bot will take slightly longer to respond for the first time.

If you have set some memory variables at the beginning of the conversation using the Initialize skill, these variables are removed with the use of *RESET CONVERSATION* action or when all the memory is reset using *EDIT MEMORY* action. In such cases, you can use the *GO TO* action to explicitly execute the Initialize skill.

For more information, see the following video.

Example Use Cases

The following table lists the use cases for Initialize skill and an example for each:

Use Case	Example
Retrieve or transform context information only once instead of multiple times in various skills	Additional context information about the current user like office location can be fetched using API calls at the beginning of the conversation and stored in memory for later use during the bot conversation.
Personalize bot behaviour by fetching context information at the beginning of the conversation	The information whether a current user is an employee or manager, fetched via API call at the beginning of the conversation, is stored in the memory and this memory field is used to trigger different skills for employees and managers.
Enable static information to be initialized in a central place	The base URL for external links returned in various skills of the chatbot is initialized at the beginning of the conversation and can be easily replaced later in a central place or can be determined dynamically.

Constraints

- You can have one Initialize skill per bot version.
- You cannot switch the Initialize skill type to business or floating.
- You only need to define the Actions for the Initialize skill. *Triggers* and *Requirements* tabs are not enabled.

Related Information

Messaging Channels [page 181]
Client Information [page 194]

4.3 Conditions

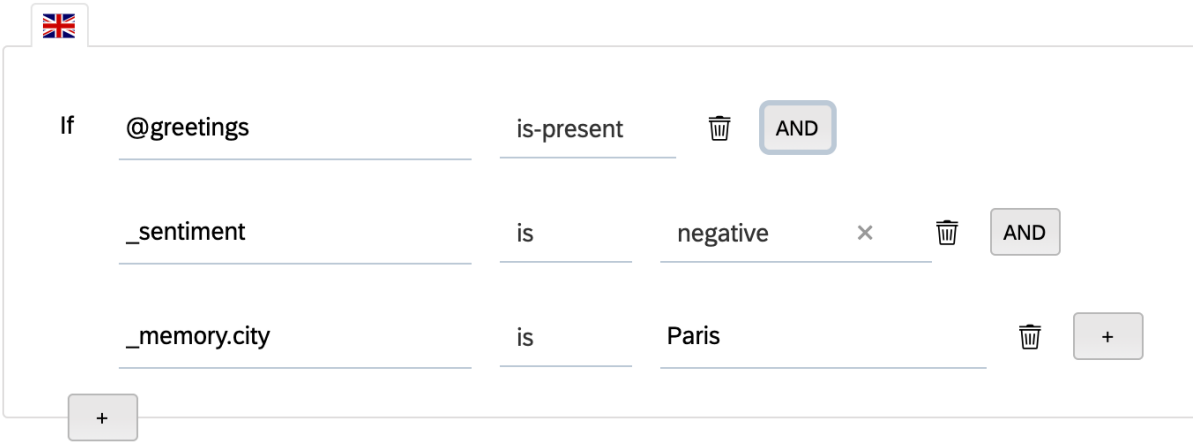
A condition is a test that can be evaluated to be either true or false.

You can find conditions in the following different parts of a skill [page 87]:

- Triggers [page 99]
- Requirements [page 100]
- Actions [page 108]

In the example below, we test the following:

- If the intent is greetings
- If the sentiment analyzed in the sentence is negative
- If the value saved for city in the conversation memory is Paris



Composition of a Condition

A condition is made up of three parts: A left operand, an operator, and a right operand. For example, in the condition *if #location.raw is Paris*, the left operand is *#location.raw*, the operator is *is*, and the right operand is *Paris*.

Left Operand

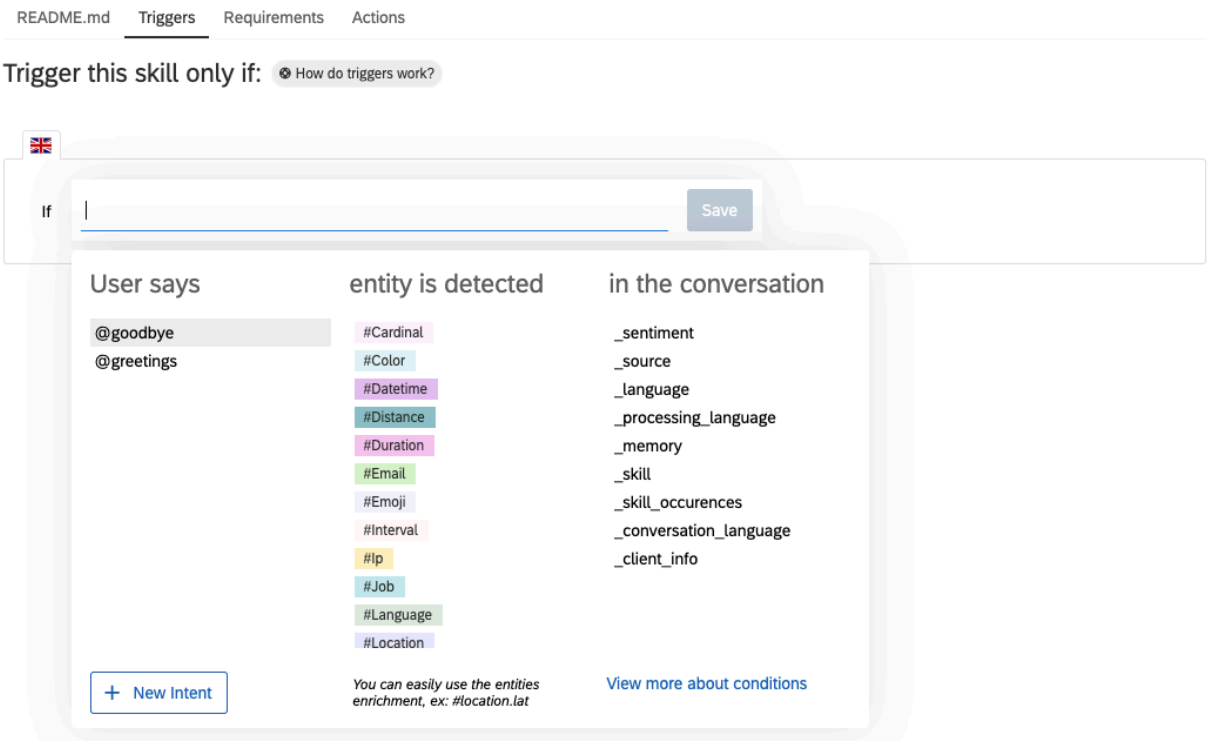
As the left operand, you can use any value of text analyze (intents, entities, etc.) from your user input and any value from the conversation state [page 132] (the last skill, memory values, etc.).

Below are a few rules to distinguish the left operand categories:

- Operands starting with @ get the associated intent (for example, *@greetings*)
- Operands starting with # get the associated entity (for example, *#location*) and test the *raw* field unless you've specified one

- Operands starting with _ get the associated field in the text analyze JSON or in the conversation state

> **i Note**
>
> If you wish, you can write the entire path. For example, if you need the latitude, *#location.lat* is the same as *nlp.entities.location.lat*. Or, if you need to access the description of the first intent detected, you can write *nlp.intents[0].description*.



Operator

You can choose from the following operators. The regex syntax follows the Ruby regex syntax.

Operator	Description
is	Test the equality between two values.
is-not	Test the inequality between two values.
in	Check if a value is in a list of elements.
not-in	Check if a value is not in a list of elements.
matches	Match a value with a regular expression.
matches-not	Check if the value doesn't match with a regular expression.

Operator	Description
lower-than	Test if the value is lower than another. This operator works only with numerical values.
greater-than	Test if the value is greater than another. This operator works only with numerical values.
is-present	Test if the value is present in the conversation state [page 132].
is-absent	Test if the value is absent from the conversation state [page 132].

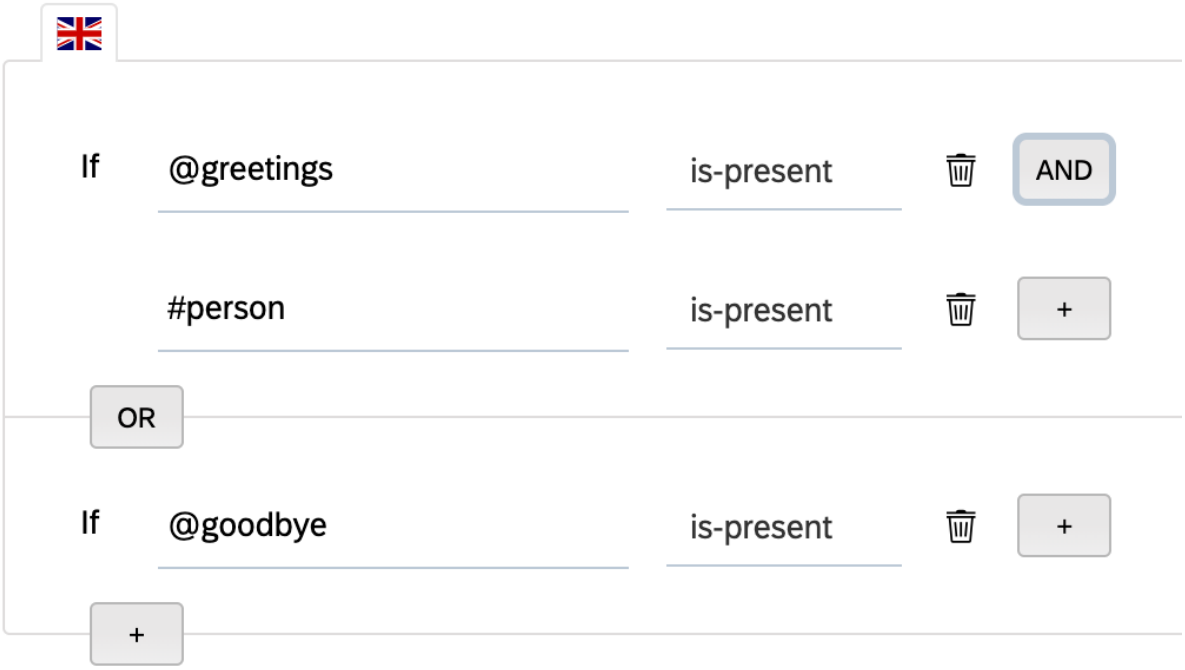
Right Operand

The right operand can either be a free input or a finite list, depending on what you've picked as the left operand. For example, if the left operand is *_sentiment*, the right operand is limited to what the SAP Conversational AI API can return (in this case, from *very positive* to *very negative*). However, if the left operand is *_memory.my_value.my_key*, the right operand isn't dependent on the SAP Conversational AI API, so any format is supported.

For more information about entity enrichment and other keys with a finite list of possible values, see the Glossary in the API Reference.

Complex Conditions

You can create multiple layers of conditions using *and* and *or*. For example, in the following screenshot, the first group (inline) is an *and* condition group, while the second group (entire block) is an *or* condition group.



Next Step

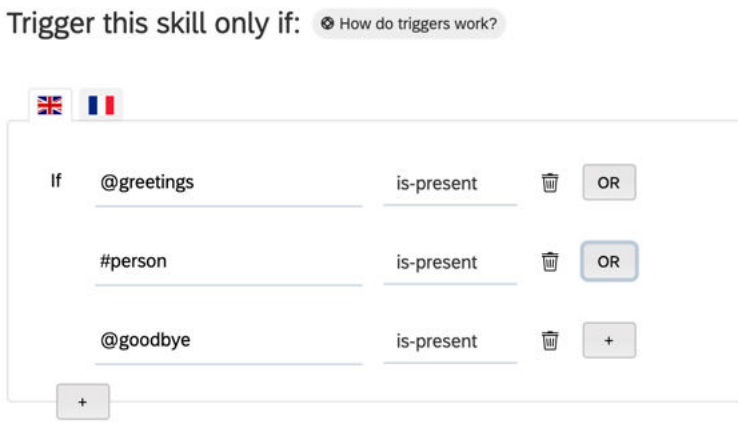
Triggers [page 99]

4.4 Triggers

Triggers are conditions [page 96] that determine whether the bot should execute the current skill or not. If the triggers for the skill are validated, the bot executes this skill over other skills.

> **i Note**
>
> If a skill is already running or is on top of the skill stack (of the Conversation State [page 132]), then both triggers and requirements of the one which is on top, are evaluated first.

You define the triggers for a skill by clicking the skill on the *Build* tab and then opening the *Triggers* tab.



If a skill has no triggers, it will never be executed by a user input. In this case, it will only be executed if it is at the end of a redirection by another skill.

Skills with the skill type *Fallback* do not have triggers because they are automatically triggered when no other skill is triggered or if an error occurs (for example, if two skills are triggered at the same time). Remember that your bot can only have one fallback skill.

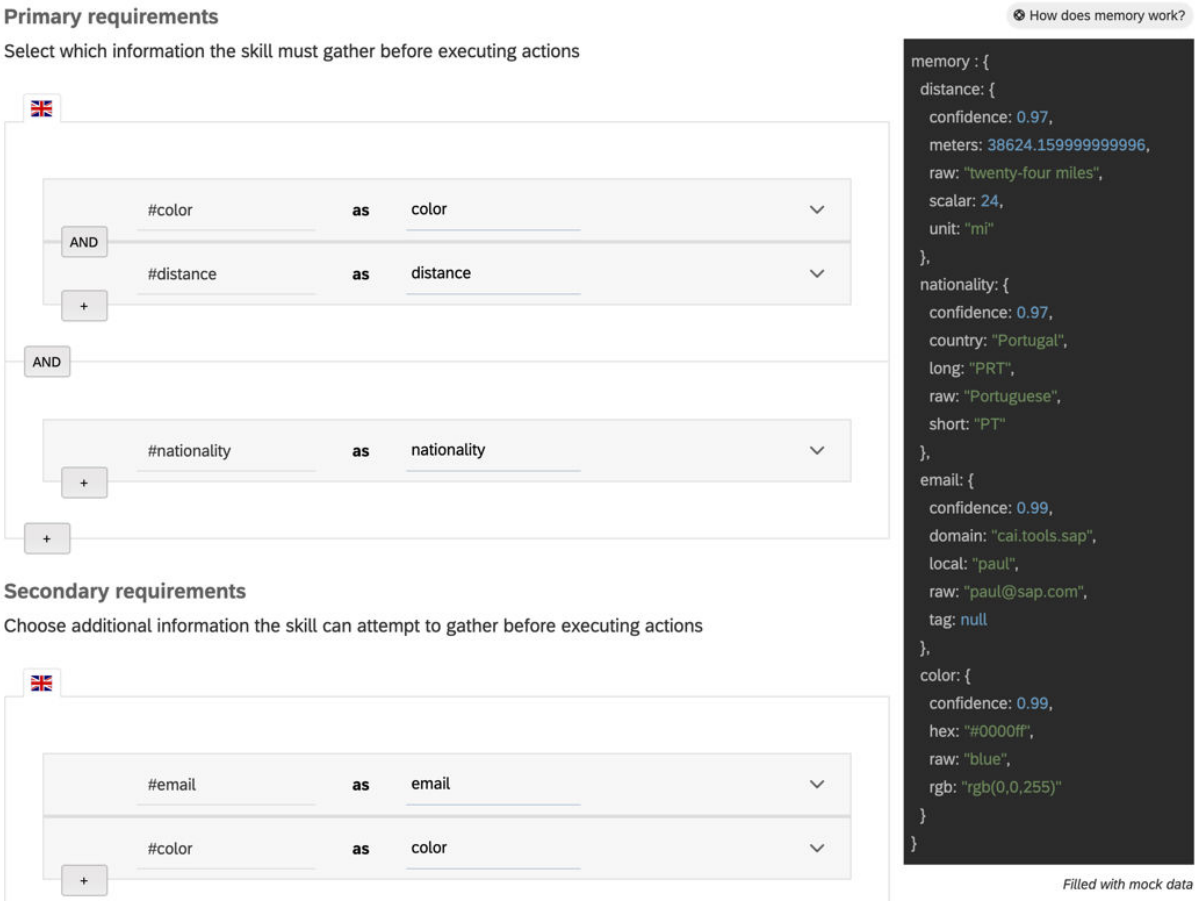
4.5 Requirements

Requirements are either intents [page 27] or entities [page 31] that your skill needs to retrieve before executing actions [page 108]. Requirements are pieces of information that are important in the conversation, and that your bot can use, for example, the user's name or a location.

Requirements can be mandatory or optional. Mandatory information is defined under *Primary requirements*.

The additional pieces of information that the user might provide are considered as optional. They are defined as *Secondary requirements* and can be used to further filter the data. For example, the user provides name or location under *Primary requirements* and currency under *Secondary requirements*, the currency is considered optional. The values for the optional entities are evaluated only after the primary requirements are evaluated and stored. The bot generates the result based on the information provided. If the user does not provide the secondary requirements, the skill will still work.

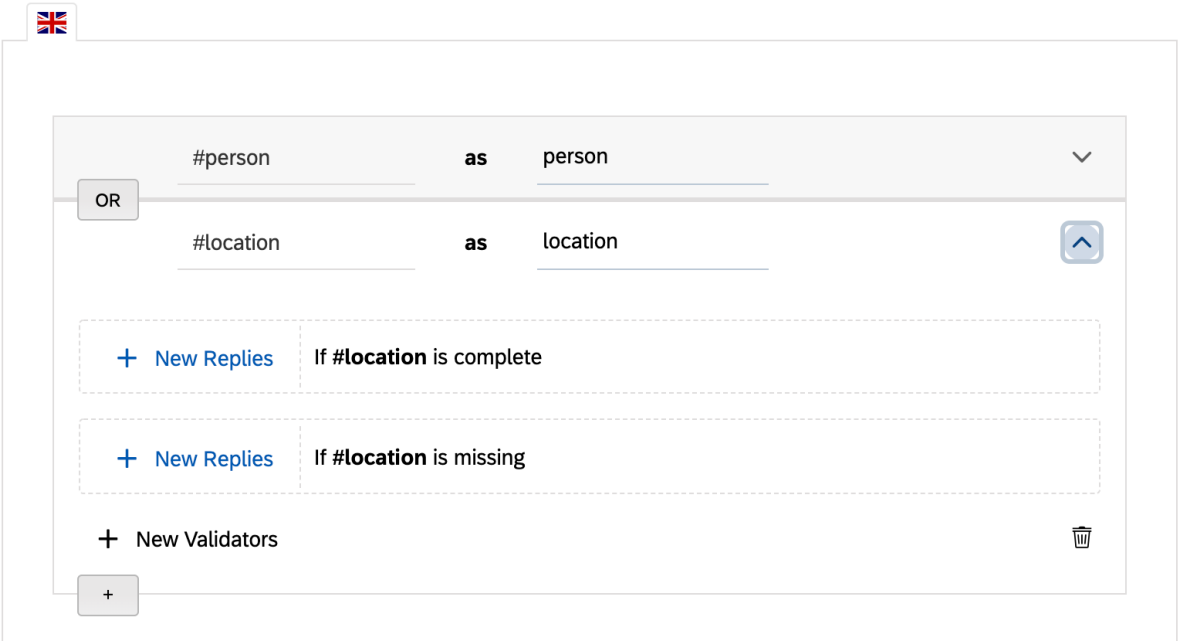
Once a requirement is completed, the associated value is stored in the bot's memory for the entire conversation. If the user doesn't provide values for the optional entities, the associated values for these entities will not be stored.



Composition of a Requirement

A requirement is made up of the following:

- Data to retrieve (that is, an entity or intent)
- Key to store the retrieved data in the bot's memory
- Optional actions to execute to retrieve the information
- Optional actions to execute when the information is retrieved and the requirement is completed
- Optional conditions to validate the data provided by the user, with associated actions to execute if the validation fails



Optional Actions to Execute to Retrieve the Information (Only for Primary requirements)

These actions are executed when the requirement is not yet completed. It's the perfect place to define messages to ask the user for the information you need. For example, with the username as a requirement, the action would be a message asking the user for their name.

> **i Note**
>
> You cannot define actions to execute to retrieve the information for *Secondary requirements* as the skill will work even if the user doesn't provide information for these entities.

Optional Actions to Execute when the Information is Retrieved and the Requirement is Completed

These actions are executed when the requirement is completed. The skill's execution then continues as far as it can. It can chain with other requirements or with actions.

Optional conditions to validate the data provided by the user, with associated actions to execute if the validation fails

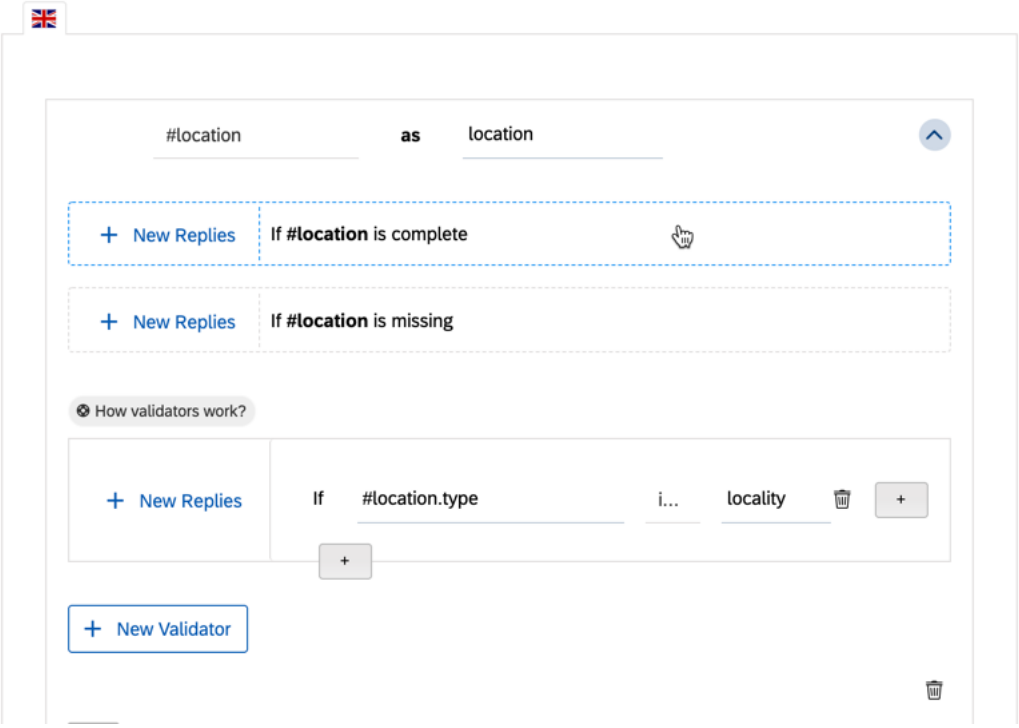
You can define validators in a requirement to validate that the user input matches your needs. These validators are made up of conditions [page 96] and actions to execute in case of a validation error.

The validation fails if the condition is *true*. In such cases, the retrieved data is not stored in memory and the requirement is not completed. For example, if we want to get a city from the user, we can create a requirement that retrieves a location entity. We can add a validator to check that this location really is a city (and doesn't, for

example, refer to a country). If the location isn't a city, we can then ask the user for a city. You could write it like this:

Sample Code

```
if #location.type is-not locality
  send_message('Can you please give me a city ?')
```

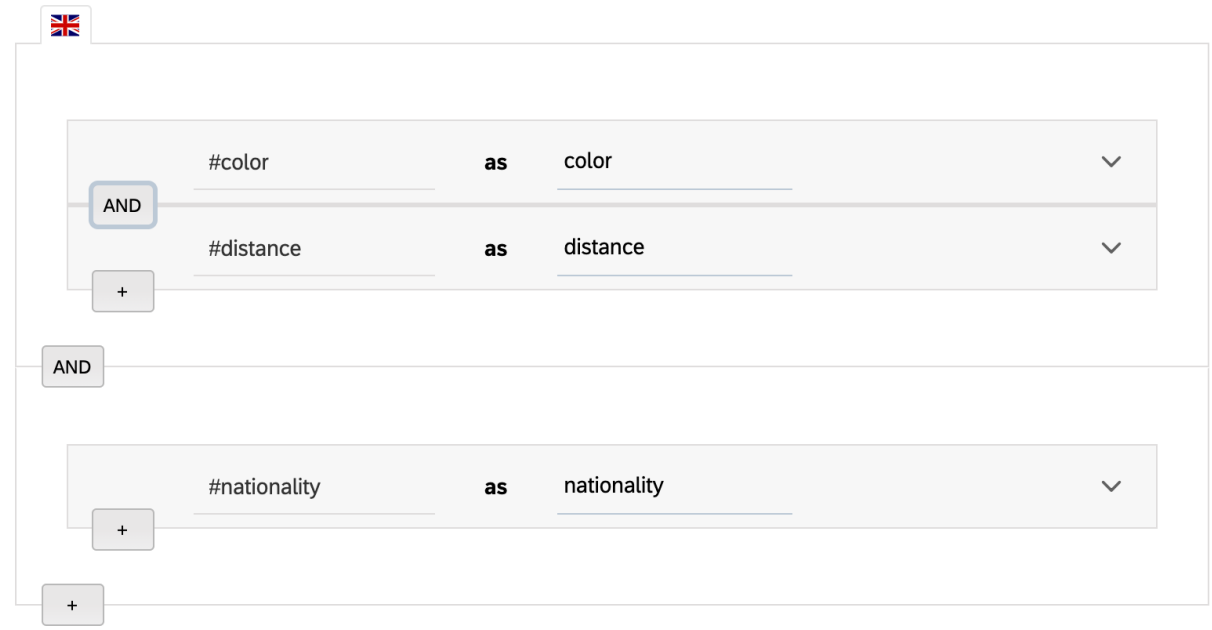


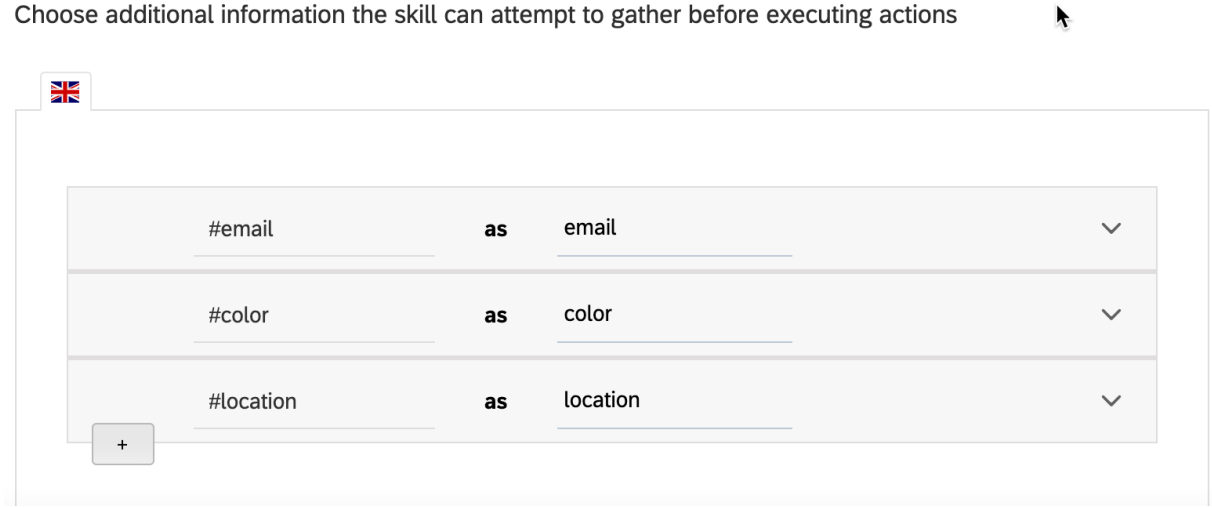
Grouping Requirements Together (Only for Primary requirements)

Like conditions, you can group requirements together with *OR* and *AND*.

Note

You can not group secondary requirements (entities) based on conditions.





Examples

Let us consider an example where the `supplier` is defined as the *Primary requirements* and the `category` is the *Secondary requirements*.

Possible combination of responses for both primary and secondary requirements

Primary requirements	Secondary requirements	Scenarios	Sample Conversation
`Supplier`	`Category`	Both values are successfully retrieved.	User: **`Show me products from supplier Avantel and category headsets.`** The bot retrieves a list of products from *supplier* `Avantel` and *category* `Headsets`.
`Supplier`	`Category`	Value for secondary requirements (`category`) couldn't be stored due to validation failure.	User: **`Show me products from Avantel in category headsets.`** Bot: **`I found Supplier Avantel.`** **`Category headset does not exist. Please suggest another category.`** User: **`Laptops`** The bot retrieves a list of products from *supplier* `Avantel` *category* `Laptops`. **i Note** If you have defined multiple entities for the secondary requirements, and there are multiple validation failures, then only the first failure is considered.

Primary requirements	Secondary requirements	Scenarios	Sample Conversation
`Supplier`	`Category`	User doesn't provide the primary requirements (`supplier`).	User: **`Show me products from category headsets.`** Bot: **`Please provide the supplier.`** User : **`Avantel`** The bot retrieves a list of products from *category* `Headsets` and *supplier* `Avantel`.
`Supplier`	`Category`	Value for secondary requirements `category` couldn't be stored due to validation failure and user doesn't provide the primary requirements (`supplier`	User: **`Show me products from category headsets.`** Bot: **`Category headset does not exist. Please suggest another category.`** User: **`Laptops`** Bot: **`Who is the supplier?`** User: **`Avantel`** The bot retrieves a list of products from *supplier* `Avantel` and *category* `laptops`.

Primary requirements	Secondary requirements	Scenarios	Sample Conversation
`Supplier`	`Category`	Value for primary requirements `supplier` couldn't be stored due to validation failure.	**User**: **`Show me products from supplier Avantel and category headsets.`** **Bot**: **`Supplier Avantel does not exist. Please suggest another supplier.`** **User**: **`Talpa`** The bot retrieves a list of products from *category* `Headsets` and *supplier* `Talpa`.
`Supplier`	`Category`	Values for primary requirements `supplier` and secondary requirements `category` couldn't be stored due to validation failure.	**User**: **`Show me products from supplier Avantel and category headsets.`** **Bot**: **`Category headset does not exist. Please suggest another category.`** **User**: **`Laptops`** **Bot**: **`Please provide value for supplier.`** **User**: **`Avantel`** **Bot**: **`Supplier Avantel does not exist. Please suggest another supplier.`** **User**: **`Talpa`** The bot retrieves a list of products from*supplier* `Talpa` and *category* `Laptops`.

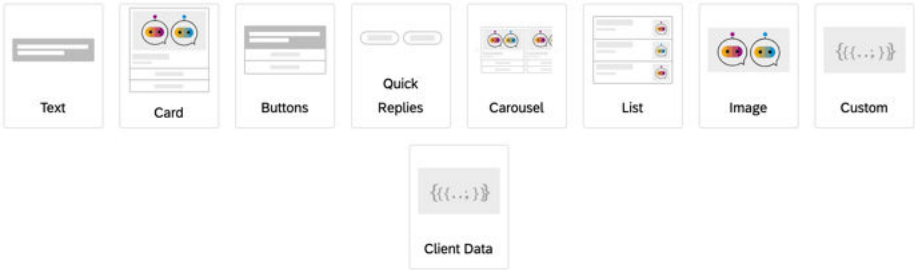
4.6 Actions

An action is something that your bot executes at a specific point when executing a skill [page 87]. To add actions to askill, open the skill on the *Build* tab and then go to *Actions* and click *+ New Message Group*.

Action Categories

An action can be one of the following:

- Send message to the user
- Connect external service
- Fallback (that is, redirect the conversation to a human agent)
- Go to another skill
- Edit the bot's memory for the current conversation
- Change language
- Reset Conversation



A message group can contain one or more actions. You can easily reorder actions within a message group by drag and drop; look for the *Move* icon below the action on the left. You can also reorder entire message groups by drag and drop; look for the *Move message group* icon at the bottom of the message group on the left.

Send Message to the User

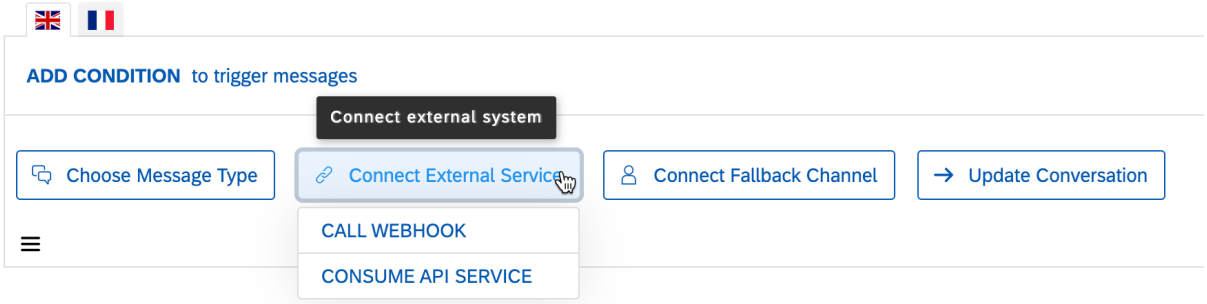
Various formats exist, enabling you to build an awesome user experience for your bots.

If your bot is connected to a channel through the Bot Connector, the message type is adapted to the channel constraint and transformed, so the look and feel will probably change compared with what you see on the SAP Conversational AI platform. For more information, see Messages [page 114] > *Formats*.

You can dynamically inject the content gathered from the conversation in the bot replies by using double brace syntax. For example, if your bot asks for the user's name as a requirement, the name is added to the bot's memory once the requirement is completed. You can then create a text message (or any other message actually) filled with "Hello {{memory.username.raw}}", where {{memory.username.raw}} is replaced with the actual username. For more information, see Messages [page 114] > *Variables*.

Connect External Service

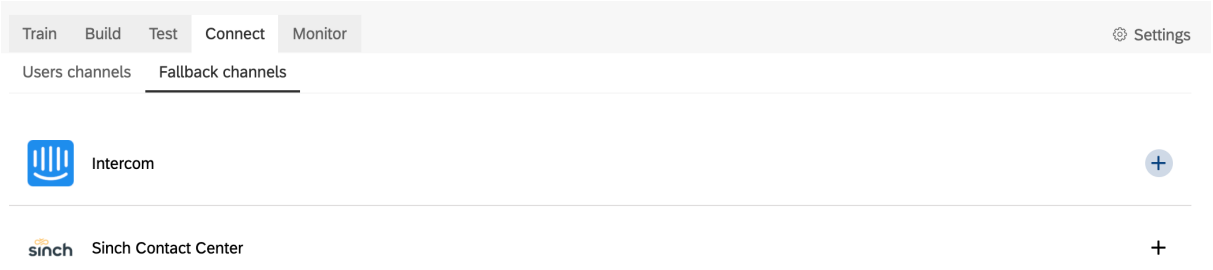
At many points in your conversation, you most likely want to retrieve business information or connect to an external system to perform actions. You can do this through *Connect External Service*. Either you can call a webhook that expects a JSON response in correct CAI format back, or you can consume any JSON response from an API service. For example, a webhook is a simple HTTP call to your back end. To configure your HTTP call, click *CALL WEBHOOK* in the Bot Builder.



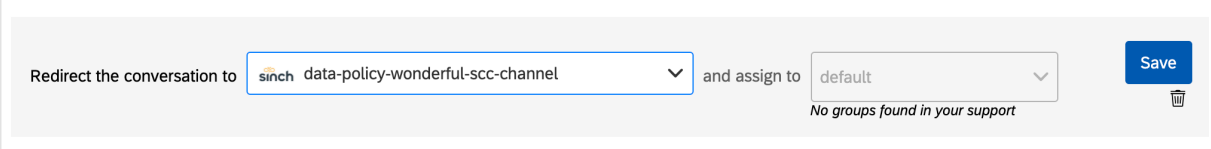
When your URL is called, a default body is sent with the complete conversation state. You can send back messages you want to send to the user, as well as an updated conversation state. For more information, see Connect to External Service [page 136].

Fallback

This action lets you redirect the conversation to a human agent. First, you need to connect the fallback channel where you want SAP Conversational AI to redirect the message. You can do this on the *Connect* tab by selecting a fallback channel and following the instructions. After connecting the fallback channel, remember to activate it by checking the input.

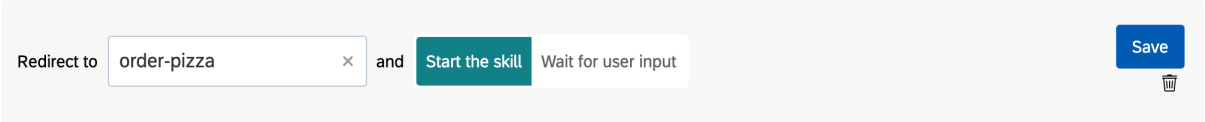


In a skill, you can configure a fallback action by selecting your fallback channel and the group to which you want to redirect the conversation. (Usually, your support center is organized into different groups.) For more information, see Fallback Channels [page 205].



Go to Another Skill

You can use this action to indicate which skill should be executed next.



You have two options:

- Start the skill
  This bypasses the triggers of the skill and directly tries to resolve the requirements and actions.
- Wait for user input
  This serves as an indicator. The next user input will try to enter this skill in priority, but triggers will be applied.

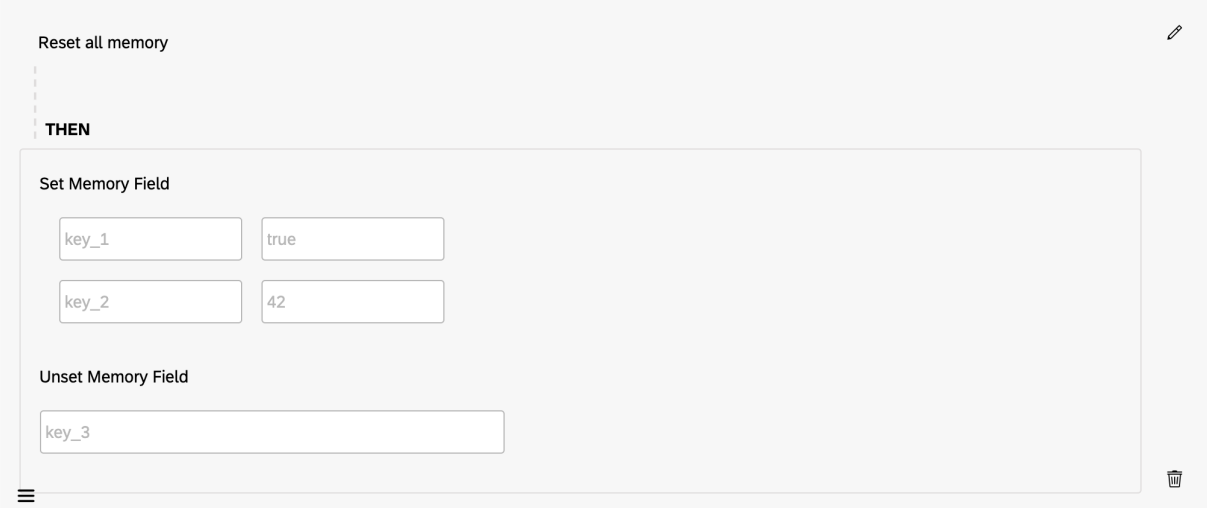
> **i Note**
>
> You can use a variable instead of the name of the skill, as described above in *Send message to the user*.

Edit the Bot's Memory for the Current Conversation

This action lets you do three different things:

- Reset the entire memory of the conversation
- Set values in the memory
- Unset a key in the memory

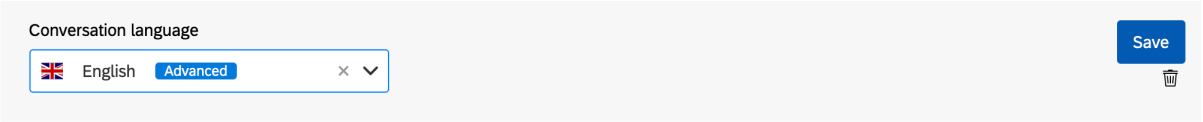
First the memory is reset, then the new values are set, and finally the specified keys are unset. For more information, see Memory Management [page 111].



> **i Note**
>
> You can also use variables, as described above in *Send message to the user*.

Change Language

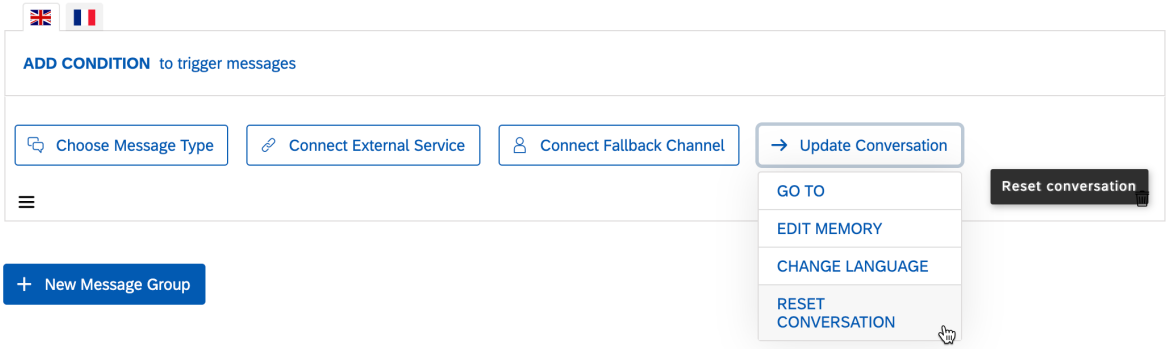
This action lets you change the language of the conversation. It can be especially useful if your user asks, for example, *Can you speak Spanish?*



Reset Conversation

You can use this action to clear the skill stack, memory and language information. This lets you reset the conversation and ensure that any new utterance after this action is evaluated as a new conversation (with the same conversation ID).

> → Remember
>
> Using Reset Memory erases all the keys but `skill_stack` information is not erased, and the current skill still exists in the stack waiting for requirements to be fulfilled.
>
> Avoid having the *Trigger Skill* action after the *Reset Conversation* action in your conversation flow. Since the trigger skill action by-passes the NLP , the bot will not be able to respond due to missing language that has been reset by the Reset Conversation action.



4.7 Memory Management

Your Conversation Memory

Each conversation with a unique user has a memory object from the beginning to the end of the conversation. This memory persists during the entire conversation; you can update it at any time or clear it whenever you want.

When a new conversation starts, the memory is an empty object (unless you want to start a new conversation with prefilled keys). The memory is stored in your conversation state [page 132]. It is returned in the default body of a webhook and after the Bot Builder API call. See also Connect to External Service [page 136] > *Body configuration*.

For example, your memory object could look like this:

Sample Code

```
"memory": {
  "person": {
    "fullname": "John",
    "raw": "John",
    "confidence": 0.95
  },
  "orderId": "ED456-G"
}
```

Lifecycle Storage

The memory key is created when the value can be filled by the requirement, or through a configuration of an edit memory action (see below), or through your code in a webhook call.

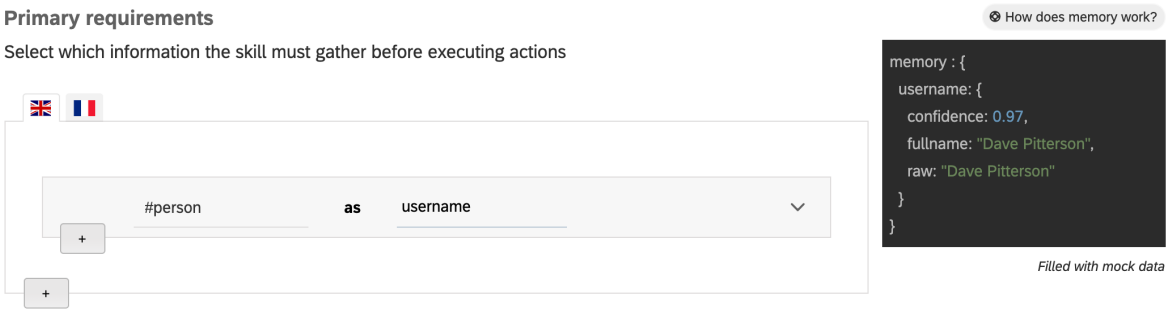
The memory key and value never change and are never overwritten during the entire conversation, unless you configure an update in an edit memory action.

How to Manipulate Memory in the Platform

The memory can be filled automatically through the requirements or manually through a configuration of an edit memory action.

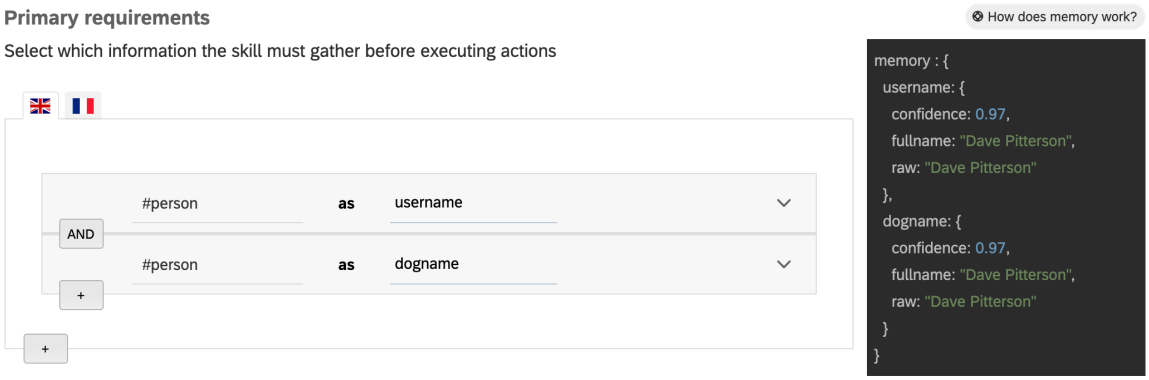
Filling the Memory Through the Requirements

A requirement can be either an entity or an intent that is detected in the user input. When you create a requirement, it is automatically detected and saved in your memory with the key that you choose.



You can reuse information at different moments in the conversation. For example, if you need the name of the user in two different skills, you create the same requirement *#person* (the entity that represents the person's name) in each skill. If you start the conversation with the first skill, the bot asks for the user's name. If the user replies with her name, the requirement is completed and the name is saved in the memory. When the conversation enters the second skill, the bot doesn't ask again for the user's name because the key is already filled in the memory.

Here's another example: You need to store the same entity *#person* twice, but for different purposes. You want the name of your user and also his dog's name. You therefore create a requirement with *#person* saved as *username*, and a second requirement with *#person* saved as *dogname*.



Filling the Memory Through an Edit Memory Action

Click *Edit memory action*. You have the following options:

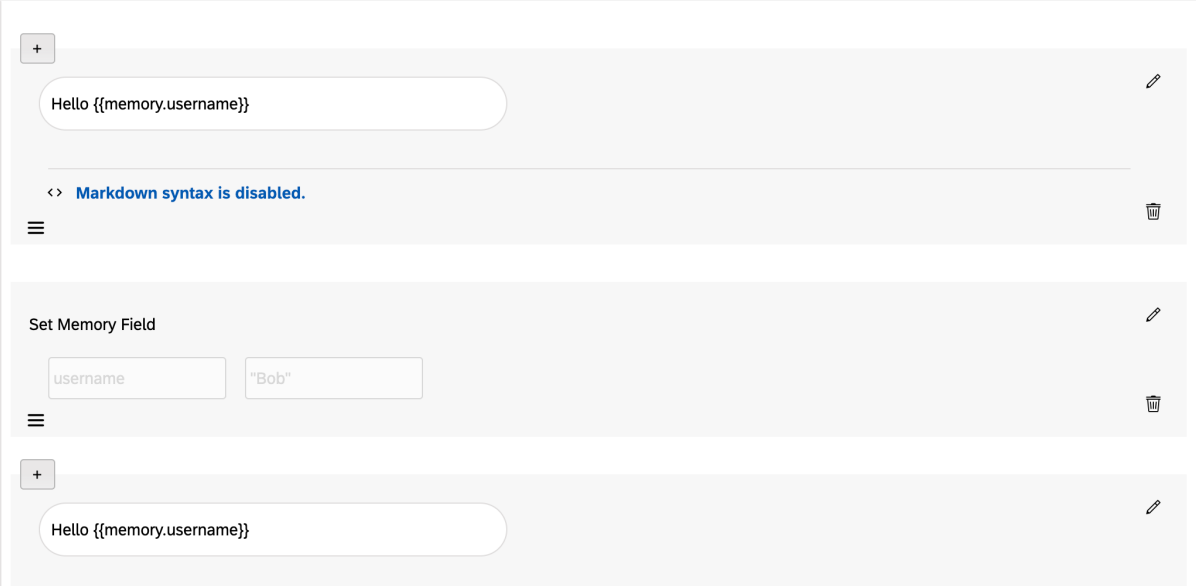
- You can reset all the memory (that is, erase all the keys) and reset new fields
- You can just set a new field and unset others

Be sure that the value is proper JSON.

These memory modifications must be done synchronously. For example, if you configure a text message with a variable *Hello {{memory.username}}*, then edit the memory by replacing the value of username with Bob, and then configure a new message *Hello {{memory,username}}*, you will have these bot replies:

Hello John (assuming John was given previously by the user)

Hello Bob



How to Manipulate Memory in Webhook Custom Code

You can edit the memory in your code during a webhook call. To understand how to format your response, see Connect to External Service [page 136]. Here's an example of how to format the return of your webhook call and update the memory of the conversation.

Sample Code

```
{
  "conversation": {
    "memory": {
      "username": "bob"
    }
  }
}
```

`memory` will replace the actual memory of your bot (so be careful that you don't lose everything if you just want to change one of your memory keys to add all your other keys).

You can also update the memory through an API call. For more information, see Update a conversation in the API Reference.

How to Start a Conversation with Memory

You can start a conversation with prefilled information in the memory, and not wait until the first user input is analyzed and the first skills are triggered. However, this is only possible if you are using the Bot Builder directly (without the Bot Connector). To understand how to use the Bot Builder API to do this, see /Dialog (Text) in the API Reference.

If you're using Webchat, you can easily start a conversation by sending information in the memory. For more information, see Webchat [page 206] and scroll down to *Bot memory management*.

4.8 Messages

How to Create Messages on the Platform

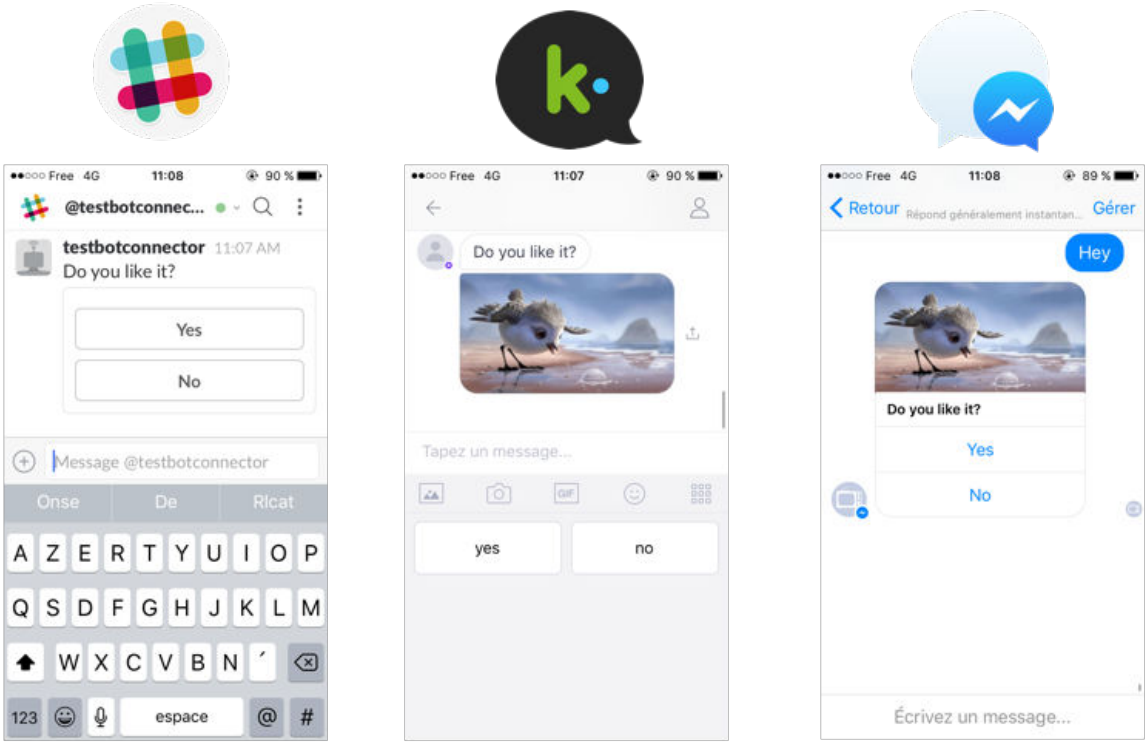
On the *Actions* tab of a skill [page 87] (or on the *Requirements* tab), you can choose to send messages.

Formats

Many different formats are supported, enabling you to build an awesome user experience for your bots. The following table lists the various formats and their advantages. For more information, see Message Types [page 120]

Format	Description
Text	Great for simple informative messages, even if the 640-character limit is quite high. We recommend keeping them short if you want your users to read them.
Card	Very useful for presenting a product because you can include an image, title, subtitle, and so on.
Buttons	Practical if you want to guide the user in the conversation with a few limited choices.
Quick replies	Appear as buttons in the chat with predefined user responses but disappear once clicked. Great if you don't want the user to have to scroll up the conversation and click a button again.
Carousel	A succession of cards that you can scroll from right to left, usually used for presenting multiple products.
List	Same purpose as a carousel but presented as a vertical list so that you can see everything at once, unlike with the carousel, where you have to scroll. However, a list is not quite as big as a carousel, and the images are smaller.
Image	A great way to post entertaining GIFs.
Client Data	Allows you to return additional data to the SAP Conversational AI Web Client in JSON format. Messages of this type are not displayed on screen.
Custom	Allows you to define the above mentioned message types with unified scripting syntax to consume your bot memory and API service response directly into the platform and create responses dynamically. Additionally, if you want to use custom messages supported by specific channels (like Blocks for Slack, Message Templates for Facebook Messenger or Flex Messages for Line), select the *Custom* type in the dropdown inside this tile to define the responses.

If your bot is connected to a channel through the Bot Connector, these message types are adapted to the channel constraint and transformed, so the look and feel will probably change compared with what you see on the SAP Conversational AI platform.



> **i Note**
>
> If you specify an image, the image must have the protocol HTTPS to be displayed correctly when the action is triggered.

Character Limits

On the platform, a character limit is displayed for every message. For example, a text message has a limit of 640 characters. This isn't a real limitation; you can still create a text message with more characters. It serves as an indication based on what Facebook Messenger will accept. So if you're using Messenger, it's a good idea to observe the character limit; otherwise your messages won't be posted in the user's conversation.

Markdown

When creating text messages or quick replies, you can opt to use markdown to format text as **bold**, *italics*, or as a hyperlink . This requires you to select the *Enable Markdown syntax* checkbox.

For **bold**, add two asterisks (**) or two underscores (__) before and after a word or phrase. For example, "Tell me what you want, what you **really, really** want" will be rendered as "Tell me what you want, what you **really, really** want".

For *italics*, add a single asterisk (*) or single underscore (_) before and after a word or phrase. For example, "This is how you _italicize_ text" will be rendered as "This is how you *italicize* text".

> **i Note**
>
> You cannot change the font size and color of the text. You are limited to the basic Markdown features that are available.

For hyperlinks , use [link text](URL). For example, "Find us at [SAP Conversational AI](https://cai.tools.sap)" will be rendered as "Find us at SAP Conversational AI ". If you don't provide a link text, the URL itself will be rendered as the link. For example, "Find us at [](https://cai.tools.sap)" will be rendered as "Find us at https://cai.tools.sap ".

For a preview of how your text message or quick reply will be rendered, simply save it.

Markdown in the text messages and quick replies that you create in SAP Conversational AI is supported in the following channels:

- Facebook Messenger [page 183]
- Skype [page 184]
- Slack [page 203]
- Telegram [page 203]
- Webchat [page 206]
- SAP Conversational AI Web Client [page 185]

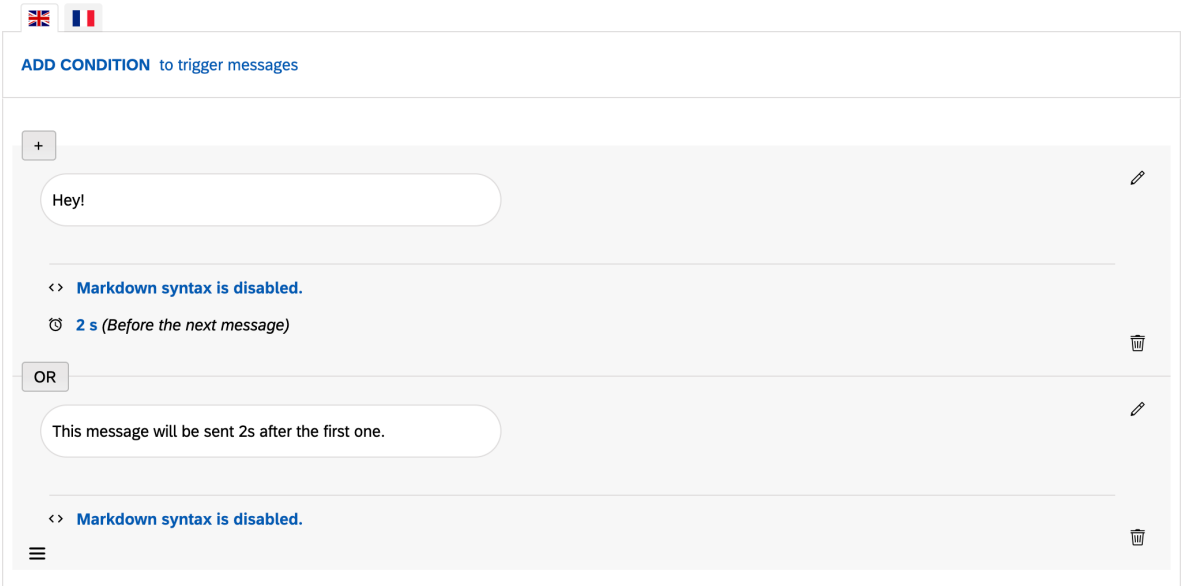
If you've connected your bot to a channel that doesn't support bold or italics, the formatting will be removed and replaced with single quotes (') instead of italics, and double quotes (") instead of bold, so that the formatted words are still given special attention. For example, "Tell me what you want, what you **really, really** want" will be rendered as "Tell me what you want, what you "really, really" want". If a channel doesn't support hyperlinks, the hyperlink will be replaced with "text (URL)", for example, "SAP Conversational AI (https://cai.tools.sap)".

If you use markdown without selecting the *Enable Markdown syntax* checkbox, the characters that you entered will be passed to the channel exactly as you entered them.

Message Delay

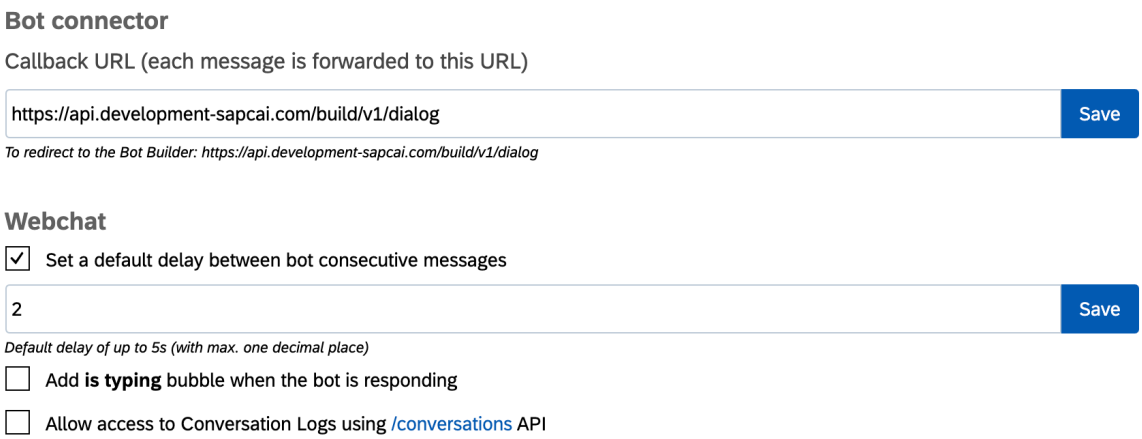
You can add a delay of up to 5 seconds between each message in a group of messages.

The main reason for adding a delay is so that users have enough time to read the message before your bot sends the next one. In a chat interface, this is especially important because each new message moves the previous message up. Also, dropping several messages with no delay can feel a little robotic. Another important reason for adding a delay is to give your bot personality. For example, you might want to add a short delay to make it look as though your bot is thinking. But be careful not to make your delays too long, so that users aren't waiting unnecessarily.



In the screenshot above, the second message will be sent to the user 2 seconds after the first message.

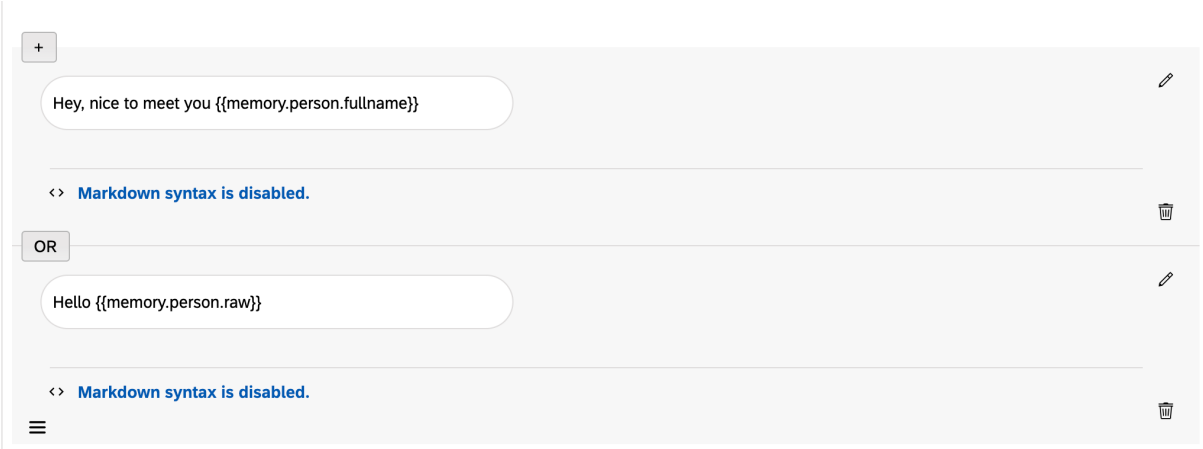
In your bot settings, you can also configure a default delay that is used if you don't set a specific delay.



If you don't set any delay, the messages are sent consecutively as usual.

Variables

You can dynamically inject the content gathered from the conversation in the bot replies by using double brace syntax. For example, if your bot asks for the user's name as a requirement, the name is added to the bot's memory once the requirement is completed. You can then create a text message (or any other message actually) filled with "Hello {{memory.username.raw}}", where {{memory.username.raw}} is replaced with the actual username. For more information, see Scripting with Variables [page 157].



Context Management

For users to meaningfully converse with your bot using natural language, your bot can recognize pronouns (like *it* or *that*) and map them to entities previously mentioned in the conversation. Similarly, if a user uses a superlative like *cheapest* or *most expensive*, or an ordinal like *first* or *second*, your bot can map the superlative or ordinal to an item in the message. For more information, and in particular, how to enable this in your bot, see References Between Entities [page 72].

How to Send Rich Messages from Your Code

For a list of the rich messages supported and their format, see Send Rich Messages [page 215].

Sometimes, you want to interact with a database or external API before sending a reply to the user. To achieve that, you can create a *CALL WEBHOOK* action to interact with your own code, implement your own logic, and send back the responses built from the data you've gathered. Here's a JS snippet as an example. It assumes that you have a *CALL WEBHOOK* action calling your `/do_some_stuff` route.

Sample Code

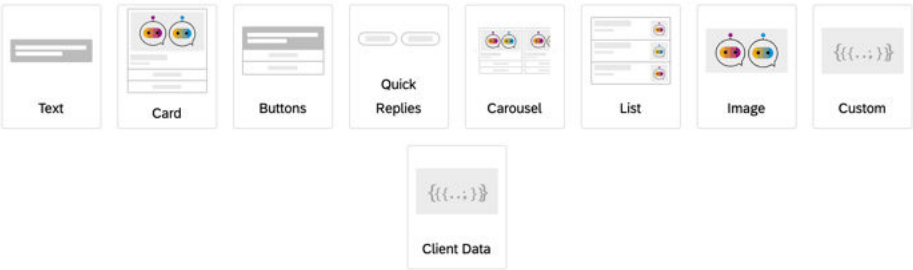
```
const express = require('express')
const bodyParser = require('body-parser')
const app = express()
const port = 5000
app.use(bodyParser.json())
app.post('/do_some_stuff', (req, res) => {
  // Do your actual logic here
  res.send({
    replies: [{
      type: 'text',
      content: 'Roger that',
    }],
  })
})
app.post('/do_more_complicated_stuff', (req, res) => {
  // Do your actual logic here
  res.send({
    replies: [{
      type: 'text',
      content: 'Roger that',
    }, {
      type: 'picture',
```

```
      content: 'The URL of my great GIF that I want to share with the world',
    }, {
      type: 'quickReplies',
      content: {
        title: 'My quick reply title',
        buttons: [{
          title: 'Choice 1',
          value: 'choice 1',
        }, {
          title: 'Choice 2',
          value: 'Choice 2'
        }],
      },
    }],
  })
})
app.listen(port, () => {
  console.log('Server is running on port 5000')
})
```

Related Information

Connect to External Service [page 136]
Development [page 195]
Messages [page 114]

4.8.1 Message Types



Text

1. Type the text message that should be your bot's response to the user. The message should have no more than 640 characters.
2. Enable Markdown [page 116] syntax if you wish to format your text.
3. You can add a delay of up to 5 seconds between each message in a group of messages.

Card

1. Add an image URL and provide the title and subtitle.
2. Add an interactive button to:
   1. *Link*
      Open a link

      > → Remember
      >
      > Ensure that you pre-pend *http* or *https* to the url that you provide.

   2. *Postback*
      Add a postback to send back to the /dialog API or a URI that is to be opened.
   3. *Phone Number*
      Provide a phone number. Tapping or clicking this number triggers a phone call using the default calling application. This feature is available for specific channels. For more information, see Send Rich Messages [page 215].
   4. *Trigger Skill*
      Enables the user to directly trigger the right action during a conversation, without having to resolve an utterance or going through the basic conversation flow.

      > i Note
      >
      > If the skill linked with the *Trigger Skill* button is deleted or deactivated, the button is not visible during the conversation.
      >
      > Typing an utterance does not trigger the skill. The user needs to click the button to trigger it.
      >
      > The buttons of type trigger skill are only valid until the next interaction. Once you click the button or type an utterance, the corresponding skill can't be triggered anymore at a later point in time, since the context in which it was created might not be valid anymore.

      > i Note
      >
      > This feature is currently supported by SAP Conversational AI Web Client, Webchat and MS Teams only.

3. You can add a delay of up to 5 seconds between each message in a group of messages.

Buttons

1. Type your bot's message asking the user to choose an appropriate option.
2. Add an interactive button to:
   1. *Link*
      Open a link

      > → Remember
      >
      > Ensure that you pre-pend *http* or *https* to the url that you provide.

2. *Postback*
   Add a postback to send back to the /dialog API or a URI that is to be opened.
3. *Phone Number*
   Provide a phone number. Tapping or clicking this number triggers a phone call is triggered using the default calling application. This feature is available for specific channels. For more information, see Send Rich Messages [page 215].
4. *Trigger Skill*
   Enables the user to directly trigger the right action during a conversation, without having to resolve an utterance or going through the basic conversation flow.

   > **i Note**
   >
   > If the skill linked with the *Trigger Skill* button is deleted or deactivated, the button is not visible during the conversation.
   >
   > Typing an utterance does not trigger the skill. The user needs to click on the button to trigger it.
   >
   > The buttons of type trigger skill are only valid until the next interaction. Once you click the button or type an utterance, the corresponding skill can't be triggered anymore at a later point in time, since the context in which it was created might not be valid anymore.

   > **i Note**
   >
   > This feature is currently supported by SAP Conversational AI Web Client, Webchat and MS Teams only.

3. You can add a delay of up to 5 seconds between each message in a group of messages.

Quick replies

Provide the following information:

1. The bot's message asking the user to choose an appropriate option.
2. Quick reply text that will be displayed to the user. You can add up to 12 quick replies.
3. Quick reply value that is sent back to the webhook for further action.

Carousel

This is a list of basic cards aligned vertically. Follow the steps [page 121] mentioned for creating a card.

List

Provide the following information:

1. An optional list header, which consists of title*, subtitle*, and an image URL*.

2. For each list item, a title, subtitle, description* and an image URL. Only the title is mandatory.
3. A button on each list item for the users to make a selection. Follow the steps for creating a button.
4. A button for the overall list, for example, to open the full list in your application or website*.
5. You can add a delay of up to 5 seconds between each message in a group of messages.

> **Note**
>
> Attribute with an asterisk (*) is a part of the enhanced message format and is currently supported in SAP Conversational AI Web Client only.

Image

Provide a URL of an image that should be displayed to the user. You can also add a delay of up to 5 seconds between each message in a group of messages.

Client Data

The content of client data messages is not displayed directly on the screen. This message type allows you to define a JSON key-value pair that is returned to the web client. The `onMessage` function implementation can use the JSON content of the client data message type to perform any type of action.

> **Sample Code**

```
{
    "anyKey": "Any JSON content"
}
```

Custom

With this message type, you can use unified scripting syntax in the existing message types to consume your bot memory and API service response directly into the platform and create responses dynamically. You can also define a *Custom* UI protocol that is supported by specific channels connected to the bot or any of the supported message protocols.

> **Note**
>
> All the existing message types like *Text*, *Card*, *Buttons* and so on, defined using the custom scripting option are fully supported by the SAP Conversational AI Web Client. However, the rendering may differ across both channels. For example, although both support scripting for buttons, Webchat renders 3 buttons while Web Client renders 11.
>
> To design messages specifically for non-SAP channels like Slack, Facebook Messenger and so on, you can select the type *Custom* in the dropdown list (within the Custom tile).

Message Type	Response Script	Example Response Script
Text	Sample Code `{ "type": "text", "content": "<Sample Text>", "markdown": "<true/false>", "delay": "<DELAY_IN_SEC>" }`	Sample Code `{"type":"text","content":"{{#compare memory.color.raw '===' 'Red'}} We have the following options. {{else}}Here is what we have for you. Take your pick. {{/compare}}", "markdown":false, "delay":null}`

Message Type	Response Script	Example Response Script
Card	**Note** These elements are rendered only if supported by the channel to which your bot is connected to. SAP Conversational AI Web Client is the only channel that currently supports all the elements. **Sample Code** `{ "type": "card", "delay": "<DELAY_IN_SEC>", "content": { "title": "<CARD_TITLE>", "subtitle": "<CARD_SUBTITLE>", "description": "<CARD_DESCRIPTION>", "imageUrl": "<LINK_VALUE>", "status": "<CARD_STATUS>", "statusState": "<''/none/information/error/success/warning>", "sections": [{ "title": "<SECTION_TITLE>", "attributes": [{ "label": "<LABEL_TEXT>", "type": "<link/text>", "value": "<DISPLAY_TEXT>" }] }], "buttons": [{ "title": "<BUTTON_TITLE>", "value": "<BUTTON_VALUE>", "type": "<postback/web_url/phonenumber/trigger_skill>" }]`	**Sample Code** `{ "type": "card", "delay": "10", "content": { "title": "{{api_service_response.default.body.value[0].Name}}", "subtitle": "${{api_service_response.default.body.value[0].Price}}", "description": "Discount_food", "imageUrl": "", "status": "", "statusState": "", "sections": [{ "title": "phonenumber", "attributes": [{ "label": "Store", "type": "phonenumber", "value": "01234567" }] }], "buttons": [{ "title": "Buy", "value": "Buy", "type": "postback" }] } }`

Message Type	Response Script	Example Response Script

	``
	  }
	}
	``	

Buttons	Sample Code
	``
	{
	  "type": "buttons",
	  "delay":
	"<DELAY_IN_SEC>",
	  "content": {
	  "title":
	"<Sample Text>",
	  "buttons": [
	  {
	  "title":
	"<BUTTON_TITLE>",
	  "value":
	"<BUTTON_VALUE>",
	  "type":
	"<postback/web_url/
	phonenumber/
	trigger_skill>"
	  }
	  ]
	  }
	}
	``
		Sample Code
		``
		{
		  "type": "buttons",
		  "delay": "",
		  "content": {
		  "title": "Here
		is what we have for
		you. Take your
		pick.",
		  "buttons": [
		  {{#eachJoin
		api_service_response.
		default.body.value}}
		  {
		  "title":
		"{{Name}} at $
		{{Price}}",
		  "value":
		"{{Rating}}",
		  "type":
		"postback"
		  }{{/eachJoin}}
		  ]
		  }
		}
		``

Quick Replies	Sample Code
	``
	{
	  "type":
	"quickReplies",
	  "delay":
	"<DELAY_IN_SEC>",
	  "markdown": "<true/
	false>",
	  "content": {
	  "title":
	"<Sample Text>",
	  "buttons": [
	  {
	  "title":
	"<BUTTON_TITLE>",
	  "value":
	"<BUTTON_VALUE>"
	  }
	  ]
	  }
	}
	``
		Sample Code
		``
		{
		  "type":
		"quickReplies",
		  "delay": "",
		  "markdown": false,
		  "content": {
		  "title": "Here is
		what we have for
		you!",
		  "buttons": [
		  {{#eachJoin
		api_service_response.
		default.body.value}}
		  {
		  "title":
		"{{Name}} at $
		{{Price}}",
		  "value":
		"Buy"
		  }{{/eachJoin}}
		  ]
		  }
		}
		``

Message Type	Response Script	Example Response Script
Carousel	Sample Code `"type": "carousel", "delay": "<DELAY_IN_SEC>", "content": [ { "title": "<CAROUSEL_TITLE>", "subtitle": "<CAROUSEL_SUBTITLE>", "imageUrl": "<LINK_URL>", "buttons": [ { "title": "<BUTTON_TITLE>", "value": "<BUTTON_VALUE>", "type": "<postback/web_url/phonenumber/trigger_skill>" } ] } ] }`	Sample Code `{ "type": "carousel", "delay": "", "content": [{{#eachJoin api_service_response.default.body.value}} { "title": "{{Name}}", "subtitle": "$ {{Price}}", "description": "{{Description}}", "imageUrl": "", "status": "Available", "statusState": "success", "sections": [{ "title": "Item Details", "attributes": [{ "label": "ReleaseDate", "type": "text", "value": "{{ReleaseDate}}" }, { "label": "Rating", "type": "text", "value": "{{Rating}}" }, { "label": "Price", "type": "text", "value": "$ {{Price}}" }] }], "buttons": [{ "title": "Buy", "value": "Buy", "type": "postback" }`

Message Type	Response Script	Example Response Script

		``
		]
		}{{/eachJoin}}
		]
		}
		``

Message Type	Response Script	Example Response Script
List	**Note** These elements are rendered only if supported by the channel to which your bot is connected to. SAP Conversational AI Web Client is the only channel that currently supports all the elements. **Sample Code** `{ "type": "list", "delay": "<DELAY_IN_SEC>", "content": { "title": "<LIST_TITLE>", "subtitle": "<LIST_SUBTITLE>", "imageUrl": "<LINK_VALUE>", "total": "<NUMBER_OF_ELEMENT_IN_LIST>", "upperBoundText": "<LIST_ENDING_INDICATOR_TEXT>", "buttons": [{ "title": "<BUTTON_TITLE>", "value": "<BUTTON_VALUE>", "type": "<postback/web_url/phonenumber/trigger_skill>" }], "elements": [{ "title": "<CARD_TITLE>", "subtitle": "<CARD_SUBTITLE>", "imageUrl": "<LINK_VALUE>", "status": "<CARD_STATUS>", "statusState": "<''/none/information/error/success/warning>", "description": "<CARD_DESCRIPTION>",`	**Sample Code** `{ "type": "list", "delay": "", "content": { "title": "Items", "subtitle": "With Price", "imageUrl": "", "total": "30", "upperBoundText": "Too many options to display", "buttons": [{{#eachJoin api_service_response.default.body.value}} { "title": "{{Name}} at ${{Price}}", "value": "{{Rating}}", "type": "postback" } {{/eachJoin}}], "elements": [{{#eachJoin api_service_response.default.body.value}} { "title": "{{Name}}", "subtitle": "{{Price}}", "imageUrl": "", "status": "available", "statusState": "success", "description": "{{Description}}", "sections": [{ "title": "Item Details", "attributes": [{ "label": "ReleaseDate", "type": "text",`

Message Type	Response Script	Example Response Script

	``
	"buttons": [
	{
	"title": "<BUTTON_TITLE>",
	"value": "<BUTTON_VALUE>",
	"type": "<postback/web_url/phonenumber/trigger_skill>"
	}
	]
	}
	]
	}
	}
	``
		``
		"value": "{{ReleaseDate}}"
		},
		{
		"label": "Rating",
		"type": "text",
		"value": "{{Rating}}"
		},
		{
		"label": "Price",
		"type": "text",
		"value": "${{Price}}"
		}
		]
		}],
		"buttons": [
		{
		"title": "Buy",
		"value": "Buy",
		"type": "postback"
		}
		]
		}
		{{/eachJoin}}
		]
		}
		}
		``

Image	Sample Code
	``
	{
	"type": "picture",
	"delay": "<DELAY_TIME>",
	"content": "<LINK_VALUE>"
	}
	``
		Sample Code
		``
		{
		"type": "picture",
		"delay": "",
		"content": "https://sample url"
		}
		``

Message Type	Response Script	Example Response Script
Custom	**Sample Code** `{ "type": "custom" }`	**Note** The following sample code is relevant for Facebook Messenger channel. **Sample Code** `{ "type": "custom", "content": { "attachment": { "type": "template", "payload": { "template_type": "airline_update", "intro_message": "Your flight is delayed", "update_type": "delay", "locale": "en_US", "pnr_number": "CF23G2", "update_flight_info": { "flight_number": "KL123", "departure_airport": { "airport_code": "SFO", "city": "San Francisco", "terminal": "T4", "gate": "G8" }, "arrival_airport": { "airport_code": "AMS", "city": "Amsterdam", "terminal": "T4", "gate": "G8"`

Message Type	Response Script	Example Response Script

		``
		},
		"flight_schedule": {
		"boarding_time": "2020-02-26T10:30",
		"departure_time": "2020-02-26T11:30",
		"arrival_time": "2020-02-27T07:30"
		}
		}
		}
		}
		}
		}
		}

4.9 Conversation State

The conversation state is the state of your bot's conversation with the user. It contains the following information:

Parameter	Description
`id`	Conversation ID provided by the Bot Connector. It is unique for each user.
`language`	Current language of the conversation, as detected by SAP Conversational AI.
`memory`	The data your bot has already collected from the user.
`skill_stack`	The last skill in the `skill_stack` has priority of execution and is popped after its actions have been executed.
`skill`	The currently active skill in the conversation.
`skill_occurences`	The number of consecutive messages handled by the current skill. It is set to 0 when the skill is done (actions are executed).
`participant_data`	An object containing the user information gathered from the channel that your bot is connected to (like Facebook Messenger, Slack, and so on). For more information, see section *User Information* in Messaging Channels [page 181].

Parameter	Description
client_info	Client-specific information which can be optionally passed by the client

Sample Code

```
{
    "id": "A_CONVERSATION_ID",
    "language": "en",
    "memory": {
      "person": {
        "fullname": "Francois",
        "raw": "Francois",
        "confidence": 0.95
      }
    },
    "skill_stack": ["get-weather"],
    "skill": "small-talk",
    "skill_occurences": 1
}
```

4.10 User Context

Your bot can access data that is associated with a user, such as user profile. This helps to enrich your bot's conversation and provide a more personalized conversational experience. If you know that your user is logged in to your website, this user context concept lets you use a token, cookie session, or user ID to authenticate the webhook call in SAP Conversational AI.

This user context concept is only available if you're using the Bot Builder API directly (not through the Bot Connector): for example, through your dedicated webchat in your website.

Note

To interact with the channel, you need to build your own custom user interface (not platform channels) or use `/dialog` endpoint directly to pass user identifier tokens or user context (like `sessionID`, `username` and so on) to the webhook servers. Webhook servers can then recognize the user identifier tokens and perform actions.

You can add these unique user identifiers by setting specific keys in the memory field when you initiate a new conversation during the first request made to the `/dialog` endpoint. This field will contain a user-defined value (for example, an authentication token or user ID of the client-side database).

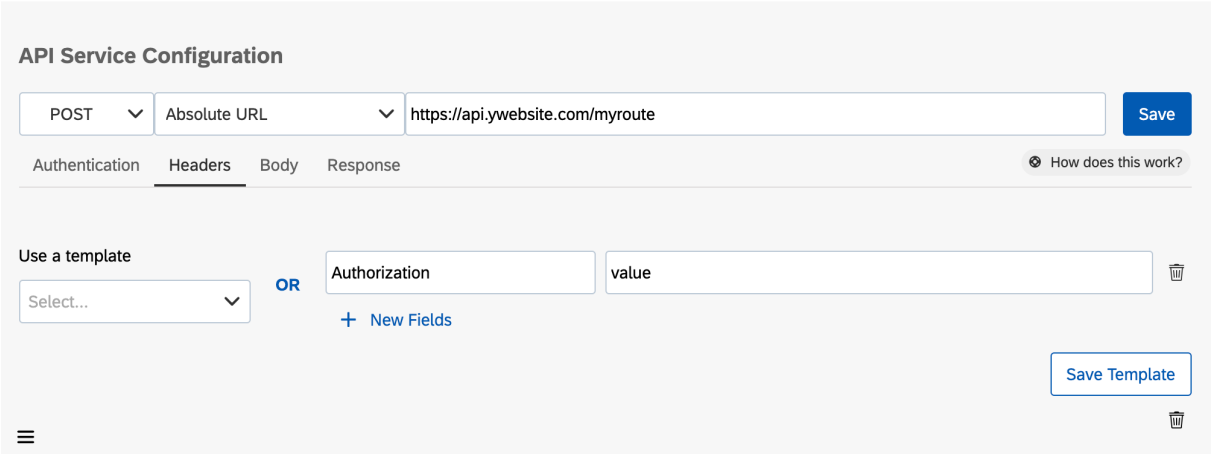
To understand how to format your API request, see /Dialog (Text) in the API Reference.

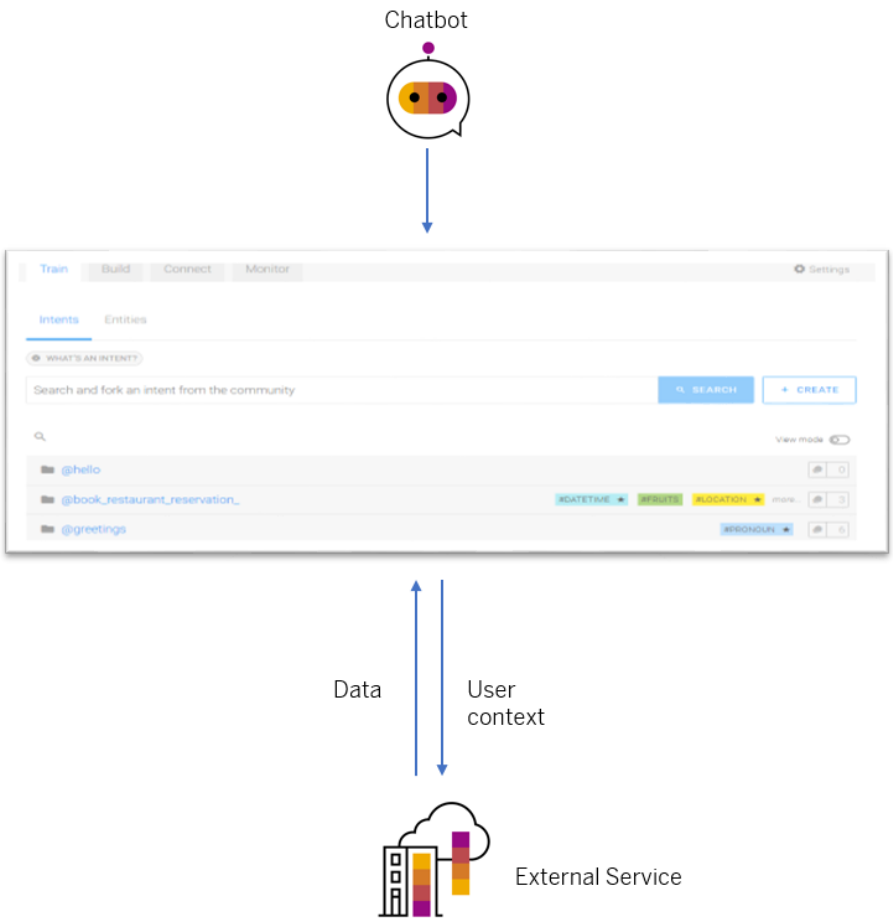
Below is an example of a POST `/dialog` request body with `memory` field. In this example, the token for the user is `myAwesomeUniqueToken`.

Sample Code

```
{
```

```
  "message": {
    "content":"Hello SAP",
    "type":"text"
  },
  "conversation_id": "CONVERSATION_ID",
  "memory": {
    "identifierToken": "myAwesomeUniqueToken"
  }
}
```





4.11 Single Sign-On with SAP Product Integration

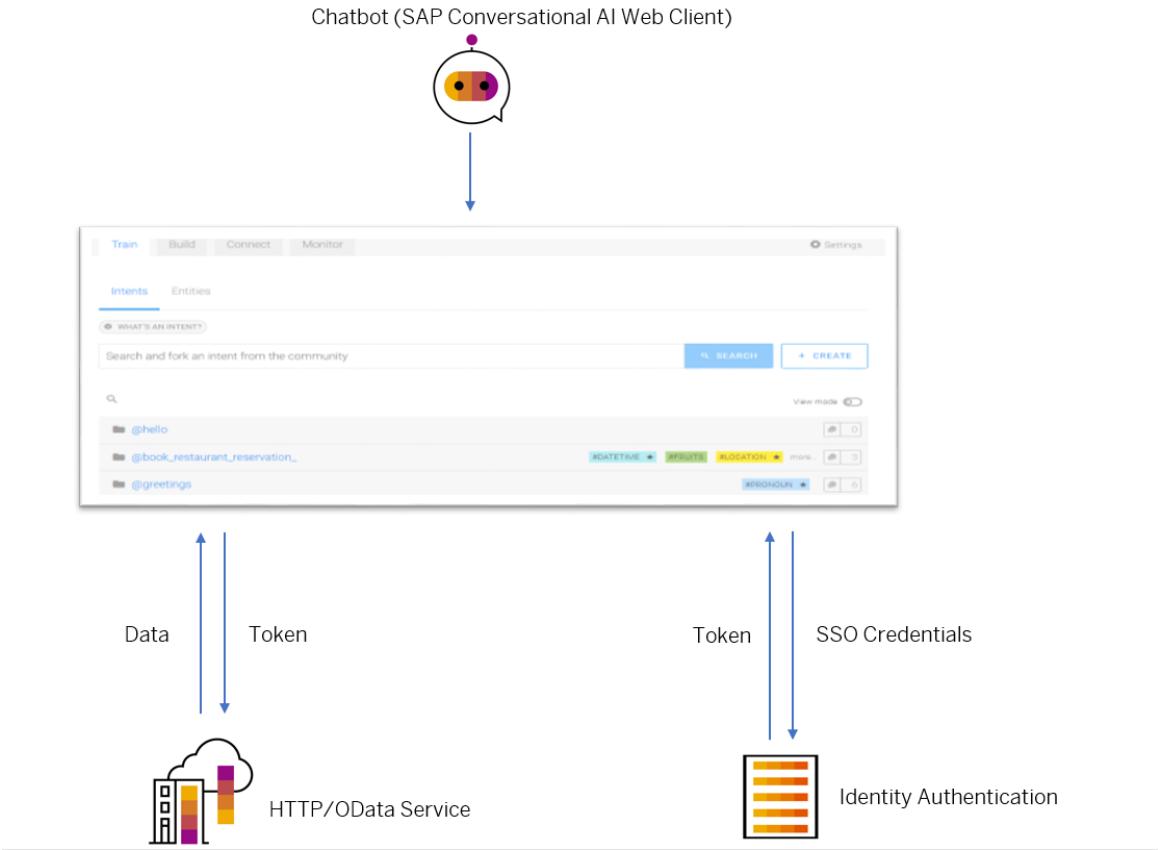
You want your bot to assist enterprise users by executing certain business operations on their behalf (for example, creating a leave request). For this, the bot will need to call an external service. The external service allows secure transmission of information by issuing a *user token* that uniquely identifies the user on that external service (JSON Web token or JWT). The user needs to log on to acquire this token.

Single Sign-On (SSO) permits users to use a set of login credentials to access the SAP Conversational AI Web Client [page 185]. Once authenticated, the business user can interact with the chatbot without providing their credentials on each log on.

To configure user authentication or single sign-on, you need to do the following:

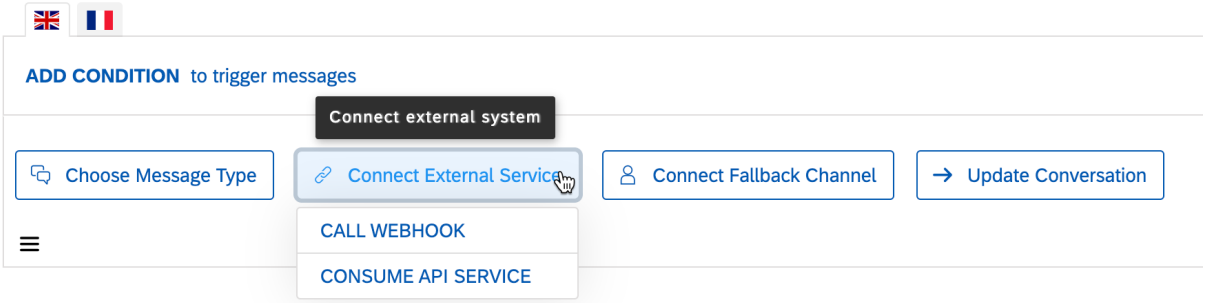
1. As an administrator, first create a SAP Business Technology Platform destination. For more information, see Managing Destinations.
2. Use this destination to configure the outbound call in the Actions tab of your bot in the SAP Conversation AI platform. For more information, see Connect to External Service [page 136].
   Single sign-on is enabled for your users.
   Once SSO is enabled, you can integrate the SAP Conversational AI Web Client into an on-premise SAP Fiori launchpad or with your web solution based in SAP Business Technology Platform. For more information, see the Configuration Guide.

> **i Note**
>
> For now, the SSO feature is only available for enterprise users of SAP products (like SAP S/4HANA, SAP SuccessFactors, and so on) to access the SAP Conversational AI Web Client .



4.12 Connect to External Service

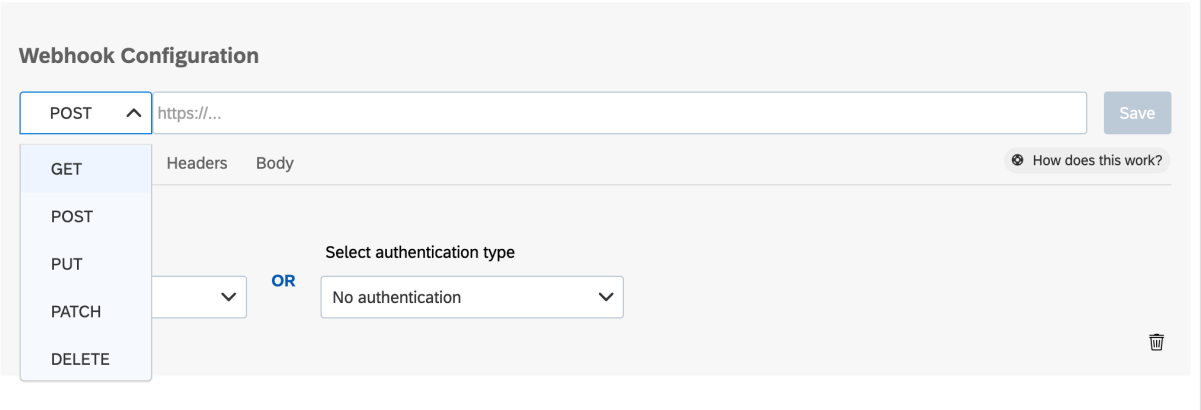
At many points in your conversation, you most likely want to retrieve business information or connect to an external system to perform actions. You can do this through *Connect External Service*. Either you can call a webhook that expects a JSON response in an appropriate format (see *Formatting the Response of the Webhook Call*), or you can consume any JSON response from an API service.



You can switch between *CALL WEBHOOK* or *CONSUME API SERVICE*.

Service	Description	When to use?
Call Webhook	A webhook is a simple HTTP call to your back end. To configure your HTTP call, click *CALL WEBHOOK* when defining requirements or actions in the Bot Builder. For more information, see Build Your Conversation [page 22] If you choose Call Webhook, the response should be in an SAP Conversational AI_specific JSON format that can be directly mapped to the user interface or will be used in the memory.	If you want to use middleware, you would call a webhook
Consume API Service	An API sends your request to the provider application and then delivers the response back to you. To configure an API request, click *CONSUME API SERVICE* when defining requirements or actions in the Bot Builder. The steps to configure the authentication, header, and body are same as the Webhook. If you choose Consume API Service, a variable called `{{api_service_response}}` will be created from a JSON object.	If you want to fetch some data (for example, a date) and operate on it, you could directly consume an API service.

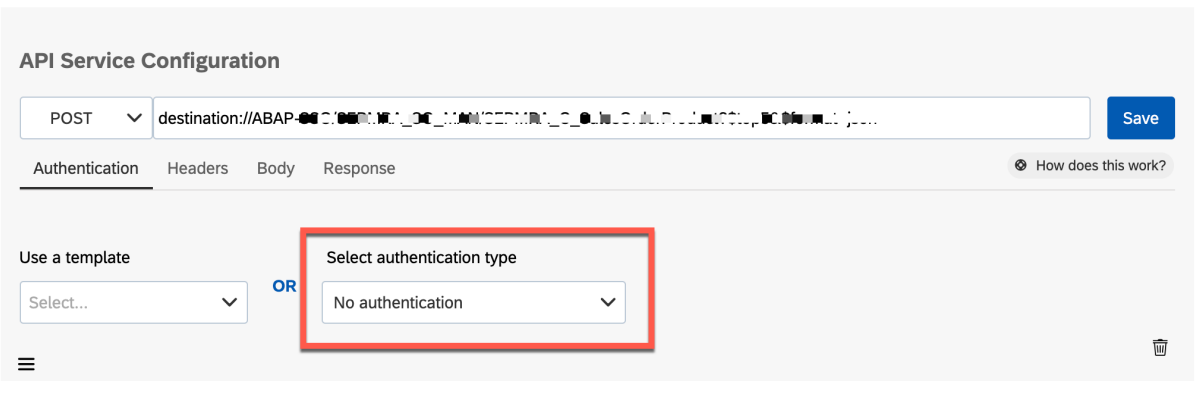
You can specify the HTTP method to use in your webhook call (GET, POST, PUT, PATCH, or DELETE).



You can provide the full URL (starting with 'https://' or a relative path or URL starting with '/') or use an SAP Business Technology Platform destination to be called by the Bot Builder. If you provide a relative URL starting with '/', then the bot webhook base URL (configurable in your bot's settings) is prepended to it. However, it will not be prepended if you use the full URL starting with 'https://'.

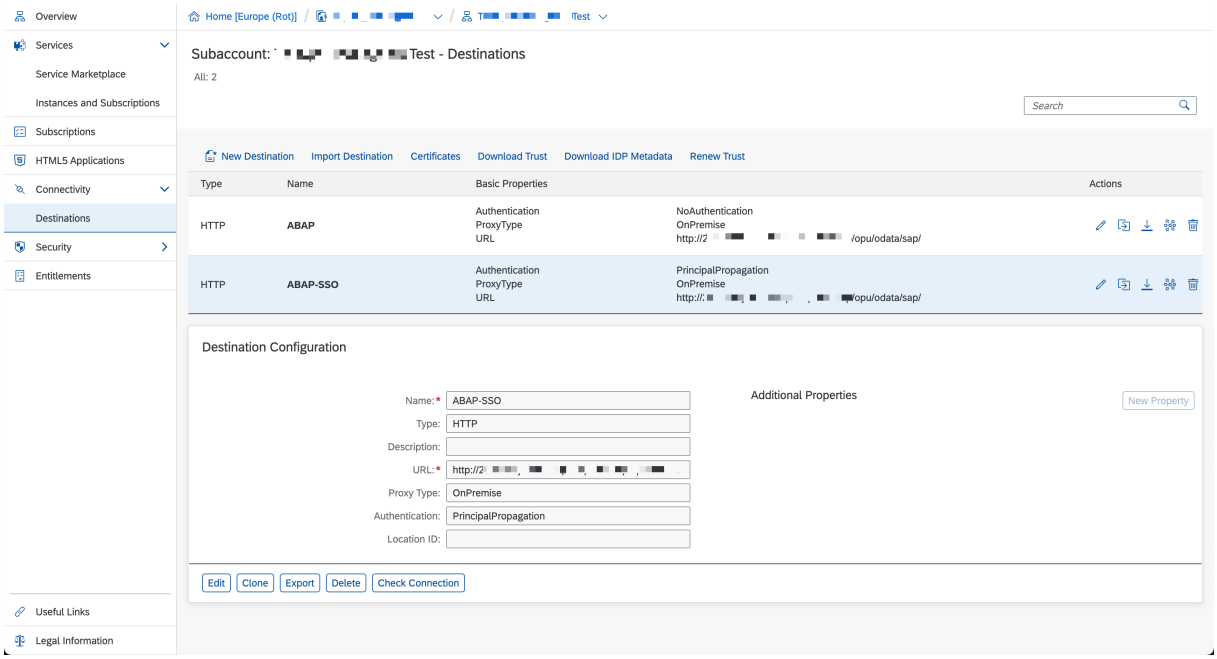
Destinations

If you have maintained SAP Business Technology Platform HTTP destinations, you must provide the exact name of your SAP BTP destination prefixed with `destination://` and select *No authentication*. This is applicable when you have not enabled system aliases.



> **i Note**
>
> You can configure a destination only if you are using the Enterprise edition of SAP Conversational AI. This is not supported in the Community edition. For more information, see Configuring the Enterprise Edition.

If you want to use SAP Business Technology Platform destinations and enable Single Sign-On, you need to integrate your bot with SAP Conversational AI Web Client. For more information, see Single Sign-On with SAP Product Integration [page 135].



Using Principal Propagation

After you have created a destination in your SAP Business Technology Platform to retrieve data from your on-premise system, all applications in your subaccount that uses the connectivity service can now access the destination. Therefore, it is recommended not to use destinations with hard coded username and password.

Instead use principal propagation to ensure that all calls made using the destination are in the name of the logged in user. This provides a secure way of forwarding the identity of a cloud user to an on-premise system. For more information, see Principal Propagation.

Destination for S/4HANA Business Data Access

This destination with principal propagation is required for processing chatbot requests of business users that access business data by calling OData services in the SAP S/4HANA system based on authorizations granted to the business user. The created destination could then be used in your bot to read data from your on-premise system. You can set the destination in the base URL in your bot *Settings* under *Versions*.

Using System Aliases

If you have enabled system aliases, choose the HTTP method and select the appropriate system alias. The URL defined for this system gets prepended to the service URL. If you want to define the complete URL (without any system alias), you must select *Absolute URL* option. In case of *Absolute URL*, the URL must start with 'https://', whereas when using a system alias, it must be a relative path starting with '/'. For more information, see System Alias Configuration [page 140].

You can define an SAP Business Technology Platform destination as a system alias. For more information, see System Alias Configuration [page 140].

Configuring the Service

The steps to configure the endpoint for both Webhooks and API service are the same, except that you can also configure the response for an API. To configure the endpoint, you need to do the following:

- Authentication configuration
- Header configuration
- Body configuration
- Response Configuration (only for API)

> **i Note**
>
> For enterprise scenarios we recommend to use *principal propagation* as authentication method instead of technical users. This will ensure that the identity of the end user interacting with the bot is used and the correct user is logged for all the requests. Consequently, the users will see only those data that they have access to.

> We recommend **not to** access via request token as the access is anonymous and no user is logged. If intercepted, these tokens can be used by end users for direct anonymous runtime access to bots. As a result, actions on the business API with the privileges of the bot creator or developer can be carried out.

Related Information

Authentication Configuration [page 142]
Header Configuration [page 144]
Body Configuration [page 144]
API Response Configuration [page 146]
Getting Response Using Webhook [page 152]

4.12.1 System Alias Configuration

System aliases are nick names representing connections to external systems which you can reuse while configuring webhooks and API service calls.

With system aliases you can define the types of external systems that your bot is using in a central place and maintain the details (like URL and authentication) separately per environment. In the enterprise edition, you can choose HTTP destinations maintained in SAP Business Technology Platform more easily.

For Existing Bots

If you enable system aliases in your current bot, your existing webhooks and API service configurations will be migrated. A system alias called *Base URL* is created and referenced wherever a relative URL is used. If basic authentication and OAuth are used, separate system aliases are created and referenced accordingly for each *Username* or *Client ID*. Authentication templates will be replaced by system aliases as well.

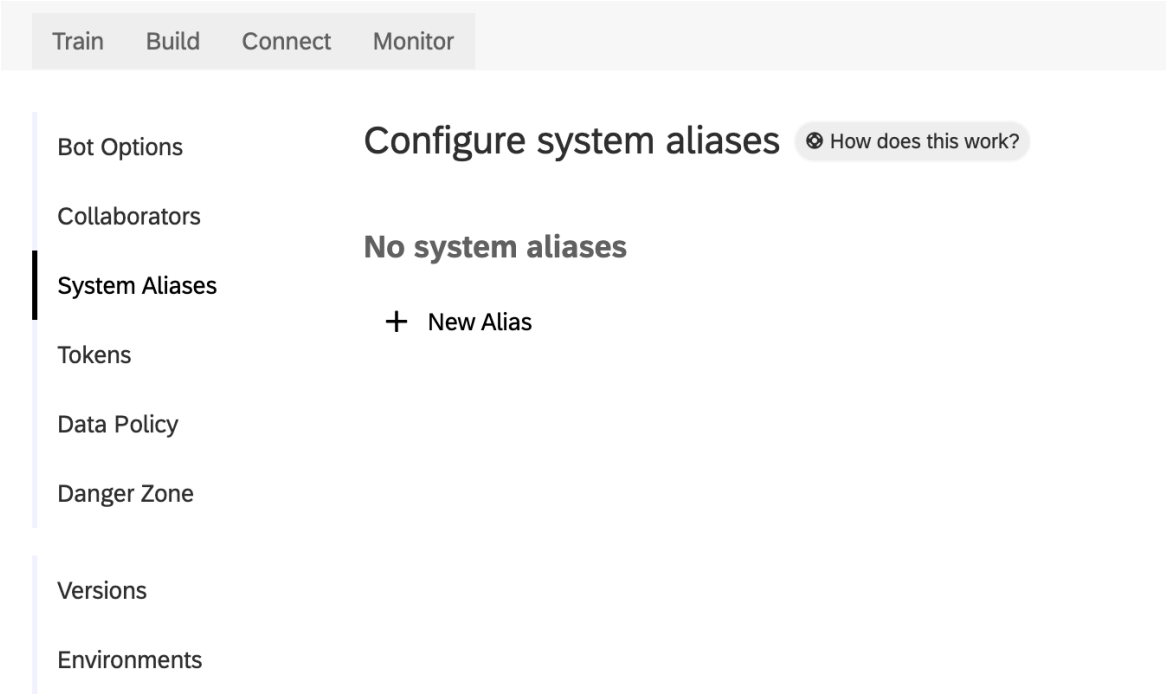
> ⚠ Caution
>
> Once you enable system aliases, this operation cannot to undone.

For enterprise tenants, a system alias is created for each destination used in the bot as well as for each authentication template when used with an absolute URL.

Whenever a system alias is created as part of this migration, the system configurations are also created for the environment, the respective bot version is deployed to.

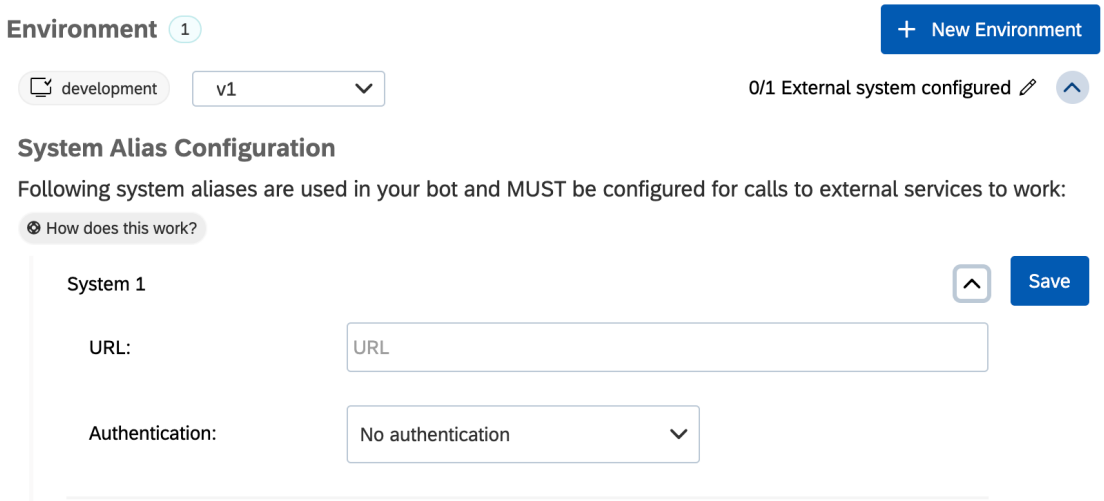
Configuring A System Alias

1. Go to your bot *Settings* and click the *System Aliases* tab. Provide a suitable name for the system alias. In case of multiple system aliases, ensure that you define distinct names for each.

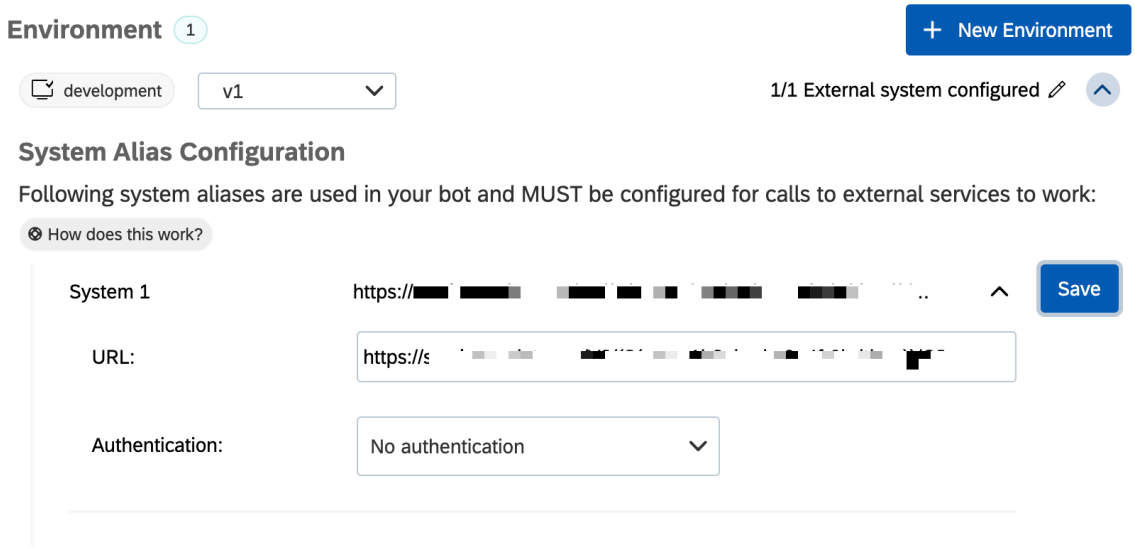


2. In the *Environments* tab, provide the URL and authentication details for the system. This enables you to maintain different systems in a productive and test environment.

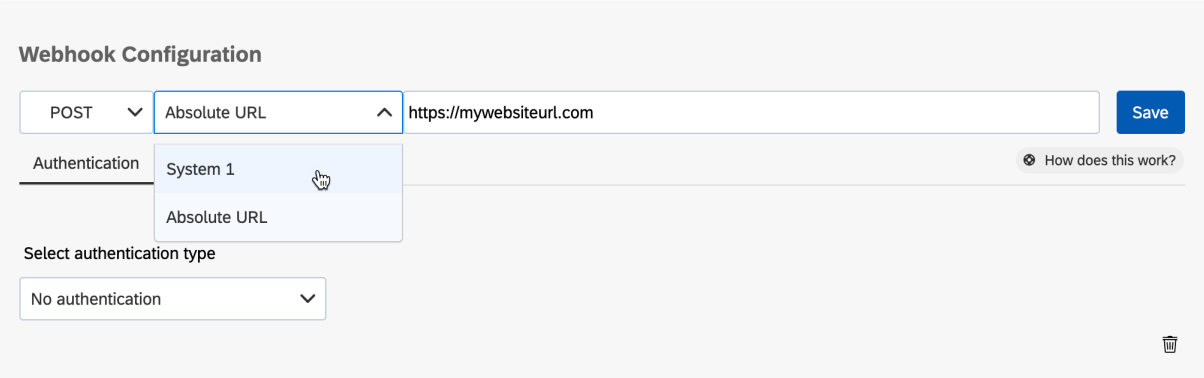
> **i Note**
>
> System aliases can only be used in combination with environments. So in case you are calling an API of SAP Conversational AI directly using the request token of a bot version, API service configurations or webhooks using a system alias will not work correctly. Please use the request token of an environment instead. Also *Chat Preview* will only work for bot versions that are assigned to an environment.



In your enterprise tenant, you can select existing HTTP destinations, which were configured in your SAP Business Technology Platform to be used as System Alias configuration. This will help you maintaining connections to the correct backend landscapes, depending on the environment the bot is running in. You need to maintain different destinations and the corresponding authorization type based on the use case. For more information, see *Destinations* under Connect to External Service [page 136].



When you configure an API service or a Webhook while defining your bot's *Actions* or fetching entity values and enrichments, you can choose the appropriate system alias that gets prepended to the service URL. If you want to define the complete URL (without any system alias), you must select *Absolute URL* option.



> **i Note**
>
> While forking a bot, only the names of the system aliases are forked and not the configurations. You need to maintain the configurations manually in the destination bot.

4.12.2 Authentication Configuration

You can define the authentication information for an API service or a Webhook to authorize your users.

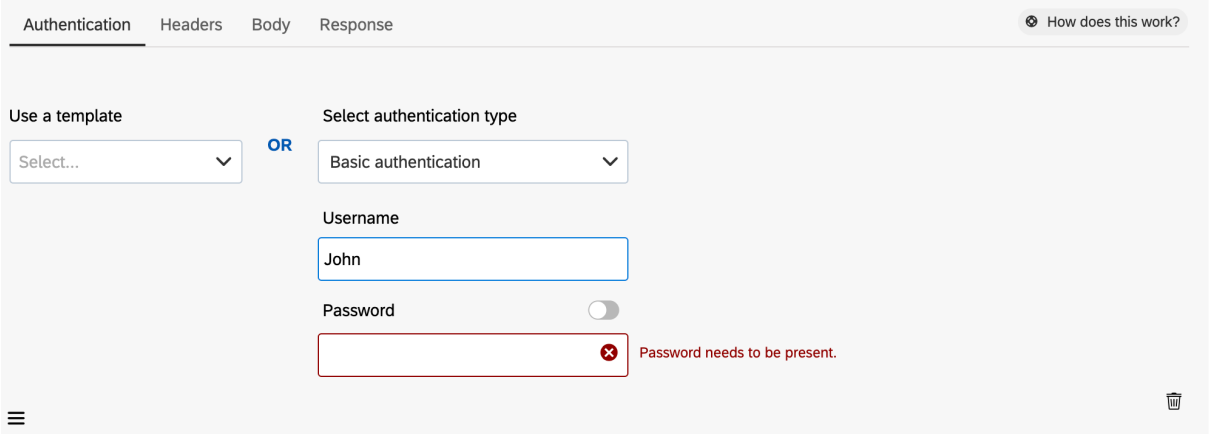
You have the following options:

- No authentication
  No authentication/authorization is passed with the request.

> **i Note**
>
> If you want to use SAP Business Technology Platform destinations, you need to select *No authentication* in the API service configuration and use the authentication that is maintained on the SCP destination directly.

- Basic authentication
  A username/password pair is passed with the request.
- OAuth 2 authentication
  A client ID, client secret, and authorization URL are passed with the request.

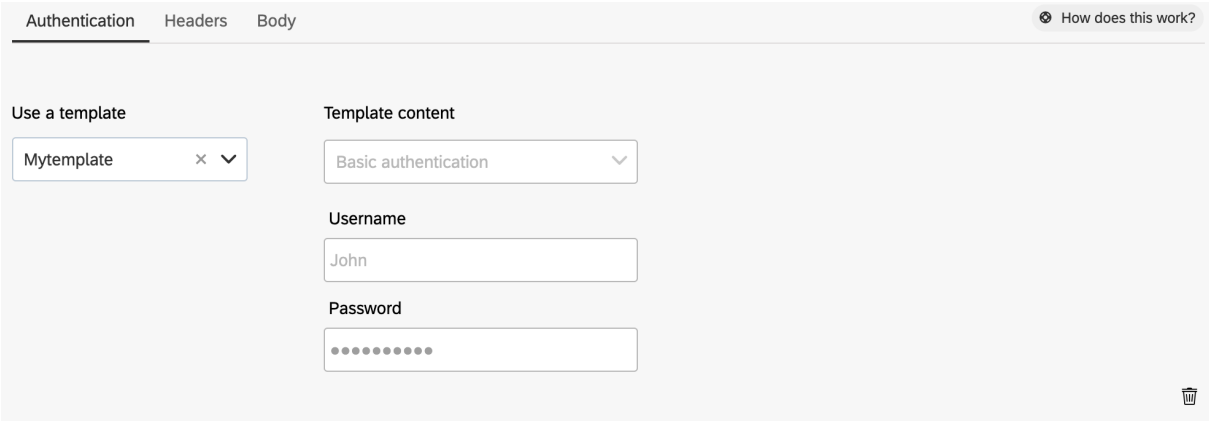
> **i Note**
>
> OAuth2 here refers to the **OAuth2 Client Credential Flow**.



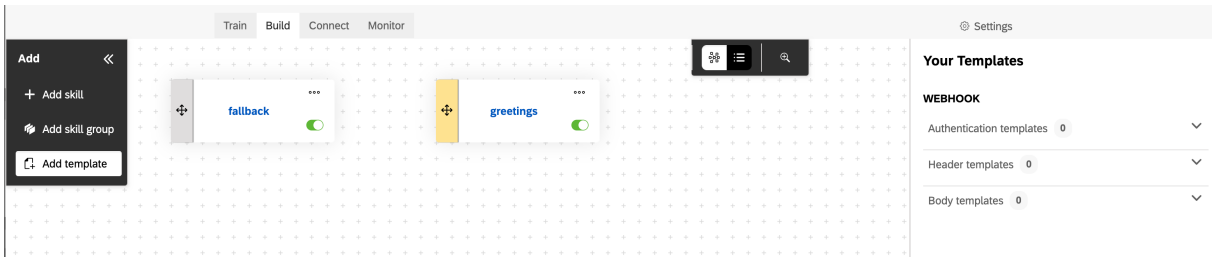
Templates

Templates enable you to reuse specific configurations of authorizations, headers, and bodies across skills.

> **i Note**
>
> If you have defined system aliases, the authentication configuration needs to be maintained separately for each system in your bot *Settings*, under the *Environments* tab. For more information, see System Alias Configuration [page 140].

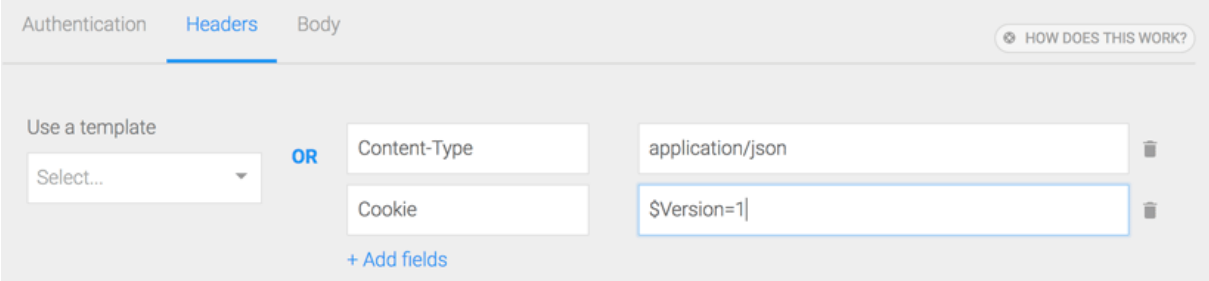


This requires you to first create the template in the skills view on the *Build* tab. You can also edit existing templates here. To create or edit a template, click *Templates* in the gray panel on the left.



4.12.3 Header Configuration

HTTP headers are accommodated by configuring a key-value pair, where you can name keys and set a value to be passed along in the header.



4.12.4 Body Configuration

The HTTP request body must be formatted as a standard JSON object. You can either receive the default Webhook body that we provide with all conversation states or create your own custom body.

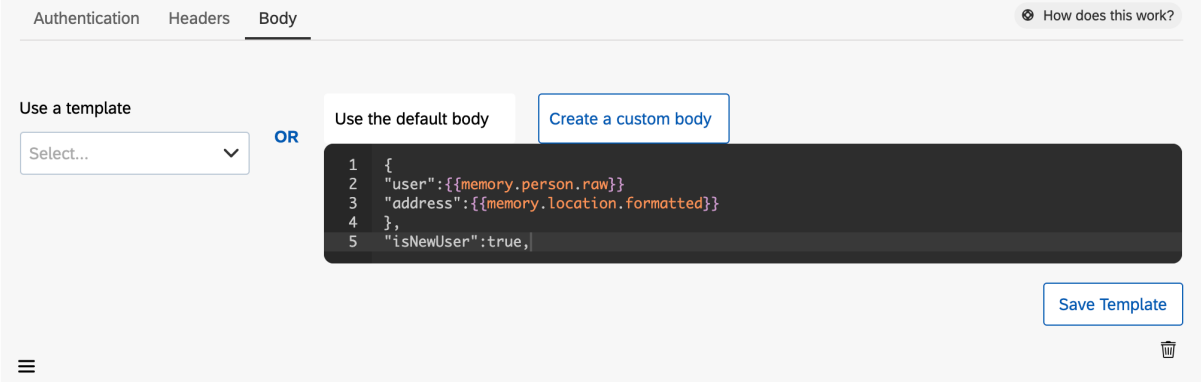
> **i Note**
>
> Default body is only relevant in a webhook and not in an API service.
>
> You can only create a custom body for your API service response. The default body format is as follows:

> **Sample Code**

```
{
  "conversation": {
    "id": "A_CONVERSATION_ID",
    "language": "en",
    "memory": {
      "person": {
        "fullname": "Francois",
        "raw": "Francois",
        "confidence": 0.95
      }
    },
```

```
      "skill_stack": ["get-weather"],
      "skill": "small-talk",
      "skill_occurences": 1
    },
    "nlp": {
      "source": "hi",
      "intents": [
        {
          "slug": "greetings",
          "confidence": 0.99
        }
      ],
      "sentiment": "vpositive",
      "entities": {},
      "act": "assert",
      "type": null,
      "version": "2.10.1",
      "processing_language": "en",
      "language": "en",
      "uuid": "96597974-3ee1-4743-8a5d-341b67effb9a"
      "status": 200,
      "timestamp": "2017-10-25T21:36:02.071243+00:00",
    }
  }
```


In custom HTTP request bodies, you can reference conversation variables (like memory variables, NLP information, and so on) in place of hard-coded values: for example, `{{memory.person.raw}}`.



! Restriction

You can only create custom body templates.

You can not delete a template if it is still in use.

i Note

When connecting to external services, you can use logical expressions and functions using the Handlebars template language in the URL, headers and body of webhooks and API service configurations. For more information, see Scripting Syntax [page 159].

While you configure the body of an API or webhook in the text editor, the syntax for the scripting functionality is highlighted.

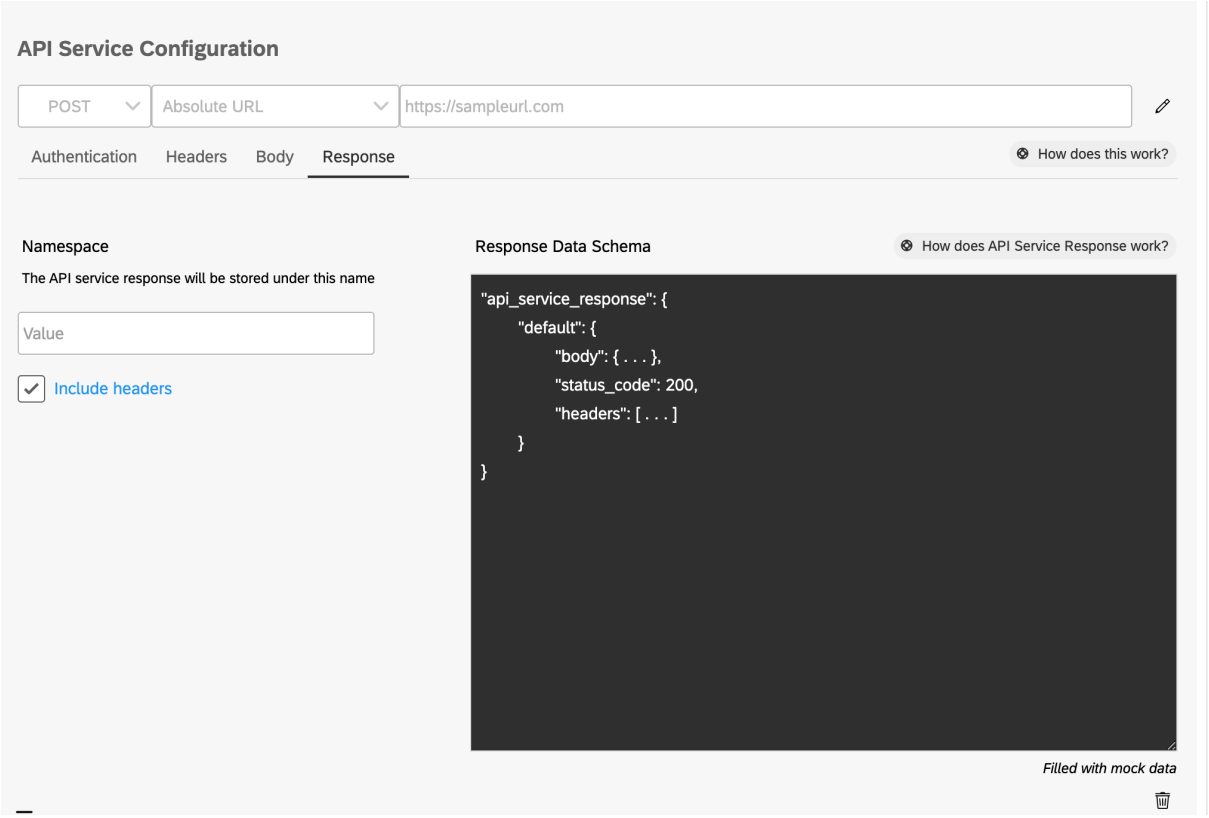
4.12.5 API Response Configuration

Configuring API Response

In addition to these configuration steps, you can also configure your response variable by giving a namespace in the *Response* tab.

If a namespace has already been provided, then the default namespace will be overwritten by the one that you provide. The JSON response of the service request will be published under this namespace.

You can use the result from `{{api_service_response}}` for another action, save it to the memory, or send it as a response using the *Choose Message Type* action. The variable `{{api_service_response}}` persists only when the skill is active during a conversation. You can persist it for longer by storing it to the memory using the *Update Conversation* action.



Example response

The following payload is returned by an OData service. In this example, the variable `{{api_service_response.products.body.d.Name}}` will get the value *Bread*, when the namespace is defined as `products`.

```
{
  "d": [
    {
      "__metadata": {
        "uri": "https://services.odata.org/V2/OData/OData.svc/Products(0)",
        "type": "ODataDemo.Product"
      },
      "ID": 0,
      "Name": "Bread",
      "Description": "Whole grain bread",
      "ReleaseDate": "/Date(694224000000)/",
      "DiscontinuedDate": null,
      "Rating": 4,
      "Price": "2.5",
[...]
```

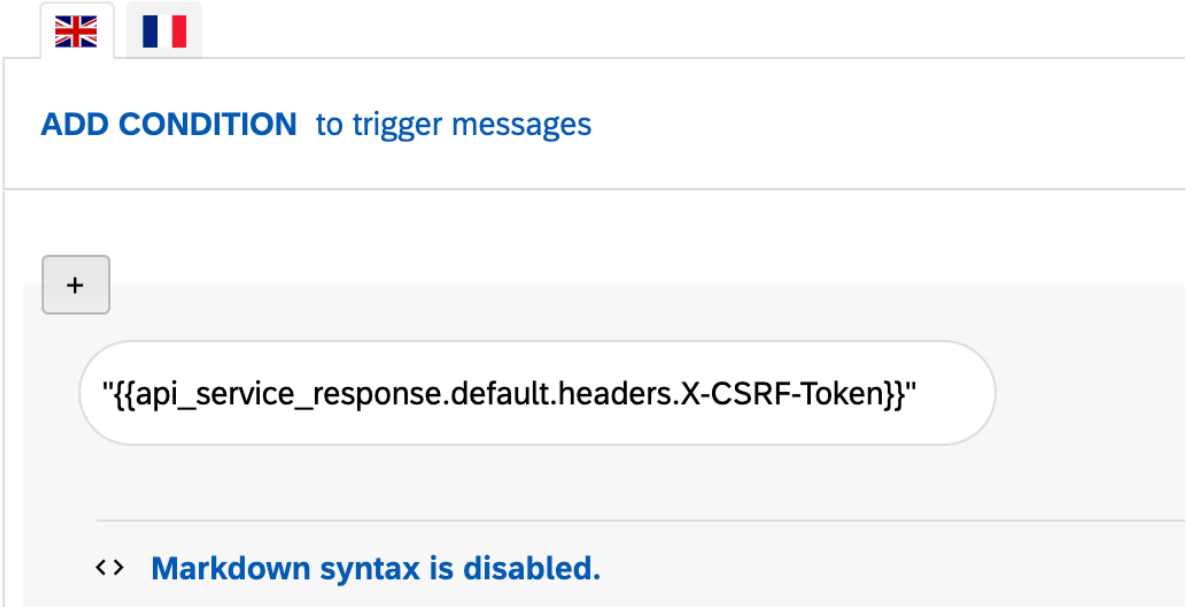

The following table explains the different parts of the variable path.

Variable	Defined by	Description
`api_service_response`	SAP Conversational AI	Variable for API service responses
`products`	Bot developer	Namespace defined in the API service configuration
`body`	SAP Conversational AI	The body of the API service call. The other option is `status_code` which returns the HTTP status code of the API call.
`d`	External service	Part of the JSON response received from the external server. Within the body of an OData response, the `v2` protocol defines that there is always a `d` node.
`Name`	External service	Part of the JSON response received from the external server and in this case, equals to the OData attribute defined in the OData service and it cannot be changed.

If needed, you can call several Consume API Service actions and enhance the `{{api_service_response}}` variable before you call a middleware with a webhook. You can aggregate the data fetched from all these actions and pass the result back.

Optionally, you can choose to include header fields in the API service Response by selecting *Include headers* check-box. This allows you to obtain header information like the cross-site request forgery (CSRF) token from a previous request that can be used in subsequent requests as an http header.

In cases where the external service returns multiple headers with the same header name, the values need to be accessed with their index, for example `{{api_service_response.default.headers.set-cookie.0}}` to get the first header value.



Configuring Response for Fetching Entity Values

If you have a large number of entity values, you can import them using a service API. For more information, see Importing Entity Values [page 36].

In addition to the above configurations, you can define and adjust the Response for the API to fetch entity values from an external system.

The JSON response of the service request will be published with the namespace: `api_service_response`.

You can use the result from `{{api_service_response}}` and adopt the result with scripting syntax.

> **i Note**
>
> The output needs to be a valid JSON array!

Example response

The following payload is returned by an OData service. In this example, the variable `{{api_service_response.body.value}}` will get an array of values like Bread, when the attribute is defined as `Name`. This is done by using the pluck-scripting syntax and wrapped as a JSON object.

Variable	Defined by	Description
`api_service_response`	SAP Conversational AI	Variable for API service responses
`body`	SAP Conversational AI	The body of the API service call. The other option is `status_code` which returns the HTTP status code of the API call.
`value`	External service	Part of the JSON response received from the external server. Within the body of an OData response, the v4 protocol defines that there is always a `d` node.
`Name`	External service	Part of the JSON response received from the external server and in this case, equals to the OData attribute defined in the OData service and it cannot be changed.

Sample Code

```
{{{json (pluck api_service_response.body.value "Name") }}}
```

Authentication | Headers | Response Customization | How does this work?

1 Fetch the data from your API service

Status: 200


```
{
    "api_service_response": {
        "body": {
            "odata.metadata": "https://services.odata
                .o[illegible]
                )) [illegible]",
            "value": [
                {
                    "ID": 0,
                    "Name": "Bread",
                    "Description": "Whole grain bread",
                    "ReleaseDate": "1992-01-01T00:00:00"
                        ,
                    "DiscontinuedDate": null,
                    "Rating": 4,
                    "Price": 2.5
                },
                {
                    "ID": 1,
                    "Name": "Milk",
                    "Description": "Low fat milk",
                    "ReleaseDate": "1995-10-01T00:00:00"
```


Your API service response is stored in the `api_service_response` key and must be an `array` of `string` values

E.g:

```
[
  "something",
  "something",
  "something again"
]
```

2 Transform your API service response to fit the format

How does this work?

Transform

```
1 {{api_service_response.body}}
```


```
{
    "odata.metadata": "https://services.odata.org/V3/(S
    [illegible]
    "value": [
        {
            "ID": 0,
            "Name": "Bread",
            "Description": "Whole grain bread",
            "ReleaseDate": "1992-01-01T00:00:00",
            "DiscontinuedDate": null,
```


Must be an array of string values

2 Select your import option

- ◯ **Replace**
  to replace with the fetched entity values
- ◯ **Merge**
  to add fetched entity values to the bot's existing entity values

4.12.6 Getting Response Using Webhook

Formatting the Response of the Webhook Call

The body format of your response should be a valid JSON and can contain two keys: `replies` and `conversation`.

Key	Required or Optional	Value
replies	Optional	Array of object
conversation	Optional	Object with a key `memory` and `language`
conversation.memory	Optional	Object filled as you want
conversation.language	Optional	String with a language ISO format

Sample Code

```
{
  "replies": [
    {
      "type": "text",
      "content": "Hello world!"
    }
  ],
  "conversation": {
    "language": "en",
    "memory": {
      "user": "Bob"
    }
  }
}
```

The conversation data that you send back will update the conversation state:

- `memory` will replace the actual memory of your bot (so be careful that you don't lose everything if you just want to change one of your memory keys to add all your other keys).
- `language` will update the language of the conversation. Each new sentence sent by the user will be processed in this language, and the bot will reply in this language.

`replies` are sent in the body of the result of the main Bot Builder and will appear in the messages key:

POST `https://api.cai.tools.sap/build/v1/dialog`

Sample Code

```
{
    "messages": [
        {
             "type": "text",
             "content": "Hello world!"
        }
    ],
    "conversation": {
        "id": "CONVERSATION_ID",
        "language": "en",
        "memory": {},
        "skill": "default",
        "skill_occurences": 1
    },
    "nlp": {
        "uuid": "b96bc782-6aba-4fac-aeaa-2326936b08bf",
        "source": "Hello SAP",
        "intents": [
            {
                "slug": "greetings",
                "confidence": 0.99
            }
        ],
        "act": "assert",
        "type": null,
        "sentiment": "neutral",
        "entities": {},
        "language": "en",
        "processing_language": "en",
        "version": "2.10.1",
        "timestamp": "2017-10-19T13:24:12.984856+00:00",
        "status": 200
    }
}
```

Formatting the Array of Replies

You can format objects in the array of reply as desired, depending on your needs when you request the Bot Builder API. If you are using the Bot Connector (that is, you have connected a channel on the SAP Conversational AI platform like Facebook Messenger, Slack, or a webchat), you need to follow the Bot Connector format. For more information, see Send Rich Messages [page 215].

Sample Code

```
{
```

```
  "replies": [
    {
      "type": "text",
      "content": "Hello world!"
    }
  ]
}
```

Running Your Basic API

When a Webhook action is triggered by a user input, SAP Conversational AI calls your API at the URL specified in the bot settings, at the endpoint specified in the action itself.

The body of the request contains various useful information, like the current skill detected and the result of the NLP analysis of the input.

Refer to the sample code in different programming languages to get started. Click on each tile to pick your coding language and follow the steps.

-
-
-
-

JavaScript (JS)

We recommend using version 6.1.0 of Node.js.

1. Create a directory `mkdir my-bot` and `cd my-bot && npm init`
2. Create your main file `touch index.js`
3. Install the dependencies `npm install express body-parser --save`
4. Open your `index.js` and copy and paste the sample code.
5. Run your code `node index.js`

Sample Code

```
const express = require('express')
const bodyParser = require('body-parser')
const app = express()
const port = 5000
```

```
app.use(bodyParser.json())
app.post('/', (req, res) => {
  console.log(req.body)
  res.send({
    replies: [{
      type: 'text',
      content: 'Roger that',
    }],
    conversation: {
      memory: { key: 'value' }
    }
  })
})
app.post('/errors', (req, res) => {
  console.log(req.body)
  res.send()
})
app.listen(port, () => {
  console.log('Server is running on port 5000')
})
```

PHP

We recommend using version 7.x of PHP.

1. Create a directory `my-bot` and `cd my-bot && touch index.php`
2. Install the dependencies `composer require slim/slim`
3. Copy and paste the sample code in the `index.php`
4. Run your server with `php -S localhost:5000 index.php`

Sample Code

```
<?php
require 'vendor/autoload.php';
use \Slim\App;

$app = new App();

$app->post('/', function ($request, $response) {
  error_log($request->getBody()->getContents());
  return $response->withJson(array(
    'replies' => [
      array('type' => 'text', 'content' => 'Roger that')
    ],
    'conversation' => array(
      'memory' => array('key' => 'value')
    )
  ));
});

$app->post('/errors', function ($request, $response) {
  error_log($request->getBody()->getContents());
  return $response;
});

$app->run();
```

Python

We recommend using version 2.7+ of Python.

1. Create a directory `mkdir my-bot` and `cd my-bot && touch bot.py`

2. Install the dependencies `pip install flask`
3. Open your `bot.py` and copy and paste the sample code.
4. Run your code `python bot.py`

Sample Code

```
from flask import Flask, request, jsonify
import json
app = Flask(__name__)
port = '5000'
@app.route('/', methods=['POST'])
def index():
  print(json.loads(request.get_data()))
  return jsonify(
    status=200,
    replies=[{
      'type': 'text',
      'content': 'Roger that',
    }],
    conversation={
      'memory': { 'key': 'value' }
    }
  )

@app.route('/errors', methods=['POST'])
def errors():
  print(json.loads(request.get_data()))
  return jsonify(status=200)

app.run(port=port)
```

Ruby

We recommend using version 2.3+ of Ruby.

1. Create a directory `mkdir my-bot` and `cd my-bot && touch bot.rb`
2. Install the dependencies `gem install sinatra`
3. Open your `bot.rb` and copy and paste the sample code.
4. Run your code `ruby bot.rb`

Sample Code

```
require 'sinatra'
require 'json'

set :port, 5000

before do
  @params = JSON.parse(request.body.read)
end

post '/' do
  content_type :json
  {
    replies: [{ type: 'text', content: 'Roger that' }],
    conversation: {
      memory: {
        key: 'value'
      }
    }
  }.to_json
end
```

```
post '/errors' do
  puts @params

  200
end
```

Exposing the Opened Port to Connect it to a Webhook

If you don't have a public server, or if you want to test your webhook during development, you need to expose the specific port of your local machine to the internet. For this you can use tools that expose the local servers to the internet.

Alternatively you can deploy your application to the SAP Business Technology Platform (Cloud Foundry) environment by registering for the SAP Business Technology Platform trial account. For a node.js application, follow the steps in the Create a Node.js Application tutorial, after deployment.

4.13 Scripting with Variables

Scripting adds more flexibility to manipulate runtime data [page 179] that is accessible in skills. This data is integrated using variables that are substituted by actual values at runtime during the conversation flow. As a bot developer you can build actions and messages using this data.

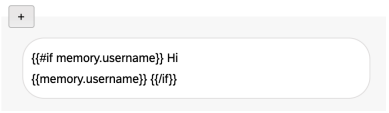
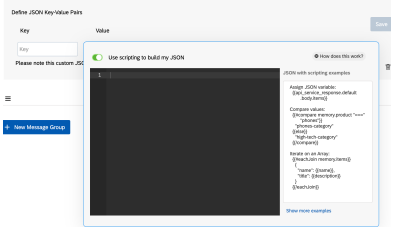
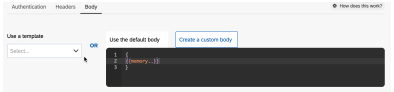
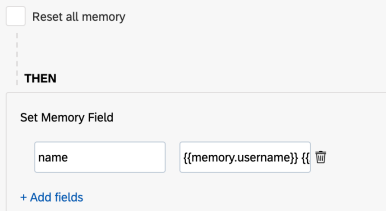
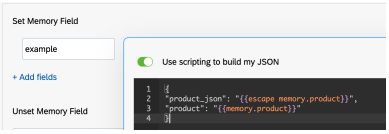
For example, if you configure a variable for the order status, then your bot can look up the back-end system and fetch the current status of the order for the customer at runtime.

You cannot store information in variables via scripting as they are always read-only. You can store information in the memory using Edit Memory actions which can then be accessed as variables.

> **i Note**
>
> Scripting is only supported in actions that support the `{{variable}}` syntax. For example, you can use scripting while defining the bot response as part of an action, but *not* while adding a condition.

Scripting syntax allows you to use logical functions and Natural Language Generation (NLG) helpers in scripts. Usage of NLG helpers can improve grammatical correctness of bot responses when using dynamic data such as variables. For more information see, NLG helpers [page 173].

The following bot actions support the scripting syntax:

Tab	Bot Actions	Example
BUILD	Send Message • Text • Client Data	 
	Connect External Service • Call Webhook (in URL, Body and Headers) • Consume API Service (in URL, Body and Headers)	 
	Update Conversation • Edit Memory (in memory keys and values) except the *Unset memory* field. You can also use scripting while assigning JSON objects to memory fields. • Go To (another skill)	 
TRAIN	Fetch Entity Values via Service API (in URL, Body, Headers and Response customization)	

Using Scripting in Edit Memory and Client Data Actions

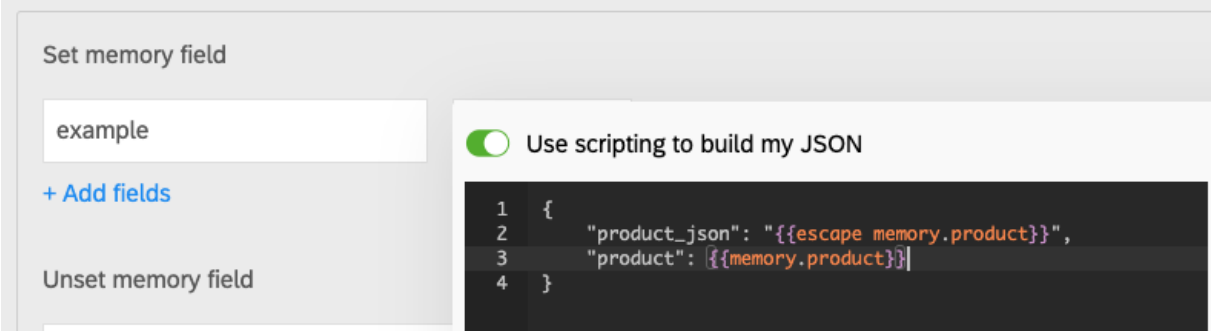
Besides using scripting and variables for replacing or formatting simple values, it can also be used to build up JSON dynamically, for example, for storing a JSON variable into the memory or for building up a JSON that is sent to the client dynamically with a loop. In such cases the combination of JSON syntax with the scripting syntax is not a valid JSON, although the result after the scripting syntax is evaluated might be a valid JSON.

In the *Edit Memory* and *Client Data* actions you need to explicitly enable the usage of scripting to build the JSON. This is necessary to disable the JSON validation and allow the scripting syntax also outside of the valid JSON.

Escaping when using scripting

When you use scripting or variables for building up JSON dynamically, it is important to make sure that any variables that might contain double quotes are escaped correctly. The escape helper should be used in cases where a variable that might contain double-quotes or JSON, is used inside a JSON string. For more information, see Working with Strings [page 164].

When using the following example in an Edit Memory action, the JSON key `product_json` will contain the product object as JSON string, whereas the `product` key will contain the object itself and allow accessing its elements, for example `{{memory.example.product.raw}}`.



4.13.1 Scripting Syntax

For more flexibility while defining actions for your skills, you can use logical expressions and functions using the Handlebars template language. With Handlebars, you can use pre-defined helpers to access and execute operations on the conversation context and to format the values as per your requirement.

Block helpers (starting with #) allow you to wrap content in a condition block or to change the context for variables used within the block, whereas other helpers return a value based on the parameters you pass. Helpers can be combined flexibly by using sub-expressions.

The Handlebars helpers that are currently supported, are explained in the following sections.

> → Remember
>
> When using a simple expression (like `{{variable}}` or `{{variable.array.[0]}}`), if the variable or array element does not exist, the variable will be ignored.
>
> When using a variable with a helper, if a required parameter of the helper is missing or is incorrect, the scripting will not be executed. For example, if you have used a helper with an incorrect parameter to define an action (*BUILD* tab), the action will be skipped.

Working with Objects

Helper	Use
`{{json object}}`	Returns a json object for "valueA".
`{{#with object}} ... {{/with}}`	Navigates the handlebars value scope into the object. Navigate into an object A with property B allows you to use `{{B}}` syntax to access an element in an array is directly instead of `{{A.B}}`
`{{lookup objectA objectB}}`	Returns the value of the object A accessed with a property value of object B.

Helper	Use

`{{flpAppState object1 object2 ...}}`	Re-formats the object so that it can directly be used in SAP Fiori Launchpad as an `AppStateData` filter.
	
	As parameters, either one or multiple objects, an array of objects or a combination of those can be passed.
	
	Each object must have the following attributes:
	
	• "service-value": value that the attribute should be compared to
	• "service-attribute": attribute that should be filtered by
	• "filter-operator": operator used for filtering, for example, "eq" is used when the value of the service attribute and the value passed as "service-value" should be exactly same. Supported filter operators are: eq, ne, lt, le, gt, ge.
	
	Objects with the same "service-attribute" will be combined in one `SelectOption` with two ranges.
	
	Input objects for flpAppState:
	
	Sample Code
	
	`{ "raw": "Notebooks", "value": "notebooks", "service-value": "C3712", "service-attribute": "CategoryID", "filter-operator": "EQ" }`
	
	Output object for flpAppState:
	
	Sample Code
	
	`{ "selectionVariant": { "SelectOptions": [ { "PropertyName": "CategoryID", "Ranges": [ { "Sign": "I", "Option": "EQ", "Low": "C3712" } ] } ] } }`

Helper	Use

	Usage of flpAppState:
	Sample Code
	``
	{
	  "fiori": {
	    "navigate": {
	      "semanticObject":
	"{{memory.SemanticObject}}",
	      "action":
	"{{memory.action}}",
	      "params": {
	        "{{memory.key1}}":
	"{{memory.value1}}"
	      },
	      "appStateData":
	{{ flpAppState memory.category }}
	    }
	  }
	}
	``
	Variations for flpAppState:
	• Working with two different properties `{{ flpAppState memory.category memory.supplier }}`
	• Working with one property but two filters `{{ flpAppState memory.category memory.category2 }}`
	• Working with one property that has two filters and another different property `{{ flpAppState memory.category1 memory.category2 memory.supplier }}`

Accessing Elements of an Array

> **i Note**
>
> The **new** syntax to access an element in an array is `{{array.[0]}}`. The **old** syntax `{{array[0]}}` is still supported for simple expressions, but cannot be used in combination with helpers and will be deprecated completely with one of the next releases.

Helper	Use
`{{first array}}`	Returns the first value inside of the "array" object provided.
`{{last array}}`	Returns last value inside of the "array" object provided.

Helper	Use
`{{itemAt array index}}`	Returns the value at specified "index" from the "array" object provided. The first element is accessed with index 0.

Working with Arrays

Helper	Use
`{{#each array}} ... {{/each}}`	Iterates over the elements in an array or object and sets the current item as context in the inner block. The following keywords can be used in the inner block: • `{{this}}` to access the current item • `{{@index}}` to get the index of the current item (starting with 0) • `{{@first}}` and `{{@last}}` to check if the current item is the first or last item • `{{@key}}` for objects only to get the attribute key
`{{#eachJoin array}} ... {{/eachJoin}}`	Iterates over the elements in an array or object and sets the current item as context in the inner block. A comma is added between the values that are returned by the inner block of each iteration, unless it returns an empty value. This helper is especially useful when building a JSON based on an array or object. Conditions in the inner block can be used to skip some items in the array. The following keywords can be used in the inner block: • `{{this}}` to access the current item • `{{@index}}` to get the index of the current item (starting with 0) • `{{@first}}` and `{{@last}}` to check if the current item is the first or last item • `{{@key}}` for objects only to get the attribute key
`{{#inArray array value}} ... {{else}} ... {{/inArray}}`	If "value" is in array this helper returns the first "..."-value else returns the second "..."-value.
`{{length array}}`	Returns the count of objects inside the "array" object provided.

Helper	Use
`{{pluck array subpath}}`	Create and return an array from child properties of the given "array". The "subpath" refers to a child property inside of an array element in "array". The "subpath" can also be used with the "." syntax to go deeper inside the object.
`{{unique array}}`	Unique helper returns an array with only unique values. It does not return a string (example with "each" and "eachJoin" for a string).
`{{sum numberArray}}`	Returns the sum of all numbers in the provided "numberArray" object.
`{{join array delimiter}}`	Joins all elements of an array into a string using the provided delimiter.

Working with Strings

Helper	Use
`{{append string suffix}}`	Appends the "suffix" to the string.
`{{lowercase string}}`	Transforms all characters in the string to lowercase letters.
`{{uppercase string}}`	Transforms all characters in the string to uppercase letters.
`{{isString value}}`	Returns true if the passed value is a string, false otherwise.
`{{occurrences string substring}}`	Searches for "substring" in "string" and returns the number of occurrences.
`{{remove string substring}}`	Removes all occurrences of "substring" within "string" and returns the result.
`{{replace string substringA substringB}}`	Searches for all occurrences of “substringA” in “string”, replaces them with “substringB” and returns the result.
`{{split string separator}}`	Splits "string" by using the separator and returns the values as an array.
`{{trim string}}`	Removes any white space characters from the beginning and the end of the string.
`{{length string}}`	Returns the length of the string.

Helper	Use
`{{escape variable}}`	Escapes any double-quotes in the variable in order to safely include it as a string within JSON. In case an object or array is passed, it will be converted to an escaped JSON string.

Working with OData Services

Helper	Use
`{{odataFilter value}}`	Returns a valid OData filter string to be used as `$filter` query parameter. For "value" parameter, either an object, an array of objects, a string or a combination of those can be passed. When an object or array of objects is passed, each object must have the following attributes: • "service-value": Value that the OData attribute should be compared to • "service-attribute": OData attribute that should be filtered by • "filter-operator": OData operator used for filtering, e.g. "eq" when the value of the OData attribute and the value passed as "service-value" should be exactly same Objects with the same "service-attribute" will be combined with an „or", all others with an „and". When a string is passed, it needs to be a valid OData filter string and will be appended with "and" to the filter string generated by the helper.

Example Context

Sample Code

```
"conversation": {
    "memory": {
        "supplier":[
       {
          "confidence": 0.99,
          "raw": "samsung",
          "service-value": 2,
          "filter-operator": "eq",
          "value": "samsung",
          "service-attribute": "SupplierID"
        },
        {
          "confidence": 0.99,
          "raw": "apple",
          "service-value": 5,
          "filter-operator": "eq",
```

```
            "value": "apple",
            "service-attribute": "SupplierID"
          }],
        "category":[
          {
            "confidence": 0.99,
            "raw": "notebooks",
            "service-value": 1,
            "filter-operator": "eq",
            "value": "notebooks",
            "service-attribute": "CategoryID"
          }],
  "nlp": {
      "entities": {
        "mass": [
          {
            "confidence": 0.99,
            "raw": "20 lbs",
            "scalar": 20,
            "unit": "lbs",
            "grams": 9071.84
          }
        ]
      }
```

Example OData Filter

`{{odataFilter memory.supplier memory.category 'myuser eq "true"'}}`

`myuser eq 'true' and (SupplierID eq 5 or SupplierID eq 2) and CategoryID eq 1`

Example usage in a URL

https://example.com/odata/?$format=json&$filter=`{{odataFilter memory.supplier memory.category 'myuser eq "true"'}}`

Resulting URL

https://example.com/odata/?$format=json&$filter=myuser%20eq%20%27true%27%20and%20(SupplierID%20eq%205%20or%20SupplierID%20eq%202)%20and%20CategoryID%20eq%201

Comparing Values

The compare helpers are used to construct logical conditions and return values depending on the comparison. The first "..." is a user defined value that will be returned if the comparison resolves as "True" otherwise the second "..." is returned.

In cases where you want to chain multiple compare helpers, you can use the “else” statement with the same or a different compare helper, for example, `{{else if condition}}` or `{{else eq value1 value2}}`.

Helper	Use
`{{#if valueA}} ... {{else}} ... {{/if}}`	Evaluates valueA for existence. If valueA is one of the values listed here (false, undefined, null, "", 0, or []) it will return the second "..."-value otherwise it returns the first "..."-value.

Helper	Use
`{{#unless valueA}} ... {{else}} ... {{/unless}}`	Evaluates valueA for non-existence. It has the inverse functionality of the "if" helper.
`{{#eq valueA valueB}} ... {{else}} ... {{/eq}}`	Evaluates if both values provided are equal (eq).
`{{#gt valueA valueB}} ... {{else}} ... {{/gt}}`	Evaluates if "valueA" greater than "valueB" (gt).
`{{#lt valueA valueB}} ... {{else}} ... {{/lt}}`	Evaluates if "valueA" is lower than "valueB" (lt).
`{{#gte valueA valueB}} ... {{else}} ... {{/gte}}`	Evaluates if "valueA" greater than or equal to "valueB" (gte).
`{{#lte valueA valueB}} ... {{else}} ... {{/lte}}`	Evaluates if "valueA" is lower than or equal to "valueB" (gte).
`{{#compare valueA operator valueB}} ... {{else}} ... {{/compare}}`	Compares "valueA" with "valueB" using a JavaScript operator as "operator" (for example, "===" or "<=").
`{{default optionalValue defaultValue}}`	Returns the "optionalValue" - if it is defined - otherwise the "default" value is returned.

Logical Operators

Helper	Use
`{{#and expressionA expressionB}} ... {{/and}}`	Returns the "..."-value if both of the given expressions resolve to True.
`{{#or expressionA expressionB}} ... {{/or}}`	Returns the "..."-value if one of the given expressions is resolved to True.
`{{#not expression}} ... {{/not}}`	Returns the "..."-value if no given expressions resolve to True

Working with Numbers

Helper	Use

`{{add valueA valueB}}` / `{{plus valueA valueB}}`
`{{subtract valueA valueB}}` / `{{minus valueA valueB}}`
`{{multiply valueA valueB}}`
`{{divide valueA valueB}}`
	Returns a number after the execution of the arithmetic function.
	Add
	subtract
	multiply
	divide valueB with valueA

`{{modulo valueA valueB}}`	Returns the remainder after division (valueA divided by valueB)
`{{ceil value}}`	Returns the next higher absolute number
`{{floor value}}`	Returns the next lower absolute number
`{{round value}}`	Returns the correct rounded absolute number
`{{formatNumber number decimalLength decimalSeparator thousandsSepatator}}`	Formats the number with the specified number of (rounded) digits after the decimal separator and the specified character is used as decimal separator. In addition, a separator for the group of thousands can be provided. All parameters are optional. Following are some examples: `{{formatNumber value 2}}`- Returns the correct rounded number at two decimal digits `{{formatNumber value decimalSeparator}}`- Returns the number with a specified decimal and thousand separator `{{formatNumber value decimalSeparator thousandSeparator}}`- Returns the number with a specified decimal and thousand separator `{{formatNumber value decimalLength decimalSeparator thousandSeparator}}`- Returns the number with specified decimal digits and separators

Dates

The date helper returns the provided "date" and "utc-offset" formatted with the provided "pattern" as utc. If no utc-offset is provided the return value will be calculated with utc "+00:00".

```
{{formatDate date pattern utc-offset}}
```

The date value can have the following formats:

- /Date(1338282808000)/
- 1544464091000+0000
- 1992-01-01T00:00:00
- 01-31-2000 (Month/Day/Year)

The pattern could have formats like these:

- "DD-MM-YYYY"
- "DD-MM-YYYY - HH:mm"
- "MM-YYYYZ"

The utcOffset should be formatted like this:

- "-02:00" or "+01:00"(for strings)
  -120 or 60 (for number values as minutes)

> **Note**
>
> Numbers between -16 and 16 will be automatically used as hours and +11 will result in "+11:00"

Examples

This section uses the above handlebars statements with some example data to elaborate the usage of the supported handlebars helpers.

Context Used

> **Sample Code**
>
> **Data**

```
{
  "api_service_response": {
    "body": {
      "results": [
        {
          "Name": "Dayum",
          "Price": "100",
          "Available":true,
          "lookMeUp": "foundMe",
          "to_ProductCategory": {
            "MainProductCategory": "Computer Systems"
          }
        },
        {
          "Name": "test",
          "Price": "200",
          "Available":true,
          "lookMeUp": "foundMe",
          "to_ProductCategory": {
            "MainProductCategory": "Computer Systems"
          }
        },
        {
          "Name": "wow",
          "Price": "50",
```

```
            "Available":false,
            "lookMeUp": "foundMe",
            "to_ProductCategory": {
              "MainProductCategory": "Computer Systems"
            }
          }
        ],
        "lookup" : {
            "foundMe": {"works": "works=true"}
          },
        "stringArray" : ["a", "b", "c", "d", "e", "e"]
      }
    }

  }
```

Handlebars

Returns the accessed object value and elevates the scope into the access object to print the result.

Sample Code

Working with Objects - with & lookup

```
"{{api_service_response.body.results.0.Name}} with: {{#with (lookup
api_service_response.body.lookup api_service_response.body.results.
0.lookMeUp)}}{{works}}{{/with}}" // handlebars statement
"Dayum with: works=true" // result
```

Gets the first item of an list or array object

Sample Code

Accessing elements - first

```
"{{first api_service_response.body.stringArray}}" // handlebars statement
"a" // result
```

Returns a specific element of a list/array object

Sample Code

Accessing elements - itemAt

```
"{{itemAt api_service_response.body.stringArray 2}}" // handlebars statement
"c" // result
```

Returns the unique values from an array using the "each" and "eachJoin" helper in combination with the "unique" helper

Sample Code

Working with arrays - unique

```
"{{#each (unique api_service_response.body.stringArray)}}{{.}}{{/each}}" //
handlebars statement
"abcde" // result

"{{#eachJoin (unique api_service_response.body.stringArray)}}
{{.}}.atIndex({{@index}}){{/eachJoin}}" // handlebars statement
```

```
"a.atIndex(0),b.atIndex(1),c.atIndex(2),d.atIndex(3),e.atIndex(4)"// result
```

Retrieves property in array of objects as list

Sample Code

Working with arrays - pluck

```
"{{pluck api_service_response.body.results "Name"}}" // handlebars statement
"Dayum,test,wow" // result

"{{pluck api_service_response.body.results
"to_ProductCategory.MainProductCategory"}}" // handlebars statement
"Computer Systems,Computer Systems,Computer Systems" // result
```

Using the pluck helper together with the json helper to get an array of sub properties. Without the json helper you get a concatenated string of the elements.

Sample Code

Working with arrays - pluck and json

```
"{{json (pluck api_service_response.body.results
"to_ProductCategory.MainProductCategory")}}" // handlebars statement
"["Computer Systems","Computer Systems","Computer Systems"]" // result
```

Returns a concatenated string of all elements inside of the list or array element. Usage of the *@index* keyword will add the index of the current iterated object.

Sample Code

Working with arrays - eachJoin

```
"{{#eachJoin api_service_response.body.results}}{"title":
"{{Name}}","subtitle": "No. {{@index}}"}{{/eachJoin}}" // handlebars statement
"{"title": "Dayum","subtitle": "No. 0"},{"title": "test","subtitle": "No. 1"},
{"title": "asdf","subtitle": "No. 2"}" // result
```

Compares two parameters with a JavaScript operator.

Sample Code

Comparing values - compare

```
"{{#compare api_service_response.body.results.0.Name "==="
api_service_response.body.results.1.Name}}This will not be rendered because
0.Name and 1.Name are different.{{else}}This will be rendered.{{/
compare}}" // handlebars statement
"This will be rendered." // result
```

Compares two numbers. This checks if valueA is bigger than valueB.

Sample Code

Comparing values - greaterThan (gt)

```
"{{#gt api_service_response.body.results.0.Price data.results.1.Price}}True:
This will be rendered.{{else}}False: This will be rendered.{{/gt}}" //
handlebars statement
"False: This will be rendered." // result
"{{#gt api_service_response.body.results.1.Price data.results.0.Price}}True:
This will be rendered.{{else}}False: This will be rendered.{{/gt}}" //
handlebars statement
"True: This will be rendered." // result
```

Checks if two expressions are true.

Sample Code

Logical operators - and

```
"{{#and api_service_response.body.results.0.Available
api_service_response.body.results.1.Available}}Both 'True'{{else}}At least
one not 'true'{{/and}}" // handlebars statement
"Both 'True'" // result

"{{#and api_service_response.body.results.0.Available
api_service_response.body.results.2.Available}}Both 'True'{{else}}At least
one not 'true'{{/and}}" // handlebars statement
"At least one not 'true''" // result
```

Transforms Date to UTC with Offset

Sample Code

Dates - formatDate

```
"{{formatDate "01-31-2000" "DD-MM-YYYYZ" 120}}" // handlebars statement
"31-01-2000+02:00" // result

"{{formatDate "/Date(1544464091000+0000)" "DD-MM-YYYY - HH:mm" "+01:00"}}" //
handlebars statement
"10-12-2018 - 18:48" // result

"{{formatDate api_service_response.body.results.0.ReleaseDate "YYYY"}}" //
handlebars statement
"1990" // result
```

Format the number with the formats as common in US and Germany.

Sample Code

Working with Objects - with & lookup

```
{{formatNumber price 2 '.' ','}}
100 -> 100.00
99999.9504 -> 99,999.95


{{formatNumber price 2 ',' '.'}}
100 -> 100,00
99999.9504 -> 99.999,95
```

```
{{formatNumber price 2 ','}}
100 -> 100,00
99999.9504 -> 99999,95

{{formatNumber price ',' '.'}}
100 -> 100
99999.9504 -> 99.999,9504
```

Natural Language Generation Helpers

If you have used variables (bot memory, API responses and so on), you can use natural language generation (NLG) helpers in scripting to improve grammatical correctness of your bot responses that are aligned with the dynamic data.

The grammar helpers can be used for grammatical functions like definite and indefinite articles and singular or plural form of words in the variable scripting syntax. The generation helpers usually take arrays as input and generate a phrase from those list or arrays.

The following NLG helpers are available and can be used for ensuring grammatically correct bot responses.

> **i Note**
>
> The script should not have newline or tab characters.

Grammar Helpers

Helper	Use
`{{definite memory.country}}` `{{definite memory.country capitalize=true}}`	Generates definite article based on the memory field and response text. Generates definite article based on the memory field and response text and capitalizes the word. Default capitalize is false.
`{{indefinite memory.seatType}}` `{{indefinite memory.seatType capitalize=true}}`	Generates indefinite article based on the memory field and response text. Generates indefinite article based on the memory field and response text and capitalizes the word. Default capitalize is false.
`{{pluralize memory.materialName quantity=5}}` `{{pluralize memory.materialName quantity=5 include-quantity=false}}`	Pluralizes the word appropriately based on quantity. Default include-quantity is true. Pluralizes the word appropriately based on quantity and includes the quantity in response

Generation Helpers

Helper	Memory contents	Result	Use
`{{list memory.seatTypes}}` `{{list memory.seatTypes conjunction='or' separator=';'}}`	Sample Code `seatTypes [ "window", "aisle", "extra leg room" ]`	*window, aisle and extra leg room* *window; aisle or extra leg room*	Generates natural language list of array values separated by a separator and a conjunction. The separator and conjunction may be specified, the default values are a comma and "and".
Here are your orders with `{{ frame filter }}`	Sample Code `filter: { label: "price", value: "200 USD", operator: "<" }`	*Here are your orders with price less than 200 USD*	Generates a frame or operator phrase with label, value and operator. The values are extracted from a memory object which has a key-value structure.
`{{ frame label=field, value=limit,operator=comparison,hint=type`	Sample Code `field: "quantity", limit: "200", comparison: "<", type: "countable"`	*Here are your orders with quantity fewer than 200*	As above, but the values are taken from individual memory variables. The hint specifies the kind of value involved; it can be 'value' (default), 'countable' and 'date'. For details, see list of operators below.
Here are your orders with `{{ frame filtersList operator="<"}}`	Sample Code `filtersList: [ { label: "price", value: "200 USD", operator: ">" }, { label: "delivery date", value: "05/17/2020", operator: ">", hint: "date"}`	*Here are your orders with price less than 200 USD and delivery date before 05/17/2020*	Generates a list of frames from a list of key-value structures. Note that the 'label'/'value'/ 'operator'/'hint' arguments may be specified in this use case as well. Their values will be used for all frames, overwriting the values given by the memory object.

Helper	Memory contents	Result	Use
Here are your orders with `{{ frame filtersList label_property="name"}}`	Sample Code `filtersList: [ { name: "price", value: "200 USD", operator: "<"}, { name: "delivery date", value: "05/17/2020", operator: "<=", hint: "date"}`	*Here are your orders with price less than 200 USD and delivery date before or on 05/17/2020*	As above, but the `label_property` argument is used to indicate that the property `name` should be used for the label. For value, operator and hint, the equivalent arguments `value_property`, `operator_property` and `hint_property` can be specified.

Examples

Context Used

Sample Code

```
{
memory: {
sortCriteria: { label: "products", value: "200 USD", operator: "=" },
airlines :[ 'United', 'American', 'Alaska'],
airlinesSchedule : [ { label : 'united', time : '5 PM', operator: '>' },
{ label : ['American', 'Alaska'], time : '8 PM', operator: '>' }, { label :
'Delta', time : '12pm', operator : '=' } ],
seatType1 : "window",
seatType2 : "aisle",
seatTypes : ["window", "aisle"],
numberOfSeats : 2,
class : "business",
accomidationType : "seat"
} }
```

Example	Result
Your reservation for `{{pluralize memory.accomodationType quantity=memory.numberOfSeats include-quantity=true}}` is successful. You got `{{indefinite (list memory.seatTypes conjunction='and')}}`.	*Your reservation for 2 seats is successful. You got a window seat and an aisle seat.*
Here are the products with `{{frame searchCriteria}}`	*Here are the products with price less than or equal to 200 USD.*

Example	Result
Here are `{{pluralize 'airlines' quantity=(length airlines) include-quantity=true}}` between SFO and SEA : `{{list airlines}}`. Their schedules are : `{{frame airlinesSchedule hint='date'}}`.	*Here are 3 airlines between SFO and SEA : United, American and Alaska. Their schedules are : United after 5pm, American after 8PM and Alaska before 12pm).*

Operators

The following operators are supported for frame helper.

List of operators

Operator	Hint	DE	EN	ES	FR
=	value	""	""	""	"à"
	countable	""	""	""	"à"
	date	"am"	""	""	"le"
!=	value	"nicht"	"not "	"no"	"différent de"
	countable	"nicht"	"not "	"no"	"différent de"
	date	"nicht am"	"not "	"no"	"différent du"
>	value	"größer als"	"greater than"	"mayor que"	"supérieur à"
	countable	"mehr als"	"more than"	"más de"	"plus de"
	date	"nach dem"	"after"	"después del"	"après le"
>=	value	"größer als oder gleich"	"greater than or equal to"	"mayor que o igual a"	"supérieur ou égal à"
	countable	"mehr als oder genau"	"more than or equal to"	"más de o igual a"	"plus de ou égal à"
	date	"nach dem oder am"	"after or on"	"después del o el"	"après le ou le"
<	value	"geringer als"	"less than"	"menor que"	"inférieur à"

Operator	Hint	DE	EN	ES	FR
	countable	"weniger als"	"fewer than"	"menos de"	"moins de"
	date	"vor dem"	"before"	"antes del"	"avant le"
<=	value	"kleiner oder gleich"	"less than or equal to"	"menor que o igual a"	"inférieur ou égal à"
	countable	"weniger als oder genau"	"fewer than or equal to"	"menos de o igual a"	"moins de ou égal à"
	date	"vor dem oder am"	"before or on"	"antes del o el"	"avant le ou le"
==	value	"ist"	"is"	"es igual a"	"est à"
	countable	"ist"	"is"	"es igual a"	"est à"
	date	"ist am"	"is"	"es el"	"est le"
=!=	value	"ist nicht"	"is not"	"no es igual a"	"n'est pas à"
	countable	"ist nicht"	"is not"	"no es igual a"	"n'est pas à"
	date	"ist nicht am"	"is not"	"no es el"	"n'est pas le"
=>	value	"ist größer als"	"is greater than"	"es mayor que"	"est supérieur à"
	countable	"ist mehr als"	"is more than"	"es más de"	"est plus de"
	date	"ist nach dem"	"is after"	"es después del"	"est après le"
=>=	value	"ist größer als oder gleich"	"is greater than or equal to"	"es mayor que o igual a"	"est supérieur ou égal à"
	countable	"ist mehr als oder genau"	"is more than or equal to"	"es más de o igual a"	"est plus de ou égal à"

Operator	Hint	DE	EN	ES	FR
	date	"ist nach dem oder am"	"is after or on"	"es después o el"	"est après le ou le"
=<	value	"ist geringer als"	"is less than"	"es menor que"	"est inférieur à"
	countable	"ist weniger als"	"is fewer than or equal to"	"es menos de"	"est moins de"
	date	"ist vor dem"	"is before"	"es antes del"	"est avant le"
=<=	value	"ist geringer als oder gleich"	"is less than or equal to"	"es menor que o igual a"	"est inférieur ou égal à"
	countable	"ist weniger als oder genau"	"is fewer than or equal to"	"es menos de o igual a"	"est moins de ou égal à"
	date	"ist vor dem oder am"	"is before or on"	"es antes del o el"	"est avant el ou le"

Nesting

If you have added a script that includes NLG helpers, all the other placeholders should be resolved *first* before the NLG helpers are resolved. In most cases you don't need to use NLG helpers as part of the non-NLG helpers. Following are a few modalities in using NLG helpers.

- NLG helpers cannot be sub-expression of non-NLG helper or can be condition expression.
- NLG helpers can have other handlebar helpers as sub-expressions and can be used in all block expressions.
- NLG helpers can be nested with other NLG helpers.

Following are some examples of both valid and invalid usage of NLG helpers:

Valid Usage	Invalid Usage

NLG helper using handlebar helper (pluralize helper using length)
`{{#if (length phoneList)}}`
We offer `{{pluralize 'brands' quantity=(length phoneList) }}`. They are `{{list phoneList conjugation='and'}}`
`{{/if}}`
	If block condition expression
	`{{#if (definite phoneList)}} . ---→ Invalid`
	We offer `{{pluralize 'brands' quantity=(length phoneList) }}`. They are `{{list phoneList conjugation='and'}}`
	{{/if}}

NLG helper looping block handler
`{{#with Order}}`
Your order `{{orderid}}` for `{{pluralize item quantity=orderquantity}}` of `{{price}}` is `{{status}}` and expected to deliver on `{{formatDate deliveryDate 'MMDDYYYY'}}`
`{{/with}}`
	NLG helper as sub-expression
	`{{#with Order}}`
	Your order `{{orderid}}` for `{{pluralize item quantity=orderquantity}}` of `{{price}}` is `{{status}}` and expected to deliver on `{{formatDate deliveryDate (pluralize 'MMDDYYYY')}}`
	`{{/with}}`

4.13.2 Runtime Data Accessible

Variables

You can create all of the variables listed in the following table while creating skills (*BUILD* tab) and the actions associated with them.

Variable	Description
{{conversation_id}}	ID of the current conversation
{{participant_data}}	An object filled with participant information that is provided by the channel connected through the Bot Connector. For example, for those channels that supply a username, it is returned as `userName`. You can easily use `{{participant_data.userName}}`. For more information, see section *User Information* in Messaging Channels [page 181].

Variable	Description
{{client_info}}	Client specific information which can be optionally passed by the client (when using /dialog endpoint) or is provided by the channel. For example, for channels that automatically detect the user language, it is returned as language and can be accessed with `{{client_info.language}}`. For more information, see Client Information [page 194] under *SAP Conversational AI Web Client*.
{{memory}}	The complete memory object. You can access each element, for example, `{{memory.person.raw}}`. Here, `person` is the alias of a requirement.
{{skill_occurences}}	Number of consecutive occurrences of the current skill
{{language}}	Language ISO code of the current conversation
{{current_message}}	The message source sent by the user
{{#entity}}	An alias of the `nlp.entities.entity[0]` object. You can access each element, for example, `{{#entity.confidence}}`, where `confidence` is `nlp.entities.entity[0].confidence`. Here an entity can be a gold or a custom entity like color, phone-type and so on. For example, `{{nlp.entities.color[0].raw}}` `=={{#color.raw}}` **i Note** This variable can not be used in combination with helpers, for example `{{#if #entity.confident}}` cannot be used
{{message_received_at}}	Timestamp of when the message was received
{{nlp}}	The complete `nlp` object. You can access any property of it, for example, `{{nlp.sentiment}}`.

5 Bot Connector

5.1 Messaging Channels

You can connect your bot to any of the supported channels such as Amazon Alexa, Facebook Messenger, Slack and so on.

The messages that you create using the standard message formats (under Message Types or through scripting) or the Bot Connector API are rendered in the connected channel based on the message format supported by the channel.

If a channel doesn't natively support a message format, the Bot Connector will handle it and rewrite the content to have a readable message everywhere. For example, if buttons are not supported by the channel, they may be displayed as links in the channel.

User Information

The user information is stored in the object `participant_data` and is scoped by the channel that your bot is connected to. If your user is using different channels to access your bot, the user information will be distinct depending upon the channel. The following table shows the fields shared by each channel that are stored in the `participant_data` object (for example, {{`participant_data.userName`}}).

Channel	User Information
Amazon Alexa	email, userName (Firstname + Lastname)
Facebook Messenger	userName (Firstname + Lastname)
LINE	userName (display name)
Webchat	None
SAP Conversational AI Web Client	email, last_name (familyName), fullName (first part of given-Name split by space)
SAP CoPilot	None
SAP Jam Collaboration	userName (Firstname + Lastname) , jamId (Id)

Channel	User Information
Microsoft (Skype, Microsoft Teams) **i Note** Cortana channel has been retired by Microsoft since January 31, 2021. As a consequence, it is no longer supported by SAP Conversational AI.	email, firstName, lastName, userName (Firstname + Lastname)
Slack	userName (realName), email
Telegram	None
Twilio	None
Twitter	userName (name)

> **i Note**
>
> There is no `participant_data` object for fallback channels.

Client Information

If your bot is connected to a channel, the client specific information is passed and stored in the `client_info` parameter.

> **i Note**
>
> Currently, this feature is supported by SAP Conversational Web Client only.

The web client automatically detects the users' language and it can be accessed at runtime using `{{client_info.language}}`. For more information, see Client Information [page 194] under *SAP Conversational AI Web Client*.

Client information can also be accessed while defining conditions [page 96], using the scripting syntax (see Runtime Data Accessible [page 179]), within Initialize Skill [page 94] or any other skill.

5.1.1 Amazon Alexa

You can connect your bot to Amazon Alexa devices and leverage the voice command feature for your bot.

Please follow the steps to configure the channel as provided under the *Connect* tab.

As an enterprise user, if you want to integrate your bot with Amazon Alexa, your enterprise tenant should be allowlisted.

If your tenant is not allowlisted yet, please raise a ticket titled as SAP Conversational AI – Alexa Custom Tenant Allowlisting for the component `CA-ML-CAI-CON`. Please mention your custom tenant URL.

The following table lists the Message Types [page 120] that are supported:

Message Type	Supported
Text	No
Card	No
Buttons	No
Quick Replies	No
Carousel	No
List	No
Image	No
Custom	No
Voice	Yes
Video	No

5.1.2 Facebook Messenger

You can connect your bot to Facebook Messenger that can interact with your end users.

Please follow the steps to configure the channel as provided under the *Connect* tab.

The following table lists the Message Types [page 120] that are supported:

Message Type	Supported
Text	Yes
Card	Yes
Buttons	Yes
Quick Replies	Yes
Carousel	Yes
List	Yes
Image/GIF	Yes

Message Type	Supported
Custom Only the *Custom* option within the *Custom* tile is supported. See Message Templates for Facebook Messenger.	Yes
Voice	No
Video	Yes

5.1.3 LINE

You can connect your bot to LINE that can interact with your end users.

Please follow the steps to configure the channel as provided under the Connect tab.

The following table lists the Message Types [page 120] that are supported:

Message Type	Supported
Text	Yes
Card	Yes
Buttons	Yes
Quick Replies	Yes
Carousel	No
List	No
Image/GIF	Yes
Custom Only the *Custom* option within the *Custom* tile is supported. See Flex Messages for Line.	Yes
Voice	No
Video	Yes

5.1.4 Microsoft (Skype, Teams)

You can connect your bot to Microsoft (Skype, Teams) that can interact with your end users.

Please follow the steps to configure the channel as provided under the *Connect* tab.

> **i Note**
>
> SAP Conversational AI is available for Skype but **not** for Skype for Business. Skype has a limitation of 100 users per bot. You will not be able to publish your bot to more users.
>
> Cortana channel has been retired by Microsoft since January 31, 2021. As a consequence, it is no longer supported by SAP Conversational AI.

The following table lists the Message Types [page 120] that are supported:

Message Type	Supported
Text	Yes
Card	Yes
Buttons	Yes MS Teams supports the button type Skill Trigger [page 121]. Therefore, the usage of Disambiguation Skill [page 90] is also supported.
Quick Replies	Yes
Carousel	Yes
List	Yes
Image/GIF	Yes
Custom Only the *Custom* option within the *Custom* tile is supported	Yes
Voice	No
Video	No

5.1.5 SAP Conversational AI Web Client

The SAP Conversational AI Web Client is a conversational user interface for connecting to SAP Conversational AI chatbots via the SAP Conversational AI Web Client channel. It is a rich web client capable of rendering the bot responses using SAP Fiori-compliant UI controls (see SAP Fiori Design Guidelines).

Mutiple browsers are supported by SAP Conversational AI Web Client. For more information, see Browser Support [page 7].

Please follow the steps to configure the channel as provided under the *Connect* tab.

The following table lists the Message Types [page 120] that are supported:

Message Type	Supported	Details
Text	Yes	
Card	Yes	
Buttons	Yes	A maximum of 3 buttons are displayed. SAP Conversational AI Webchat supports the button type Skill Trigger [page 121]. Therefore, the usage of Disambiguation Skill [page 90] is also supported.
Quick Replies	Yes	
Carousel	Yes	
List	Yes	
Image/GIF	Yes	Image formats that are supported by Web Client are - *BMP*, *GIF*, *ICO*, *JPEG*, *PNG*, *SVG*, and *TIFF*). This depends on the browser that you use. For more information, see Image file type and format guide .
Custom	Yes	All the existing message types like *Text*, *Card*, *Buttons* and so on, defined using the custom scripting option are fully supported by the SAP Conversational AI Web Client. However, the *Custom* option within the custom tile is not supported. For message types *Card*, *Carousel* and *List*, you can define more UI attributes using *Custom* tile (through scripting) as compared to messages defined using the relevant tile. For example, attributes like `status`, `statusState`, `section` and so on can be only defined using the Custom message type and is only supported for SAP Conversational AI Web Client.
Voice	No	

Message Type	Supported	Details
Video	Yes	The video URLs that are currently supported are - *YouTube*, *Vimeo*, *Daily Motion*, *SAP Video*, and *SAP TV Video*. The supported video formats are - *MP4*, *WEBM*, *OGV*, and *3GP*.

SAP Conversational AI Web Client follows specific design guidelines and limits while rendering different message types. For more information, see UX Recommendations for Web Client [page 200].

Related Information

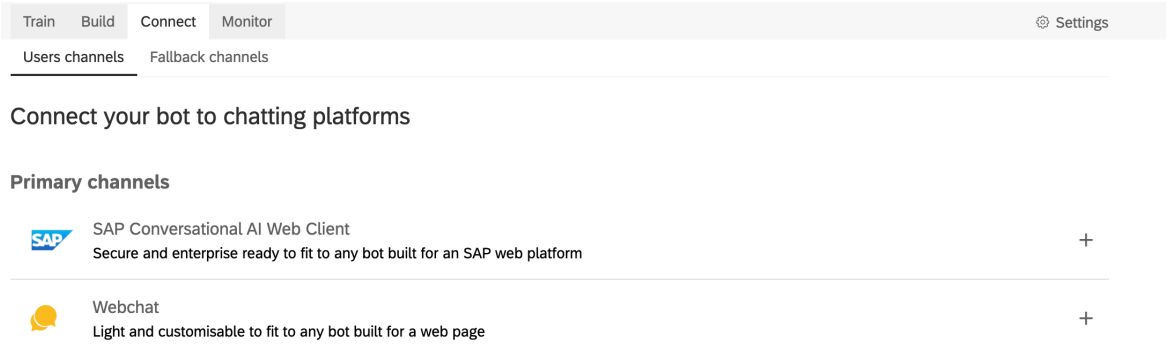
Single Sign-On with SAP Product Integration [page 135]
Connect to External Service [page 136]

5.1.5.1 How to Use the Web Client

Create a Channel

Procedure

- Go to the *Connect* tab of your bot and create an *SAP Conversational AI Web Client* channel.



5.1.5.2 Integration Settings

The SAP Conversational AI Web Client offers two methods of integration. The first uses a snippet with a single channel ID, and the second uses an application ID.

Using a Snippet with a Single Channel ID

Context

When the SAP Conversational AI Web Client connects to a single channel, it can be integrated into any Web page, without requiring user authentication.

Procedure

- Simply provide a technical name for your channel and paste the provided snippet (displayed when you hit *Create*) into your Web application. You're now ready to chat with your bot.

  When using the SAP Conversational AI Web Client from an enterprise tenant on a non-authenticated web site, add the parameter `data-use-public-api=true` to your code snippet. For example:

  Sample Code

  ```
  <script
  src="https://tenant.domain.com/resources/public/webclient/bootstrap.js"
  data-channel-id="XXXXXX"
  data-token="XXXXXX"
  data-expander-type="CUSTOM"
  data-use-public-api=true
  id="cai-webclient-custom">
  </script>
  ```

Using an Application ID

Prerequisites

To use the application ID method, the SAP Conversational AI Web Client needs to be integrated into the main Web application or application shell of a supported SAP product (for example, the SAP Fiori launchpad of an

SAP S/4HANA system) or into a Web application that meets the same security and authentication requirements. For information on requirements and the integration process, see the Configuration Guide for SAP Conversational AI.

Context

The application ID method offers a very flexible way of mapping a channel to a given Web application. It also supports mapping of multiple channels to a single SAP Conversational AI Web Client. When you use this method, end users are offered a channel selection menu when they open the web client.

Procedure

- Provide two mandatory parameters and one optional channel title:

  Configuration

Field	Description	Example
Name	The technical name of the channel	`my-new-channel`
Application ID	The ID of the application where the SAP Conversational AI Web Client is integrated. The application ID depends on the type of SAP product or the type of integration (snippet or pre-integrated). For SAP S/4HANA systems with SAP Fiori launchpad as the front end, the application ID is `<BOT_OWNER><SID><CLIENT>`. The application ID will automatically be prefixed with the bot owner slug (the user or organization owning the bot) to ensure its uniqueness. **i Note** The BOT_OWNER only appears in the community tenant. There is no BOT_OWNER in the enterprise tenant.	`org-ABC123`

Field	Description	Example
Channel Title	A title that will be displayed in the channel selection menu when more than one channel is mapped to a given application ID.	`My Channel Title`

5.1.5.3 Enabling Role Based Access to Bots

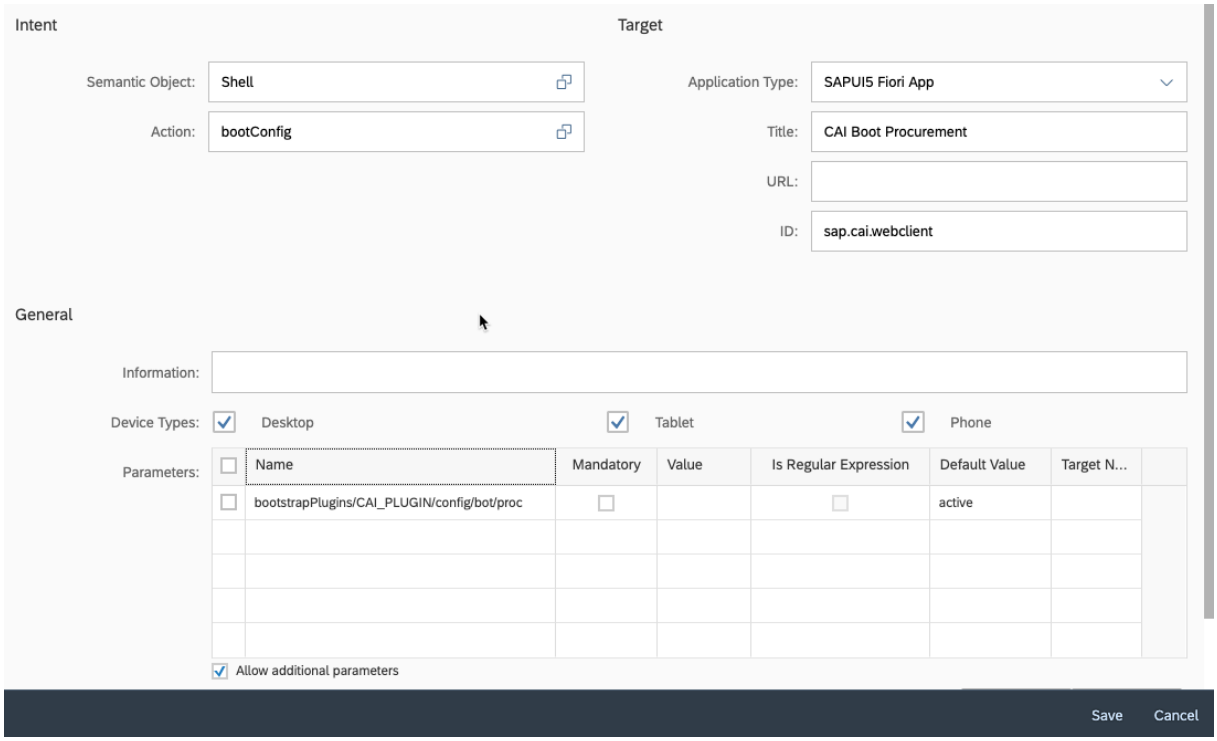
You can provide users access to different bots in the SAP Fiori Launchpad based on user role assignment and using the target mappings as described.

As an SAP Fiori Launchpad administrator, you can assign one or more bots to the same user by creating Fiori catalogs with the following configuration:

> → Remember
>
> User must also be assigned a catalog that adds the Shell plugin.

Field	Value
Semantic Object	Shell
Action	bootConfig
Application Type	SAP UI5 Fiori App
Title	your configuration name
ID	sap.cai.webclient

Then add a parameter `bootstrapPlugins/CAI_PLUGIN/config/bot/<bot>` with value *active*.



The `<bot>` is a string that will be used to identify the *Webclient* channel *applicationId* following the pattern `<SID><CLIENT>-<bot>`.

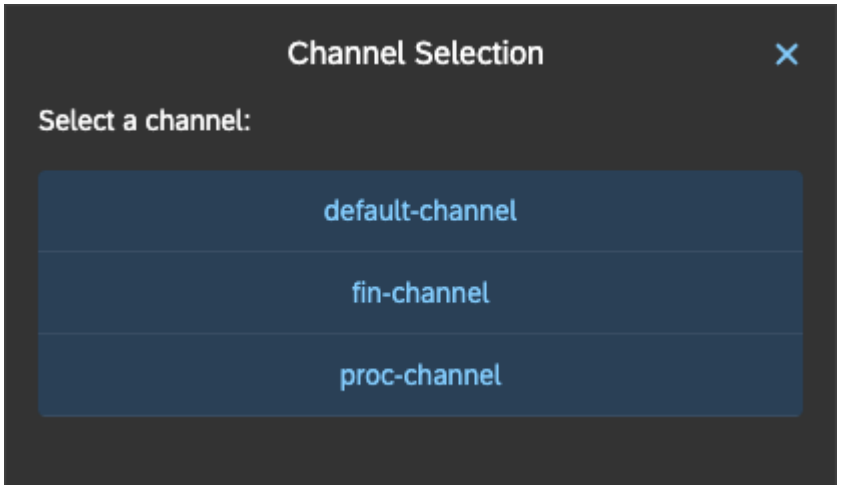
Example

The procurement specific catalog contains a parameter named `bootstrapPlugins/CAI_PLUGIN/config/bot/proc` set to *active* in the system `WMT240`. This will refer to the channel with *application Id* `WMT240-proc`. Once you assign a user to this catalog, the user will be able to use this channel.

> **i Note**
>
> Values of SystemId (`SID`) and Client (`CLIENT`) for your ABAP front-end server can be retrieved with the API `sap.ushell.Container.getLogonSystem()`.
>
> The BOT_OWNER is only added to the *applicationId* in the community tenant. There is no BOT_OWNER in the enterprise tenant.

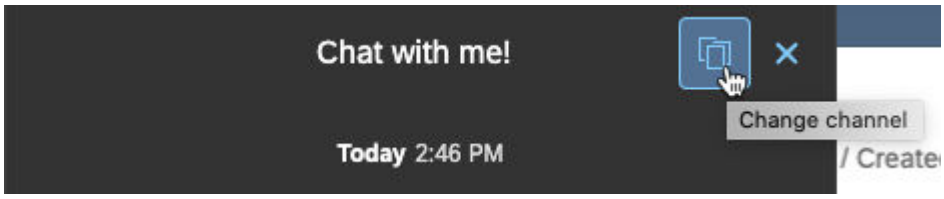
Assigning Multiple Bots

If you have assigned multiple bots to a user, a list of all channels are displayed in the FLP. The user needs to select a channel by clicking the appropriate title.



Next time, when the user logs in, the same bot is launched by default.

The user can select a different bot using the channel selector.



Changing the Default Channel

The default channel, the one with application id set to `<SID><CLIENT>` (like `UYR490`) is always added by default. It is possible to opt-out the default channel by assigning the user a catalog containing the same *bootConfig* catalog as defined above with parameter `bootstrapPlugins/CAI_PLUGIN/config/defaultBot` set to *inactive*.

5.1.5.4 Customizing Your Channel

When using snippet integration (either for a single channel or with an application ID), you can customize the look and behavior of your SAP Conversational AI Web Client.

- **Color Scheme**
  You can either use the default SAP theme or create your own custom scheme by specifying the different basic and accent colors to be used.
  You can define a custom cascading style sheet (CSS) using the editor or provide a link to a CSS to fine tune the look and feel of the web client. Please ensure that the stylesheets are accessible according to CORS principle.

  > **i Note**
  >
  > Custom CSS supports CSS variables, however the CSS input editor flags them as error or warning during syntax validation.

  For more information, see the following video.

- **Header Customization**
  You can define the header title and logo of the web client.
- **Message Settings**
  You can define the pictures that are displayed next to the bot replies and user messages. You can also define the input placeholder as well as a welcome message that will be displayed each time the user opens the SAP Conversational AI Web Client.
- **General Settings**
  The *General Settings* let you define the following:
  - The lifetime of a conversation with a bot
  - The language of your permanent static menu if you have created one
  - The behavior of the Web Client when the web application is opened by the user
  - The maximum length of a user message

  Web Client Standalone Page
  A URL for a stand alone web page is generated for the Web Client channel. The web page can be accessed using a URL that is generated based on your tenant. As a bot developer, you can use this link to provide direct access to the Web Client channel to your users or testers.
  The web page has the following features:
  - The page design reacts to custom themes defined in channel configuration
  - The page is mobile responsive
  - The standalone page uses a page header similar to the SAP Fiori Shell Bar
  For more information, see the following video.

  Rendering Web Client in a Custom Container
  When you configure the Web Client channel, by default, a script is generated that is used to render Web Client in an overlay. By using some additional parameters, you can:
  - render Web Client in a specific zone or container in your web application
  - control the size of the container
  - hide the header by setting the `data-hide-header` as `true`

  Since it is not a floating chat window, the user does not have the option to open or close it using a button. You need to insert the `<div>` tag with an `id` and specify a value for the `id`. You use this as the value for the `data-inplace-container-id` parameter. For example:

Sample Code

```
<div id="webclient-container" style="width:800px;height:
1000px;border:solid" />
```

Sample Code

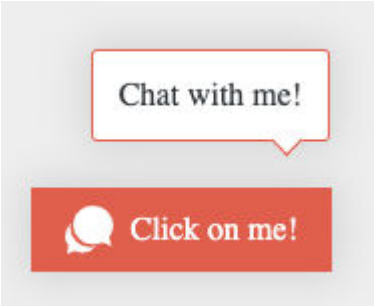
```
<script
src="https://<url>/webclient/bootstrap.js"
data-channel-id=<channel_id>
data-token=<token> data-inplace-container-id="webclient-container" data-
hide-header=true
data-expander-type="CAI"
data-expander-
preferences="JTdCJTIyZXhwYW5kZXJMb2dvJTIyJTNBJTIyaHR0cHMlM0ElMkYlMkZjZG4uY2
FpLnRvb2xzLnNhcCUyRndlYmNoYXQlMkZ3ZWJjaGF0LWxvZ28uc3ZnJTIyJTJDJTIyZXhwYW5kZ
XJUaXRsZSUyMiUzQSUyMkNsaWNrJTIwb24lMjBtZSElMjIlMkMlMjJvbmJvYXJkaW5nTWVzc2Fn
ZSUyMiUzQSUyMkNoYXQlMjB3aXRoJTIwbWUhJTIyJTJDJTIydGhlbWUlMjIlM0ElMjJERUZBVUx
UJTIyJTdE"
id="cai-webclient-custom">
</script>
```

For more information, see the following video.

5.1.5.5 Opening the Web Client

When you use the snippet approach to integrate the SAP Conversational AI Web Client into your Web application, you can create your own button and implement the events to open and close the web client using the JavaScript API (see the Development [page 195] section below).

Alternatively you can use the default button provided by the SAP Conversational AI Web Client, which appears automatically in your Web application. You can specify the button label, a logo, and call to action text, such as the following:



5.1.5.6 Client Information

As a bot developer, you can access the language setting detected by SAP Conversational AI Web Client (at runtime) and set this as the default language for the conversation before the bot's first response.

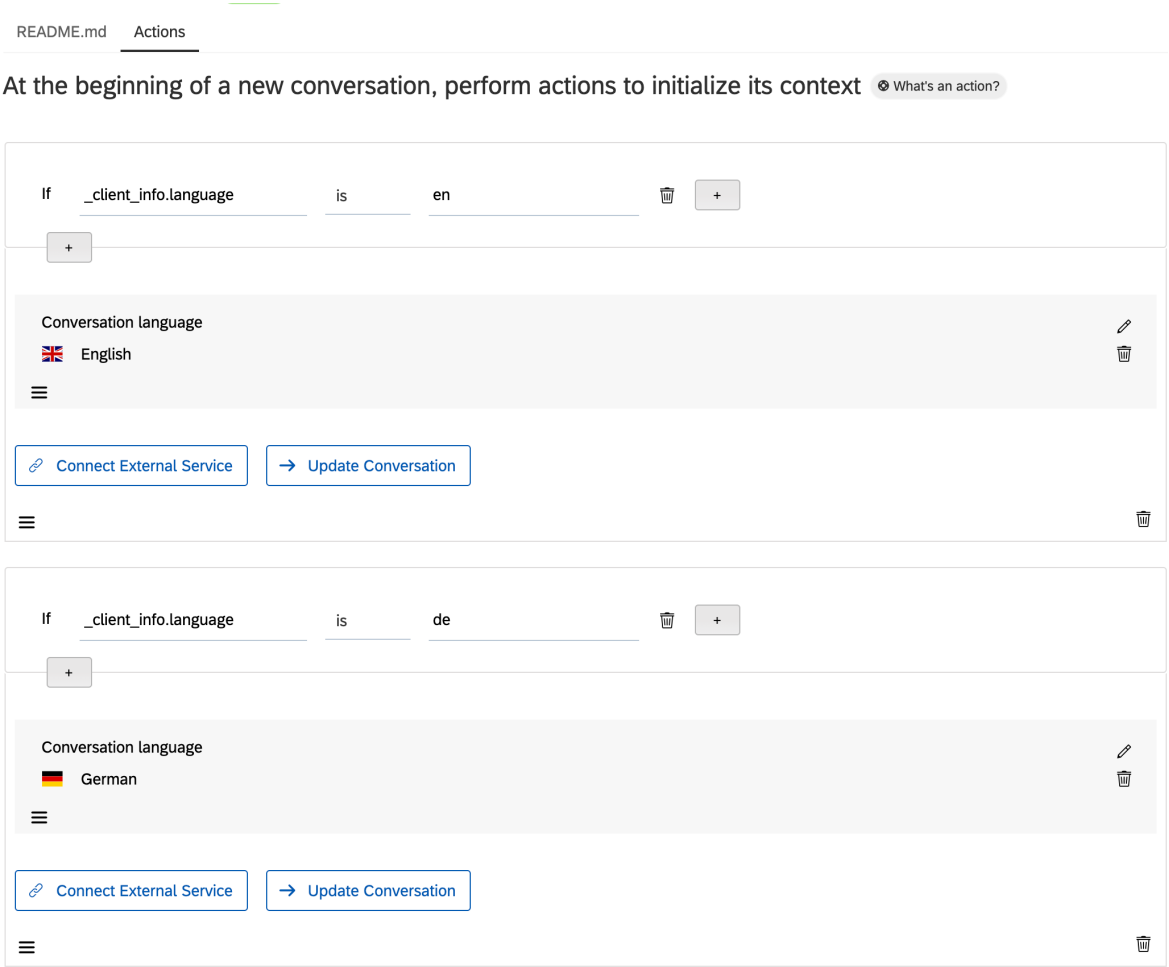
Web client determines and passes the language as part of the client information (`client_info` parameter). For more information see Messaging Channels [page 181].

The language detection for web client depends on the your user settings or OS or browser settings instead of NLP.

- For a bot integrated with SAP S/4HANA and SAP S/4HANA Cloud, the language setting will be the language configured in your user's profile.
- For a bot integrated into a non-UI5 application, the conversation is set to the language defined in your user's OS or browser settings.

As a bot developer, you can access the language setting passed by SAP Conversational AI Web Client and initialize your bot's conversation in this language. You can do this by defining the Initialize Skill [page 94] and using the Change Language [page 108] action. When the user enters the first expression in the conversation, the initialize skill is executed. This intializes the conversation language while skipping the natural language processing (NLP) step. As a result, the bot responds in the language that is defined in the *Change Language* action.

> **i Note**
>
> For a bot integrated with SAP S/4HANA and SAP S/4HANA Cloud, if there is a change in the browser language or user profile language, make sure that you refresh the browser so that the language setting is updated and is available at runtime in the `client_info` parameter.



You can also override the language setting passed in `client_info` parameter or enforce a conversation language using the Web Client Bridge API. For more information, see Development [page 195].

For more information, see the following video.

5.1.5.7 Development

The SAP Conversational AI Web Client offers a public JavaScript API, which is available at runtime in your Web application as soon as the SAP Conversational AI Web Client is loaded.

This API is available at the JavaScript object path: `window.sap.cai.webclient`

The API has the following methods:

```
/**
 * Opens the web client
 */
- show()

/**
 * Hides the web client
 */
- hide()

/**
 * Toggles show/hide of the Web Client. This can be attached to a button in the
application shell.
 */
- toggle()

/**
 * Sets the theme of the WebClient
 *
 * @param {string} themeName The name of the theme to be set
 */
- setTheme(themeName)

/**
 * Sends the given message to the webchat to post
 *
 * @param {string} message the string message to send to the webchat
 */
- sendMessage(message)
/**
 * Overwriting the language setting passed in client_info and optionally set it
 * as the conversation language of the bot.
 *
 * @param {string} message the string message to send to the webchat
 */
- getUserLanguage(determinedLanguage)
```

SAP Conversational AI Web Client Bridge API

In certain scenarios, when the hosting page uses the public API to interact with the `webclient` object, the `webclientBridge` may be called by the `webclient` during specific steps at runtime.

This bridge API implementation must be available before the `webclient` is loaded and can be used for the following purposes:

- Overwriting specific channel properties
- Providing context-specific information before any message is sent to the bot
- Overwriting the language setting passed in `client_info` parameter and optionally set this as the conversation language for the bot

The `webclient` expects to find these functions in a JavaScript object available at `window.sapcai.webclientBridge`.

getApplicationId

The `applicationId`, as defined when the SAP Conversation AI Web Client channel is created, can be set in the dataset of the bootstrap script tag, as in the following example:

Example: script

```
<script src="tenant_url/resources/public/webclient/bootstrap.js" data-application-id="<bot_owner>-<applicationId>" ></script>
```

`getApplicationId()` function in the bridge API. Alternatively, it can be set by implementing the

Example: getApplicationId

```
const getApplicationId = () => {
    return "bot_owner-applicationId"
}
```

getChannelPreferences

When the SAP Conversational AI Web Client starts using a specific channel, it first fetches the preferences as defined in the channel.

If implemented in the `webclientBridge`, the `getChannelPreferences()` function will be called. This function expects a JSON object or a promise resolving to a JSON object.

The properties returned by the `getChannelPreferences()` function will then overwrite the channel properties.

Example: getChannelPreferences

```
function getChannelPreferences() {
    return {
        "accentColor": "#123456",
        "headerTitle": "My very own Bot"
    }
}
```

The supported properties are as follows:

Key	Type	Description
accentColor	CSS Color	The accent color of the web client
botMessageBackgroundColor	CSS Color	The background of a bot reply
botMessageColor	CSS Color	The font color of a bot reply
complementaryColor	CSS Color	The secondary accent color
backgroundColor	CSS Color	The background color of the web client
headerTitle	String	The title of the web client
userInputPlaceholder	String	The input placeholder

Key	Type	Description
botPicture	URL	The picture displayed next to the bot replies
userPicture	URL	The picture displayed next to the user message
welcomeMessage	String	A welcome message displayed on first load

getMemory

Before any message is sent to the bot, the `webclient` will try to call the `getMemory()` function from the bridge API, if it exists.

This allows the hosting page to provide context information to the bot. This function expects a JSON object or a promise resolving to a JSON object.

Key	Required	Description
memory	Required	An object like `{ "firstName": "John", "lastName":"Doe" }`
merge	Optional	A boolean: If set to true, the payload is merged with the existing memory, overriding common keys, but keeping the ones absent from the payload. If set to false or missing, the memory is replaced entirely by your payload.

The `webclient` expects the `memory` object to be returned in less than 10 seconds. Otherwise, it will send the message without the `memory` object.

onMessage

The `webclient` will render any replies from the bot in the `webclient` window. However, you may also want to take action based on the returned message. This is implemented using the `onMessage` function.

Each time the `webclient` receives a reply from the bot, it will call the `onMessage()` function from the `webclientBridge`, if the function is implemented.

This will pass the `channelId` as an argument and the message in a JSON format

onMessage Definition

```
/**
 * Execute some business logic once a message is received
 * @param {array} aMessages a JSON array of messages
 */
function onMessage(aMessages)
```

The `onMessage` function is used with the client data message type, which you select on the *BUILD* tab under *Actions* > *ADD CONDITION to trigger messages* > *SEND MESSAGE*. Messages of this type are not

displayed on screen. Instead, they return additional data to the web client in JSON format. The web application then implements the `onMessage` function of the bridge API and uses the client data content as context or to trigger specified actions.

Example: Client Data for Showing an Alert

```
window.sapcai.webclientBridge.onMessage = (payload) => {
  payload.messages.forEach(function(message) {
    if (message.attachment.type === 'client_data') {
      alert(['Received custom json content',
            JSON.stringify(message.attachment.content, null, 2)])
    }
  })
}
```

getUserLanguage

If implemented in the `webclientBridge`, the `getUserLanguage()` function will be called. This function expects a JSON object or a string.

Example 1: Overwrite the Language Passed in client_info with 'FR'

```
const getUserLanguage = (determinedLanguage) => {
      return `FR`;
}
```

Example 2: Overwrite the Language Passed in client_info with 'FR' and Set It As the Conversation Language

```
const getUserLanguage = (determinedLanguage) => {
      return {
          language: `FR`,
    forceLanguage: true
      };
}
```

> **i Note**
>
> If the language is set to a different value in the initialize skill using the *Change Language* action, it is overriden with the value set using the Web Client Bridge API, as described in the above example.

Example 3: Use the Language Determined by Web Client and Set It As the Conversation Language

```
const getUserLanguage = (determinedLanguage) => {
      return {
          language: determinedLanguage,
    forceLanguage: true
      };
}
```

Related Information

Messages [page 114]
Send Rich Messages [page 215]

5.1.5.8 Constraints

SAP Conversational AI Web Client is the front-end component to chat with your SAP Conversational AI-based bots. While it is very close to the SAP CoPilot digital assistant, it does not support all the features offered by the latter.

Here is a list of the features that are not yet supported by the SAP Conversational AI Web Client:

- In-place navigation (such as *Open in App* button on a card in SAP CoPilot)
- Human to human chats
- Header and View More for the *List* view
- Paging, search, and View More for the buttons
- Voice input using the microphone
- Date picker
- Contact card
- Detailed view for the Object card or List items
- Attach and share objects in chat
- Help button
- Clear conversation option
- Drag and dock the digital assistant within the SAP Fiori launchpad

5.1.5.9 UX Recommendations for Web Client

SAP Conversational AI Web Client supports all the Message Types [page 120]. While using the *Custom* message type, Web Client renders elements according to the following UI guidelines.

Message Type	UI Element Restrictions	Character Length Restrictions	Recommendations
Text	-	640 characters	
Card	Displays a maximum of: • 1 section • 5 content pairs or attributes* • 3 buttons	• Title Displays up to 3 lines of available space, maximum 640 characters • Sub-title Displays up to 2 lines of available space, maximum 640 characters	
Buttons	Displays a maximum of 15 buttons 5 buttons are displayed initially, next 10 buttons are displayed on the click of *View More*	• Message: 640 characters • Title: 80 characters	

Message Type	UI Element Restrictions	Character Length Restrictions	Recommendations
Quick Replies	Displays a maximum of 11 buttons	• Message: 640 characters • Title: 80 characters	
Carousel	For up to 8 cards, the paging indicators are displayed as dots. Dots are replaced with numerical indicators in the format *1 of n* for more than 8 cards.	• Title Displays up to 3 lines of available space, maximum 640 characters • Sub-title Displays up to 2 lines of available space, maximum 640 characters • Details Displays a maximum of 1 section and 5 content pairs or attributes*	We recommend having a maximum of 12 cards in a carousel
List	Displays a maximum of 12 list items 6 items are displayed initially, next 6 items are displayed on the click of *View More*	• Title Displays up to 3 lines of available space, maximum 640 characters • Sub-title Displays up to 2 lines of available space, maximum 640 characters • Details Displays a maximum of 1 section and 5 content pairs or attributes*	

*Content pairs or attributes are currently only supported while scripting using the type *Card*, *List* or *Carosuel* under the *Custom* message tile.

5.1.6 SAP CoPilot

You can connect your bot to SAP CoPilot that can interact with your end users.

Please follow the steps to configure the channel as provided under the *Connect* tab.

The following table lists the Message Types [page 120] that are supported:

Message Type	Supported
Text	Yes

Message Type	Supported
Card	Yes
Buttons	Yes
Quick Replies	Yes
Carousel	No
List	Yes
Image/GIF	No
Custom	No
Voice	No
Video	No

5.1.7 SAP Jam Collaboration

You can connect your bot toSAP Jam Collaboration that can interact with your end users.

To connect your chatbot to SAP Jam Collaboration, you need to be an SAP Jam administrator. For information about SAP Jam Collaboration, please see SAP Jam Collaboration on SAP Help Portal.

Please follow the steps to configure the channel as provided under the *Connect* tab.

The following table lists the Message Types [page 120] that are supported:

Message Type	Supported	Details
Text	Yes	
Card	Yes	
Buttons	Yes	Only the button type *postback* is supported. The button types *web_url* and *phone number* are not supported. If the button types *web_url* and *phone number* are used, the buttons will behave like *postback* buttons.
Quick Replies	Yes	
Carousel	Yes	
List	Yes	

Message Type	Supported	Details
Image/GIF	No	
Custom	No	
Voice	No	
Video	No	

5.1.8 Slack

You can connect your bot to Slack that can interact with your end users.

Please follow the steps to configure the channel as provided under the *Connect* tab.

The following table lists the Message Types [page 120] that are supported:

Message Type	Supported
Text	Yes
Card	Yes
Buttons	Yes
Quick Replies	Yes
Carousel	No
List	Yes
Image/GIF	Yes
Custom Only the *Custom* option within the *Custom* tile is supported. See Blocks for Slack.	Yes
Voice	No
Video	No

5.1.9 Telegram

You can connect your bot to Facebook Messenger that can interact with your end users.

Please follow the steps to configure the channel as provided under the *Connect* tab.

The following table lists the Message Types [page 120] that are supported:

Message Type	Supported
Text	Yes
Card	No
Buttons	No
Quick Replies	No
Carousel	No
List	No
Image	Yes
Custom	No
Voice	No
Video	No

5.1.10 Twilio

You can connect your bot to Twilio that can interact with your end users.

Please follow the steps to configure the channel as provided under the *Connect* tab.

The following table lists the Message Types [page 120] that are supported:

Message Type	Supported
Text	Yes
Card	No
Buttons	No
Quick Replies	No
Carousel	No
List	No
Image/GIF	Yes
Custom	No
Voice	No

Message Type	Supported
Video	No

5.1.11 Twitter

You can connect your bot to Facebook Messenger that can interact with your end users.

Please follow the steps to configure the channel as provided under the *Connect* tab.

The following table lists the Message Types [page 120] that are supported:

Message Type	Supported
Text	Yes
Card	No
Buttons	No
Quick Replies	Yes
Carousel	No
List	No
Image/GIF	No
Custom Only the *Custom* option within the *Custom* tile is supported	Yes
Voice	No
Video	No

5.1.12 Fallback Channels

You can configure fallback channels to let you redirect the conversation to a human agent. First, you need to connect the fallback channel where you want SAP Conversational AI to redirect the message. You can do this on the*Connect* tab by selecting a fallback channel and following the instructions. After connecting the fallback channel, remember to activate it by checking the input.

Two fallback channels can be configured:

- Intercom
- Sinch Contact Center

When the fallback action is triggered, the bot doesn't reply, but instead sends the conversation history to your support channel, where a human agent writes a reply that is redirected to the user. When the human agent closes the ticket or conversation in your support center, the bot is able to talk to the user again.

> **i Note**
>
> If you are using the *Sinch Contact Center*, you have the option to select:
>
> - *Send bot memory to fallback channel* to pass the bot memory to this channel. Please be aware that this might include personal or sensitive data, so it is recommended that you review the GDPR compliance for Sinch and decide which information should be added in the memory upon enabling this option.
> - *Send user info to fallback channel* to pass the user information (user name and email address) to this channel so that the user can be identified. Please note that user information is personal data, so it is recommended to review the GDPR compliance for Sinch before enabling this option. For more information, see User Information [page 181].

5.1.13 Webchat

The Webchat channel is developed by the SAP Conversational AI team and is an open-source project on GitHub. You can use the default version of the webchat that we provide in the platform or customize the open-source version by forking it and deploying it on your side.

Mutiple browsers are supported by Webchat. For more information, see Browser Support [page 7].

Supported Message Types

You can connect your bot to Webchat that can interact with your end users.

Please follow the steps to configure the channel as provided under the *Connect* tab.

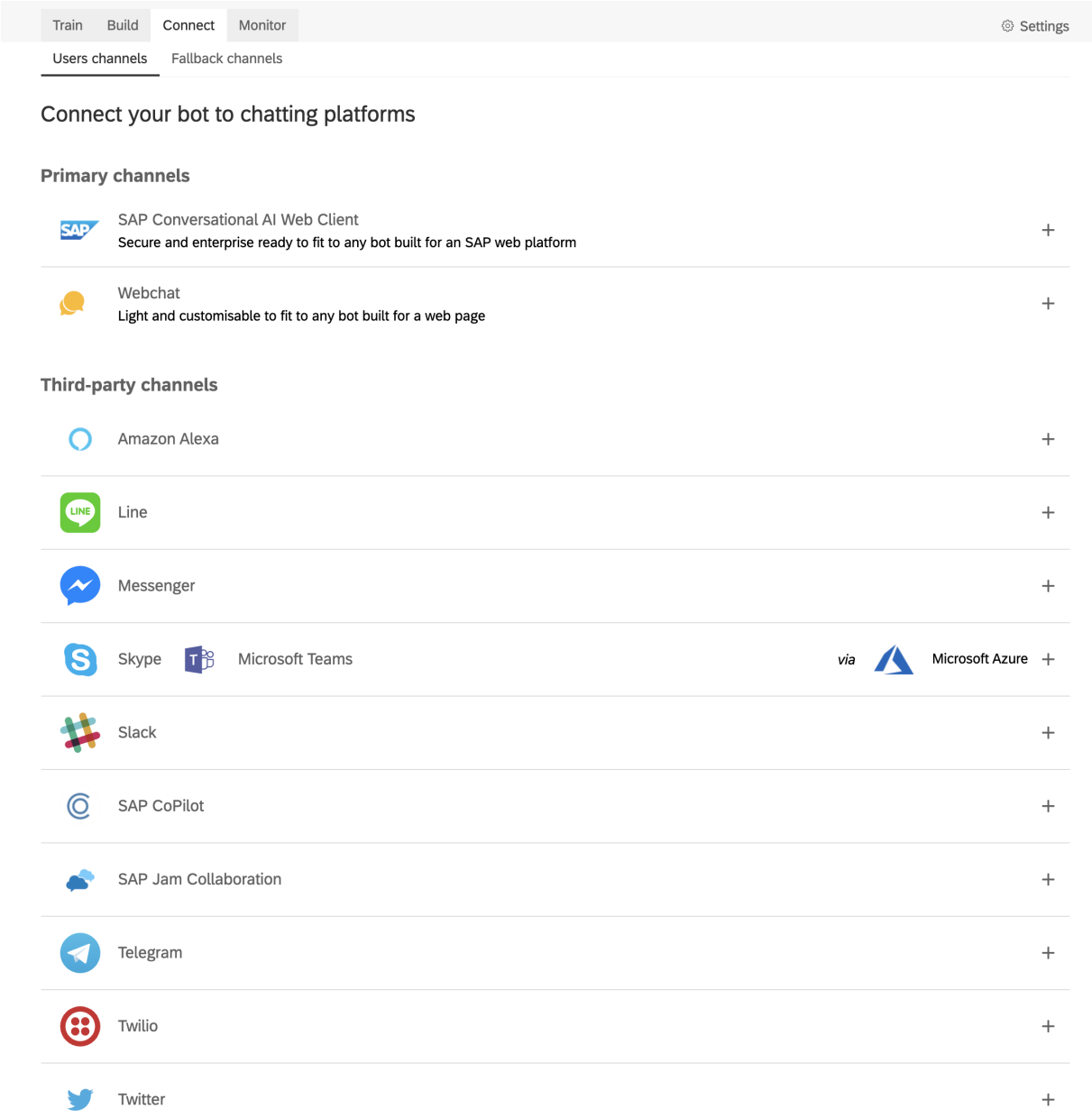
The following table lists the Message Types [page 120] that are supported:

Message Type	Supported	Details
Text	Yes	
Card	Yes	
Buttons	Yes	A maximum of 3 buttons are displayed. SAP Conversational AI Webchat supports the button type Skill Trigger [page 121]. Therefore, the usage of Disambiguation Skill [page 90] is also supported.
Quick Replies	Yes	A maximum of 11 quick replies are displayed.

Message Type	Supported	Details
Carousel	Yes	We recommend having a maximum of 12 cards in a carousel
List	Yes	We recommend having a maximum of 12 list items
Image/GIF	Yes	
Custom Only the *Custom* option within the *Custom* tile is supported	Yes	
Voice	No	
Video	No	

Default Hosted Version

On the *Connect* tab of your bot, activate the *Webchat* channel.



How to Use It

1. Configure details like color, title of the button, bot picture, user picture, and so on. If you wish, you can restrict messages from users to 512 characters or less. For example, you may want to do this if users tend to add a lot of details that obscure the intent of the request.
2. Add the following script to your web page to get the Webchat.

```
<script src="https://cdn.cai.tools.sap/webchat/webchat.js"
channelId="CHANNEL_ID"
token="TOKEN_ID"
id="cai-webchat"
></script>
```

> **Note**
>
> You can find `CHANNEL_ID` and `TOKEN_ID` when creating a Webchat channel in the Bot Connector. The `CHANNEL_ID` and `TOKEN_ID` uniquely identify the channel your bot is connected to. This is the only information needed to connect your bot to the channel (in non-authenticated scenarios). We do not recommend adding the script directly to your web page, where these tokens could be discovered and copied.

Bot Memory Management

You might want to send custom data from your website to the bot, like the name of the logged-in user, their ID, and the page they are currently viewing (for example, to send product suggestions). To do that, you can define a `window.webchatMethods.getMemory` function. The webchat will call this function before sending user messages. It will then send your arbitrary payload along with the message to the bot.

If you use the Bot Builder (which we highly recommend!), your payload is put in the conversation memory. This enables you to access this data in the Bot Builder. Let's say you send this as the payload: `{ "userName": "Dominik", "userId": 123456 }`. You can then send this as a greeting message: `Hello {{ memory.userName }}! How do you do?`

`window.webchatMethods.getMemory` is called with the parameter `conversationId` and must return a JSON object or a promise resolving to a JSON object:

> **Sample Code**

```
{
  "memory": { "userName": "Dominik" },
  "merge": true
}
```

Key	Required	Value
memory	Required	An object like `{ "userName": "Dominik" }`
merge	Optional	A boolean: If set to true, the payload is merged with the existing memory, overriding common keys, but keeping the ones absent from the payload. If set to false or if missing, the memory is replaced entirely by your payload.

If your `getMemory` function takes more than 10 seconds, the message is sent anyway, without waiting for your function to finish.

> **Example**
>
> Here's a simple example.
>
> > **Sample Code**

```
<html>
  <head>
    <script>
      window.webchatMethods = {
        // called at each user message
```

```
        getMemory: (conversationId) => {
          const memory = { userName: 'Dominik Bousquet', userId: 123456 }
          return { memory, merge: true }
        }
      }
    </script>
  </head>
  <body>
    <script src="https://cdn.cai.tools.sap/webchat/webchat.js"
      channelId="<channelId>"
      token="<token>"
      id="cai-webchat"
    ></script>
  </body>
</html>
```

Example

Here's an example to retrieve the user information from the cookie and page URL.

Sample Code

```
window.webchatMethods = {
  getMemory: (conversationId) => {
    const getCookie = (name) => {
      const value = document.cookie.match('(^|;) ?' + name + '=([^;]*)(;|
$)')
      return value ? value[2] : null
    }
    const userName = getCookie('userName')
    const memory = { userName, currentUrl: window.location.href }
    return { memory, merge: true }
  }
}
```

Example

Here's an example to retrieve the user information from an API call.

Sample Code

```
window.webchatData = {}
window.webchatMethods = {
  getMemory: (conversationId) => {
    if (window.webchatData.savedUserData) {
      return { memory: window.webchatData.savedUserData, merge: true }
    }
    return new Promise((resolve, reject) => {
      axios.get('/current_user')
        .then((response) => {
          const memory = { userName: response.data.name, userId:
response.data.id }
          window.webchatData.savedUserData = memory
          resolve({ memory, merge: true })
        })
        .catch(reject)
    })
  }
}
```

Example

Here's an example with the page URL information and reset memory information.

Sample Code

```
window.webchatData = {}
window.webchatMethods = {
  getMemory: (conversationId) => {
    if (!window.webchatData.oriUrl) {
      window.webchatData.oriUrl = window.location.href
    }
    // merge: false - reset the conversation if the user
    // switched to another page since the first message
    if (window.webchatData.oriUrl !== window.location.href) {
      return { memory: {}, merge: false }
    }
    return { memory: { userName: 'Dominik' }, merge: true }
  }
}
```

Open-Source Version

If you want to customize the style or add new functionalities that don't exist in the default hosted version, you can fork the open-source version on GitHub at SAPConversationalAI/Webchat .

How to Use It

Please see the README.md. The open-source version is developed in ReactJS.

5.2 Getting Started with the Bot Connector

Note

Open Source Bot Connector has been archived and there will be no further enhancements or maintenance of the repository by SAP Conversational AI. Please migrate to the bot connector available on our bot building platform (hosted in on SAP Business Technology Platform), which offers integration with a wide range of channels, governed by secure data processing as per SAP product standards.

Whenever a message is posted on one of the channels to which your bot is connected, it receives a POST request at the endpoint that you've set in the platform. To reply, you need to make a POST request with your bot's request token (which you can find in your bot settings). In the following example, we use SDKs to make it simpler.

Receive Messages and Send Hello World

1. Copy and paste this snippet to a file.
2. Replace `YOUR_REQUEST_TOKEN` with your bot's request token.
3. Install the dependencies by running the following command:
   - JavaScript (JS): `npm install sapcai express body-parser`
   - PHP: `composer require sapcai/sdk-php`
   - Python: `pip install sapcai flask`
   - Ruby: `gem install Sapcai sinatra`

JS

Sample Code

```
var express = require('express')
var bodyParser = require('body-parser')
var sapcai = require('sapcai').default
var connect = new sapcai.connect('YOUR_REQUEST_TOKEN')
var app = express()
/* Server setup */
app.set('port', 5000)
app.use(bodyParser.json())
app.post('/', function(req, res) {
  connect.handleMessage(req, res, onMessage)
})
function onMessage (message) {
  // Get the content of the message
  var content = message.content
  // Get the type of the message (text, picture,...)
  var type = message.type
  // Add a reply, and send it
  message.addReply([{ type: 'text', content: 'Hello, world' }])
  message.reply()
}
app.listen(app.get('port'), function () { console.log('App is listening on
port ' + app.get('port')) })
```

PHP

Sample Code

```
<?php
use Sapcai\Client;
// Start Slim server
$app = new \Slim\App();
// Instantiate the Connect Client
$connect = Client::Connect($_ENV["YOUR_REQUEST_TOKEN"]);
//Handle / route
$app->post('/', function ($request, $response) {
  $connect->handleMessage($body, 'replyMessage');
```

```
});
function replyMessage ($message) {
  // Get the content of the message
  $text = $message->content;
  // Get the type of the message (text, picture,...)
  $type = $message->type;
  $message->addReply([(object)['type' => 'text', 'content' => 'Hello,
world']]);
  $message->reply();
}
// Run Slim server
$app->run();
```

Python

Sample Code

```
from sapcai import Connect
from flask import Flask, request, jsonify
connect = Connect('YOUR_REQUEST_TOKEN')
def bot(request):
  message = connect.parse_message(request)
  # Get the content of the message
  content = message.content
  # Get the type of the message (text, picture,...)
  type = message.type
  # Add a reply, and send it
  replies = [{type: 'text', content: 'Hello, world'}]
  connect.send_message(replies, message.conversation_id)
  return jsonify(status=200)
app = Flask(__name__)
@app.route('/', methods=['POST'])
def root():
  return bot(request)
app.run(port='5000')
```

Ruby

Sample Code

```
require 'sinatra'
require 'sapcai'
connect = Sapcai::Connect.new('YOUR_REQUEST_TOKEN')
set :port, 5000
post '/' do
  connect.handle_message(request) do |message|
    # Get the content of the message
    content = message.content
    # Get the type of the message (text, picture,...)
    type = message.type
    # Add a reply, and send it
    replies = [{type: 'text', content: 'Hello, world'}]
    connect.send_message(replies, message.conversation_id)
  end
```

```
end
```

5.3 Receive Messages

When you receive messages from the Bot Connector, the body of the request contains useful information that your bot can leverage to reply to the sender.

Format

Your bot receives the message in the same format, irrespective of the channel. The content of the message is as follows:

Sample Code

```
{
  "chatId": "XXXXXX"
  "senderId": "XXXXXXX",
  "mentioned": true,
  "origin": "XXXX",
  "message": {
    "participant": "XXXXXX",
    "conversation": "XXXXXX",
    "receivedAt": "XXXXXX",
    "attachment": {
      "type": "text",
      "content": "Hello, world!",
    },
  },
}
```

Attributes

Below is the attribute of the payload that your bot receives:

Key	Value
chatId	[String] The channel's native ID of the chat
senderId	[String] The channel's native ID of the sender
mentioned	[Boolean] Whether the bot is mentioned or not

Key	Value
origin	[String] The origin of the message ("messenger", "slack", etc.)
message	[Object] The message itself

Below is the exact content of the message itself:

Key	Value
participant	[String] The ID of the participant in the Bot Connector
conversation	[String] The ID of the conversation in the Bot Connector
receivedAt	[String] The date when you received the message
attachment	[Object] An object containing the type and content of the message

5.4 Send Rich Messages

To send a message, you need to make a `POST` request to /messages API using an appropriate token [page 262] (which you can find in your bot settings) and send a specific payload for each message type.

Button Types

Some of the message types support Buttons [page 121]. The value for this in the payload is referred to as `BUTTON_TYPE`. Based on the button type you choose, you can replace `BUTTON_TYPE` with any of the following values:

Button Type	Attribute Value	Description
Postback	postback	This is the basic type. When this button is tapped, the value is sent as a normal incoming message.
Link	web_url	Depending on the channel, when this button is tapped, the URL in the value field is loaded.

Button Type	Attribute Value	Description
Phone Number	phonenumber	Depending on the channel, when this button is tapped, the phone number in the value field is called.

Delay

You can also set an optional delay of between 0 and 5 seconds for each of your messages. This delay is applied before sending the next message.

If you don't set a delay, the messages are sent consecutively in the order you specified, with no wait time.

Text Message

Sample Code

```
{
  "type": "<text>",
  "markdown": true,
  "delay": 2,
  "content": "<MY_TEXT>"
}
```

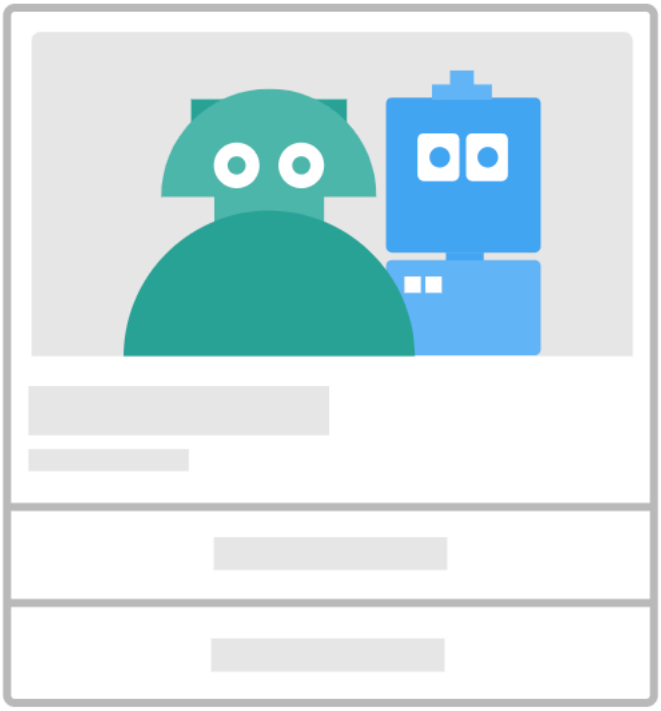
Quick Replies

Sample Code

```
{
```

```
    "type": "quickReplies",
    "markdown": true,
    "content": {
      "title": "<TITLE>",
      "buttons": [
        {
          "title": "<BUTTON_TITLE>",
          "value": "<BUTTON_VALUE>"
        }
      ]
    }
  }
```

Cards



Sample Code

```
  {
    "type": "card",
    "content": {
      "title": "<CARD_TITLE>",
      "subtitle": "<CARD_SUBTITLE>",
      "imageUrl": "<IMAGE_URL>",
      "buttons": [
        {
          "title": "<BUTTON_TITLE>",
          "type": "<BUTTON_TYPE>",
          "value": "<BUTTON_VALUE>"
        }
      ]
```

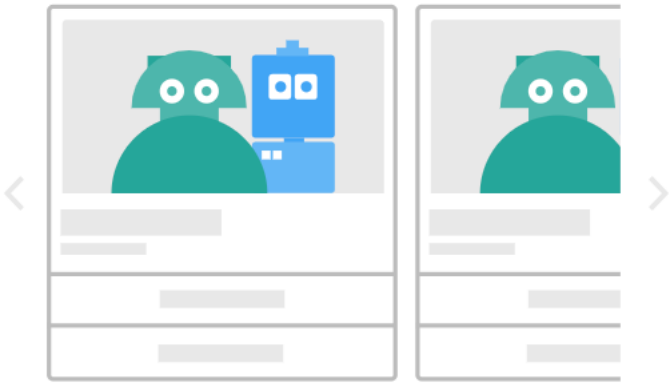
```
    }
  }
```

Buttons

Sample Code

```
{
  "type": "buttons",
  "content": {
    "title": "<BUTTON_TITLE>",
    "buttons": [
      {
        "title": "<BUTTON_TITLE>",
        "type": "<BUTTON_TYPE>",
        "value": "<BUTTON_VALUE>"
      }
    ]
  }
}
```

Carousel

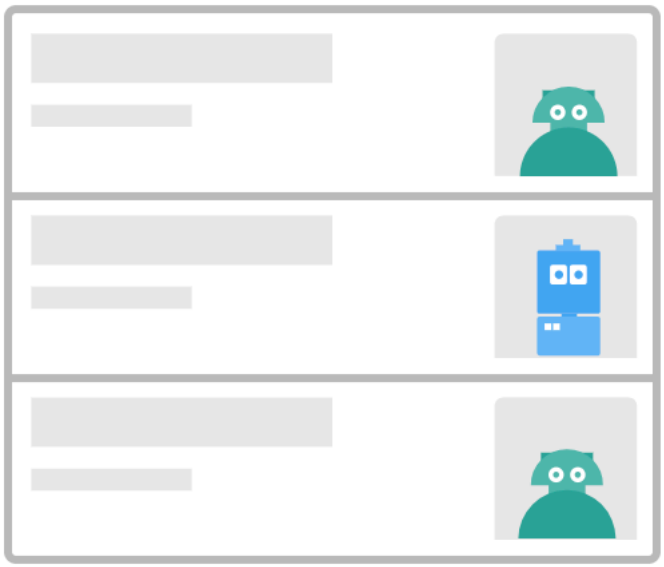


> **Note**
>
> The buttons array is a required attribute in the carousel JSON. If you do not wish to add buttons in the carousel, you can keep the buttons array empty.

> **Sample Code**

```
{
  "type": "carousel",
  "content": [
    {
      "title": "<CARD_1_TITLE>",
      "subtitle": "<CARD_1_SUBTITLE>",
      "imageUrl": "<IMAGE_URL>",
      "buttons": [
        {
          "title": "<BUTTON_1_TITLE>",
          "type": "<BUTTON_1_TYPE>",
          "value": "<BUTTON_1_VALUE>"
        }
      ]
    }
  ]
}
```

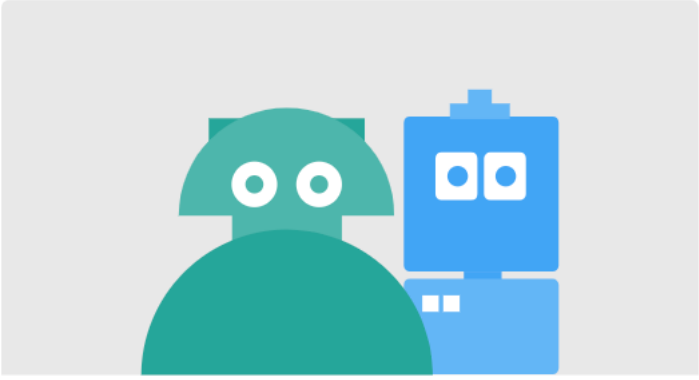
List



Sample Code

```
{
  "type": "list",
  "content": {
    "elements": [
      {
        "title": "<ELEM_1_TITLE>",
        "imageUrl": "<IMAGE_URL>",
        "subtitle": "<ELEM_1_SUBTITLE>",
        "buttons": [
          {
            "title": "<BUTTON_1_TITLE>",
            "type": "<BUTTON_TYPE>",
            "value": "<BUTTON_1_VALUE>"
          }
        ]
      }
    ],
    "buttons": [
      {
        "title": "<BUTTON_1_TITLE>",
        "type": "<BUTTON_TYPE>",
        "value": "<BUTTON_1_VALUE>"
      }
    ]
  }
}
```

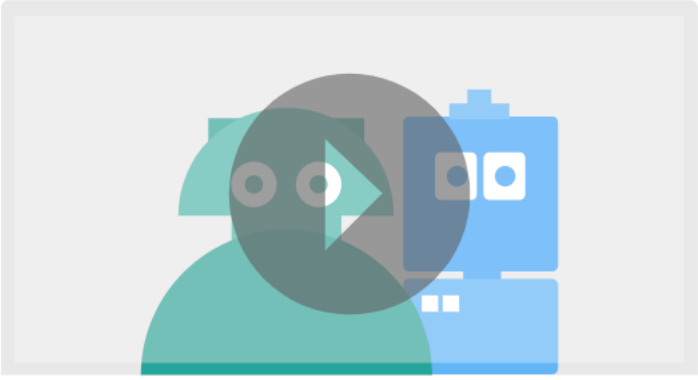
Picture



Sample Code

```
{
  "type": "picture",
  "content": "<IMAGE_URL>",
}
```

Video



Sample Code

```
{
  "type": "video",
  "content": "<VIDEO_URL>",
}
```

Note

Currently videos can only be added via *Custom* action or webhook.

Custom

The *Message Type Custom* (under Custom action tile) can be used to pass channel-specific JSON to the channel. It can be used either via the *Custom* action or webhook.

You can define custom messages for Facebook Messenger, Slack, Twitter and Webchat.

> **i Note**
>
> For message types *Card*, *Carousel* and *List*, you can define more UI attributes using *Custom* tile (through scripting) as compared to messages defined using the relevant tile. For example, attributes like `status`, `statusState`, `section` and so on can be only defined using the Custom message type and is only supported for SAP Conversational AI Web Client.

- Slack
  Blocks are not yet supported for Slack. Please refer to https://api.slack.com/messaging/composing/layouts .
- Facebook Messenger

  > **Sample Code**
  >
  > ```
  > {
  > "type": "custom",
  > "content": {
  > "attachment": {
  > "type": "template",
  > "payload": {
  > "template_type": "airline_update",
  > "intro_message": "Your flight is delayed",
  > "update_type": "delay",
  > "locale": "en_US",
  > "pnr_number": "CF23G2",
  > "update_flight_info": {
  > "flight_number": "KL123",
  > "departure_airport": {
  > "airport_code": "SFO",
  > "city": "San Francisco",
  > "terminal": "T4",
  > "gate": "G8"
  > },
  > "arrival_airport": {
  > "airport_code": "AMS",
  > "city": "Amsterdam",
  > "terminal": "T4",
  > "gate": "G8"
  > },
  > "flight_schedule": {
  > "boarding_time": "2020-02-26T10:30",
  > "departure_time": "2020-02-26T11:30",
  > "arrival_time": "2020-02-27T07:30"
  > }
  > }
  > }
  > }
  > }
  > }
  > ```

  Please refer to https://developers.facebook.com/docs/messenger-platform/reference/templates
- Twitter

Please refer to https://developer.twitter.com/en/docs/direct-messages/api-features

- Webchat
  In case you are using open-source Webchat, you can define a customized response by using *Custom* message type.

> **Note**
>
> You need to follow a specific format to define channel-specific custom bot responses. This format applies to all channels for which custom message type is supported. See format in image below.

```
{
  "type": "custom",
  "content": {
    {{!-- INSERT YOUR CHANNEL-SPECIFIC MESSAGE PAYLOAD HERE --}}
  }
}
```

Client Data

The payload content of client data messages can be anything that the `onMessage` function could reuse.

> **Sample Code**

```
{
  "type": "client_data",
  "content": {
    "elements": [{
      "key": "myKey",
      "value": {
        "key1": "value1",
        "key2": "value2"
      }
    }]
  }
}
```

Related Information

Messages [page 114]
Development [page 195]

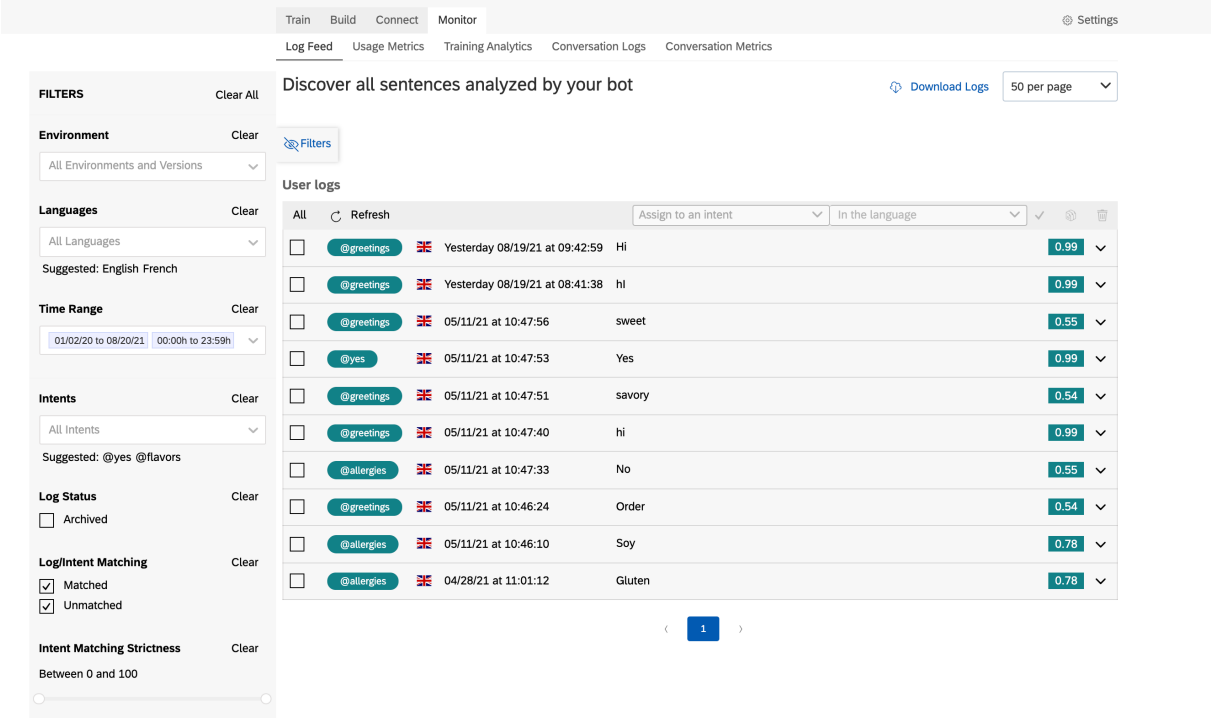
6 Monitoring and Analytics

6.1 Log Feed

The *Log Feed* shows all the conversations that users have with your chatbot and classifies them to one of your bot's intents.

A user can say the same thing in different ways. If your bot is unable to answer the user's question or retrieve an appropriate response, you can directly map that conversation to an intent from the log feed. This improves the performance and the bot can learn directly from what a user says.

> **i Note**
>
> In case you have disabled the user conversation data from being stored while creating the bot, no data is visible under *Log Feed*. This is applicable only for Webchat and SAP Conversational AI Web Client channels and is not applicable for third party channels.



1. Click the *Monitor* tab. A list of all the sentences that were analyzed by your bot is visible in the *Log Feed*.
2. You can apply multiple filters in the *FILTERS* panel on the left. The logs are displayed accordingly.

> **i Note**
>
> The language filter corresponds to the processing language and not detected language anymore. All the existing logs will be filtered based the detected language, however, all the new logs will correspond to the processing language.

3. In the left panel, select whether you want to view only the *Matched* logs or the *Unmatched* ones. You can further filter the logs that were matched to a specific intent.

   > **i Note**
   >
   > You cannot set the *Intent Matching Strictness* filter if the *Unmatched* filter is checked.

4. Set the *Intent Matching Strictness*. This indicates the range of confidence score between which your bot should match the conversation to one of your bot intents.
   Based on the filters applied, the logs are displayed.
5. Click a conversation.
6. Choose the bot version and select the intent to which this conversation should be mapped.
7. Click the blue check mark on the right.
   The conversation is mapped to the intent and is no longer available in the Log Feed.
   You can delete the conversations that are not relevant to be classified under any intent and archive the ones that are too old.
   For more information, refer to the tutorial Improve Your Chatbot Accuracy by Monitoring User Activity.

> **i Note**
>
> Duplicate logs are not visible in the *Log Feed* under the *Monitor* tab. They will be merged with the existing logs.

6.2 Usage Metrics

Are Usage Metrics Available for My Bot?

All metrics are extracted from the conversations that users have with your bot through the Bot Builder.

> **i Note**
>
> In case you have disabled the user conversation data from being stored while creating the bot, the data for entity values, enrichments, user utterances is not visible under *Usage Metrics*. This is applicable only for Webchat and SAP Conversational AI Web Client channels and is not applicable for third party channels.

How you're using the platform	Usage metrics
I'm using the Bot Builder and Bot Connector	Available
I'm using the Bot Builder directly through the API (with the `/dialog` endpoint)	Available
I'm using the NLP API only (with the `/request` endpoint)	Not available
I'm using the NLP API and Bot Connector	Not available

All metrics are filtered by one of the languages of your bot (except for some graphics, where indicated) and a time range that you can select. For quicker loading, the default time range is last week. To change the time range, click *SHOW FILTERS*. Also for better loading, the metrics are fetched asynchronously.

Metrics Type

Conversations

A conversation is a sequence of interactions between your bot and your users. When no new messages appear in the conversation for 15 minutes, we consider the conversation to be over. The conversation ID can be the same for a user who has 3 conversations with your bot. This happens when your bot is connected to Facebook Messenger. If a user has one long conversation with your bot, we split this long conversation into several parts to understand when the user starts a real new conversation with your bot.

Users

A user can have several conversations with a bot. Users are unique by channel. This means that if your bot is connected to two different channels, the same person is considered as user A in the first channel and as user B in the second channel.

Messages Received

All messages sent by your users are considered as messages received when the users type a sentence, but also when they click on a button or quick reply.

Average Messages by Conversation

Taking all conversations into account, this is the average number of messages received from the user in each conversation.

Most Used...

- Intents
  How many times intents are detected in all sentences. If multiple intents are detected, each intent is counted.
- Entities
  How many gold and custom entities are detected in all sentences.
- Skills
  How many times your skills have been triggered, that is, how many times a user enters a skill and follows the conversation flow, no matter how many interactions the user has in this skill. However, it doesn't mean that the user succeeds in reaching the end of the skill.

For more information, refer to the tutorial Improve Your Chatbot Accuracy by Monitoring User Activity.

6.3 Training Analytics

On the *Monitor* tab, the *Training Analytics* section helps you to build a great dataset for your bot. These analytics are only available for bots with at least 4 intents and at least 30 expressions per intent.

Your dataset (that is, all the intents and entities that you created and trained) is a fundamental element of your bot. If your bot isn't well-trained (meaning your dataset isn't well-structured or is incomplete), your bot won't be able to correctly understand messages from its users, resulting in a disappointing conversational experience.

Your Dataset Benchmark

At the top of the page, you can run a benchmark. It will trigger several processes to measure the performance of your dataset and give you insights on how to improve your intent classification and your custom entity detection.

A benchmark can take several hours, depending on the size of your bot.

You can only run one benchmark at a time for your bot.

For more accurate results, we use your bot training data and a validation file that you provide. Since we take your bot training data at a time *t*, and provide tips and insights based on this data, we advise you not to update your dataset during the benchmark; otherwise the insights will be less accurate.

Your Intent Classification Benchmark

Your Bot Training Data

All the expressions inside each intent are evaluated. This results in three metrics between 0 and 1 for each intent (precision, recall, and F1 score) and three global metrics for the entire dataset.

Note that the scores may differ slightly if you run the benchmark again on the same dataset.

Your Validation File

A validation file is composed of sentences with their corresponding intents. It reflects the reality, so it's important to build this file with real sentences that users actually sent to your bot.

Each sentence is tested with your training dataset, and we check if the first intent returned is the right one. Once the evaluation is done, we also get three metrics between 0 and 1 for each intent (precision, recall, and F1 score) and three global metric.

How Do I Create a Validation File?

For multilingual bots, please upload one file for each supported language.

File format

Your file must be a valid CSV file. It must end with .csv and the separator must be a semi-colon ";". If you want to include quotes in your expressions or intents, you must add two double quotation marks before and after the quoted word(s).

Sample Code

```
"intent";"expression"
    "greetings";"Hello"
    "greetings";"Hi"
```

```
"weather";"What's the weather in Paris?"
"translation";"What does ""Bonjour"" in French mean?"
```

Content

The goal of this file is to represent reality, that is, to show how users use your bot. Real user entries should include dedicated vocabulary, typos, and so on. The proportion to which each intent is present in your file should also reflect the way real users use your bot. Here are some guidelines:

- Try to represent almost all intents in your bot. It's okay if a few intents, far from the core of your bot's use case, are missing. However, at least 85% of the intents should be represented in the validation file.
- Provide many sentences for each intent. Don't choose some sentences over others.
- Check that all sentences in your file match an existing intent in your bot.
- Avoid duplicate sentences.

To ensure that your validation file reflects the way that people actually use your bot, we recommend creating your file as follows:

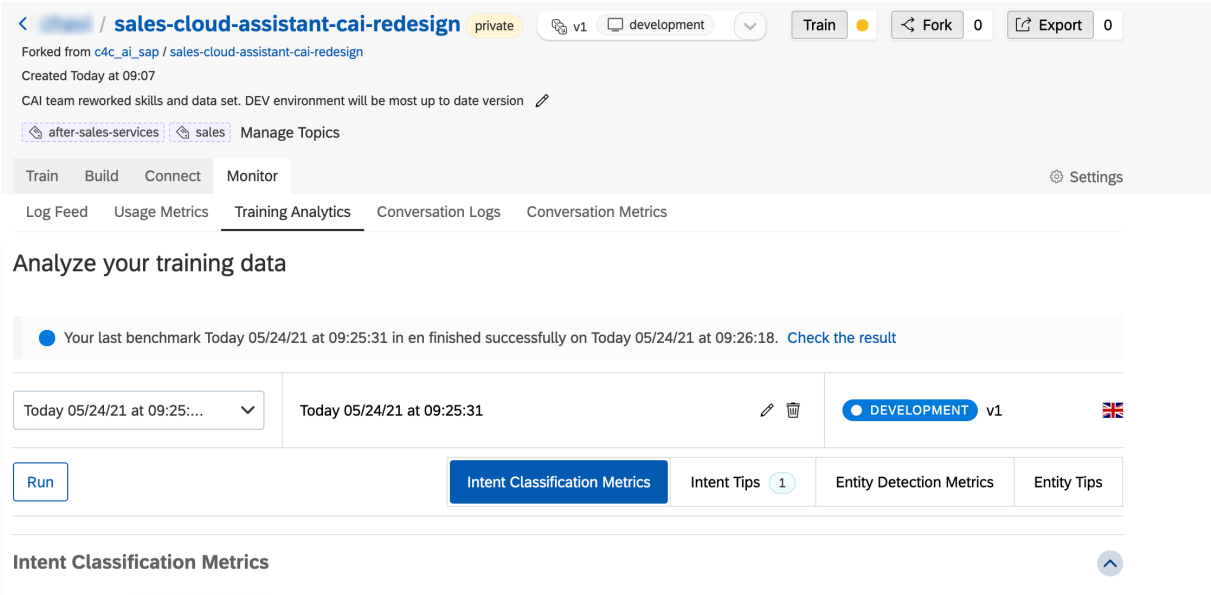
1. On the *Monitor* tab, go to *Log Feed* and filter matched **and** unmatched production logs from the past 1 to 3 months.
2. Export these logs by clicking *Merge duplicate logs on a single line*.
3. Check manually (yes, you need to be the final validator!) that each sentence matches the right intent.
4. Create the final validation file with these sentences and intents.

If your file doesn't include at least 85% of your intents, you need to pick sentences from your logs to complete your file and reach approximately 85%. You can do this as follows:

1. Go back to your *Log Feed* page and search for the specific intents that are missing.
2. Select between 3 and 10 sentences for each missing intent, and add these sentences to your validation file.

Final step

Upload your file to the platform. Before you launch training, make sure you name the training as shown in the following figure:



We'll analyze it and provide feedback. For example, we may suggest adding more sentences. You can still run a benchmark at any time; these guidelines are just suggestions.

Your Benchmark Scores

Classification metrics are used for both intent classification and entity detection. They are called *Precision*, *Recall* and *F1-score*, and are shown as values between 0 and 1, but they can be interpreted as percentages.

Precision

A metric that is calculated per intent. For each intent, it measures the proportion of correct predictions out of all of the times the intent was declared during the benchmark. It answers the question *Out of all the times my bot predicted this intent, how many times was it correct?*. Low precision usually signifies the relevant intent needs cleaning, which means removing sentences that don't belong to this intent.

For your bot users, a low precision means *The bot always thinks I'm talking about A, no matter what I say!*

Recall

A metric calculated per intent. For each intent, it measures the proportion of correct predictions out of all of the entries belonging to this intent. It answers the question *Out of all of the times my bot was supposed to detect this intent, how many times did it do so?*. Low recall usually signifies the relevant intent needs more training (for example, by adding more sentences to enrich the training).

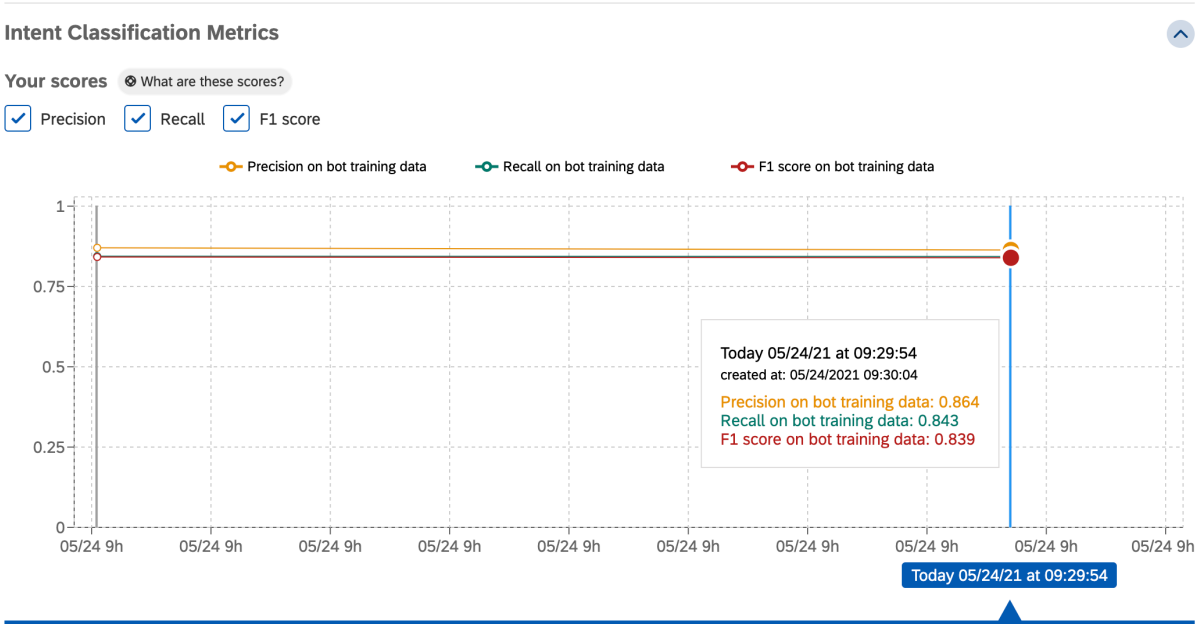
For your bot users, a low recall means *I can't get the bot to understand that I want to do B!*

F1 Score

The harmonic mean of precision and recall. It's a good indication of the performance of each intent, ranging from 0 (bad performance) to 1 (good performance). The F1 scores for each intent can be averaged to create a global indication for the performance of your bot.

For your bot users, a low F1 score means *This is completely useless!*

The following figure is an example of intent classification metrics.



- Precision (indicated with yellow) is the ratio of times you are right in trusting the prediction. If the precision of an intent is 0.6, you would be right to think it is indeed this intent around 60% of the times.
- Recall (indicated with green) is the ratio of times an intent has been correctly predicted. If the recall of an intent is 0.6, it means 60% of the sentences tagged as your intent have been correctly predicted.
- F1-score (indicated with red) is a "harmonic score". It is an appropriate average between the precision and recall. If the F1-score is 0.6, this means around 60% of both expectations and predictions of your intent would be good predictions.

You can see the evolution of your scores through time using the metrics graph. A dot represents one of the three classification scores for one specific benchmark, the score being between 0 and 1.

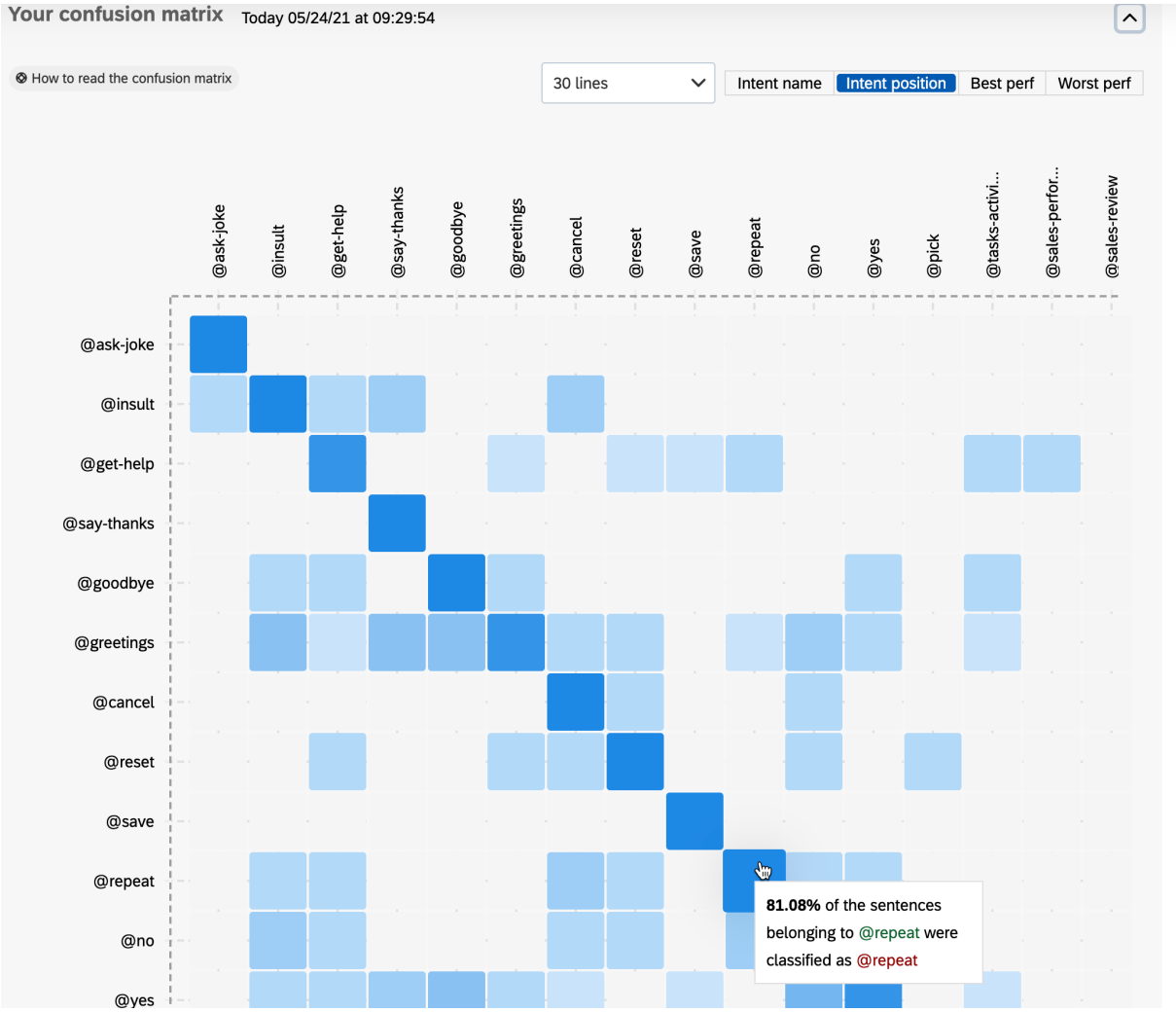
You can filter the metrics to focus on only one or two of them, and see the evolution of these through your different benchmarks. A good evolution is indicated by an increasing curve towards 1.

Your Intent Confusion Matrix

Your confusion matrix is used to gain further insight into intents that may clash and get confused. The element at the intersection of row A and column B signifies the percentage of sentences that should be classified as A, but are classified as B.

You can order the confusion matrix by intent name and by performance. If you don't have any problems between your intents, you should have a confusion matrix with a beautiful diagonal since 100% of expressions match the right intent, as expected.

Sort by F1 score or Ranking. Approach intent by intent; start with intent that is core to the business use case. Select the intent that has high ranking but low F1. Analyze the confusion matrix and overlap of intents (vertical and horizontal).



> → Tip
>
> When you click a single intent in your benchmark, the same line will be in focus in your confusion matrix.

For more information, refer to the tutorial Improve Your Chatbot Accuracy by Monitoring User Activity.

Tips to Improve Your Intent Classification

In addition to benchmark metrics, we provide step-by-step suggestions to improve your dataset in a more accurate way. If the suggestions are not meaningful for you, click *Next tips*.

Here's a prioritized list of the suggestions we may make.

- Remove expressions
  We can detect that a lot of testing examples of some intents are falsely predicted as another intent. Moreover, we check if the number of training examples of this intent is more than 50% larger than the median number of examples in your dataset (it is said to be unbalanced). As a result, the algorithm may learn to increase the importance and detection rate of this intent. To prevent that, we advise removing any misclassified examples.

- Avoid duplicates
  Machine learning algorithms are excellent at predicting the results of data that they encountered during the training step. Duplicates could end up in the training set and testing set, and abnormally improve the benchmark results.
- Add expressions
  We check if some intents have a low recall (see definition above). Since there is no balance problem in your dataset, our machine learning strategy is unable to capture the globality of the semantic complexity of this intent. You may be able to solve this by adding more training examples.
- Merge intents
  Two intents may be too close semantically to be efficiently distinguished. A significant part of the error of one intent is directed toward the second one, and vice versa. Merging them may help improve the bot's flow.
- Split intent
  If an intent has both low precision and low recall, while the recall scores of the other intents are acceptable, it may reflect a use case that is too broad semantically. Try splitting this intent into several intents.

Tips to Eliminate Intent Overlap and Confusion

- Look for words or phrases that are common across intents and analyze the number of times such words or phrases appear across overlapping intents. Rebalance as required.
- Ensure that a minimum 10% of all sentences within an intent have key words or phrases. For example: *Address* intent could have minimum 10% sentences with keywords such as *location* or *headquartered*.
- Edit sentences (if required) and move sentences from one intent to another as appropriate.
- Look for sentence structures and similar patterns across overlapping intents. Re-balance by adding and/or editing sentences. This is explained using the following example:

Example sentence	Where?	Machine interprets as
Show me all Talpa deals	in intent 1	Show `pronoun` (all) `object` (Talpa) `object` (deals)
Show me all Avantel contracts	in intent 2	Show `pronoun` (all) `object` (Avantel) `object` (contracts)

  First split the sentence into words and understand the root of each word. Then replace the name of each entity with sufficient training examples.

- If there is heavy overlap across multiple intents, group together all sentences from the intents. View it afresh and logically categorize into clusters.
  For example, group everything related to *customer* under Cluster A, everything related to *market information* under Cluster B and so on. Identify appropriate intents.
- Overlap across intents could also be caused by an imbalance in the number of sentences. Wherever possible, it is recommended to have similar number of sentences across intents.

Reality Check

This helps you to ensure that your training dataset represents reality as far as possible.

Before you can carry out a reality check, you must first upload a validation file (see *How do I create a validation file?* above).

Under *Make your dataset closer to reality*, choose an intent in the dropdown. You'll then get metrics about how close your training dataset is to reality (from your validation file), as well as suggestions to improve your dataset. For example, you can find the length of the expressions in your training dataset compared with the medium length of the sentences sent by your users. You can also find the most important words in your intent compared with the most important words when your users chat. If some words are missing in your dataset, or if some words are never used by your users, we provide tips to help you solve the issue.

Your Entity Detection Benchmark

We split the expressions inside each intent into two parts: 90% is used for training, 10% is used to evaluate the custom entity detection. The evaluation is simple: We detect each custom entity in each sentence, based on the knowledge we have from the training dataset. We check if each word has been properly detected as a custom entity or as a simple word. We repeat this process five times to enforce randomness in the splits. This results in three metrics between 0 and 1 for each entity (precision, recall, F1 score, ranking, and size) and three global metrics for the entire dataset.

Your Entity Detection Confusion Matrix

Your confusion matrix is used to gain further insight into entities that may clash and get confused. The element at the intersection of row A and column B signifies the percentage of entities that should be detected as A, but are detected as B.

> → Tip
>
> When you click a single entity in your benchmark, the same line will be in focus in your confusion matrix.

Tips to Improve Your Entity Detection

- Remove values
  Too many words are tagged as custom entities in your chatbot. Custom entities should be used and tagged on words only if you really need them to detect and retrieve key information from your users.
- Add different values
  You're using the same value too many times in this entity. This can be intentional if you want to check that something is present or not (and you don't need to detect several values). If this is the case, please ignore this tip. If not, you may want to either delete this entity because you're not really using it, or add different values.

- Remove mistagging errors
  A custom entity is always confused with another one. You may have a tagging issue. For example, some values may be tagged in both entities, or an entity is mistagged. If it's not a mistagging issue, the entities may be too similar; check whether you can merge them.

Tips to Improve Entity Performance

- Missing entity tagging should be fixed.
- Ensure consistency in usage of entity values. For example, *SAP CAI* versus *SAP Conversational AI* as product name.
- Avoid repetitive names. Train the bot with a variety of names or values.
- Analyze entities overlap in confusion matrix.
- Ensure that the precision is at a minimum of 90% and F1 score is 70% and above.

General Recommendations

- If required, create new bot version(s) to experiment with actions that improve performance and run benchmark. It is not necessary that every action or a change in the data set should result in improved performance. Certain actions that improve data set consolidation and robustness could actually result in a dip in performance. It is better to retain such actions that improve data set, although may be accompanied by loss of a few percentage points. Lesser confusion or overlap in confusion matrix and not F1 score alone, should determine whether the actions performed are right and to be retained.
- Continuously monitor the chatbot usage. Analyze log feed, tag entities and annotate sentences to the right intents to continually improve data set.

6.4 Conversation Logs

As a bot developer, you can access the complete conversation between your bot and the users. The *Conversation Logs* tab shows each utterance of your user and the bot replies.

You can analyze if your bot is able to understand the user's questions and respond with an appropriate answer. It helps you determine the flow of skill execution based on user utterances. This helps you to enhance the quality and accuracy of the bot response and improve the conversation design.

> **i Note**
>
> For all bots (irrespective of their owner being a user or an organization), collaborators cannot access the *Conversation Logs* tab. To access the conversation logs, you need to be a part of a team that has been assigned *Read* permissions for this tab. The default permission for all users is set to *No access*. For more information, see Bot Permissions [page 243]

> In case you have disabled the user conversation data from being stored while creating the bot, no data is visible under *Conversation Logs*. This is applicable only for Webchat and SAP Conversational AI Web Client channels and not for third party channels.

How to filter conversation logs

Upon accessing the conversation logs tab, there are some predefined filters that are already set. You can define the filters as per your need to fetch the conversations and their details. Based on this, all the conversations are displayed in the conversation panel.

> **i Note**
>
> All the filters that you set, are applied to the conversation logs (using `AND` function), however, if you have chosen multiple values for a filter, both values are considered (using `OR` function).

Filter	Details
Environment	Shows conversations based on the selected bot environment and version
Time range	Shows conversations within the selected period
Language detected	Shows conversations based on the selected language of the bot
Timezone	Shows conversations within the selected time zone
Conversation ID	Shows conversation associated with the provided conversation ID
Skills	Shows conversations in which the selected skill was triggered. You can select multiple skills. The conversations corresponding to only one of the selected skills will be displayed
Intents	Shows conversations in which the selected intent was triggered. You can select multiple intents. The conversations corresponding to only one of the selected intents will be displayed.
Entities	Shows conversations in which the selected entity was present. You can select multiple entities. The conversations corresponding to only one of the selected entities will be displayed.

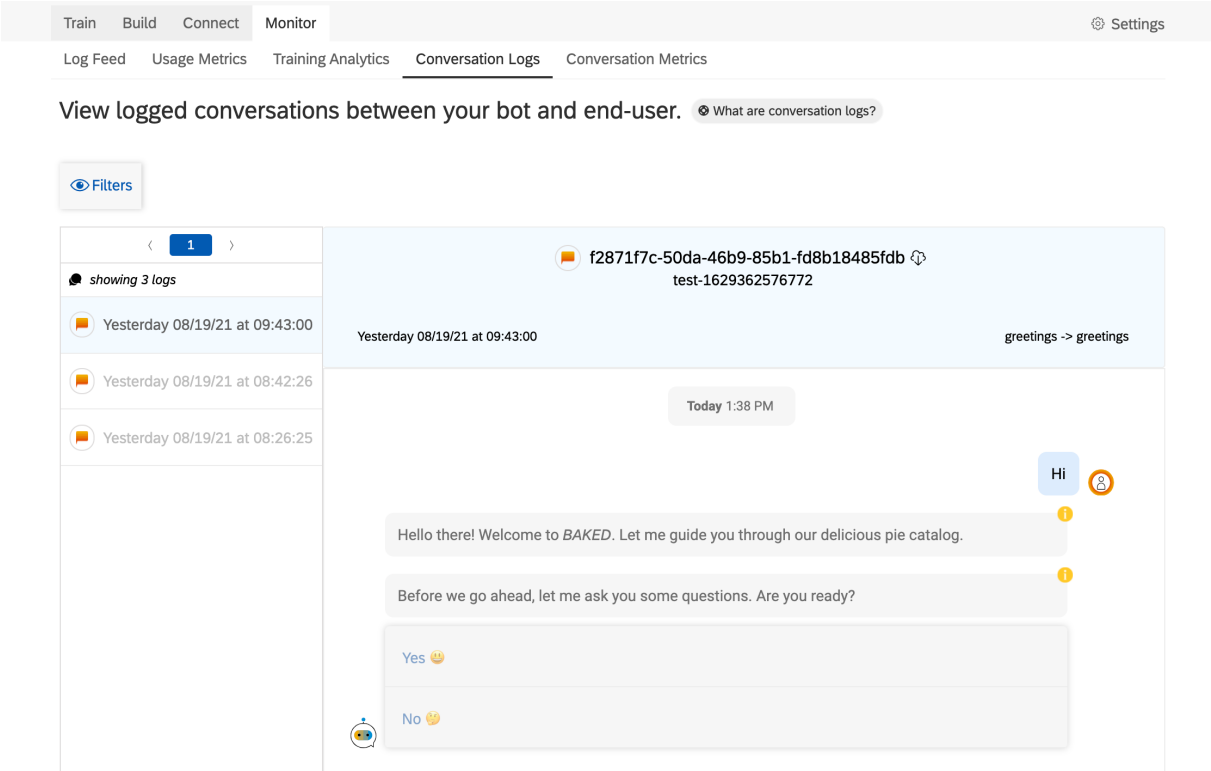
Click *Download filtered logs* at the bottom of the filter panel. All the filtered conversation logs are downloaded into a .CSV file with the following fields - *Conversation Id*, *Bot Id*, *Version*, *Environment Id*, *Received At*, *Conversation Thread Id*, *Message*, *Reply Type*, *Replies*.

What conversation data is visible

Based on the filters applied, the conversations are listed with the recent one first.

Each conversation has a unique conversation ID. It stores data related to a single conversation globally. This data is visible to the bot developer based on the permissions that are assigned at the bot level. For more information, see Permissions at Bot Level [page 242].

Click each conversation thread to see the details. You can see all the user utterances and the bot replies.



You can download the selected conversation thread into a .CSV file with the following fields - *Conversation Id*, *Bot Id*, *Version*, *Environment Id*, *Received At*, *Conversation Thread Id*, *Message*, *Reply Type*, *Replies*.

For more information, refer to the tutorial Improve Your Chatbot Accuracy by Monitoring User Activity.

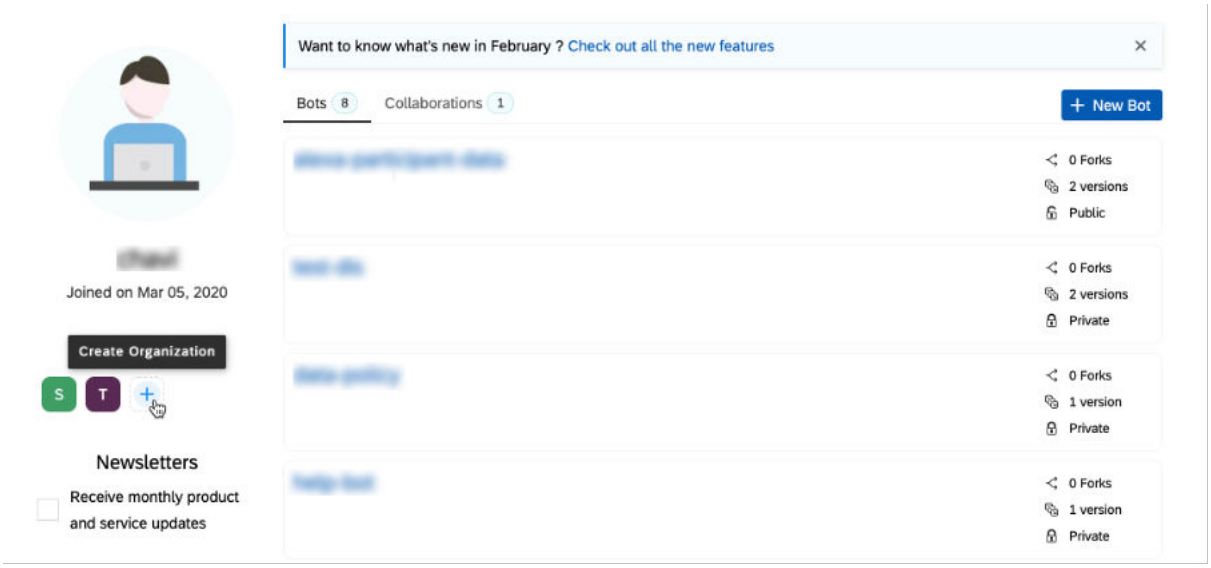
7 Collaboration

7.1 Organizations

Organizations are shared accounts, allowing groups of people to collaborate on several bots at the same time. Your user account (which is your identity on SAP Conversational AI) can be a member of any number of organizations.

Create an Organization

From your profile, you can create private and public organizations.



Public organizations, their public bots, and members are visible to all. Private organizations, their bots, and members are visible only to the members of the organization.

Manage an Organization

An organization must always have at least one administrator. If an organization has only one administrator, the administrator will be unable to update their own role, remove themselves from the organization, or delete their account until they make another member an administrator.

Add or Remove Members

If you are the administrator of an organization, you can add and remove members from the organization.

To add members, go to the *Members* tab and click *+ NEW MEMBER*. An email is sent to the bot developer or the team member requesting for consent. The member is added after the request is approved. Note that the email is valid for seven days from the initiation of the request.

To remove a member, go to the *Members* tab, click the overflow menu (three dots) to the right of the user's name, and choose *REMOVE FROM ORGANIZATION*.

> **i Note**
>
> If a member leaves the company, the administrator has to manually remove the member from the organization, otherwise the bot will be still accessible with the disabled credentials.

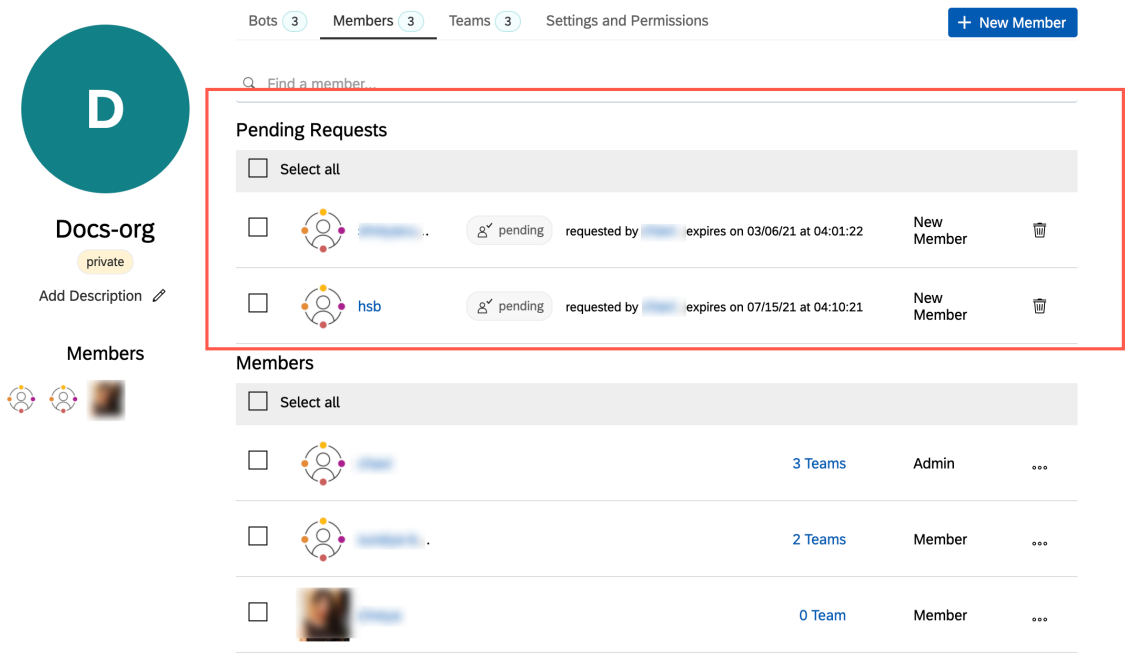
Manage Roles

If you are the administrator of an organization, you can change the role of a team member to an administrator or vice-versa. To change the role, go to the *Members* tab, click the overflow menu (three dots) to the right of the user's name, and choose *CHANGE ROLE*. Select the role that you want to assign and click *UPDATE ROLE*.

An email is sent to the bot developer or the org member requesting for consent. The role is updated after the request is approved. Note that the email is valid for seven days from the initiation of the request.

Manage Requests

As an org administrator, you can see all the pending requests under the *Members* tab of the organization. You can delete the pending request that has been sent to the bot developer or the team member.



Change the Organization's Settings

If you are the administrator of an organization, you can change the organization's settings, such as change the organization's name, assign permissions, make the organization private or public, and delete the organization. You do this on the *Settings and Permissions* tab.

If you change an **organization** from private to public, the existing private bots of the organization remain private, that is, they remain visible only to the members of the organization. (Note: To change a **bot** from private to public, go to the settings for the bot and click *Danger Zone* and then *MAKE PUBLIC*.)

If you change an **organization** from public to private, the existing public bots of the organization are also made private.

> **i Note**
>
> Only the bots that are assigned to the production organization provided by SAP are entitled for production use. The link to the production organization is sent to you via email. In case you have already started developing a bot, you can move your existing bots under this organization. For more information see 2831752.

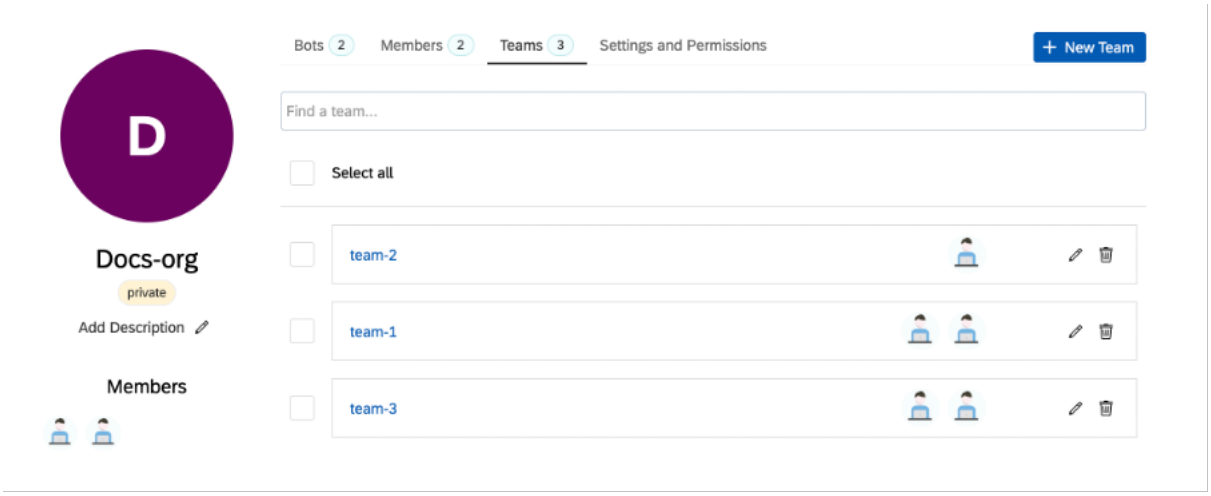
Create and Manage Teams

If you are the administrator of an organization, you can create and manage teams within the organization and assign additional permissions to them. See also Permissions at Organization Level [page 241].

Create Teams

You can create teams and add members on the *Teams* tab.

An email is sent to the bot developer requesting for consent. The member is added after the request is approved.. Note that the email is valid for seven days from initiation of the request.



Tip: You can also add members to teams on the *Members* tab.

Manage Teams

You can change the name of an existing team and delete teams on the *Teams* tab.

Transfer a Bot to an Organization

You can transfer an existing FAQ or an actions bot from your user account to another user account, an organization account, or from an organization account to another organization account.

> **i Note**
>
> If you are the only administrator of an organization and you are leaving the company, make sure that you transfer the bot ownership to another team member.

Prerequisites

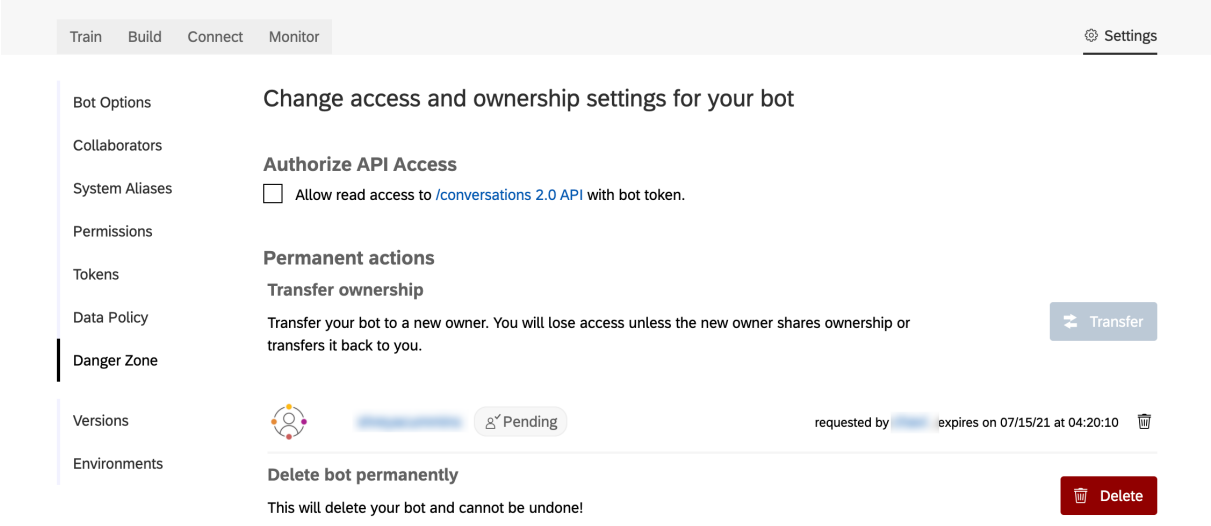
If you transfer a bot from your user account to an organization account, you must be an administrator or belong to a team with *Create bot* permission in the target organization.

If you transfer a bot from an organization account to another organization account, you must be an administrator or belong to a team with *Create bot* permission in both the sending organization and the target organization.

For more information, see Permissions at Organization Level [page 241].

To Transfer a Bot to an Organization:

1. Go to the *Settings* page for the bot.
2. Under *Danger Zone*, click *Transfer*.
   An email is sent to the bot developer or the organization administrator requesting for consent. The bot is transferred after the request is approved. Note that the email is valid for seven days from initiation of the bot transfer request.



Manage Requests

All the pending requests for bot transfer are visible under the *Danger Zone*. As an org administrator or a member with *Create bot* permission, you can delete the pending request that has been sent to the bot developer or the org member.

7.2 Permissions at Organization Level

If you are the administrator of an organization, you can assign one of the following permissions as the base permission to all members of the organization. You can also assign one of the following permissions as an additional permission to a team of one or more members of the organization. You do this on the *Settings and Permissions* tab.

- No access
  Members of the organization who have only this permission cannot see or access any of the organization's private bots. However, they can access its public bots.
- Read only
  Members of the organization who have only this permission can access all of the organization's bots in read-only mode, that is, they can view, search, and filter all of the organization's bots. They can also fork any of the organization's bots, but only to a destination where they have read/write access.
- Read and write
  Members of the organization who have only this permission can access all of the organization's bots in read/write mode, that is, they can edit, delete, and fork any of the organization's bots, as well as reload tokens.
- Create bot (+ Read and write)
  Members of the organization who have this base permission have the same access as members with *Read and write* permission, plus they can create new bots and transfer existing bots.

Administrators of an organization always have all rights, that is, they have *Create bot (+ Read and write)* permission, plus they can change the organization's settings, as well as manage teams and members.

Base Permissions

The base permission is the default permission granted to all members of the organization and applied to all bots within the organization.

Base permissions

Define the permissions granted to all members and applied to all bots within the organization.

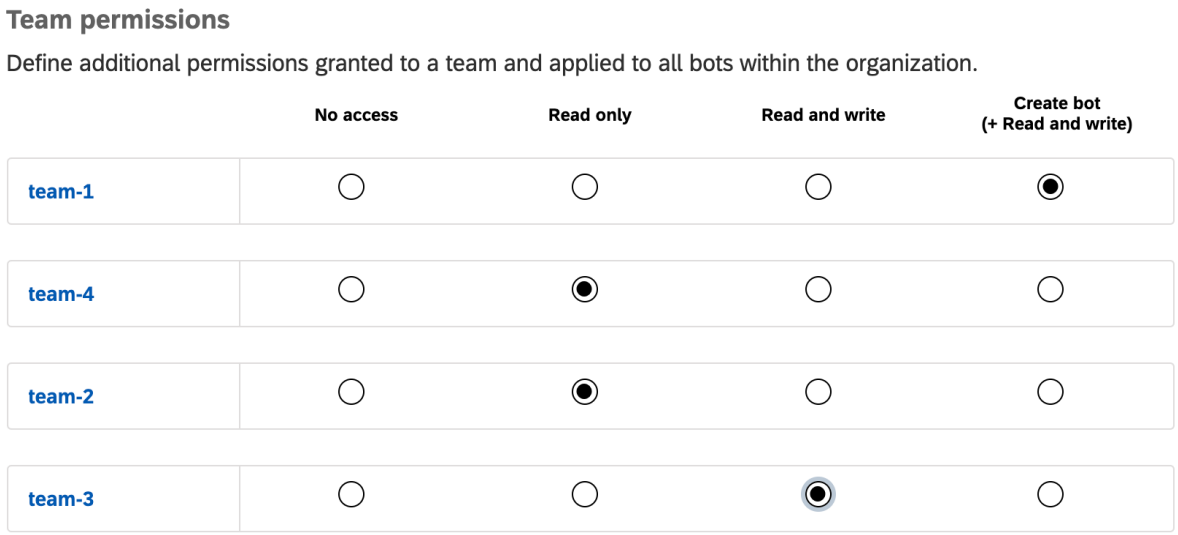
	No access	Read only	Read and write	Create bot (+ Read and write)
All members	○	○	○	◉

For existing organizations, the default base permission is *Create bot (+ Read and write)*. For new organizations, the default base permission is *No access*.

Team Permissions

Team permissions let you give additional permissions to teams of one or more members of the organization. Team permissions are applied to all bots within the organization.

You create teams and add members to teams on the *Teams* tab. (Tip: You can also add members to teams on the *Members* tab.) You then assign the additional permission to the team on the *Settings and Permissions* tab.



Since these additional permissions are provided only to teams, if you want to assign an additional permission to only one member of the organization, you simply create a team with only that member.

When you create a new team, the default permission for the team is set to the base permission. The base permission is also the minimum credential that can be set for the team permission.

> **Example**
>
> In your organization, the base permission is set to *No access*. You create two new teams: *Team 1* and *Team 2*. Since the base permission is set to *No access*, the default permission for *Team 1* and *Team 2* is also set to *No access*.
>
> You change the team permission for *Team 2* to *Read and write*. You then change the base permission to *Read only*. The team permission for *Team 1* is updated accordingly to *Read only*.

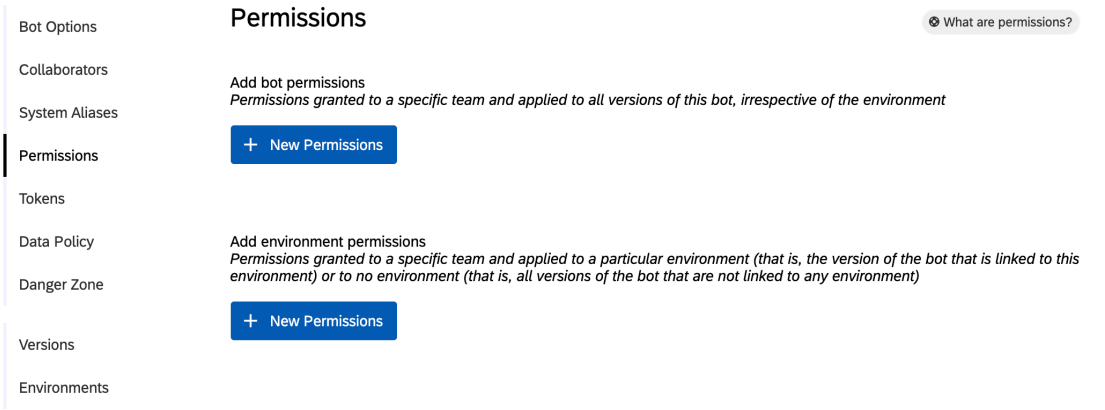
> **Note**
>
> For bot conversation logs you need to provide permissions under bot settings.

7.3 Permissions at Bot Level

If you are the administrator of an organization, or if you have Read and Write permissions for the *Settings* module, you can assign one of the following permissions as the base permission to all members of the organization or to a team for a given bot.

You can also assign one of the following permissions as an additional bot permission to a team of one or more members of the organization as an environment permission (see below). You do this on the bot *Settings and Permissions* tab for the bot.

- Read only
  The team can access the bot in read-only mode, that is, they can view, search, and filter the bot. They can also fork the bot, but only to a destination where they have read/write access.
- Read and write
  The team can access the bot in read/write mode, that is, they can edit, delete, and transfer the bot. They can also fork the bot, but only to a destination where they have read/write access.



> **i Note**
>
> - Request tokens for at least one module of the bot are displayed to the team with Read Only access, but it can be reloaded only if you are the administrator of an organization or if you have Read and Write permissions on *Settings*.
> - Dev tokens are displayed and can be reloaded only if you are the administrator of an organization or if you have Read and Write permissions on *Settings*.

Bot Permissions

Bot permissions are granted to a specific team and applied to all versions of the bot, irrespective of the environment.

You can set these permissions individually for the following modules. For example, you can set Read only for the Train module, but Read and write for the Connect module:

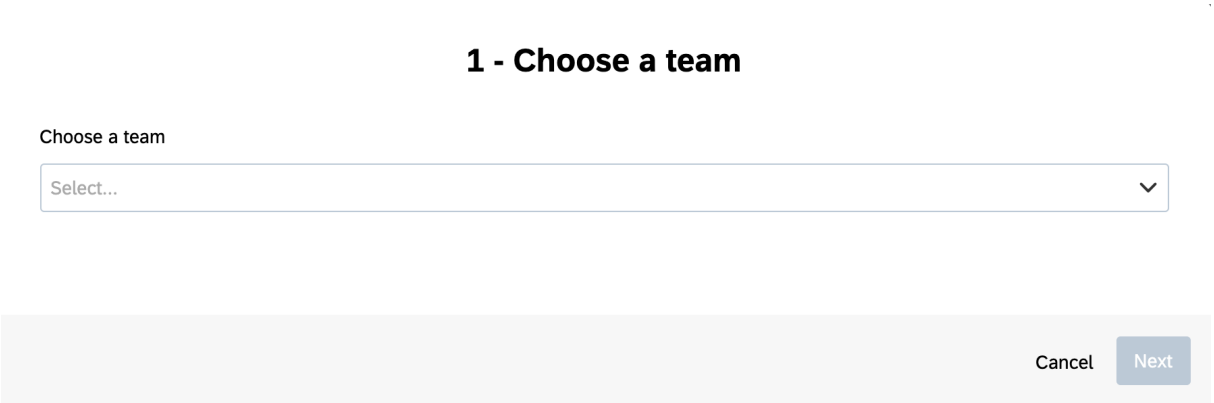
- Train
  Includes the *Train* tab, NLP settings, log feed, and training analytics.
- Build
  Includes the *Build* tab and bot builder settings.
- Connect
  Includes the *Connect* tab and bot connector settings.
- Monitor
  Includes permissions for each tab under *Monitor* - Log Feed, Usage Metrics, Training Analytics, and Conversational Logs.
- Settings
  Includes the bot settings and permission management.

> **i Note**
>
> When adding only *Bot Permissions*, the permissions for the log feed and training analytics under *Monitor* tab will be prioritized.
>
> When adding only *Environment Permissions*, the permissions for the log feed and training analytics under *Train* tab will be prioritized.

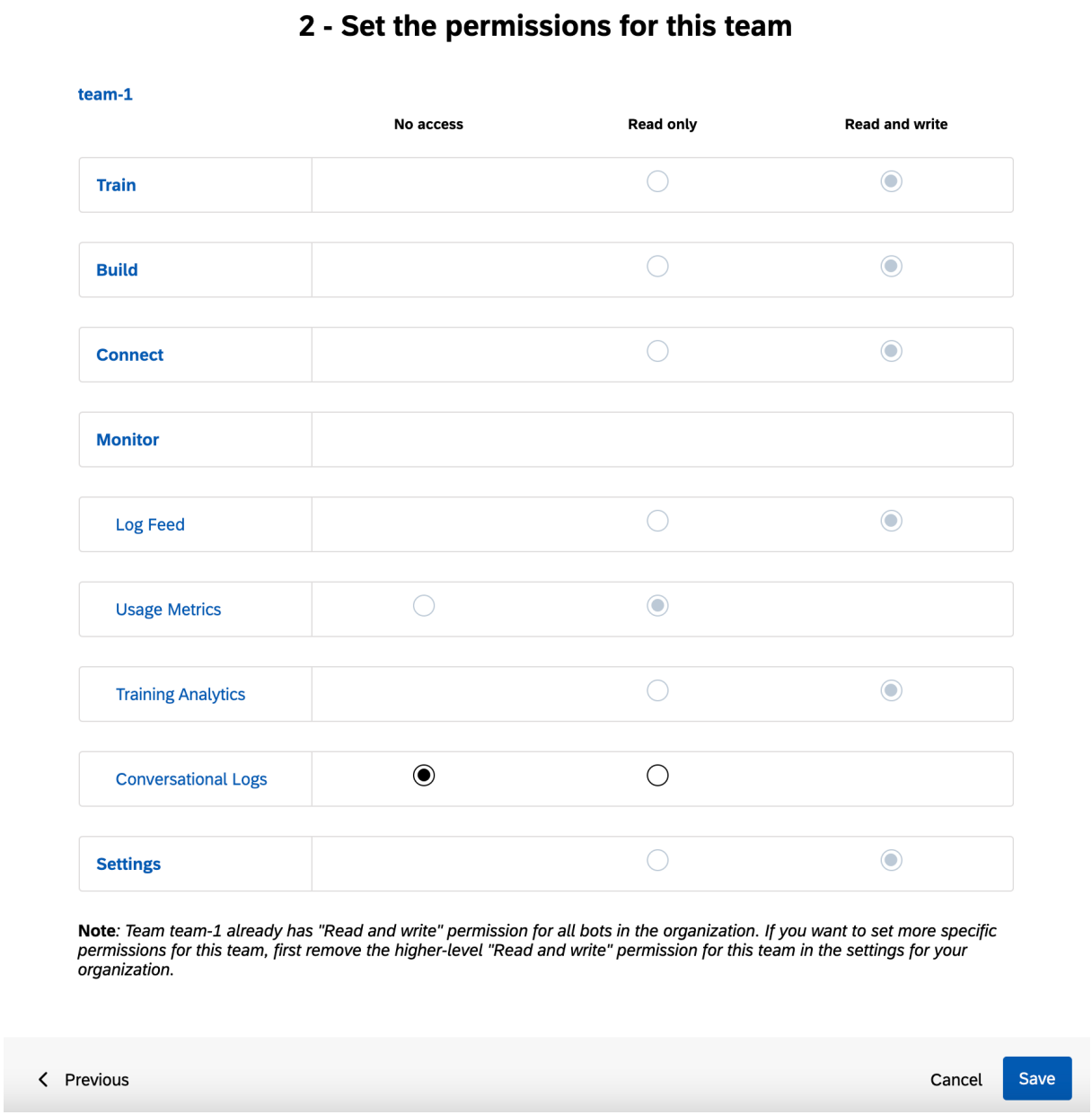
- #unique_86/unique_86_Connect_42_subsection-im1 [page 244]
- #unique_86/unique_86_Connect_42_subsection-im2 [page 245]

Click the shape for more information.

Choose Team



Set Permissions



> **i Note**
>
> - Permissions that you apply to *Build* are also replicated to *Train*.
> - Permissions that you apply to *Settings* are also replicated to *Train* and *Build*.

By default, the bot permissions correspond to the permissions defined at organization level. See Permissions at Organization Level [page 241]

Note that the bot permissions correspond to the minimum environment permissions for a given team.

Environment Permissions

Environment permissions are granted to a specific team and applied to a particular environment (that is, the version of the bot that is linked to this environment) or to no environment (that is, all versions of the bot that are not linked to any environment). Examples of environments are Production, Development, and so on.

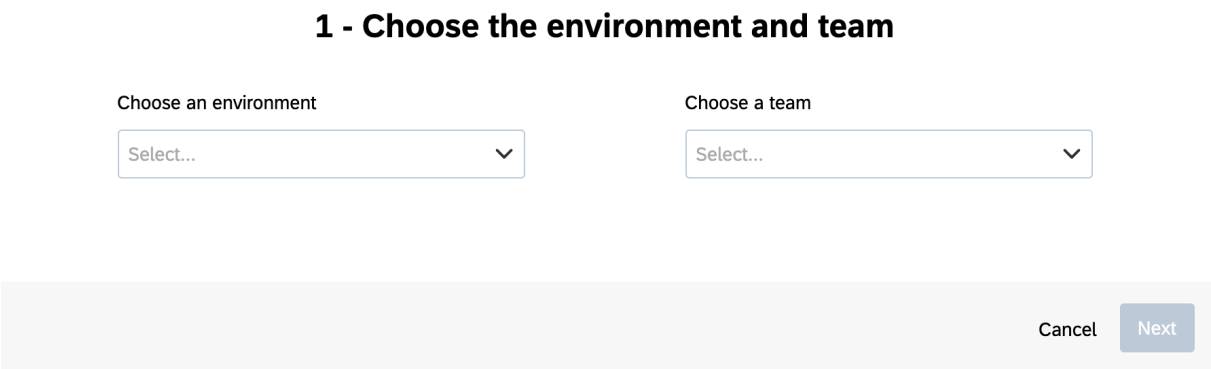
You can set these permissions individually for the following modules:

- Train
  Includes the *Train* tab, NLP settings, log feed, and training analytics.
- Build
  Includes the *Build* tab and bot builder settings.

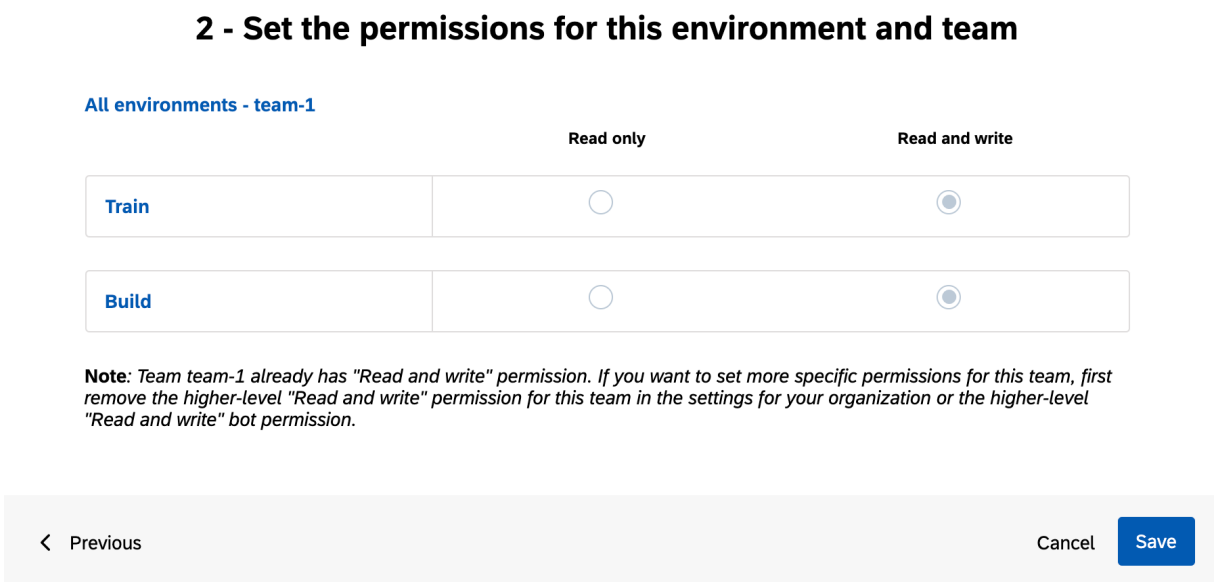
- #unique_86/unique_86_Connect_42_subsection-im3 [page 246]
- #unique_86/unique_86_Connect_42_subsection-im4 [page 247]

The following image contains links to more information.

Choose Environment



Set Permissions



> **i Note**
>
> Permissions that you apply to *Build* are also replicated to *Train*.

By default, the environment permissions correspond to the permissions defined in the bot permissions. If no bot permissions are set, the permissions defined at organization level are applied.

8 Bot Management

8.1 Versions and Environments

You can use versions and environments to manage and update large, complex chatbots in an organized way that doesn't expose working drafts of a chatbot to users.

What Is a Version and Why Are They Useful?

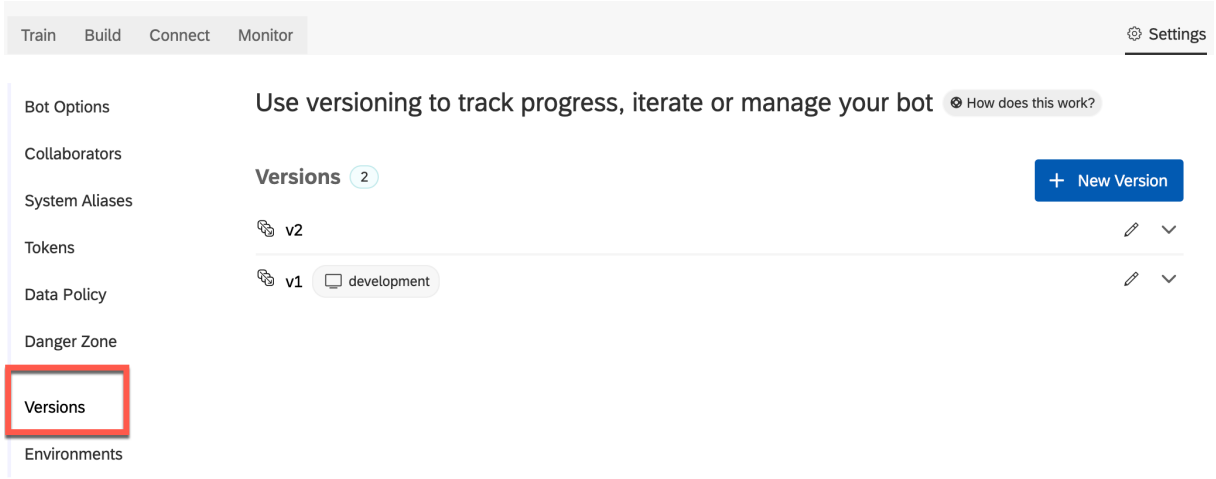
A version is a package of your bot training dataset and skills. Each version is independent of the others and can be managed individually. For example, you might want to create a new version prior to major updates to your training dataset or skills. Or you might want to create two or more variants of the same core bot for different audiences.

When you create a new bot, by default, your bot has only one main version *v1* and is assigned to the *DEVELOPMENT* environment.

How Do I Create a New Version?

You can create a new version under *VERSION SETTINGS* or in the dropdown near the bot name. Click *CREATE NEW VERSION* and select the version you want to copy. This copies the *Train* and *Build* tabs of the source version and creates a new version that you can then name. The new version is a pure copy; you can update the new version or the old one separately.

> **i Note**
>
> If you have only one version of your bot, this version cannot be deleted. You also cannot delete a version that is associated to an *Environment*. You first need to modify the version associated to the environment (or delete the environment). However, if you have more than one version (not linked to an environment), it can be deleted.


Train
Build
Connect
Monitor
Settings
Bot Options
Collaborators
System Aliases
Tokens
Data Policy
Danger Zone
Versions
Environments
Use versioning to track progress, iterate or manage your bot
How does this work?
Versions
2
New Version
v2
v1
development

Train Build Connect Monitor Settings

Bot Options
Collaborators
System Aliases
Permissions
Tokens
Data Policy
Danger Zone
Versions
Environments

Use versioning to track progress, iterate or manage your bot How does this work?

Version 1 New Version

v1 development

Tokens

Request token

You can use the request token above to make runtime API requests on:

https://api.cai.tools.sap/build/v1/dialog
to use bot builder (API Reference)
https://api.cai.tools.sap/v2/request
to analyze text (API Reference)

Note: Default environment configuration will be automatically applied if no environment is assigned.

Default language

Language

English Advanced

Bot builder

Bot webhook base URL

Save

Follow an https format: https://my-api.com

NLP

Context management

Resolve pronouns
Map pronouns to entities in the conversation history

Resolve descriptions
Map superlatives to list replies in the conversation history

Training Mode

Automatic
Training is automatically triggered by any change to the bot

Manual
You decide when you want to update your training

Intent

Matching Strictness

0 50 100

Entity

Free and Gold Entities Matching Strictness

0 100

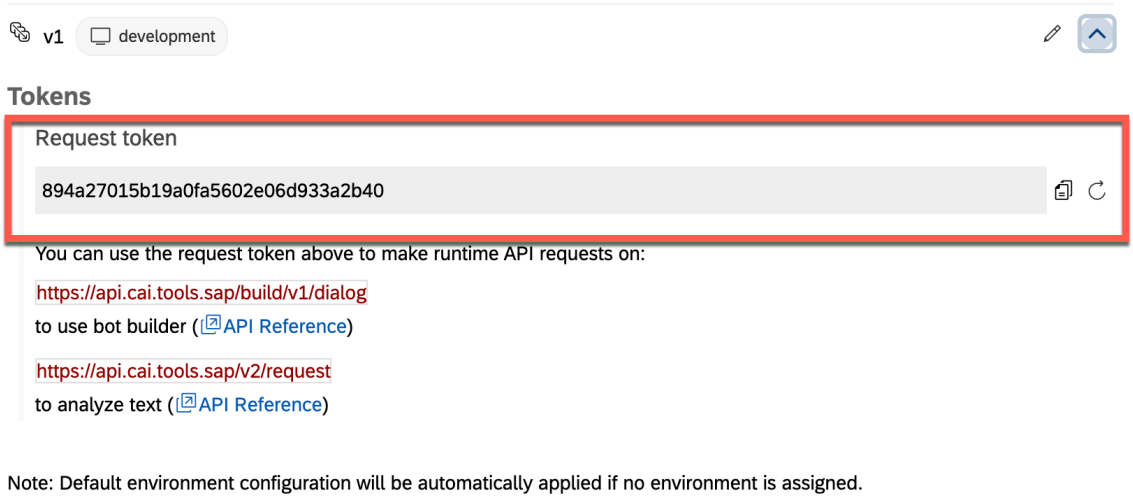
Restricted Entity Matching Strictness

0 100

> **i Note**
>
> Bots are restricted to eleven or fewer versions.

Version Request Token

Each version has a dedicated request token. This means that if you want to analyze a text with the `/request` endpoint or use the Bot Builder API with the `/dialog` API, you have to provide the request token from the version that you want to use.

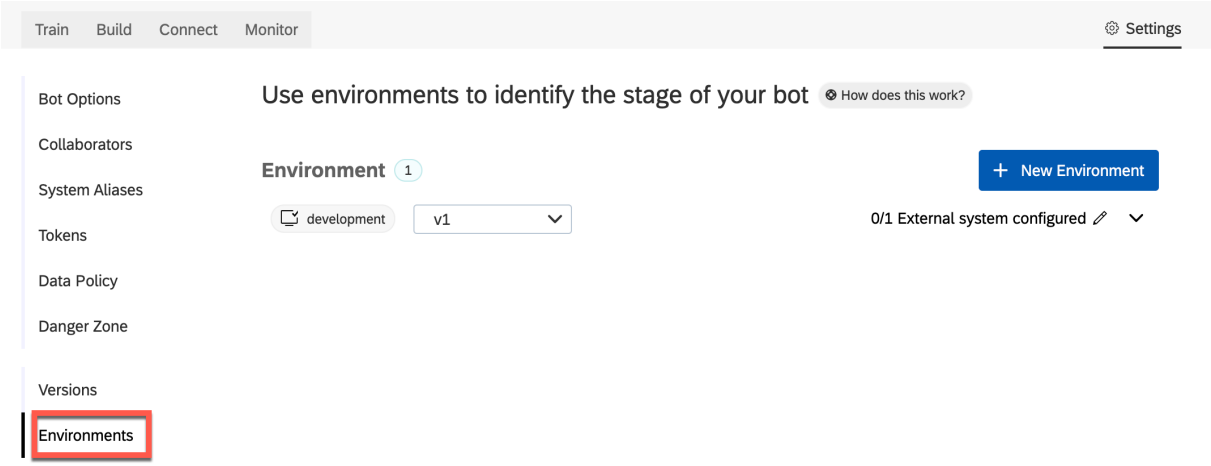


> **i Note**
>
> We strongly recommend to use the request token of an environment, as this allows you to define certain configurations for that specific environment. When using a request token of a version, the default environment will be used for such environment-related configurations. For example, when using a version request token for a bot that uses System Aliases, the configurations from the default environment will be used.

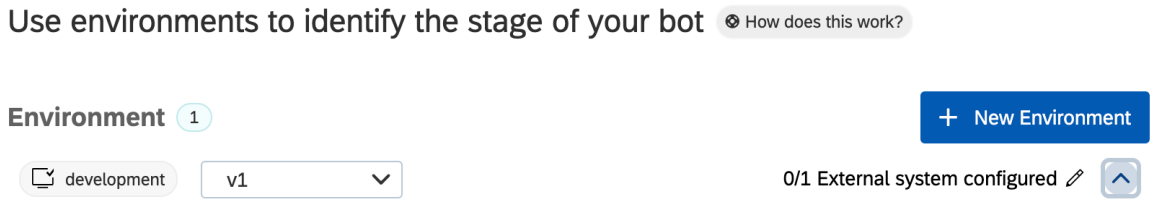
What Is an Environment?

Environments are configurations applied to specific versions and help you to seamlessly deploy your chatbot in production. They are best leveraged as specific consumption environments (for example, Development, Staging, and Production).

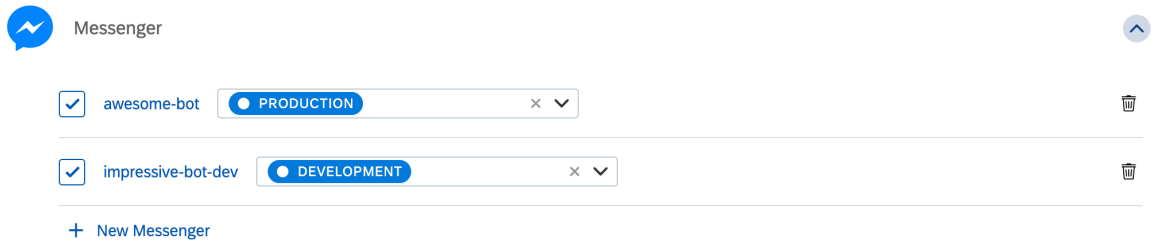
When you first create a bot, your first version *v1* is by default associated and linked to the environment *DEVELOPMENT*. You can create and name additional environments in the *VERSION SETTINGS* area under *Environments*.



Each environment is always linked to a specific version.



On the *Connect* tab, you can connect each channel to a specific environment (if a channel is not explicitly linked to an Environment, the Default Environment is used). This means that you can have a Facebook Messenger channel for your *DEVELOPMENT* environment and link this environment to version *v2*, and have a Facebook Messenger channel for your *PRODUCTION* environment and link this environment to version *v1*.



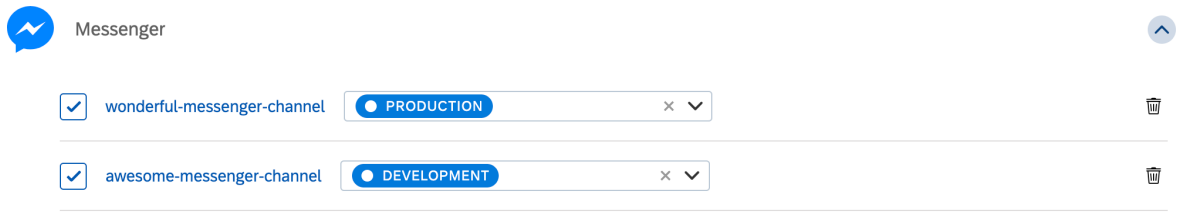
Environment Request Token

Each environment has a dedicated request token. This means that if you want to analyze a text with the `/request` endpoint or use the Bot Builder API with the `/dialog` API, you can either provide a version token or an environment token.

If you provide an environment token in your request, it uses the version of your chatbot that is linked to this environment.

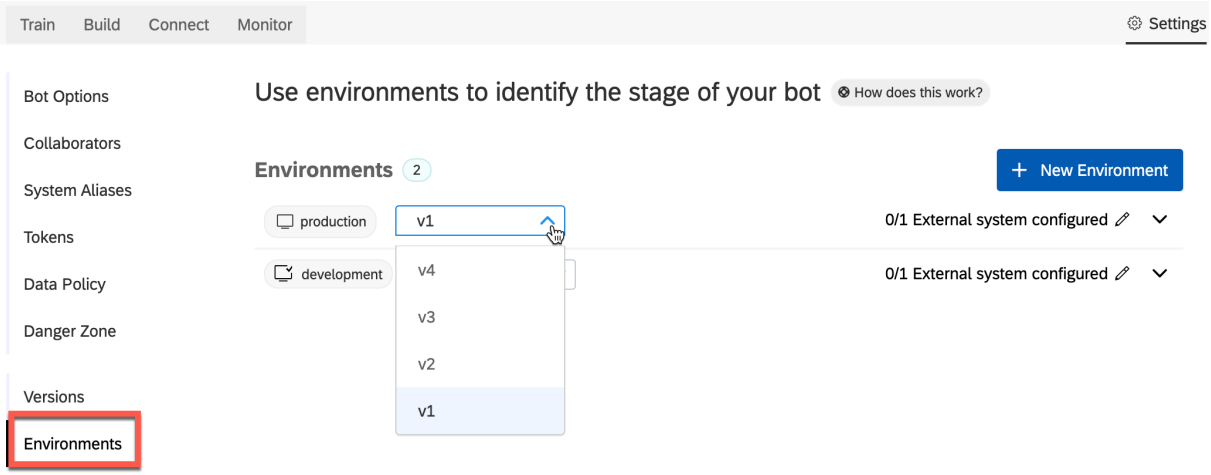
How Do Versions and Environments Work?

Since each environment is linked to a version, it's really easy to deploy a new version to a production environment. Here you have two Facebook Messenger channels: one for the *DEVELOPMENT* environment and one for the *PRODUCTION* environment. The *v1* version of my bot is in the *PRODUCTION* environment. The users chatting with my Facebook Messenger page *Awesome-bot* are chatting with this version.



Let's say I'm working on a different version *v2* and I'm testing it on the *DEVELOPMENT* environment with another Facebook Messenger page. I'm pretty comfortable with this new version and now I want to deploy it to the *PRODUCTION* environment.

On the *Settings* page for my bot, I go to *VERSION SETTINGS* and change the version that is linked to the *PRODUCTION* environment.



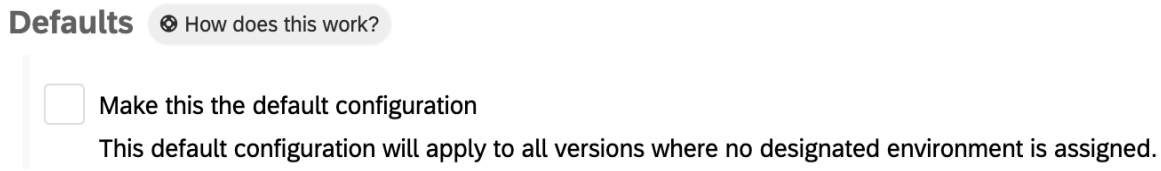
Now the *PRODUCTION* environment is linked to the *v2* version of my bot. The users on my Facebook Messenger page *Awesome-bot* can now talk to the new version of my bot.

You can assign a version to multiple environments. However, you can assign only one version to each environment.

Default Environment

You can select one environment as the default environment, this configuration is used where no designated environment is assigned for a version. This is especially the case when you're using a request token of a version or *Chat Preview* to test a bot version which does not have an environment assigned. In these cases the default environment will be used, for example when using System Aliases.

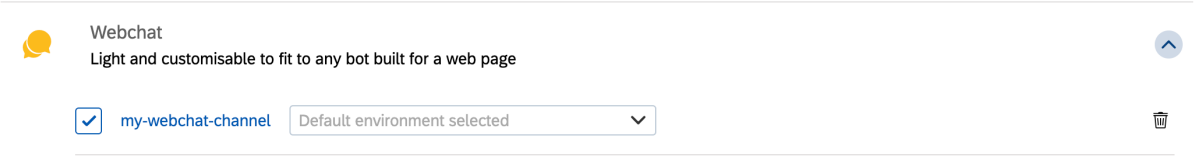
For newly created bots, the *Development* environment is automatically selected as the default environment, but you can change this by selecting the *Make this the default configuration* for another environment.



The current default environment can be identified by the checkmark next to it.

> **i Note**
>
> You can't deselect the default environment without setting another environment as default. When you set a new environment as default, the checkmark from first default environment will be unchecked automatically.
>
> If you want to delete an environment that has been set as default, you first have to set another environment as the default one.

The default environment will also be used when creating a channel without a dedicated environment. In this case, the version assigned to the default environment and the configuration are used when chatting with your bot through that channel.



How Does the Bot Builder API (Without Bot Connector) Work?

If you're directly using the Bot Builder endpoint `/dialog` without a channel in the Bot Connector, the best practice is to use environment request tokens and not version tokens.

In your code, when you request SAP Conversational AI and send a message, it will always be on the same environment (for example, the *PRODUCTION* environment). When you need to deploy a new version of your chatbot to your users, you just need to go to the *Settings* page for your bot and, under *VERSION SETTINGS*, change the version that is linked to the *PRODUCTION* environment.

You don't need to change the request token in your code because it's the same environment that you're requesting; you simply switch to a different version on the *Settings* page.

Monitoring

On the *Monitor* tab, you can filter all of the metrics by environment and version. For example, you can opt to see only the log feed for the *STAGING* environment or only the usage metrics for the *PRODUCTION* environment.

8.2 Forking Bots, Skills, Intents, and Entities

When you start using SAP Conversational AI, you may want to use someone else's bot as a starting point to get up and running quickly. As you use SAP Conversational AI more frequently, you may want to reuse previous bots and customize them for particular use cases. You may also want to reuse individual building blocks like skills, intents, or entities across multiple bots. You can do this through forking.

> **i Note**
>
> If you simply want to make a change to a deployed version of a bot, or test something out, we recommend creating a new version of the bot instead of forking it. For more information, see Versions and Environments [page 248].

Forking a bot, skill, intent, or entity creates a personal copy of it. Forking is a one-way operation that behaves like copying a file on your local file system. It doesn't establish a direct link between the original and your copy.

At the highest level, you can fork a bot. If you want to reuse more granular building blocks, you can fork individual skills, intents, or custom entities. Forking behaves slightly differently, depending on what is forked.

Remember, you cannot fork a bot into an existing bot.

Forking a Bot

Resources Forked

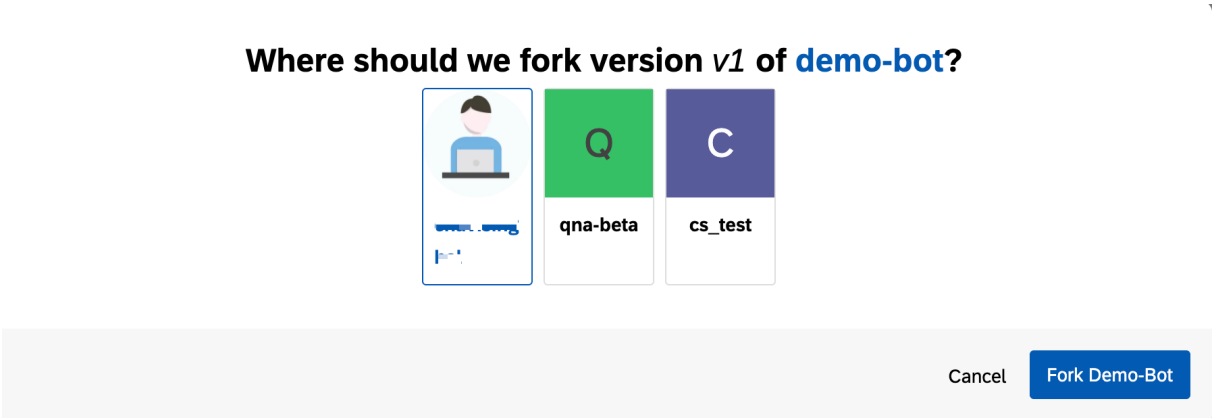
Forking a bot creates a personal copy of all resources that belong to the bot: intents, entities, skills, skill groups, body/header templates in every language and data policy settings. The following are not copied: Channels (on the Connect [page 181] tab), usage metrics, conversation logs, training analytics benchmarks, versions and environments, authentication templates, roles and permissions, and runtime data (user and conversation data that is visible in the Log Feed [page 224].)

The configuration of the bot is partially copied (default language, system alias (name only) and matching strictness).

> **⚠ Caution**
>
> Currently, not copied are callback URL, message delay between messages, context management settings, training mode, data policy, and collaborators.

Restrictions and Edge Cases

Forking a bot for which versioning is enabled creates a personal copy only of the currently selected version. This means that a new bot is created based on the version. The version name as well as the assigned environment are not forked.



Permissions/Destinations

You can fork any bot for which you have at least read permissions.

> **! Restriction**
>
> If you are collaborator for a private bot, but you are not a part of the organization to which the bot is assigned, you will not be able to fork it. If the bot is a part of a private organization, it is a private bot and it can be forked (within the organization) by the owner of the bot or a collaborator who is also part of the organization.

A public bot can always be forked. A private bot in a public organization can be forked within the organization for which you are a member.

You can fork bots into your own account or in an organization where you have *Read and Write* permissions. If the bot is a part of a private organization, it is a private bot and it cannot be forked.

Bot visibility	Environment	Status	Can it be forked?	Condition
Public	Organization/Account	Owner/Member/ Collaborator	Yes	None
Private	Public/Private Organization	Owner/Member	Yes	Can be forked within the organization
Private	Public/Private Organization	Collaborator who is not a part of the organization	No	Not applicable
Private	Account	Owner	Yes	Into an organization you have *Read and Write* permissions to

Forking a Skill, Intent, or Entity

Destinations

You can fork a skill, intent, or custom entity to the following destinations::

- To any version of a bot for which you have *Read and write* permission.
  The skill, intent, or entity is forked into the specific version of the target bot (which you selected) in that account.
- From any bot for which you have *Read* permission.
  This includes private bots from your account or any organization.

If languages are enabled in the source bot that are not enabled in the destination bot, the data for the language is copied over, but it is hidden until the language is enabled in the destination bot.

See also *Forking a skill*, *Forking an intent*, and *Forking an entity* below.

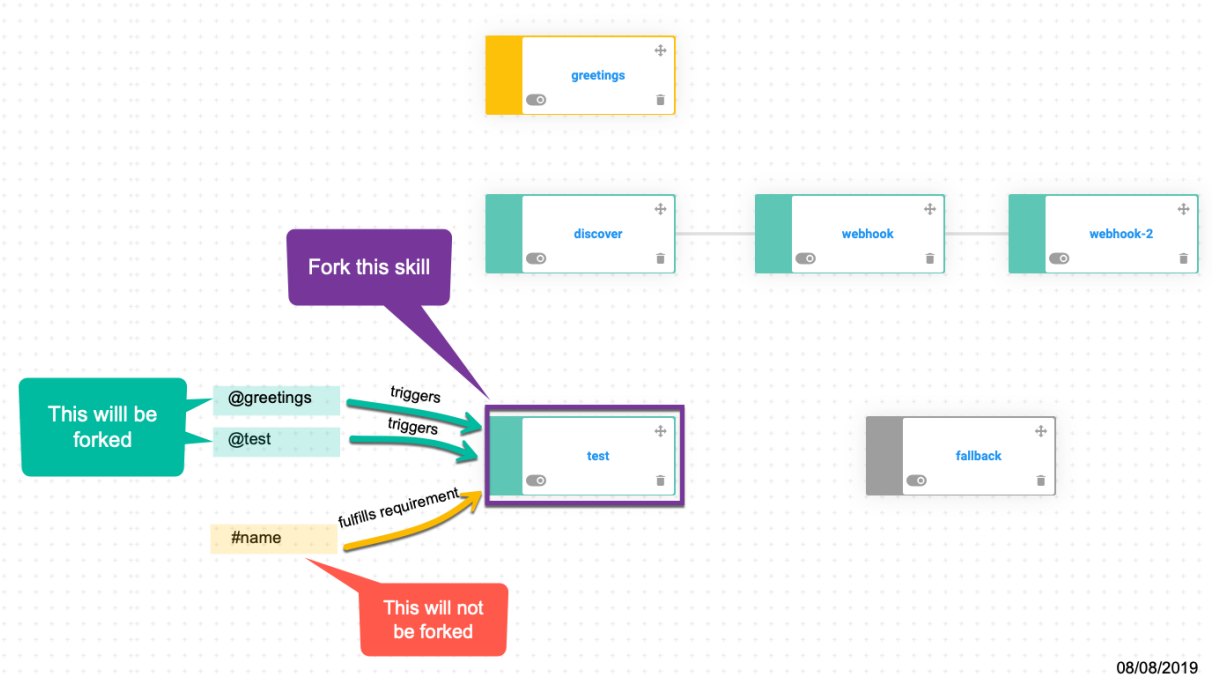
Forking a Skill

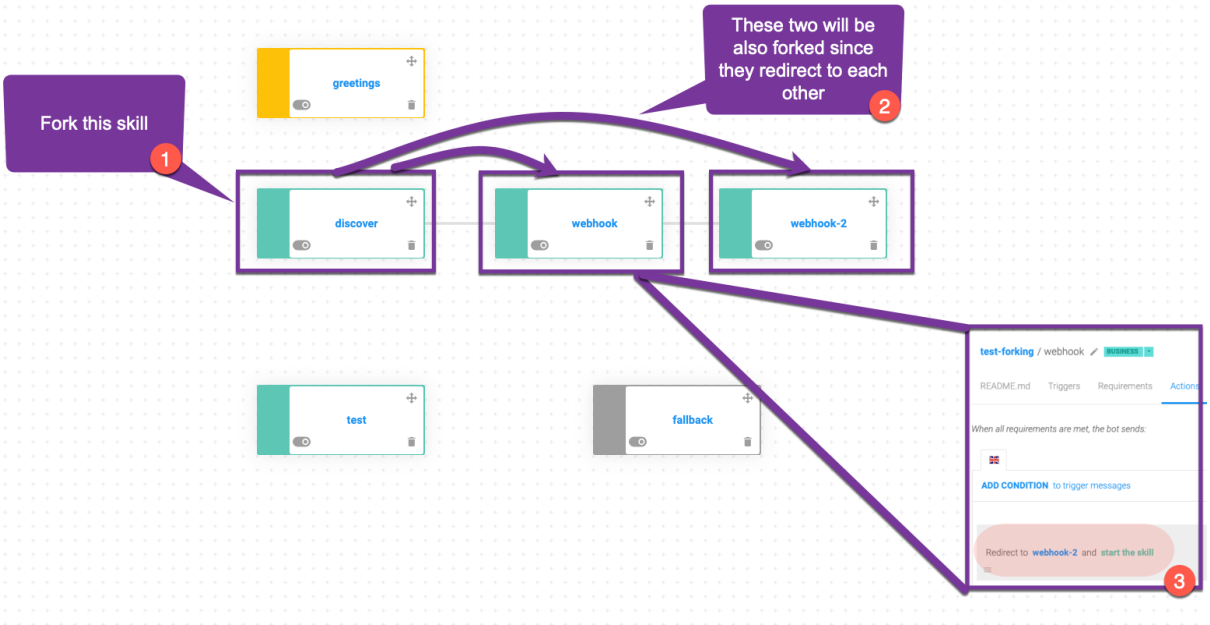
Resources Forked

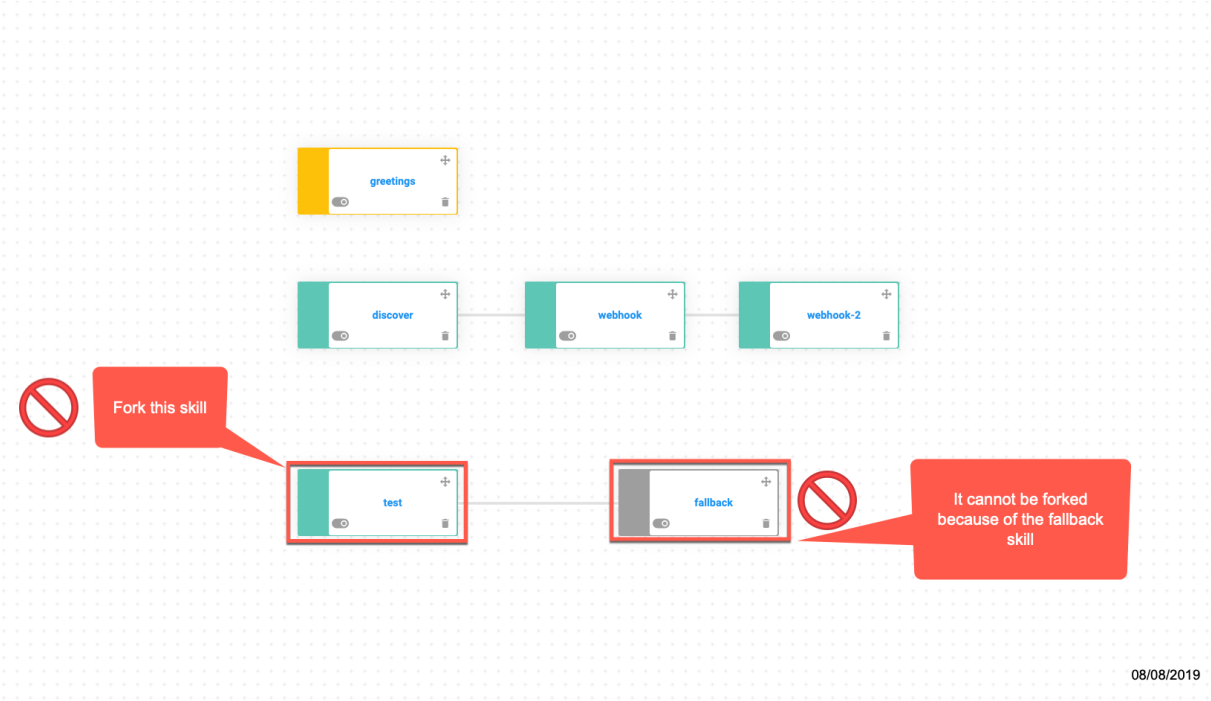
When you fork a skill, it is forked with its corresponding state (activation state, README.md, triggers, requirements, actions, markdown settings, change of language) as well as its associated resources. Here is a list of associated resources that are forked and that aren't.

Associated resources	Are they forked?	Details
Skill (Redirect to)	Yes	If the skill redirects to other skills (except the fallback skill), these skills are also copied. If a skill with the same name already exists in the destination bot, it is given a postfix as the existing ones are not overwritten.
Intent (requirement or trigger)	Yes	Only if an intent is a requirement or a trigger
Intents (with same name and in the requirement or trigger)	No	If there is already an intent with the same name in the destination bot as in the source bot then the intent is not forked. Instead, the newly created skill points to the intent of the target bot.
Body and header templates	Yes	All body and header templates used in webhook calls are forked over.
Entity (tagged in an intent which is forked over)	Yes	If the entity is tagged in an intent which is a trigger or requirements, it is forked over.

Associated resources	Are they forked?	Details
Entity (requirement or trigger)	No	If an entity is a requirement or trigger, it is not forked.
Webhook authentication information and templates	No	All webhook authentication and templates in webhook calls are not forked currently due to security risks







Constraints

For the memory variables that are copied over, there is currently no check if a variable with the same name exists in the destination bot. If it does, then the two variables with the same name will be treated as one variable.

You cannot fork a skill that redirects to the fallback skill or calls a fallback channel.

Entire skill groups cannot currently be forked.

Forking an Intent

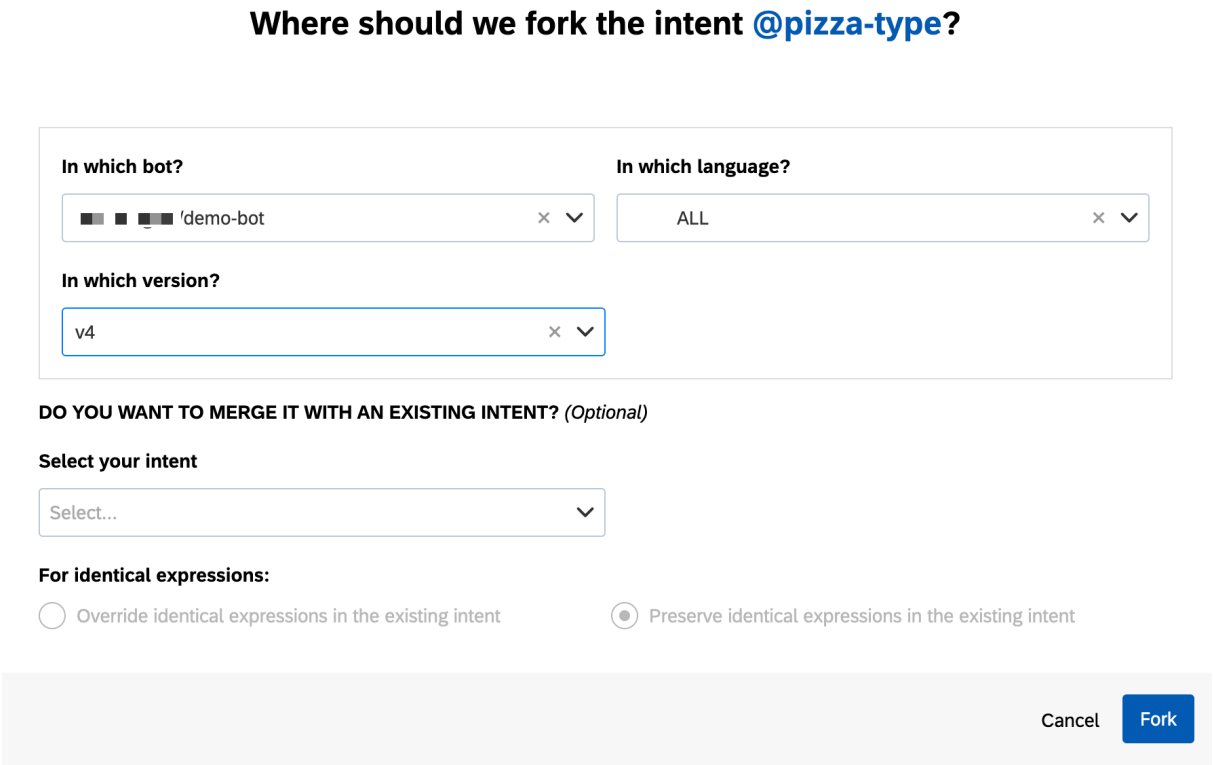
Resources Forked

Forking an intent creates a copy of the localized expressions together with the entities that have been tagged. For custom entities, all values are copied over.

By default, all languages of the intent are copied over, though you can choose to only copy expressions and entities in a specific language.

Options

By default, forking creates a new intent in your destination bot. Instead of creating a new intent, you can choose to merge the expressions of the source intent in the destination intent. In both cases, tagged entities are copied over unless they also exist in the new bot, in which case they are just linked instead of copied.



Forking an Entity

Resources Forked

Forking a custom entity creates a copy of the entity with its corresponding training data, custom enrichments, and fuzzy matching strictness. All intents or skills in which the entity is used do not get copied over automatically. For free entities, the training data annotated in intents is lost.

By default, entities and their values in all languages are copied over. However, you can choose to copy them in a specific language.

> **i Note**
>
> By default, entities and their values in all languages of the entity are copied over. However, you can choose to copy in a specific language.

Options

Forking without merging

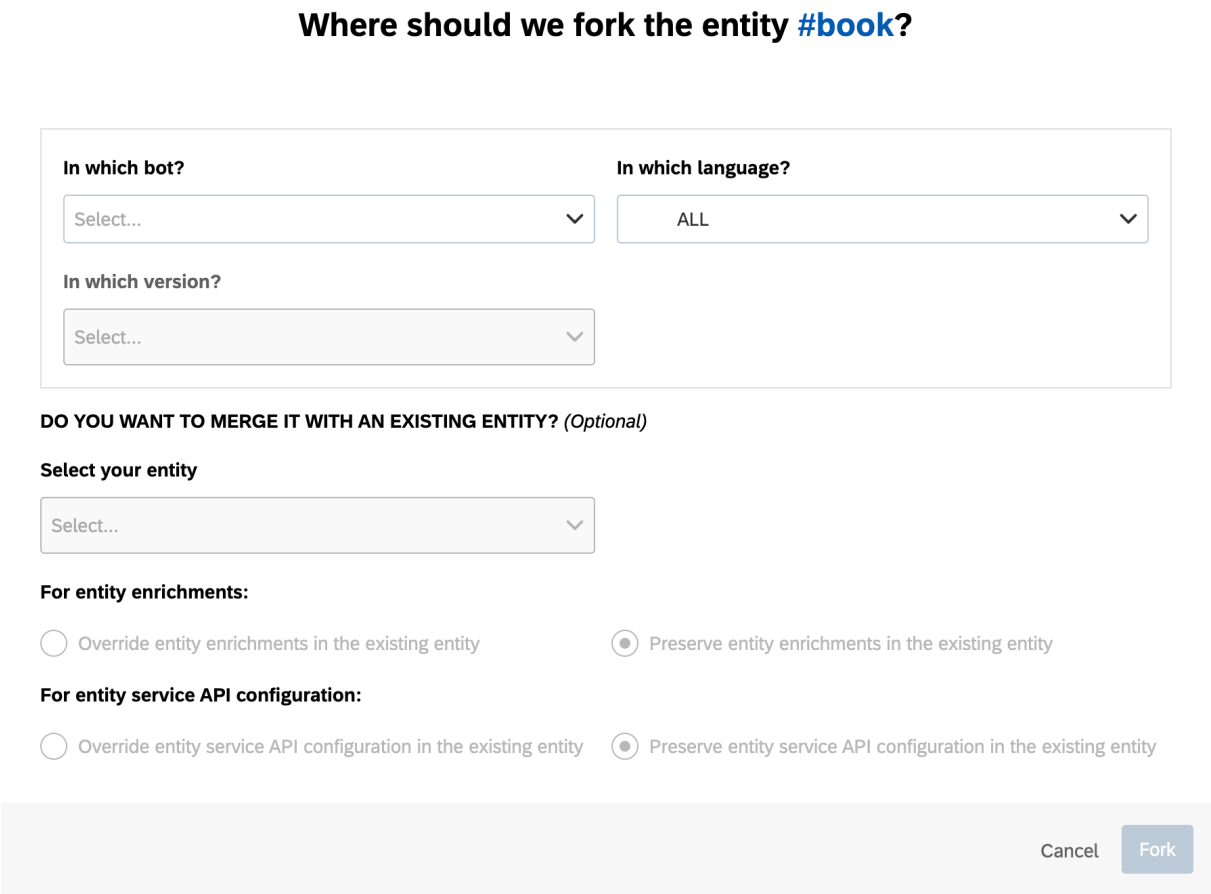
Forking creates a copy of the entity in your destination bot. Its corresponding entity values, enrichments and Service API configurations are also copied over. For example, if you fork an entity "phone", and it already exists in the destination bot, then the entity "phone-1" is created and all the values, enrichments are also copied over.

Forking and merging

Instead of creating a new duplicate entity, you can choose to merge the entity values of the source entity within the destination entity.

Here you have two options, each for entity enrichments and entity service API configuration:

- Override: All the enrichments or entity service API configurations from the source bot are retained
- Preserve: All the enrichments or entity service API configurations from the destination bot are retained



Forking Bots vs. Creating a New Version

When to fork	When to version
You want to import a third-party bot for testing and modification (for example, movie bot)	You want to make a change to a deployed version (isolation)
You want to try something out with a bot in which you only have *Read* access	You want to test something out in the same bot

Related Information

Skills [page 87]
Intents [page 27]
Entities [page 31]
Organizations [page 237]
Versions and Environments [page 248]

8.3 Authentication

Bot developers can authenticate the API calls made by their servers or clients to SAP Conversational AI using one of the following methods.

- OAuth tokens (recommended)
- Bot (developer / request) tokens

Depending on token you choose, you are allowed to make different requests.

With your *request* token, you can make requests in the RUNTIME API , to analyse text or start a conversation. When you create a new bot, by default, your bot has only one main version v1 and is assigned to the development environment. You will have one request token for each version and for each environment of your bot.

With your developer token, you can make requests on every endpoint our API provides.

OAuth vs Bot tokens

OAuth Token	Developer / Request token
Token automatically expires after twelve hours and needs to be regenerated every twelve hours	Token once generated remains the same through out the lifetime of a bot and regeneration of the token has to be triggered manually
More Secure	Less Secure
No need to update credentials in the middleware every time a new token is generated	Must update token in middleware every time a new token is generated
Certificate based token generation is useful for server to server communication	No certificate based token generation possible

> **i Note**
>
> As of February release, you must use OAuth token to authenticate API calls to new bots.

Type of APIs

SAP Conversational AI provides two type of APIs:

Designtime	Runtime
APIs used to configure the bot, for example, /entities, /intents, /dataset and so on	APIs used to interact with the bot, for example, /dialog or /request
Works with developer token of the bot	Works with request, version or environment token of the bot

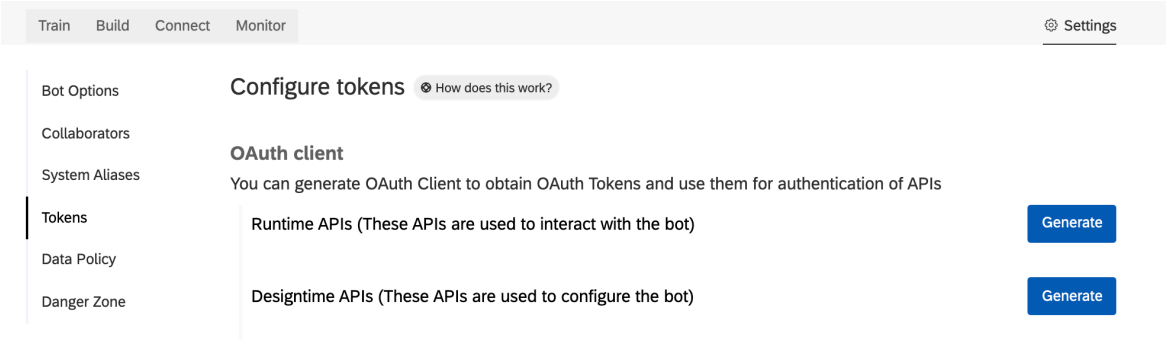
Authentication Using OAuth Token

You need to first generate an OAuth client for designtime or runtime APIs. Using the client credentials or client certificate, the OAuth token is generated that can be used for calling the SAP Conversational APIs.

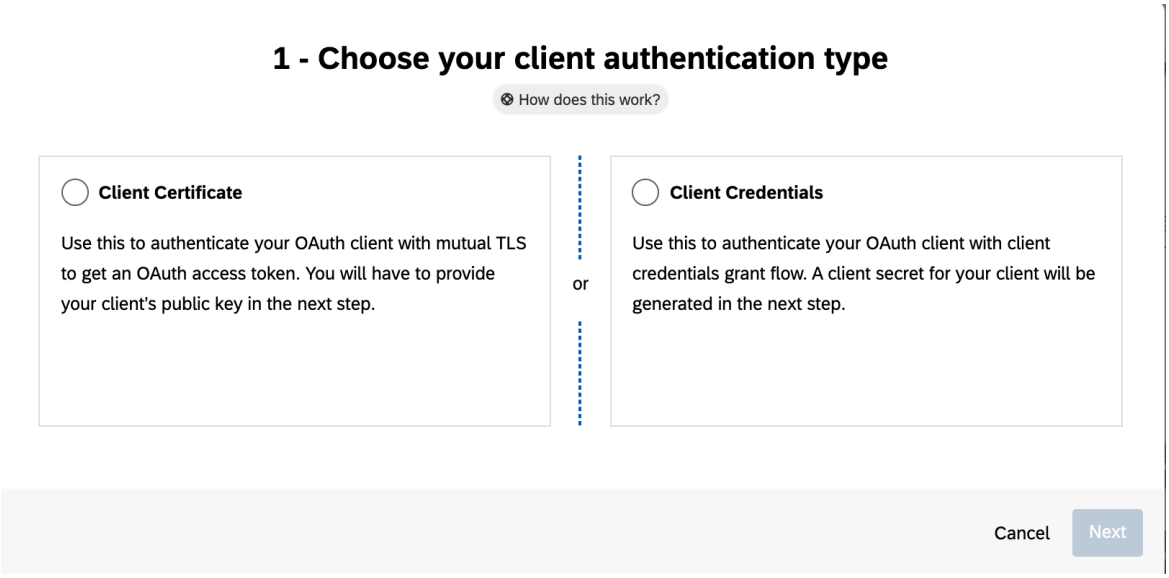
> **i Note**
>
> Before you proceed, please ensure that you have enabled Cloud Foundry for the subaccount subscribed to SAP Conversational AI application in SAP Business Technology Platform. For more information, see Creating a Subaccount and Enabling Cloud Foundry.

Generate OAuth client

1. Go to your bot *Settings* and click *Tokens*.
2. Click *Generate* under the Runtime APIs or Designtime APIs.



3. Choose if you want to authenticate using a client certificate or client credentials and click *Next*.



If you choose *Client Certificate*, you need to paste the certificate code in the text box.

> **i Note**
>
> You can use DigiCert G2 signed certificate only.

Once the client is generated based on any one of the options that you selected, the following fields are displayed:

Field	Use
Auth URL	URL to generate OAuth token
Client ID	Identifier of the OAuth client to be passed in `client_id` field in the request body
Client secret (only in case of client credentials)	Secret for OAuth Client to be passed in `client_secret` field in the request body

> **i Note**
>
> For security reasons, *Client Certificate* and *Client Secret* will be visible only once after generation. They will not be displayed subsequently. Please note the secret immediately after creating an OAuth Client.

> **→ Remember**
>
> If you don't need the OAuth token anymore, it is recommended that you delete the token to avoid any security issues.

Generate OAuth token

You have two options to generate an OAuth token.

- Client Credentials

Sample Code

```
curl -X POST <Auth_URL> -d
'grant_type=client_credentials&client_id=<client_id>&client_secret=<client_
secret>'
```

- Client Certificate

```
curl --cert cert.pem --key key.pem -X POST <Auth_URL> -d
'grant_type=client_x509&client_id=<client_id>'
```

  Here `cert.pem` is the public key certificate and `key.pem` is the private key of the certificate.

If the request is successful, the token is generated and included in the response.

> **Note**
>
> OAuth tokens expire after twelve hours. The value for *expires_in* field indicates the time (in seconds).

Use the OAuth token for calling APIs

To call an API, you need to paste the token in the Headers of the API call.

Designtime APIs	Runtime APIs

Sample Code
`Authorization: Bearer <Designtime_OAuth_Token>`
`X-Token: Token <Developer_Token>`
	Sample Code
	`Authorization: Bearer <Runtime_OAuth_Token>`
	`X-Token: Token <Request_Token>`

Authentication Using Bot Token

In your bot *Settings* choose an appropriate token to be used for making the API calls depending on the type of API (designtime or runtime). Paste the token in the header of your API call.

Designtime APIs	Runtime APIs

Sample Code
`Authorization: Token <Developer_Token>`
or
`X-Token: Token <Developer_Token>`
	Sample Code
	`Authorization: Token <Request_Token>`
	or
	`X-Token: Token <Request_Token>`

8.4 Transporting Bots From User Interface

8.4.1 Overview

You can export your bot from source tenant as a .zip file and import it into the target tenant (enterprise or community) of your choice.

Before you start the export or import process, please ensure that you have fulfilled the pre-requisites as mentioned in Requirements [page 267].

> → Remember
>
> You can export and import only one version at a time from the user interface.

Resources that are not Exported

Not all bot resources are exported using this functionality. After the bot is successfully imported into your target tenant, you need to manually configure the resources that were not exported.

The following table lists the bot artifacts that are not exported -

Resources	Are they Exported?	Details
Environment	No	Default environment is created for a new bot in the target tenant
Tokens	No	You need to regenerate the tokens for the target bot
System Aliases	No	You need to reconfigure the system aliases for the target bot
Log Feed information	No	Only the tenant specific conversations are visible in the Log Feed
Webhook and API authentication information and credentials of the source bot, authentication templates, requirements (under *Skills*) for consuming API authentication	No	You need to reconfigure the authentication information and credentials, authentication templates and requirements for consuming API authentication for the target bot
Channels which the source bot is integrated with	No	You need to integrate your bot to the channels again in your target tenant
Collaborators	No	The target bot does not have any collaborators after it is imported into your tenant

> → Remember
>
> The import will fail if-

- the exported file (.zip) is modified
- a different bot with the same name already exists in the target tenant
- a different version with the same name is already available in the target tenant
- the same bot and version combination already exists in the target tenant

Importing Bots from Community vs Transporting Bots as File

Besides transporting your bot as a .zip file across your community and enterprise tenants, an enterprise user also has the option to import the existing bots (public or private) from the SAP Conversational AI community tenant to the enterprise tenant in the SAP Business Technology Platform. Based on your requirements, you can choose the appropriate method that suits your business needs. The following table lists the differences between these options -

Importing Bots from Community	Transporting Bots as File
You can import the selected bot version from community to enterprise only and not from enterprise to community	You can transport bot versions across community and enterprise tenants
A new bot is created in the enterprise tenant for each bot version that is imported	A new bot is created in the target tenant with initial import, however, with subsequent imports, new versions of the same bot are created
You can choose to copy the Log Feed information of the source bot	Log Feed information of the source bot cannot be copied

8.4.2 Requirements

Before you start the export and import process from the user interface, please ensure that you meet the following requirements.

> **i Note**
>
> You can export and import bots between your enterprise or community tenants or among your enterprise tenants. Therefore, the following requirements are applicable based on the tenant that you are using for import or export.

Requirements for Export

Enterprise Users	Community Users
Your own enterprise tenant in the SAP Business Technology Platform from where you want to export the bot For more information, see Tenant Onboarding	Your own user account or an organization in SAP Conversational AI platform from where you want to export the bot
Bot(s) with version(s) created in your enterprise that belongs to an organization or an individual user account	Bot(s) with version(s) created in your community tenant that belongs to an organization or an individual user account
You are the bot owner (for individual user account) or the administrator of the organization from where you want to export the bot	You are the bot owner (for individual user account) or the administrator of the organization from where you want to export the bot

Requirements for Import

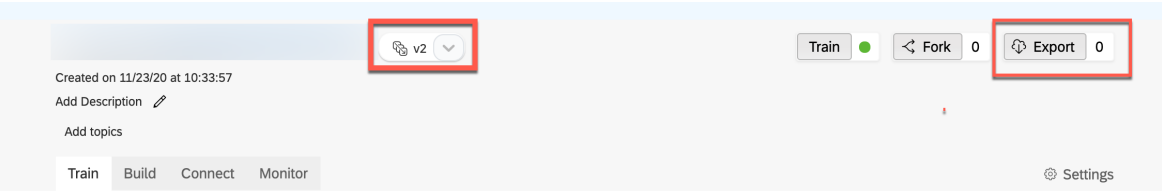
Enterprise Users	Community Users
A target enterprise tenant in the SAP Business Technology Platform where you want to import the bot For more information, see Tenant Onboarding	Your own user account or an organization in SAP Conversational AI platform into which you want to import the bot
You are the account owner (for individual user account) or the administrator of the organization into which you want to import the bot	You are the account owner (for individual user account) or the administrator of the organization into which you want to import the bot

8.4.3 Export and Import your Bots across Tenants

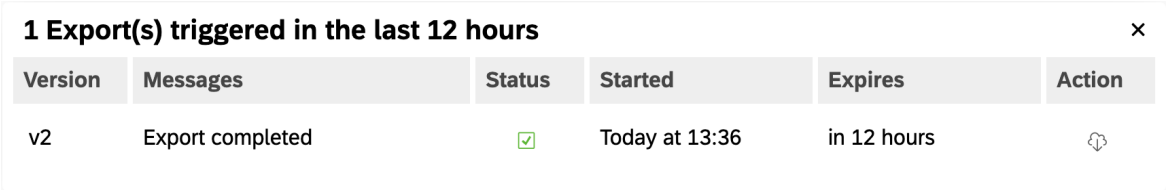
Export a Bot

1. Open the bot that you want to export. Select the required version and click *Export* in the top right corner of the screen.
   The number to the right of the export button indicates the number of export requests that have been triggered for this bot.

   > **i Note**
   >
   > You can only see your own export requests and not the ones triggered by others.



2. Click the number to the right of the export button to see the details of the export request/s for all the versions.

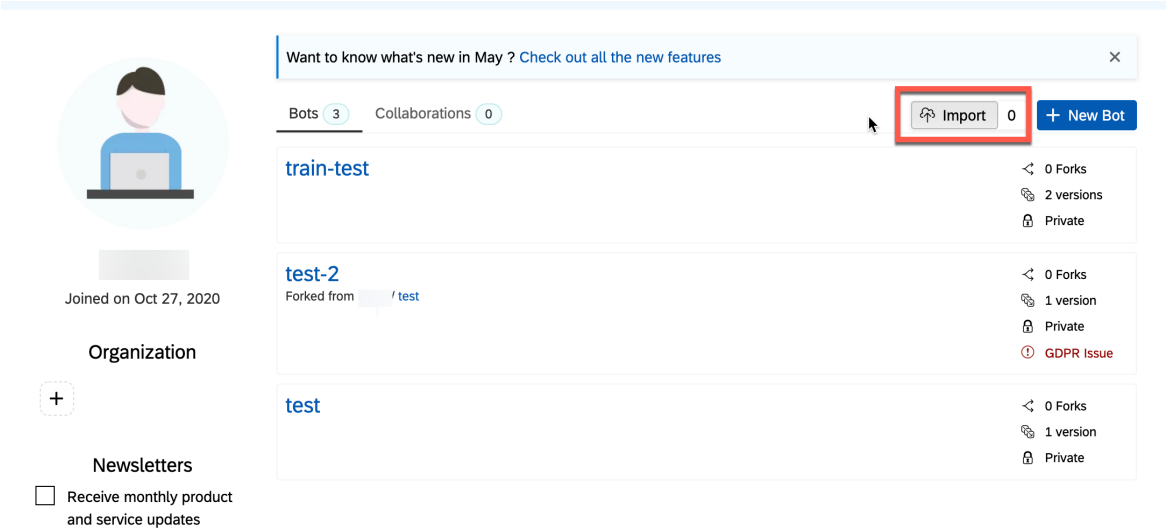


3. After the file is exported, the download icon in the *Action* column is active.
4. Click the download icon.
   A .zip file named as *export_<UUID>.zip* is downloaded and is available in your local (downloads) folder. The file should be downloaded and saved within the next twelve hours of generation after which the export request will expire. You need to re-trigger the export process to download the file if it wasn't downloaded earlier.

> **i Note**
>
> You can change the name of the file as per your choice but the file type must be .zip. The file name can be 5 to 255 characters long (including .zip) and can contain letters, numbers, spaces, underscores, dots and hyphens.
>
> The content of the file must not be modified, else the import will fail.

Import a Bot

1. In your profile page, click *Import* on the top right.
   The number to the right of the import button indicates the number of import requests that have been triggered for this account.



2. Upload the .zip file from your local folder.
3. Click the number to the right of the import button. You can see the details of all the import request/s triggered for this account (including the ones triggered by others).

> **i Note**
>
> All the import requests triggered by other bot developers or org administrators are displayed.

1 Import(s) triggered in the last 12 hours ×

Bot name	Version	Started	Status	Messages
-participant-data	v2	Today at 13:27		Import is in progress

4. Refresh the page. The imported bot is visible in your profile page.

> **i Note**
>
> After you export your bot from the source tenant and import it into the target tenant, it is not required to manually train the dataset in the target tenant. However, the bot behaviour may slightly vary if the manual training is launched without any change in the dataset.

For more information, see the following video:

9 Getting Started with FAQ Bot

9.1 Introduction and Prerequisites

A FAQ bot retrieves answers to users' questions from one or more documents (.csv file) that you upload. The document must include predefined pairs of questions and answers. This allows your bot to map the user's query to the best match and retrieve an answer without interpreting the intent of the question.

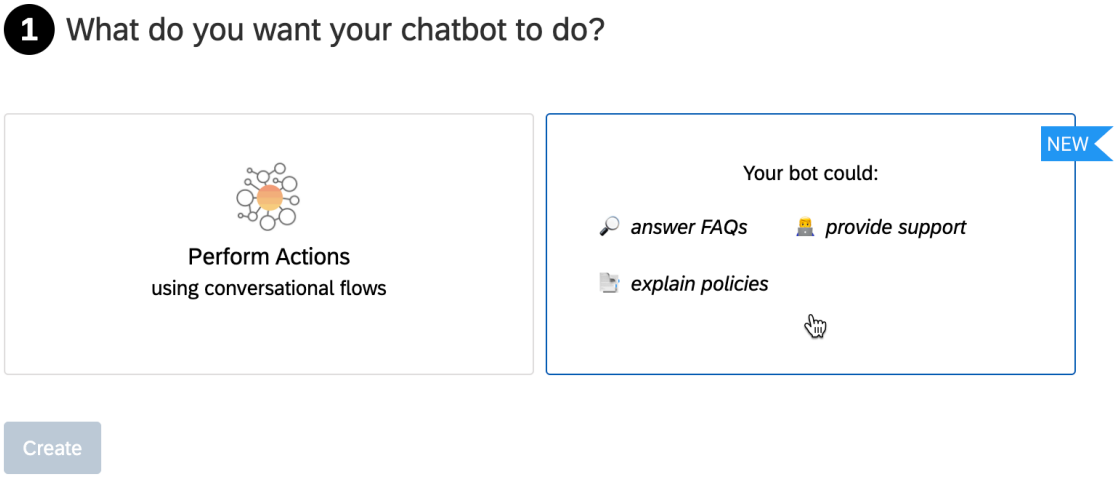
To ease the complexity of the bot, the intents and entities are pretrained, and the bot includes a set of predefined skills. However, you can design the bot responses as per your business needs.

> **i Note**
>
> The FAQ bot is currently supported for advanced languages only. For more information, see Advanced Level Languages [page 80]

9.2 Create an FAQ Bot

Procedure

1. Click the blue *+ New Bot* button on the right side of the *Bots* tab.
2. On the next page, click the *Retrieve Answers* tile.



3. Enter a name, description (optional), and other data, and click *Create*. For more information, see Create Your Chatbot [page 9].

Results

Your bot opens on the *Train* tab.

9.3 Upload Your FAQ Document

Once you've created your bot, it will open on the *Train* tab, where you can upload your FAQ files (.csv extension).

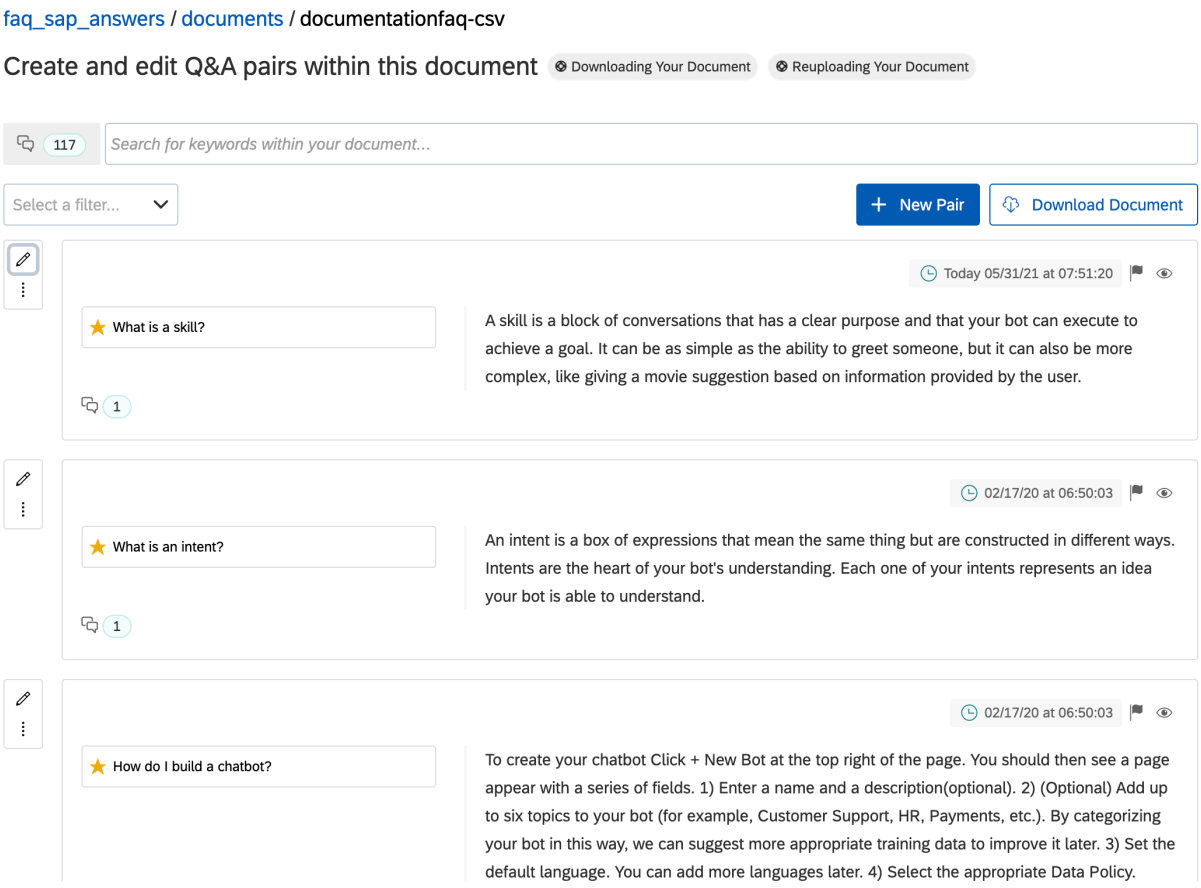
Document Requirements for FAQ File

- The file is a UTF-8 encoded CSV file. For more information, see CSV UTF-8 standards .
- Your file has either comma, semicolon, or pipe separated values, based on the default settings maintained in your computer
- Your file size is no more than 10 MB
- Your document has no more than 20,000 questions and answers.
- Answers have no more than 2000 characters.
- Questions have no more than 500 characters
- The FAQ file has two columns: one for questions and one for answers.
- You have used Markdown syntax for adding hyperlinks to text.

Template for FAQ file

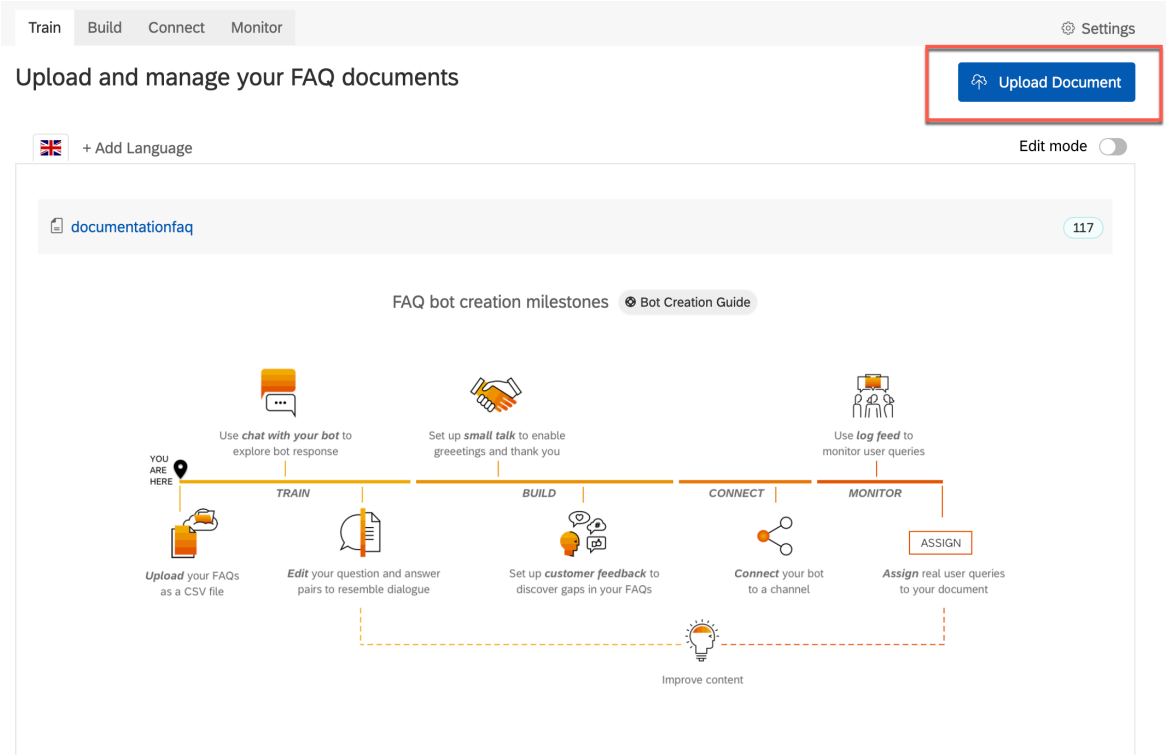
The FAQ file (.csv extension) must have two columns for questions and answers. For each question (in the left column), an answer should be provided in the right column, as shown in the following table.

Questions	Answers
Q1	A1
Q2	A2
Q3	A3
Q4	A4
Q5	A5
Q6	A6



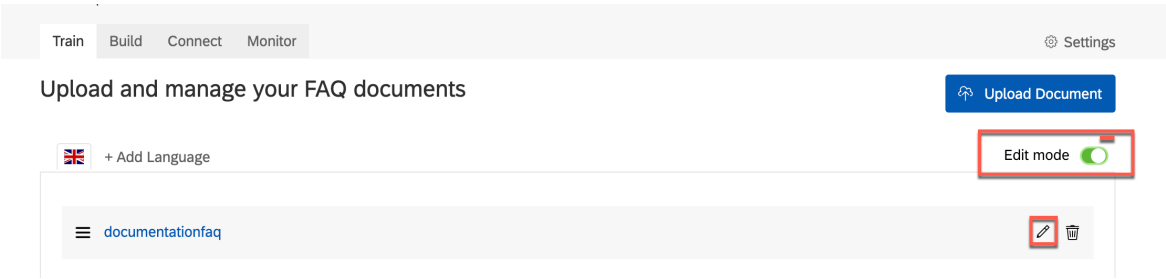
To upload your document:

1. On the *Train* tab, click *Upload Document*.
   Either drag and drop the file or upload it from your local drive.



You'll know when your file has finished uploading because the progress icon will stop spinning, and a message will tell you that your file has been uploaded.
The file extension -csv is removed from your file name.

2. Optionally, you can change the name of the uploaded file. Switch on the edit mode and click the pencil icon on the right. Type the new file name and press `Enter`.
   Note that if you provide a file name that is same as an already uploaded file, -1 is automatically appended for uniqueness, for example *bot-build-1*.
   The FAQ file name is changed.



3. To upload another file, click *Upload Document* again.

> **! Restriction**
>
> You cannot upload two files with the same name.

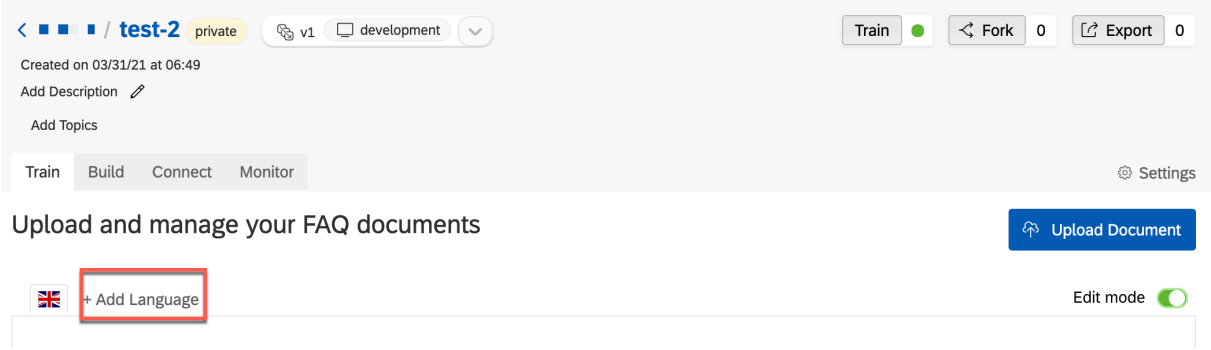
> **i Note**
>
> For better content management, you may want to upload multiple files. But although the content is in separate files, it is treated as if it were all in one file.

4. Newly uploaded files always appear on top of the list as they are sorted by creation date. If you want to change the order of the files, switch to edit mode, grab the icon on the left to drag and drop the file as needed.

To upload your files in multiple languages

You can upload files in four languages: German, French, Spanish, and English.

Click *+Add Language* and select the language in which you wish to upload your content.



A tab is created for the selected language. Click the appropriate language tab and upload the file that has content in that particular language.

> **i Note**
>
> The language tabs are automatically added to the skills as well. If you want change the default configurations, you must do this for each language individually.

Related Information

Upload a Document with Alternative Questions [page 276]

9.4 Edit your FAQ Document

After uploading your FAQ document, you can add or remove question and answer pairs and perform various other actions directly on the screen. Here are some of the things you can do:

Action	Description
Search	Search for FAQ pairs with keywords. All the FAQ pairs that have the keyword are displayed.

Action	Description
Edit	Edit the content and enter multiple questions for the same answer. You can mark one of them as the display question.
Flag	This is to notify your team member that a question and answer pair has been added or changed.
Disable matching	You can do this to ensure that the content is not included in the bot's response. This is useful when the content is still in the draft state and not ready to be included in the bot's response.
Filter	Filter the content based on the status.
Delete	Delete the question and answer pairs
Download	Download the content in .csv format. For security purpose, the content in the downloaded document may include some special characters like backslashes. These are removed upon re-uploading.

9.5 Upload a Document with Alternative Questions

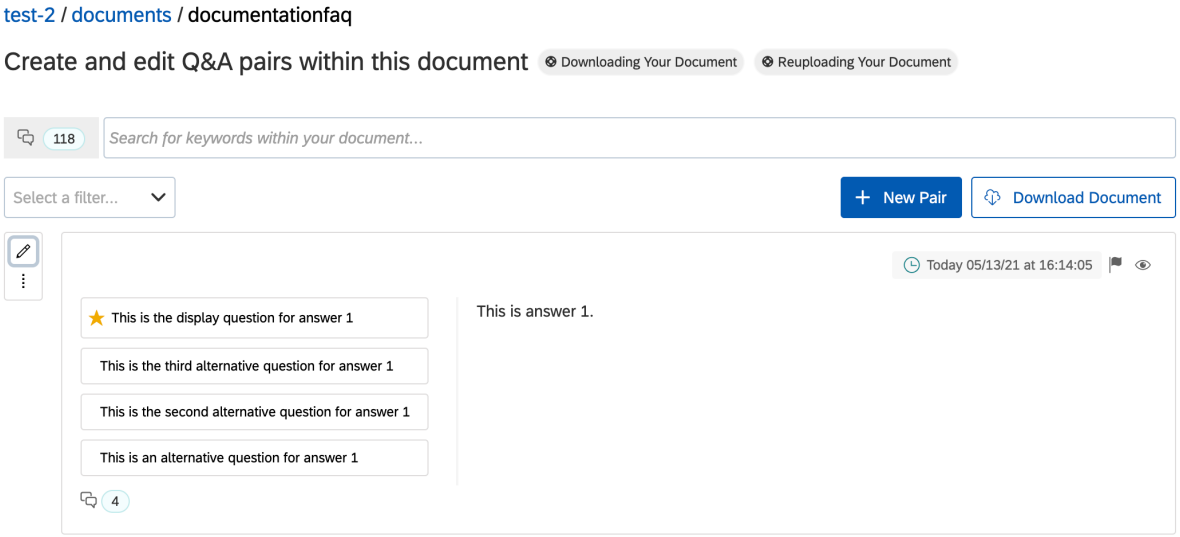
You can upload a file that includes multiple or alternative questions for the same answer and use that content to train the bot.

Template for FAQ file with alternative questions

The FAQ file (.csv extension) must have two columns for questions and answers. If there are multiple questions that have the same answer, add them in the format shown in the following table. In this example, questions 2 and 3 are the alternative questions for answer 1 while questions 5 and 6 alternative questions for answer 2.

Questions	Answers
Q1	A1
Q2	
Q3	
Q4	A2
Q5	
Q6	

Please download the sample template that can be used to create the FAQ document with alternative questions.



For more information, see Upload Your FAQ Document [page 272]

9.6 Start Chatting with Your Bot

Once you've uploaded your file, you can start chatting with your bot. We've pre-configured the following settings:

- When a user asks a question that is similar to a question in the Q&A document, you'll see the top mapped response. If the Q&A result matches with a 90% confidence or higher, your bot responds with one answer. If the result is lower than 90%, the bot responds with a piece of text and gives the user the option of choosing from the top three best matched questions.
- If no answer is found, the following message is sent: *I was not able to find what you were looking for in my document.*

You can design the conversation as per your business needs.

Explore your Bot Responses

You can chat with your bot or use the *Expression Analysis* console to see how your bot responds.

Expression Analysis

The Expression Analysis console shows you the top ten matches including the question, the answer, and the confidence score of your bot's response, independent of the way you have configured your bot. This can be helpful for training purposes. You can use this information to decide if the ranking is in a suitable range and if a sentence should be added as an alternative question.

Chat Preview

This shows you the bot's response as your end users will see it. This helps you to decide if you want to add additional messages like a prompt, or change the way the content is displayed.

Recommended Q&A Best Practices

- Beware of answers that require context of the question. For example, *Can I fly first class? Yes.* In this scenario, it would be best to show the question and the answer or change the answer so that it rephrases the question.
- The length of the message should be ideal for a chatbot to display in the chat window. Therefore, it is recommended that you limit the answer to 640 characters.
- If you have an answer that is longer than the recommended limit, summarize the key points and redirect the user to a web-page or an online resource for more information. You must use markdown to hyperlink a word. For more information, see Messages.
- Make sure your questions are natural language queries and not key words.

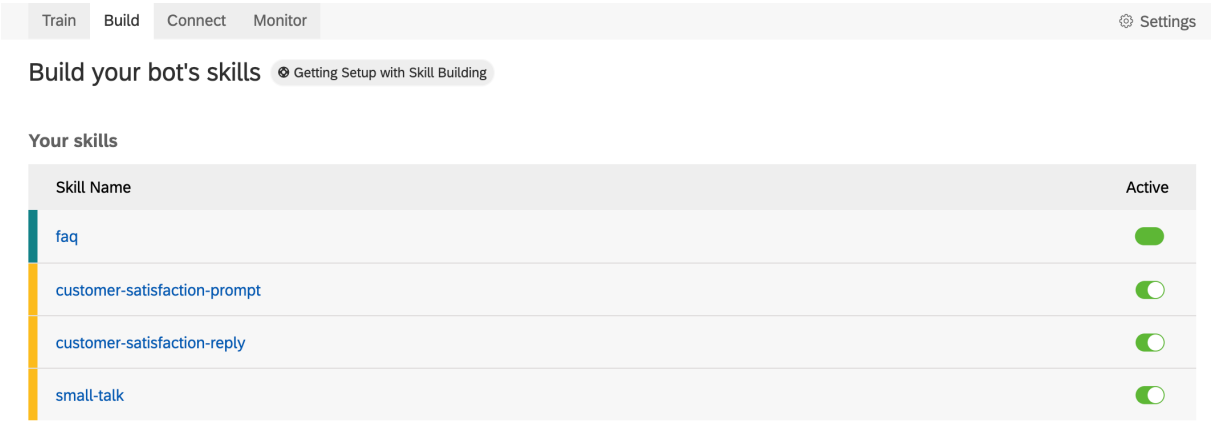
9.6.1 Skills

The FAQ bot includes a set of predefined skills that allow you to design and manage the conversation flow.

There are four predefined skills that can be used out of the box or adapted as per your business needs.

- FAQ
  Enables you to send customized message using the confidence score of the FAQ text.
- Customer Satisfaction Prompt
  Enables your bot to ask for feedback when the user selects a response.
- Customer Satisfaction Reply
  Enables your bot to reply when the user selects a response.
- Small talk
  Enables your bot to greet the user with *hello*, *thank you* or *goodbye*.

> **i Note**
>
> You can choose to deactivate the two customer satisfaction skills and the small talk skill. You can add the responses defined in these skills within the FAQ skill itself.



Related Information

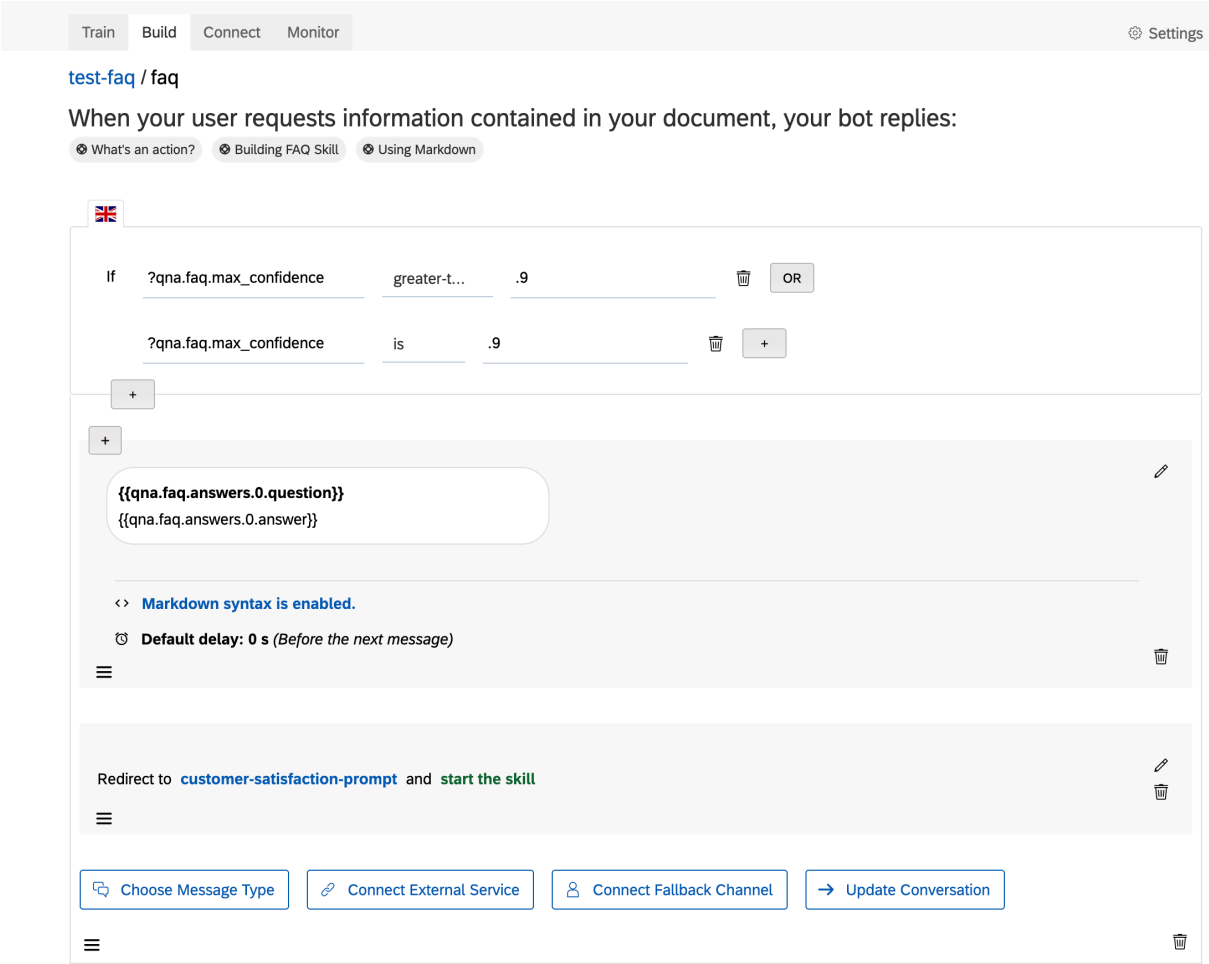
FAQ Skill [page 279]
Customer Satisfaction Prompt Skill [page 283]
Customer Satisfaction Reply Skill [page 284]
Small Talk Skill [page 285]

9.6.1.1 FAQ Skill

The FAQ skill enables your bot to match the user's utterance to the most accurate question and answer pair in the FAQ document that you have uploaded. This is determined by a number called *confidence score* or the matching score. The confidence score indicates the accuracy with which your bot pairs the query with the most matching answer. The greater the confidence score, the higher the accuracy of the response.

Typing a question mark lets you choose the maximum confidence score or minimum confidence score (`?qna.faq.max.confidence`). By default, three different ranges of the score are defined as follows:

Scenario	Confidence score	Result
If the bot most likely knows the answer to the user's question	90–100%	The bot returns an accurate response and then routes to the customer satisfaction prompt skill.
If the bot isn't sure and needs some more user input	5–90%	The bot returns multiple questions with highest scores. It prompts the user to choose the most appropriate question and then provides a suitable response.
If the bot doesn't know the answer	0–5%	The bot redirects the user to the fallback channel.



> **i Note**
>
> You can adjust the confidence score as per your requirement and add more ranges using the `?qna.faq.max.confidence` operand.

Based on the system's confidence in its match to a question, you can send a specific message to the user. You can define up to ten responses for a given user question. Please note that the index begins at zero.

Q&A Pair	Question	Answer
1	`{{qna.faq.answers.0.question}}`	`{{qna.faq.answers.0.answer}}`
2	`{{qna.faq.answers.1.question}}`	`{{qna.faq.answers.1.answer}}`
3	`{{qna.faq.answers.2.question}}`	`{{qna.faq.answers.2.answer}}`

Q&A Pair	Question	Answer
4	`{{qna.faq.answers.3.question}}`	`{{qna.faq.answers.3.answer}}`
5	`{{qna.faq.answers.4.question}}`	`{{qna.faq.answers.4.answer}}`
6	`{{qna.faq.answers.5.question}}`	`{{qna.faq.answers.5.answer}}`
7	`{{qna.faq.answers.6.question}}`	`{{qna.faq.answers.6.answer}}`
8	`{{qna.faq.answers.7.question}}`	`{{qna.faq.answers.7.answer}}`
9	`{{qna.faq.answers.8.question}}`	`{{qna.faq.answers.8.answer}}`
10	`{{qna.faq.answers.9.question}}`	`{{qna.faq.answers.9.answer}}`

Additionally, you can:

- Include the question in the bot's response.
- Set up a fallback channel to redirect your user to a live agent if the response is in the lowest confidence range.

Include the Question in the Bot's Response

You can include the question as well as the answer in the bot's reply to the user. We store up to the top ten responses for a given user question if you increment what you find in the preconfigured FAQ skill you can display more answers.

In a message block, include the following text. Note that the index begins at 0.

Top ranked question: `{{qna.faq.answers.0.question}}`

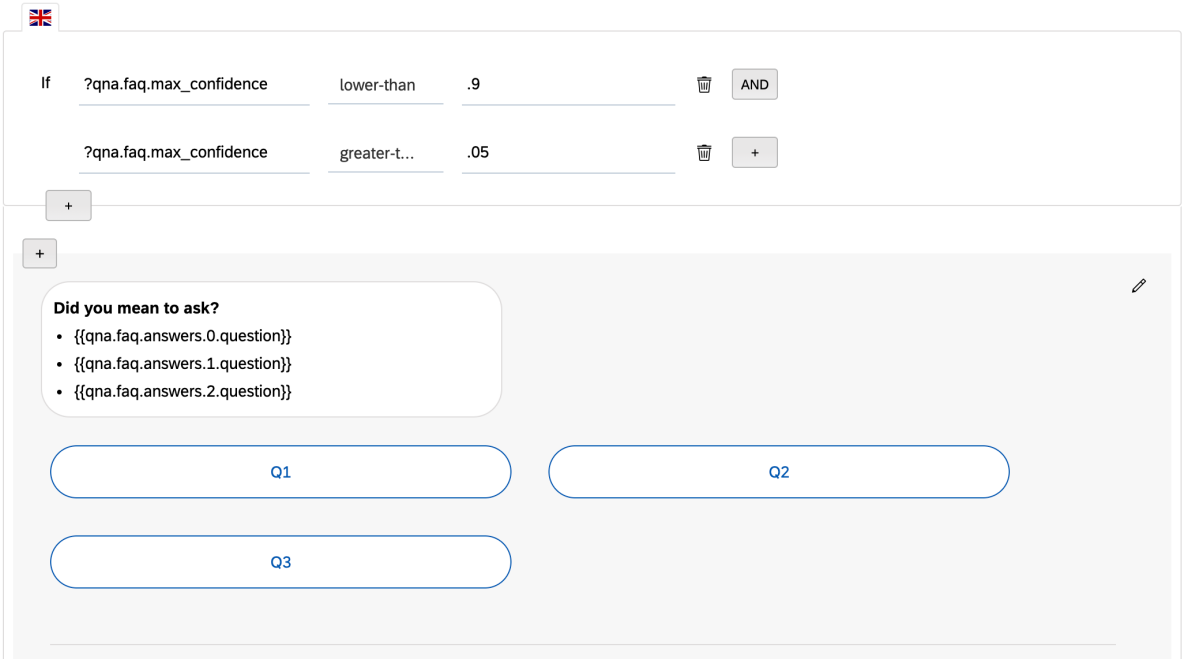
Top ranked answer: `{{qna.faq.answers.0.answer}}`

Second ranked question: `{{qna.faq.answers.1.question}}`

Second ranked answer: `{{qna.faq.answers.1.answer}}`

Third ranked question: `{{qna.faq.answers.2.question}}`

Third ranked answer: `{{qna.faq.answers.2.answer}}`



> **i Note**
>
> You can choose to send messages in different formats like text, card, buttons, list, and so on. For more information, see Messages [page 114].

Set up a Fallback Channel

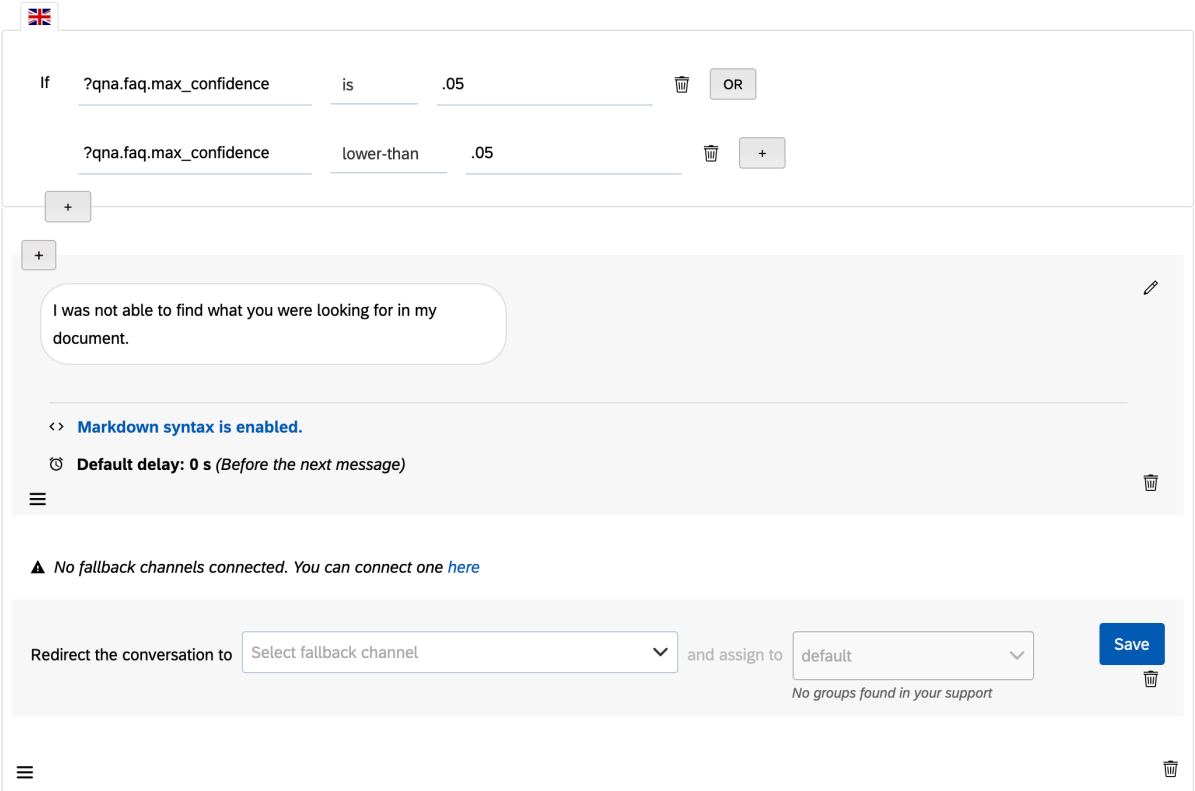
If your bot us unable to match the user's query to a suitable question and answer pair, it redirects the user to a fallback channel.

Before you proceed, you must first configure a fallback channel. For details of how to do this, refer to Actions [page 108].

On the *Build* tab, open the FAQ skill. Go to the response defined for the conversation with the lowest confidence score.

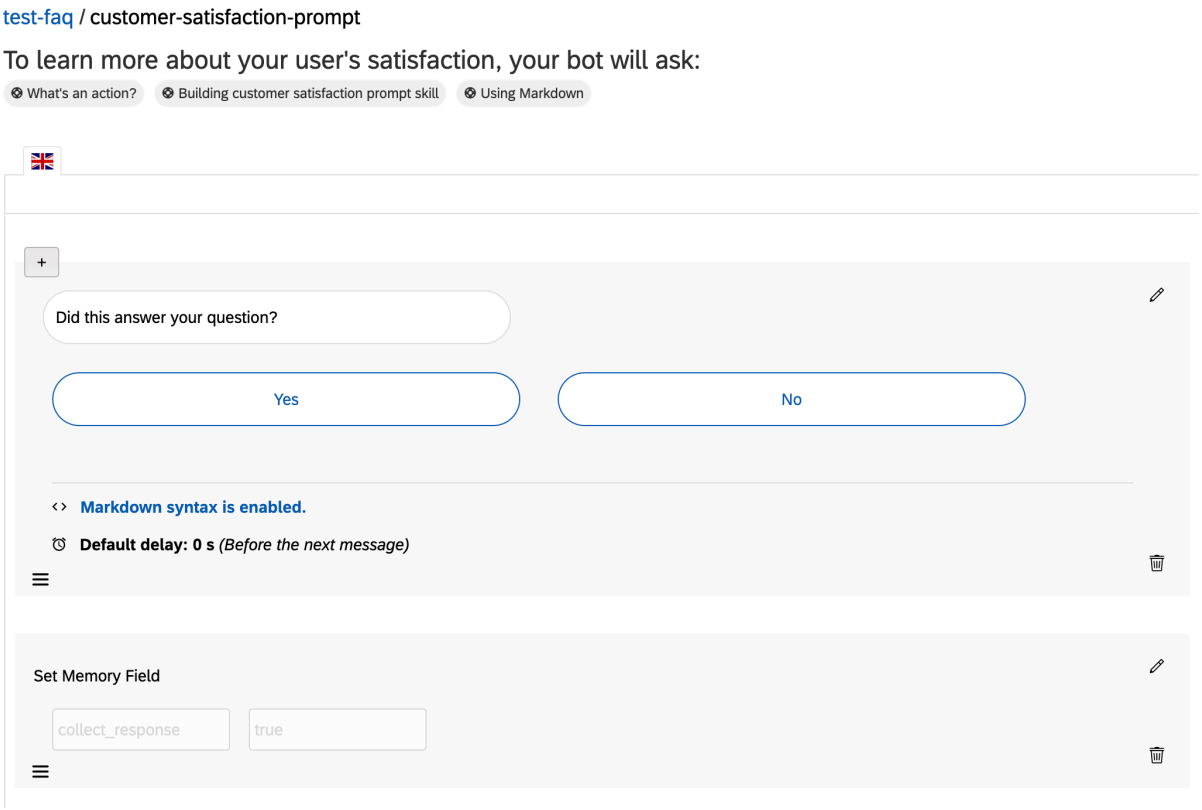
Click the *Fallback* button. Select the fallback channel and the group to which you want to redirect the conversation.

> **i Note**
>
> You can edit the default message and adapt it as per your requirement.



9.6.1.2 Customer Satisfaction Prompt Skill

In this skill, you define the message that is sent to your user to ask for feedback when the user chooses a response. If you change or accidentally erase one of the configurations, here is a reminder of what the default settings looked like.



> **⚠ Caution**
>
> We recommend not to change the postback value of this skill as the response will not work with our pre-trained classifier.

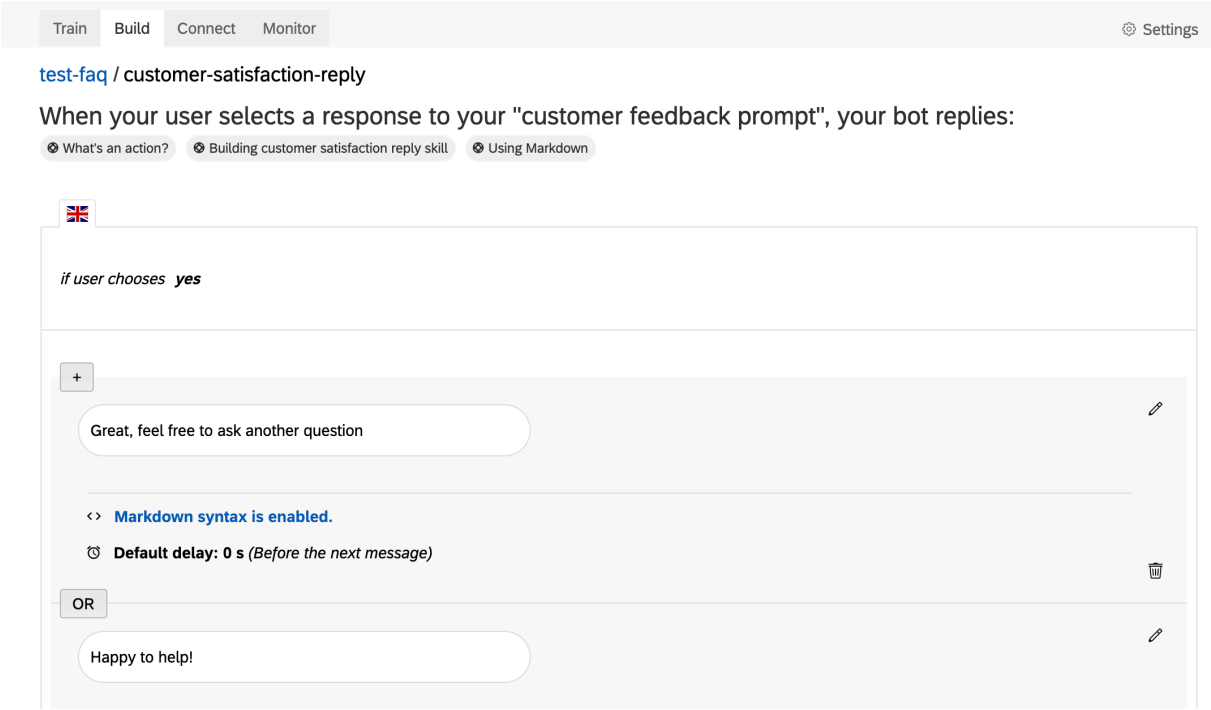
> **ℹ Note**
>
> You can choose to send messages in different formats like text, card, buttons. list and so on. For more information, see Messages [page 114].

Related Information

Memory Management [page 111]

9.6.1.3 Customer Satisfaction Reply Skill

In this skill you define the message that is sent to your user, after receiving the user's feedback. If a user says something that wasn't helpful, you can guide the user to valuable information or sources – for example, a link to documentation, or the email address of someone they can reach out to. You can even set up a fallback to connect the user to a human who will pick up the conversation. For more information, see Actions [page 108].



> **i Note**
>
> You can choose to send messages in different formats like text, card, buttons. list and so on. For more information, see Messages [page 114].

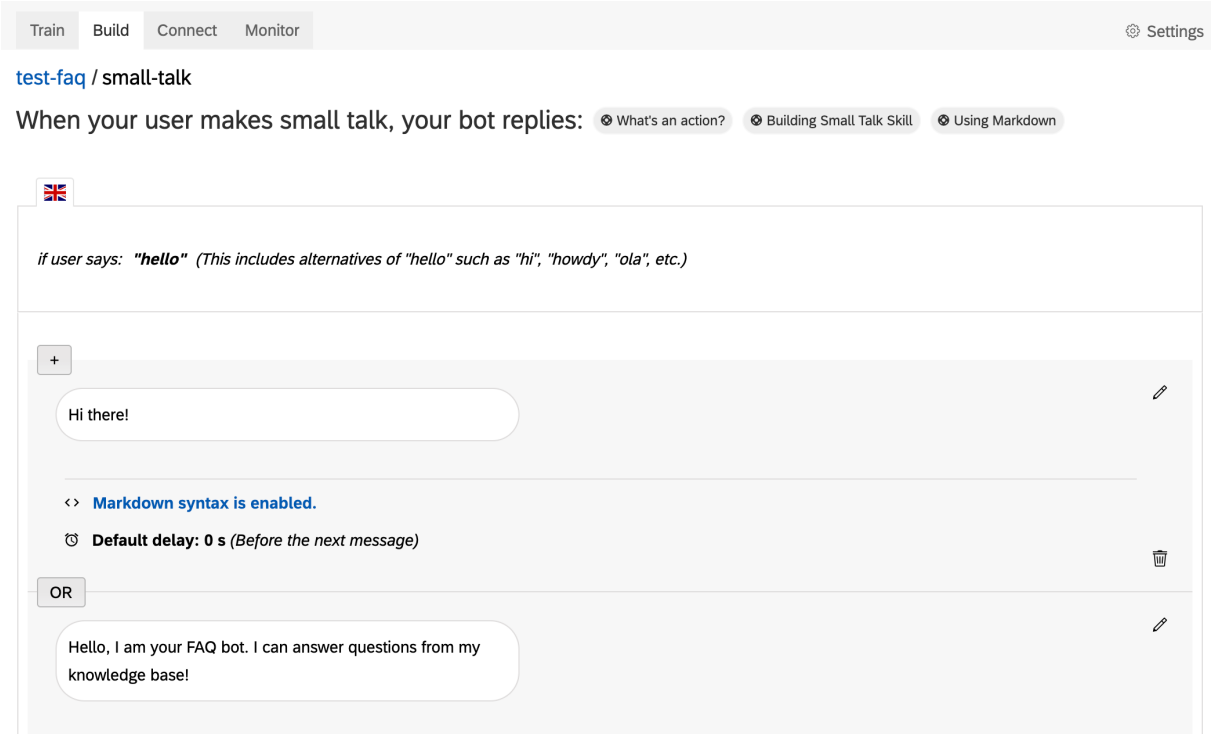
9.6.1.4 Small Talk Skill

Small talk adds human quality to your bot and can reduce the frequency with which the fallback channel is triggered.

While defining the small talk messages, think about your users and their expectations from the bot. The bot should be able to connect to the users with appropriate greetings. For example, when a user says *hello*, the bot should be able to introduce itself and explain what it can do. Provide alternative ways to greet the users.

A goodbye message to end a conversation is an opportunity to collect feedback on the overall bot experience or leave the user with a parting thought.

Make sure that every small talk leads to a call to action. For example, *Hello! How can I help you?*



> **i Note**
>
> You can choose to send messages in different formats like text, card, buttons, list, and so on. For more information, see Messages [page 114].

9.6.2 Custom Normalization

Custom normalization transforms words or word sequences into a canonicalized (standard or normal) format.

During a chat, a user might type phrases which are different from what the content creator (bot developer) had in mind, for example, S4 HANA (without a backslash and with a space) instead of S/4HANA.

The custom normalization feature lets your bot identify and correctly process these cases without imposing on your users how they should phrase their information needs.

Besides creating these equivalency classes, custom normalizations can also be used to create sets of synonyms. This feature is especially useful when the covered terms are domain specific or derived from a non-standard vocabulary.

Before you use the API, it is important to understand how it can impact your chatbot's performance. When you normalize a word, it will cause the system to override nuances that it would normally pay attention to which can lead to a higher false positive rate.

Here are some best practices and things to consider when using the API for domain specific information, abbreviations, and spelling errors.

Best Practices for Domain Specific Information

If there are variations in how people refer to terms that exist in your document, it is a good idea to add normalizations. Let's continue with the example for SAP S/4 HANA.

SAP S/4 HANA is the official name of the product but a user can refer to this product in various ways, like S4, S/4 HANA, S4HANA, S4 HANA, HANA, SAP HANA. Since the bot is not trained to consider all these words as SAP S/4 HANA, it will treat each character as special and will emphasize the differences. It will route questions about S4 HANA (with a space and without a backslash) to other instances of S4 HANA (with a space and without a backslash).

Therefore, if you have words with different variations, but they all mean the same thing, you can create a normalization to group them together.

Best Practices for Abbreviations

If your content has a lot of abbreviations and/or if you users tend to use a lot of abbreviations, you can add these to a list of normalized words. It is important to be intentional about which word you equalize as it will impact the way your bot retrieves the answer. The following example illustrates this scenario.

You can either adjust the word(s) to the abbreviation or the abbreviation to the word(s). Let's take the example of an abbreviation - *Super Fun Abbreviation Example* is abbreviated to *SFAE*.

Adjusting the word to the abbreviation

In this case, the system replaces all instances of *Super Fun Abbreviation Example* to *SFAE*. This is a stricter approach to matching user input with the content.

Question: `How can you use SFAE in your documentation?`

Answer: `You can use a super fun abbreviation example by following steps one and two and then proceed to step three.`

System Adjustment

Question: `How can you use SFAE in your documentation?`

Answer: `You can use SFAE by following steps one and two and then proceed to step three.`

If a user asks a question about *super awesome abbreviations*, there will be a weaker match as the system doesn't compare the word *fun* with *awesome* (has a similar meaning) but rather with *SFAE* (which does not have a meaning).

Adjusting the abbreviation to the word

In this case, the system replaces all instances of *SFAE* to *Super Fun Abbreviation Example*. This is a less stricter approach to matching user input with the content..

Question: `How can you use SFAE in your documentation?`

Answer: `You can use super fun abbreviation example by following steps one and two and then proceed to step three.`

System Adjustment

Question: `How can you use a super fun abbreviation example in your documentation?`

Answer: `You can use a super fun abbreviation example by following steps one and two and then proceed to step three.`

If a user asks a question about *super awesome abbreviations*, there will be a stronger match as the system compares the word *fun* (has a meaning) with *awesome*, that has a similar meaning.

Best Practices for Spelling Errors

If the content owners are in different locations, there can be inconsistencies in spellings. If a content owner makes a spelling error, the system assumes it as intentional and if the user makes the same spelling error, it will provide that as the top answer.

Example

All the content is written in American English except for one answer that is written in British English. There are two spellings of the same word - *color* (American English) and *colour* (British English).

If a user asks a question with the word *colour*, the bot may retrieve the answer that is in British English. This happens as the word *colour* is treated unique even if the rest of the question is more similar to the one in American English.

In this case, you can create a word normalization so that both *color* and *colour* will be treated equally and the spelling will not impact the rankings of the answer .

Getting Started

Creating a normalization

First, you need to provide a JSON object as the value for the `custom_normalizations` parameter in the /topic endpoint.

An example of such a JSON object is provided in Updating A Topic in the API Reference.

Using this parameter, you can do the following actions:

Update a topic with the `custom_normalizations` parameter to:

- Change the normalization
  Update the topic with a new object that will replace the existing one
- Delete the normalization
  Update the topic with `custom_normalizations = {}` (an empty object)

See current normalization configuration

In order to see which normalizations are currently set, you should call the Showing a topic endpoint.

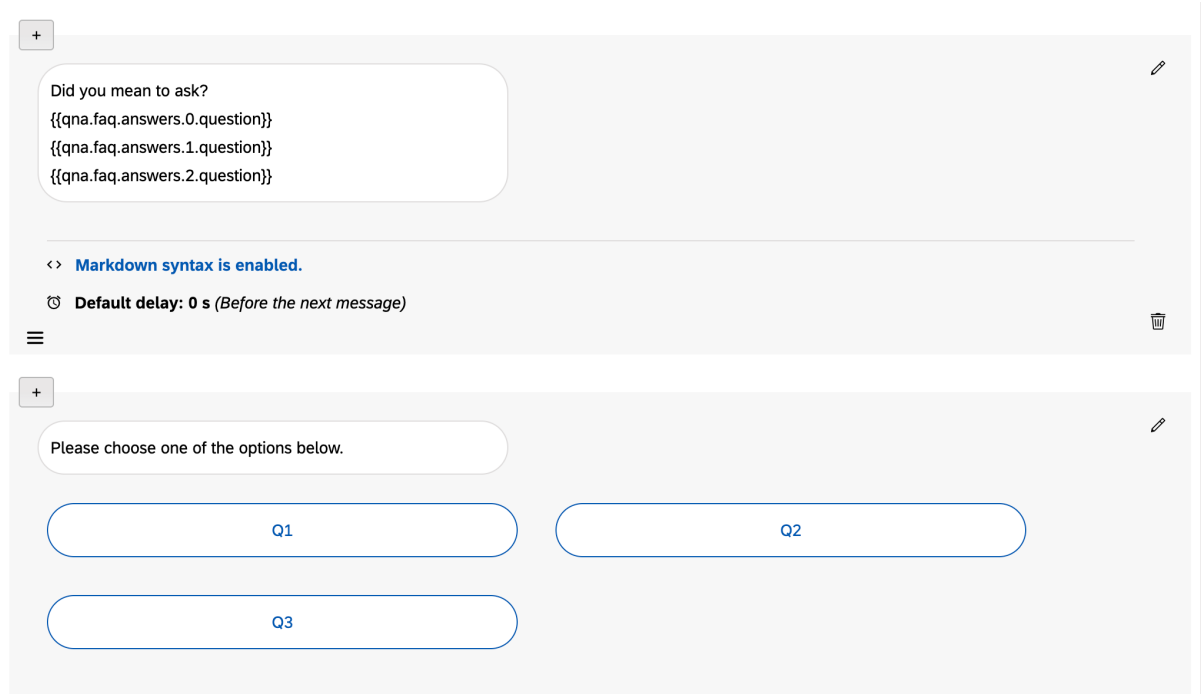
9.7 Connect Your Bot to a Channel

You can connect your FAQ bot to multiple channels in the *Connect* tab.

This lets you utilize resources like persistent static menu (on connecting your chatbot to SAP Web Client and/or Messenger) that are always available to the users, enabling them to quickly trigger certain skills at any point in the conversation. To learn how to connect your bot to different channels, see Messaging Channels [page 181].

> **! Restriction**
>
> The default response configuration for the FAQ bot is not supported by Microsoft channels (Skype, Teams).
>
> Long titles and markdown syntax are not supported in quick replies, but can be used in text messages.

To make sure that bot responses are displayed properly in the Microsoft channels, you can use the text message for the question (using markdown) and quick replies to provide options to the user in the faq skill as shown in the following figure:



9.8 Monitor Your Bot Conversations

You can monitor your bot's activity and view and analyze all its conversations with users. Based on this information, you can refine the bot activity.

Related Information

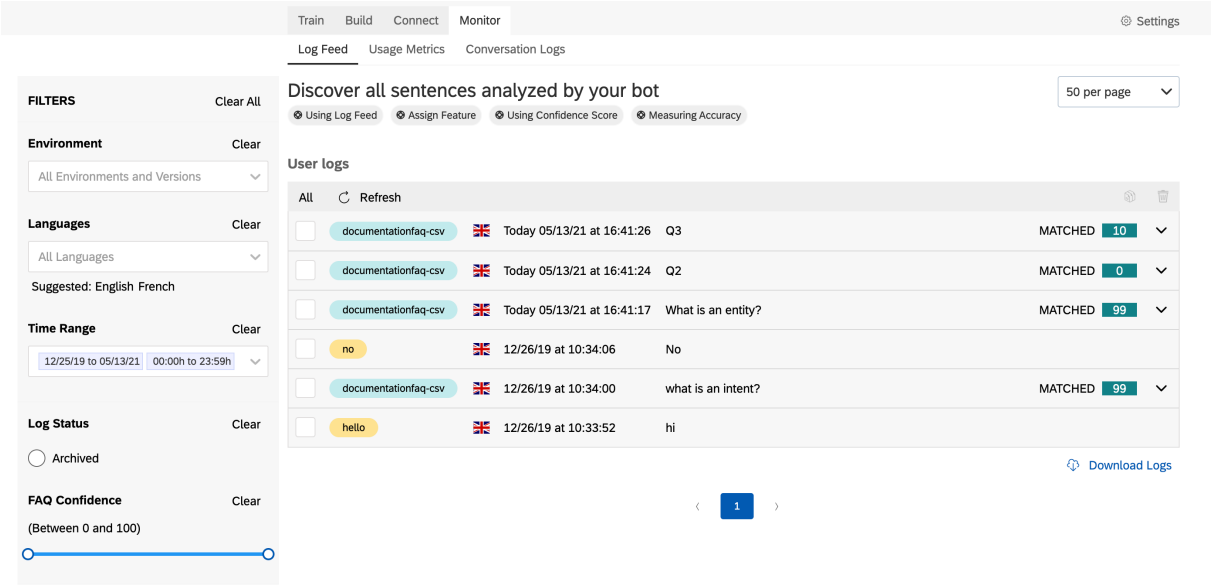
Monitor Conversations Using Log Feed [page 290]
Assign Users' Questions to an Answer [page 291]

9.8.1 Monitor Conversations Using Log Feed

1. On the *Monitor* tab, go to *Log Feed*.
   The screen shows a list of all the sentences that were analyzed by your bot.
   On the right of the screen, next to a user utterance, you'll see either the word *MATCHED* or *UNMATCHED* together with a number. This indicates which questions are not getting answered and which responses were successfully retrieved. The number is the confidence score, which means how confident your bot feels when retrieving a reply for your user's question.

   > **i Note**
   >
   > You can adjust the FAQ confidence in the *FILTERS* panel on the left. A threshold of 40 or below is considered to be a low threshold.

2. Click the arrow next to the confidence score.
   The top matched FAQ pairs are shown ranked by confidence.



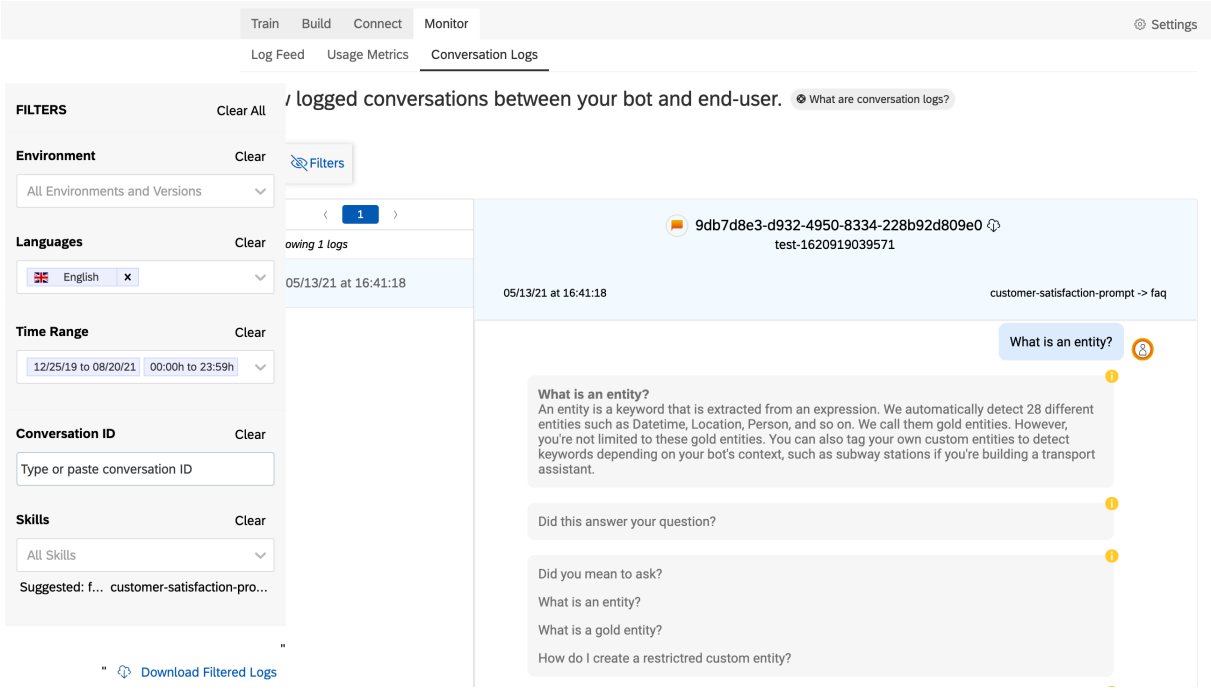
9.8.2 View Conversation Logs

As a bot developer, you can access the complete conversation between your FAQ bot and the users. The *Conversation Logs* tab shows each utterance of your user and the bot replies.

You can analyze if your bot is able to understand the user's questions and respond with an appropriate answer. It helps you determine the flow of skill execution based on user utterances. This helps you to enhance the quality and accuracy of the bot response and improve the conversation design.

How to filter conversation logs

Upon accessing the conversation logs tab, there are some predefined filters that are already set. You can define the filters as per your need to fetch the conversations and their details. You can filter the conversations based on *Environment*, *Time range*, *Language detected*, *Timezone*, *Conversation ID* and *Skills*. Based on this, all the conversations are displayed in the conversation panel. For more information, see Conversation Logs [page 234].

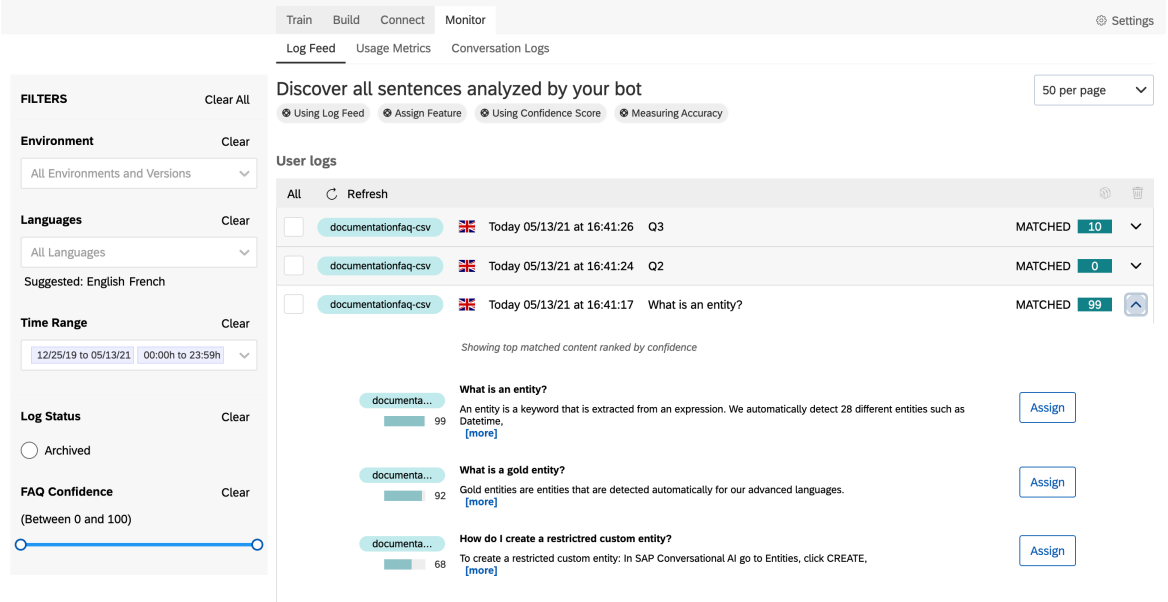


9.8.3 Assign Users' Questions to an Answer

A user can ask the same question in different ways. If your bot is unable to answer the user's question or retrieves an incorrect response, you can directly map that question to an answer in your FAQ document from the log feed. This improves the performance, and the bot can learn directly from what a user says.

1. In the *Log Feed* tab, click the arrow next to the question that was matched.
   The top matched FAQ (question and answer) pairs are shown ranked by confidence, depending on the number of results returned from the search.
2. Click *ASSIGN* next to the FAQ pair.

> **i Note**
>
> Make sure that you only assign unique sample user questions because if two answers are associated with similar or overlapping sample user questions, your bot will not know which FAQ pair should be the best response.



3. In the popup, edit the user's question if any grammatical changes are needed, and click *Assign*.
   The sample user question has been assigned to the selected FAQ pair and is added to the FAQ document that you uploaded in the *Train* tab.
   You can unassign a user's question from the log feed by clicking *Unassign*.

9.8.4 Measure FAQ Bot Accuracy

You can use the benchmarking script to analyze your FAQ bot's accuracy in responding to the user's questions.

For this, you need the gold dataset and can measure up to three levels of accuracy.

Requirements

- You have created a user account and an FAQ bot in SAP Conversational AI platform.
- You have installed Python 3 including the pip3 package manager, in order to use this report.

Gold Dataset

You need a gold dataset (CSV format) that includes two columns - questions and answers. Each row of the CSV should contain the question and the corresponding expected answer that the bot should respond with in the best case. Note that the answers in the gold dataset need to be exactly the same as in the FAQ file. This is required to generate meaningful metrics as listed below.

> **i Note**
>
> Please refer to the `gold_test_data.csv` file in the `assets/data` directory in the repository for a sample gold dataset for an FAQ bot set up with the corresponding question and answers from the `bot_faq_data.csv` file.

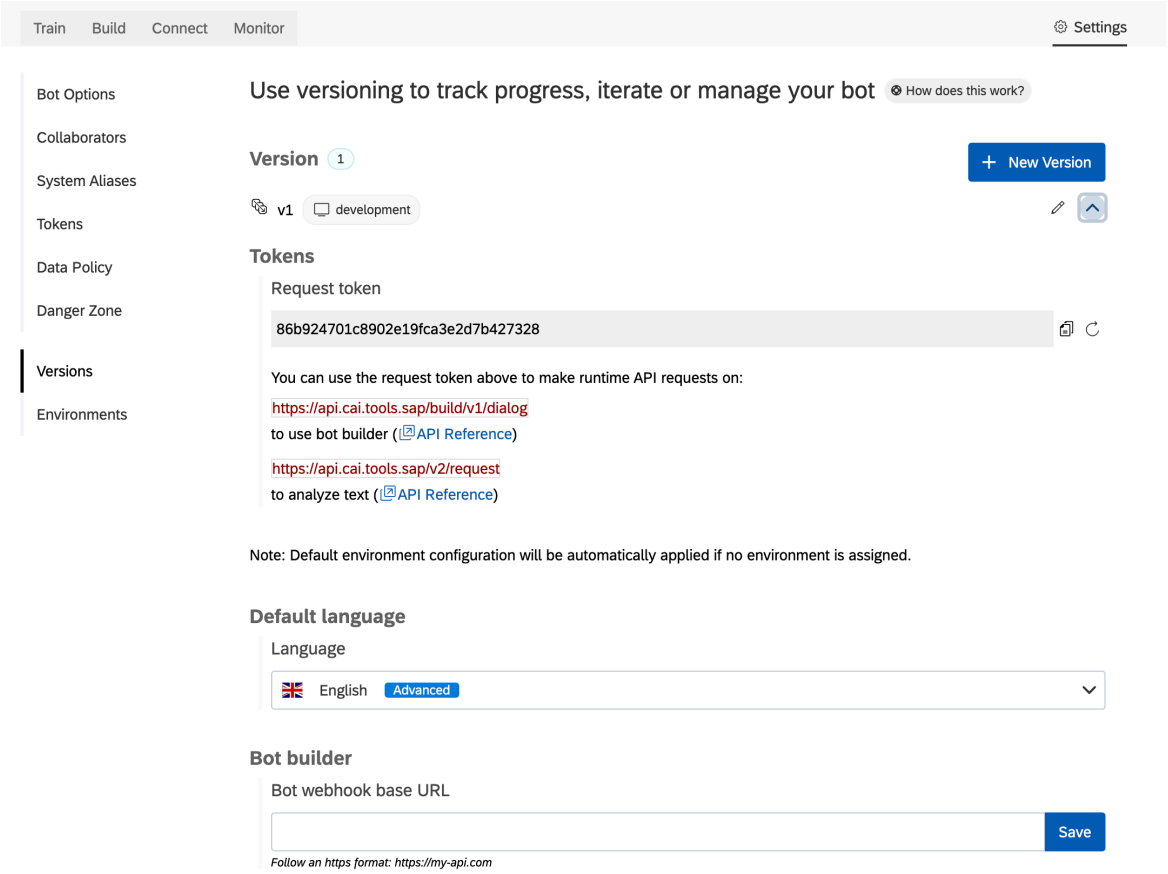
Metrics

There are three levels of accuracy or precision that the script lets you measure. At each accuracy level, the report also shows the questions that were not answered correctly.

Accuracy Level	Description
Level One	Percentage of questions in the test data to which the bot responded with the first best (most accurate) answer
Level Two	Percentage of questions in the test data to which the bot responded with one of the two best answers
Level Three	Percentage of questions in the test data to which the bot responded with one of the three best answers

How to Generate the Report

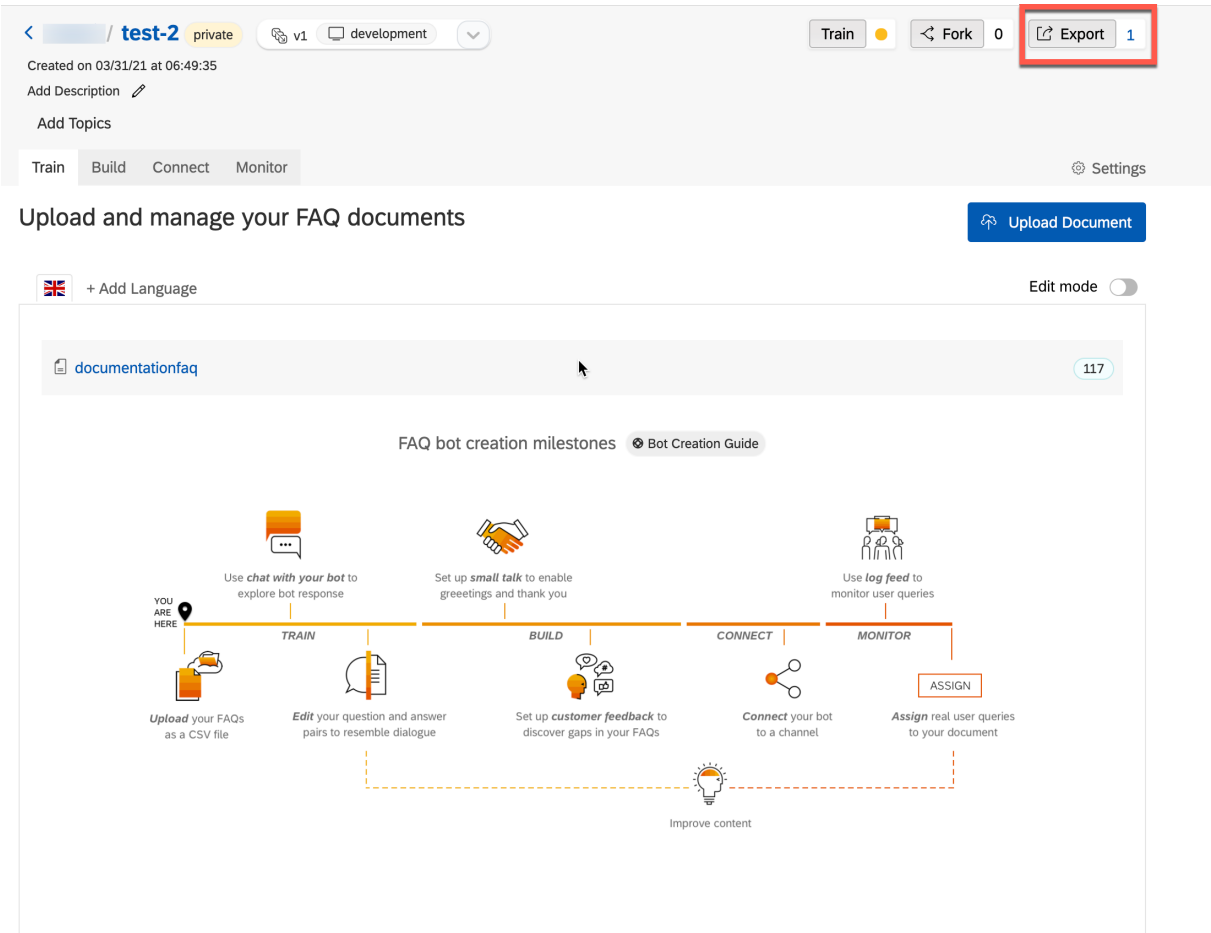
1. Clone or download the repository https://github.com/SAPConversationalAI/faq-benchmarking-sample
2. Install the dependencies by running `pip3 install -r requirements.txt` in the console.
3. For authentication, log on to https://cai.tools.sap/ and copy the *Request token* from your bot > *Settings* > *Versions* >.



4. Prepare a gold dataset. Make sure that:
   - Questions should not exist in your bot's dataset.
   - The question and answer pairs in the gold dataset must be an exact match to those defined in your FAQ file/s.
5. Run the `benchmark.py` file with the arguments `--csv_file` (the location of the CSV file stored with [question, answer]), `--request_token` and `--language`.
   ```
   python3 benchmark.py --csv_file=assets/data/gold_test_data.csv --
   request_token=e6473e3bef5f0a9e2c98a47298e596e8 --language=en
   ```

9.9 Export and Import your Bots

You can export and import your existing bots across your SAP Conversational AI enterprise tenants or community tenant. Using this feature you can replicate the changes from one tenant to another (like development, test, production) without the need to re-build the bots from scratch in the target tenant. For more information, see Transporting Bots From User Interface [page 266].


test-2
private
v1
development
Train
Fork
0
Export
1
Created on 03/31/21 at 06:49:35
Add Description
Add Topics
Train
Build
Connect
Monitor
Settings
Upload and manage your FAQ documents
Upload Document
+ Add Language
Edit mode
documentationfaq
117
FAQ bot creation milestones
Bot Creation Guide
Use chat with your bot to explore bot response
Set up small talk to enable greeetings and thank you
Use log feed to monitor user queries
YOU ARE HERE
TRAIN
BUILD
CONNECT
MONITOR
Upload your FAQs as a CSV file
Edit your question and answer pairs to resemble dialogue
Set up customer feedback to discover gaps in your FAQs
Connect your bot to a channel
ASSIGN
Assign real user queries to your document
Improve content

10 Personal Data

10.1 Update or Delete Your Personal Data

Update Your Personal Data

If you are a bot developer and you want to update your profile settings, click your avatar at the top right of the page in SAP Conversational AI and choose *Account Settings*. You can maintain your profile settings in the following tabs:

- Profile
  You can change your nickname (username), however, the email ID for your account can not be changed.
- Preferences
  You can select a different time zone and set your email preferences to receive monthly updates and/or receive the newsletter.
- Danger Zone
  You can set your *Profile Visibility* as *Private*. If you have uploaded a custom profile picture using Gravatar, it is not visible to other bot developers for a private profile, the default gravatar is visible instead. However, for a public profile, the custom profile picture is visible to everyone.
  You can import bots from other SAP Conversational AI tenants, for example enterprise tenant.
  If not needed anymore, you can choose to delete your account. All your bots and the data will be deleted.
  For more information, see privacy policy .

> ⚠ Caution
>
> All bots that you created with this account will also be deleted, except for bots that you created from within an organization if there are still members of that organization (including another administrator). If there are still members of the organization, and you are the sole administrator, you must first make another member an administrator before you can delete your account.

10.2 Access or Delete a Bot User's Personal Data

If you are a bot developer and one of your end users wants to access or delete their own personal data, please email the following information to *sap-cai-privacy@sap.com* from the email address associated with your SAP Conversational AI profile:

- Name of the channel used to access the bot
- Information required for this channel (see following table)

> i Note
>
> You can also use the following APIs to delete your user's personal data:

- /participant API to retrieve (using `GET`) or delete (using `POST`) the conversation data of your users.
- /bulk_delete endpoint API to delete the conversation logs for a bot, based on the provided timestamp.

Channel used to access the bot	Information needed to retrieve/delete end-user's conversations with bot
Amazon Alexa	Amazon user ID, for example, amzn1.ask.account.[unique-value-here]
Facebook Messenger	Facebook PSID, for example, 1254938275682919
LINE	User ID, for example, vcc1a798ab48861b186bec80b6955e3dd
Microsoft Azure	User ID, for example, user12345
SAP Jam Collaboration	Actor ID, for example, 746583
Slack	User ID, for example, W875G90RL
Telegram	Chat ID, for example, 180847183
Twilio	Phone number, for example, +11234567890
Twitter	User name, for example, johnmiller

> **i Note**
>
> For third party Fallback channels, if you have enabled transfer of personal or sensitive data, you must review the data privacy guidelines and GDPR compliance separately for deletion of personal data from the channel.

SAP CoPilot

SAP CoPilot doesn't store the end-user's conversation with the assistant beyond the UI session.

In addition, at any point the user can go to the SAP CoPilot settings and perform a clear assistant activity to delete/clear the conversations. Alternatively, the user can get a report about their personal data stored, as documented in User Offboarding in the SAP CoPilot User Help. That report also offers a function for the user to entirely offboard SAP CoPilot; this would also delete the conversations with the assistant.

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Gender-Related Language

We try not to use gender-specific word forms and formulations. As appropriate for context and readability, SAP may use masculine word forms to refer to all genders.

www.sap.com/contactsap

© 2021 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see https://www.sap.com/about/legal/trademark.html for additional trademark information and notices.

THE BEST RUN SAP