Table of Contents

- LUIS Documentation
- Overview
  - Learn about Language Understanding Intelligent Service
- Quickstarts
  - Create a new app
- Tutorials
  - Integrate LUIS with a bot using Node.js
- Concepts
  - Intents
  - Entities
  - Utterances
  - Features
  - Support for multiple languages
- How-to guides
  - Plan your app
  - Build a LUIS app
    - Start a new app
    - Add intents
    - Add utterances
    - Add entities
    - Improve performance using features
    - Train and test the app
    - Use active learning
    - Publish your app
  - Prebuilt models
    - Use prebuilt entities
    - Use prebuilt domains
    - Use the Cortana prebuilt app
  - Manage your LUIS app

- Monitor your app using the dashboard
- Manage keys
- Create subscription keys
- Manage versions
- Collaborate on a LUIS app

Reference

- Authoring APIs
- Endpoint API
- SDKs
  - Android
  - Node.js
  - Python
  - Windows

Resources

- Azure Roadmap
- LUIS FAQ

Learn how to enable natural and contextual interaction within your applications with Cognitive Services. Quick start tutorials and API references help you incorporate artificial intelligence capabilities for text, speech, vision, and search.

- **Learn about Cognitive Services**
- **Get started with the Computer Vision API**
- **Get started with the Face API**
- **Get started with the Bing Web Search API**
- **Get started with the Custom Speech Service API**
- **Get started with Language Understanding Intelligent Services (LUIS)**
- **Cognitive Services video library**

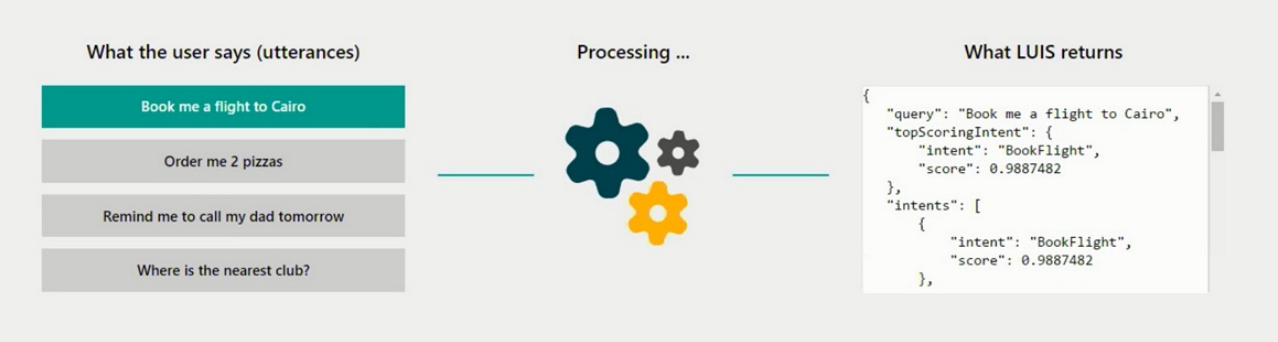
Learn about Language Understanding Intelligent Service (LUIS)

6/28/2017 • 4 min to read • Edit Online

One of the key problems in human-computer interactions is the ability of the computer to understand what a person wants. Language Understanding Intelligent Service (LUIS) enables developers to build smart applications that can understand human language and react accordingly to user requests. LUIS uses the power of machine learning to solve the difficult problem of extracting meaning from natural language input, so that your application doesn't have to. Any client application that converses with users, like a dialog system or a chat bot, can pass user input to a LUIS app and receive results that provide natural language understanding.

What is a LUIS app?

A LUIS app is a place for a developer to define a custom language model. The output of a LUIS app is a web service with an HTTP endpoint that you reference from your client application to add natural language understanding to it. A LUIS app takes a user utterance and extracts intents and entities that correspond to activities in the client application's logic. Your client application can then take appropriate action based on the user intentions that LUIS recognizes.



Key concepts

- **What is an utterance?** An utterance is the textual input from the user, that your app needs to interpret. It may be a sentence, like "Book me a ticket to Paris", or a fragment of a sentence, like "Booking" or "Paris flight." Utterances aren't always well-formed, and there can be many utterance variations for a particular intent. See Add example utterances for information on training a LUIS app to understand user utterances.
- **What are intents?** Intents are like verbs in a sentence. An intent represents actions the user wants to perform. It is a purpose or goal expressed in a user's input, such as booking a flight, paying a bill, or finding a news article. You define a set of named intents that correspond to actions users want to take in your application. A travel app may define an intent named "BookFlight", that LUIS extracts from the utterance "Book me a ticket to Paris".
- **What are entities?** If intents are verbs, then entities are nouns. An entity represents an instance of a class of object that is relevant to a user's intent. In the utterance "Book me a ticket to Paris", "Paris" is an entity of type location. By recognizing the entities that are mentioned in the user's input, LUIS helps you choose the specific actions to take to fulfill an intent. See Entities in LUIS for more detail on the types of entities that LUIS provides.

Plan your LUIS app

Before you start creating it in the LUIS web interface, plan your LUIS app by preparing an outline or schema to describe intents and entities in your application. Generally, you create an intent to trigger an action in a client application or bot and create an entity to model some parameters required to execute an action. For example, a

"BookFlight" intent could trigger an API call to an external service for booking a plane ticket, which requires entities like the travel destination, date, and airline. See Plan your app for examples and guidance on how to choose intents and entities to reflect the functions and relationships in an app.

Build and train a LUIS app

Once you have determined which intents and entities you want your app to recognize, you can start adding them to your LUIS app. See create a new LUIS app, for a quick walkthrough of creating a LUIS app. For more detail about the steps in configuring your LUIS app, see the following articles:

1. Add intents
2. Add utterances
3. Add entities
4. Improve performance using features
5. Train and test
6. Use active learning
7. Publish

You can also watch a basic video tutorial on these steps.

Improve performance using active learning

Once your application is deployed and traffic starts to flow into the system, LUIS uses active learning to improve itself. In the active learning process, LUIS identifies the utterances that it is relatively unsure of, and asks you to label them according to intent and entities. This process has tremendous advantages. LUIS knows what it is unsure of, and asks for your help in the cases that lead to the maximum improvement in system performance. LUIS learns quicker, and takes the minimum amount of your time and effort. This is active machine learning at its best. See Label suggested utterances for an explanation of how to implement active learning using the LUIS web interface.

Configure LUIS programmatically

LUIS offers a set of programmatic REST APIs that can be used by developers to automate the application creation process. These APIs allow you to author, train, and publish your application.

- LUIS Programmatic API.

Integrate LUIS with a bot

It's easy to use a LUIS app from a bot built using the Bot Framework, which provides the Bot Builder SDK for Node.js or .NET. You simply reference the LUIS app as shown in the following examples:

Node.js

```
// Add a global LUIS recognizer to your bot using the endpoint URL of your LUIS app
var model = 'https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/2c2afc3e-5f39-4b6f-b8ad-c47ce1b98d8a?subscription-key=9823b65a8c9045f8bce7fee87a5e1fbc';
bot.recognizer(new builder.LuisRecognizer(model));
```

C#

```
// The LuisModel attribute specifies your LUIS app ID and your LUIS subscription key
[LuisModel("2c2afc3e-5f39-4b6f-b8ad-c47ce1b98d8a", "9823b65a8c9045f8bce7fee87a5e1fbc")]
[Serializable]
public class TravelGuidDialog : LuisDialog<object>
{
  // ...
```

The Bot Builder SDK provides classes that automatically handle the intents and entities returned from the LUIS app. For code that demonstrate how to use these classes, see the following samples:

- LUIS demo bot (C#)
- LUIS demo bot (Node.js)

Integrate LUIS with Speech

Your LUIS endpoints work seamlessly with Microsoft Cognitive Service's speech recognition service. In the C# SDK for Microsoft Cognitive Services Speech API, you can add the LUIS application ID and LUIS subscription key, and the speech recognition result is sent for interpretation.

See Microsoft Cognitive Services Speech API Overview.

Create your first LUIS app in ten minutes

6/27/2017 • 3 min to read • Edit Online

This Quickstart helps you create your first Language Understanding Intelligent Service (LUIS) app in just a few minutes. When you're finished, you'll have a LUIS endpoint up and running in the cloud.

You are going to create a travel app, that helps you book flights and check the weather at your destination. The how-to topics refer to this application and build on it.

Before you begin

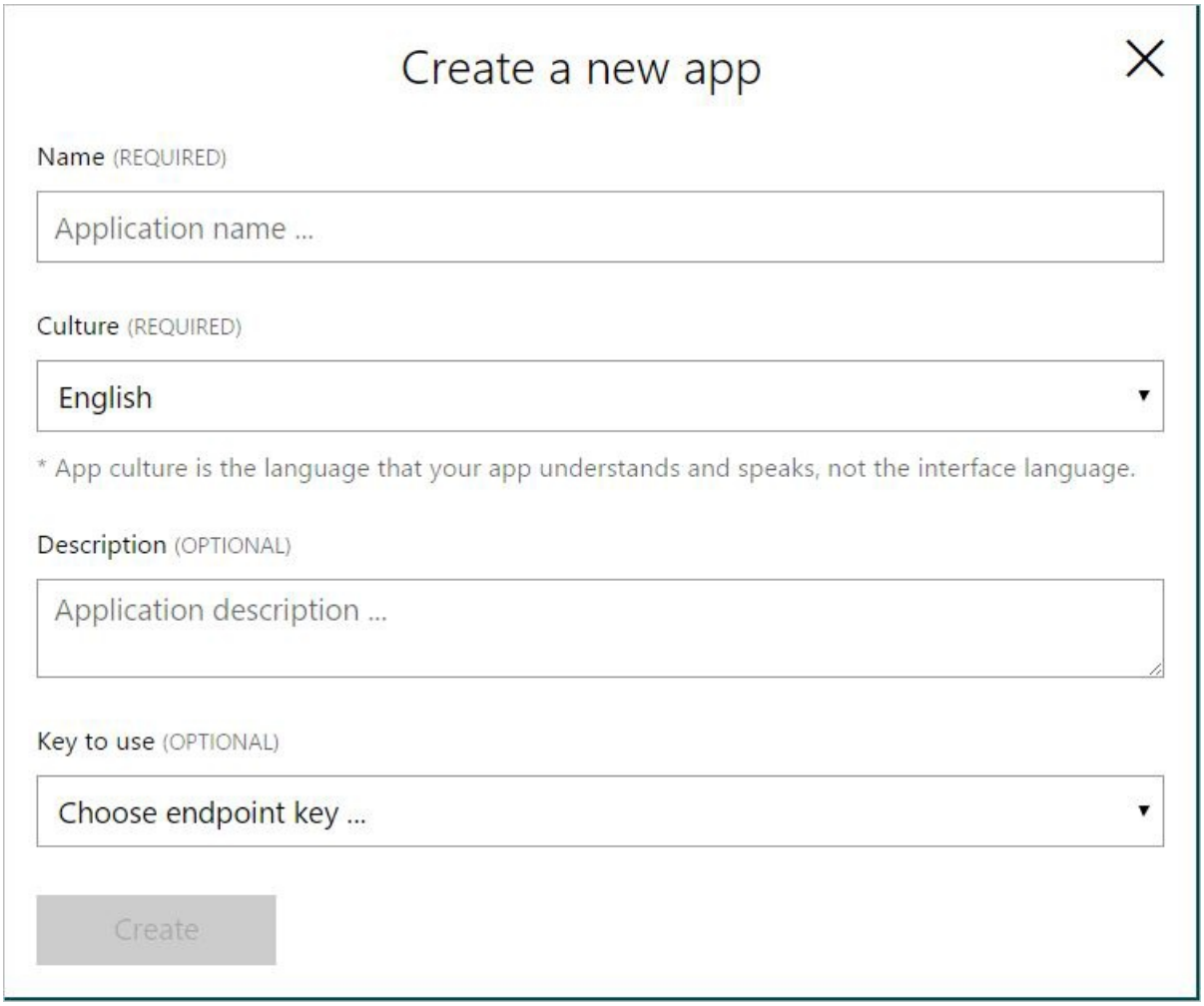
To use Microsoft Cognitive Service APIs, you first need to create a Cognitive Services API account in the Azure portal.

If you don't have an Azure subscription, create a free account before you begin.

1. Create a new app

You can create and manage your applications on **My Apps** page. You can always access this page by clicking **My Apps** on the top navigation bar of the LUIS web page.

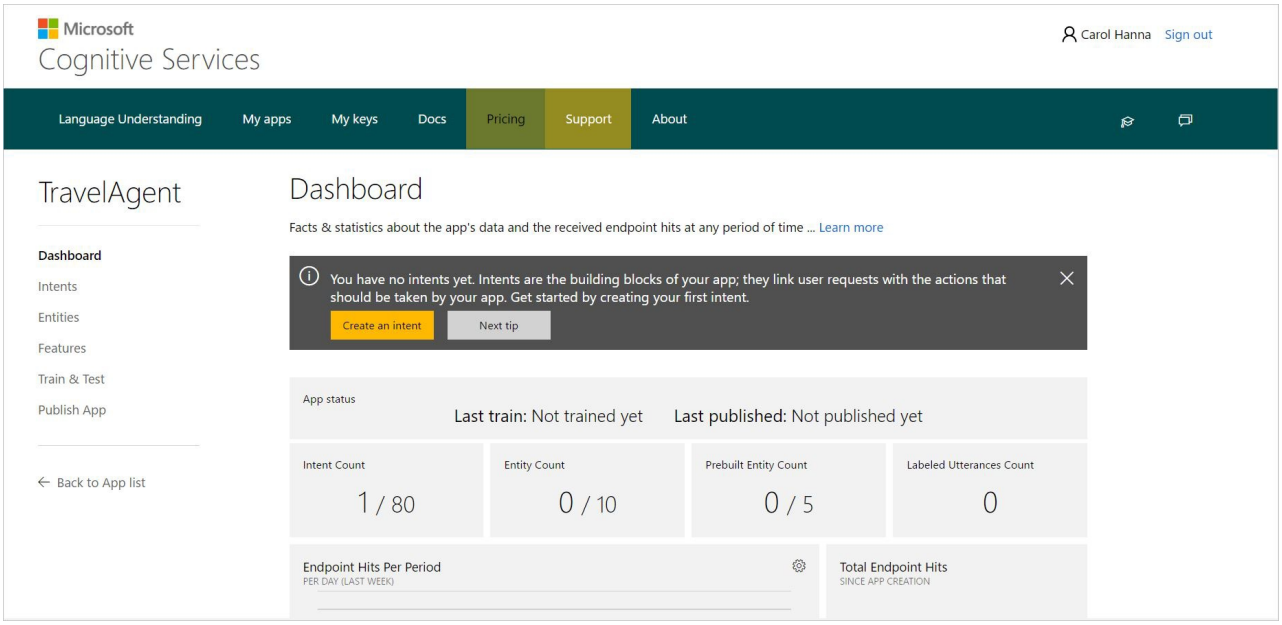
1. On the **My Apps** page, click **New App**.
2. In the dialog box, name your application "TravelAgent".



3. Choose your application culture (for TravelAgent app, we'll choose English), and then click **Create**.

> **NOTE**
> The culture cannot be changed once the application is created.

LUIS creates the TravelAgent app and opens its main page which looks like the following screen. Use the navigation links in the left panel to move through your app pages to define data and work on your app.



2. Add intents

Your first task in the app is to add intents. Intents are the intentions or requested actions conveyed by the user's utterances. They are the main building block of your app. You now need to define the intents (for example, book a flight) that you want your application to detect. Go to the **Intents** page in the side menu to create your intents by clicking the **Add Intent** button.

For more detail on how to add intents, see Add intents.

3. Add utterances

Now that you've defined intents, you can start seeding examples to every intent to teach the machine learning model the different patterns (for example, "book a flight to Seattle departing on June 8th".) Select an intent you just added and start adding and saving utterances to your intent.

4. Add entities

Now that you have your intents, you can proceed to add entities. Entities describe information relevant to the intent, and sometimes are essential for your app to perform its task. An example for this app would be the airline on which to book a flight. Add a simple entity named "Airline" to your TravelAgent app.

For more information about entities, see Add entities.

5. Label entities in utterances

Next, you need to label examples of the entities to teach LUIS what this entity can look like. Highlight relevant tokens as entities in the utterances you added.

6. Add prebuilt entities

It might be useful to add one of the pre-existing entities, which we call *prebuilt* entities. Those types of entities are ready to be used directly and don't need to be labeled. Go to the **Entities** page to add prebuilt entities relevant to your app. Add the `ordinal` and `datetime` prebuilt entities to your app.

7. Train your app

Select **Train & Test** in the left panel, and then click **Train Application** to train your app based on the intents, utterances, entities you defined in the previous steps.

8. Test your app

Once you've trained your app, you can test it by typing a test utterance and pressing Enter. The results display the score associated with each intent. Check that the top scoring intent corresponds to the intent of each test utterance.

9. Publish your app

Select **Publish App** from the left-side menu and click **Publish**.

10. Use your app

Copy the endpoint URL from the Publish App page and paste it into a browser. Append a query like "Book a flight to Boston" at the end of the URL and submit the request. The JSON containing results should show in the browser window.

Next steps

- Try to improve your app's performance by continuing to add and label utterances.
- Try adding Features to enrich your model and improve performance in language understanding. Features help your app identify alternative interchangeable words/phrases, as well as commonly-used patterns specific to your domain.

Integrate LUIS with a bot using the Bot Builder SDK for Node.js

6/29/2017 • 3 min to read • Edit Online

This tutorial walks you through creating a bot with the Bot Builder SDK for Node.js and integrating it with a Language Understanding Intelligent Service (LUIS) app.

Before you begin

Before you start, make sure you have the accounts and tools you need for working with LUIS and the Bot Framework.

LUIS prerequisites

- If you don't have an Azure subscription, create a free account before you begin. This should be the same account that you use for LUIS.

Bot Framework prerequisites

- Download the Bot Builder SDK from GitHub. Install the SDK for Node.js.
- Install the Bot Framework Emulator, which you'll use to run the bot locally.
- Create an account on the Bot Framework developer portal, where you'll register the bot.

1. Download the sample bot

Download the sample code for the LUIS demo bot (Node.js) from GitHub. This is a chat bot that can search for hotels in a specified location and provide reviews for a specified hotel. It listens for three different user intents: `SearchHotels`, `ShowHotelsReviews`, and `Help`.

2. Import the LUIS model to create the LUIS app

You can create and manage your applications on **My Apps** page. You can always access this page by clicking **My Apps** on the top navigation bar of the LUIS web page.

1. On the **My Apps** page, click **Import App**.
2. In the **Import new app** dialog box, click **Choose file** and navigate to `LuisBot.json` in the folder where you downloaded the bot in step 1. Name your application "Hotel Finder", and Click **Import**. It may take a few minutes for LUIS to extract the intents and entities from the JSON file. When the import is complete, LUIS opens the Dashboard page of the Hotel Finder app. Use the navigation links in the left panel to move through your app pages to define data and work on your app.

3. Train and publish the LUIS app

Go to **Train & Test** and click **Train your app**. Optionally, you can also test your app now, but it is not required for publishing. To publish, go to **Publish App**, select an **Endpoint Key**, select **Production** for the **Endpoint slot**, and click **Publish**.

4. Copy the LUIS endpoint URL

Once you've published the app, the **Publish App** page will display an endpoint URL. Copy this URL. You'll update the bot's code to point to it.

5. Paste the LUIS endpoint into the bot code

1. Go to the `.env` file in the sample bot. Set `LUIS_MODEL_URL` to the URL from the previous step.
2. Delete any trailing `&q=` from the URL. Here's an example of how the URL might look:

```
LUIS_MODEL_URL=https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/2c2afc3e-5f39-4b6f-b8ad-c47ce1b98d8a?subscription-key=9823b65a8c9045f8bce7fee87a5e1fbc&verbose=true&timezoneOffset=0
```

In `app.js` this URL is used to initialize the recognizer object that the bot uses to listen for the user's intent.

```
// This Url can be obtained by uploading or creating your model from the LUIS portal: https://www.luis.ai/
var recognizer = new builder.LuisRecognizer(process.env.LUIS_MODEL_URL);
```

6. Register the bot

Go to the Bot Framework developer portal and register the bot. As part of this process, you'll generate a Microsoft App ID and password. Copy and securely store the password that is shown, and enter the App ID and password in the bot's `.env` file.

7. Start the bot

Open a node.js command prompt, and run the bot: `node app.js`.

8. Start the emulator

1. Run the Bot Framework Emulator, enter `http://localhost:3978/api/messages` for your endpoint URL, and enter the Microsoft App ID and password from step 6, which you should have in the `.env` file.
2. (Optional) Take note of the URL in the **Log** panel of the emulator. There should be a message displaying the ngrok URL, similar to this example: `ngrok listening on https://ce9a9909.ngrok.io`. Go back to the Bot Framework developer portal and enter this URL in the bot's profile, appending `/api/messages` to the URL. This step is optional, but useful if you later decide to test the bot using the developer portal or connect it to channels other than the emulator.

9. Talk to the bot

1. Click **Connect** in the emulator. You can start giving the bot some requests like "Find me hotels in Paris", or "Show me reviews of the Contoso Hotel".

Next steps

- Try to improve your LUIS app's performance by continuing to add and label utterances.
- Try adding additional Features to enrich your model and improve performance in language understanding. Features help your app identify alternative interchangeable words/phrases, as well as commonly used patterns specific to your domain.

Intents in LUIS

6/27/2017 • 1 min to read • Edit Online

An intent represents a task or action the user wants to perform. It is a purpose or goal expressed in a user's input, such as booking a flight, paying a bill, or finding a news article.

In your LUIS app, you define a set of named intents that correspond to actions users want to take in your application. A travel app may define an intent named `BookFlight`, that LUIS extracts from the utterance "Book me a ticket to Paris."

EXAMPLE INTENT	EXAMPLE UTTERANCES
BookFlight	Book me a flight to Rio next week Fly me to Rio on the 24th I need a plane ticket next Sunday to Rio de Janeiro
Greeting	Hi Hello Good morning
CheckWeather	What's the weather like in Boston? Show me the forecast for this weekend
None	Get me a cookie recipe

All applications come with the predefined intent, **"None"**. You should teach it to recognize user statements that are irrelevant to the app.

You can add up to **80** intents in a single LUIS app. However, it is a best practice to use only as many intents as you need to perform the functions of your app. If you define too many intents, it becomes harder for LUIS to classify utterances correctly. If you define too few, they may be so general as to be overlapping.

> **TIP**
> In addition to intents that you define, you can use prebuilt intents from one of the prebuilt domains. See Use prebuilt domains in LUIS apps to learn about how to customize intents from a prebuilt domain for use in your app.

How do intents relate to entities?

Create an intent when this intent would trigger an action in your client application, like a call to the checkweather() function, and create an entity to represent parameters required to execute the action.

EXAMPLE INTENT	ENTITY	ENTITY IN EXAMPLE UTTERANCES
CheckWeather	{ "type": "location", "entity": "seattle" }	What's the weather like in `Seattle`?
CheckWeather	{ "type": "date_range", "entity": "this weekend" }	Show me the forecast for `this weekend`

Next steps

- Learn more about entities, which are important words relevant to intents
- Learn how to add and manage intents in your LUIS app.

Entities in LUIS

7/27/2017 • 2 min to read • Edit Online

Entities are important words in utterances that describe information relevant to the intent, and sometimes they are essential to it. Entities belong to classes of similar objects.

In the utterance "Book me a ticket to Paris", "Paris" is an entity of type location. By recognizing the entities that are mentioned in the user's input, LUIS helps you choose the specific actions to take to fulfill an intent.

You do not need to create entities for every concept in your app, but only for those required for the app to take action. You can add up to **30** entities in a single LUIS app.

You can add, edit or delete entities in your app through the **Entities list** on the **Entities** page in the LUIS app web portal. LUIS offers many types of entities; prebuilt entities, custom machine learned entities and list entities.

Types of entities

LUIS offers the following types of entities:

TYPE	DESCRIPTION
Prebuilt	Built-in types that represent common concepts like dates, times, and geography. These don't count towards the maximum number of entities you may use in your LUIS app. See Prebuilt entities for more information.
List	List entities represent a fixed set of synonyms or related words in your system. Each list entity may have one or more synonyms. They aren't machine learned, and are best used for a known set of variations on ways to represent the same concept. List entities don't have to be labeled in utterances or trained by the system. A list entity is an explicitly specified list of values. Unlike other entity types, LUIS does not discover additional values for list entities during training. Therefore, each list entity forms a closed set. Your app may use up to 50 list entities, and they don't count toward the maximum 30 you may use. Each list can contain up to 20000 items.
Simple	A simple entity is a generic entity that describes a single concept. These count towards the maximum number of entities you may use.

TYPE	DESCRIPTION
Hierarchical	A hierarchical entity defines a category and its members. It is made up of child entities that form the members of the category. You can use hierarchical entities to define hierarchical or inheritance relationships between entities, in which children are subtypes of the parent entity. For example, in a travel agent app, you could add hierarchical entities like these: • $Location, including $FromLocation and $ToLocation as child entities that represent origin and destination locations. • $TravelClass, including $First, $Business, and $Economy as child entities that represent the travel class. A hierarchical entity can consist of up to **10** child entities. These count towards the maximum number of entities you may use.
Composite	A composite entity is made up of other entities that form parts of a whole. A composite entity can consist of up to **20** child entities. These count towards the maximum number of entities you may use.

Next steps

See Add entities to learn more about how to add entities to your LUIS app.

Utterances in LUIS

6/27/2017 • 1 min to read • Edit Online

Utterances are input from the user that your app needs to interpret. To train LUIS to extract intents and entities from them, it's important to capture a variety of different inputs for each intent. Active learning, or the process of continuing to train on new utterances, is essential to machine-learned intelligence that LUIS provides.

Collect phrases that you think users will say, and include utterances that mean the same thing but are constructed differently.

How to choose varied utterances

When you first get started by adding example utterances to your LUIS model, here are some principles to keep in mind.

Utterances aren't always well formed

It may be a sentence, like "Book me a ticket to Paris", or a fragment of a sentence, like "Booking" or "Paris flight." Users often make spelling mistakes. When planning your app, consider whether or not you will spell check user input before passing it to LUIS. The Bing Spell Check API integrates with LUIS. You can associate your LUIS app with an external key for the Bing Spell Check API when you publish it. If you do not spell check user utterances, you should train LUIS on utterances that include typos and misspellings.

Use the representative language of the user

When choosing utterances, be aware that what you think is a common term or phrase might not be to the typical user of your client application. They may not have domain experience. So be careful when using terms or phrases that a user would only say if they were an expert.

Choose varied terminology as well as phrasing

You will find that even if you make efforts to varied create sentence patterns, you will still repeat some vocabulary.

Take these example utterances:

```
how do I get a computer?
Where do I get a computer?
I want to get a computer, how do I go about it?
When can I have a computer?
```

The core term here, "computer", is not varied. They could say desktop computer, laptop, workstation, or even just machine. LUIS can be quite intelligent at inferring synonyms from context, but when you create utterances for training, it's still better to vary them.

Next steps

See Add example utterances for information on training a LUIS app to understand user utterances.

Features in LUIS

6/27/2017 • 4 min to read • Edit Online

In machine learning, a *feature* is a distinguishing trait or attribute of data that your system observes.

You add features to a language model, to provide hints about how to recognize input that you want to label or classify. Features help LUIS recognize both intents and entities, but features are not intents or entities themselves. Instead, features might provide examples of related terms, or a pattern to recognize in related terms.

Types of features

LUIS offers the following types of features:

TYPE	DESCRIPTION
Phrase list	A phrase list includes a group of values (words or phrases) that belong to the same class and must be treated similarly (for example, names of cities or products). What LUIS learns about one of them is automatically applied to the others as well. • The maximum length of a phrase list is 5000 items. You may have a maximum of 10 phrase lists per LUIS app.
Pattern	A pattern specifies a regular expression to help LUIS recognize regular patterns that are frequently used in your application's domain. Some examples are the pattern of flight numbers in a travel app or product codes in a shopping app.

How to use phrase lists

For example, in a travel agent app, you can create a phrase list named "Cities" that contains the values London, Paris, and Cairo. If you label one of these values as an entity, LUIS learns to recognize the others.

A phrase list may be exchangeable or non-exchangeable. An *exchangeable* phrase list is for values that are synonyms, and a *non-exchangeable* phrase list is intended for values that aren't synonyms but are similar in another way.

Use phrase lists for terms that LUIS has difficulty recognizing

Phrase lists are a good way to tune the performance of your LUIS app. If your app has trouble classifying some utterances as the correct intent, or recognizing some entities, think about whether the utterances contain unusual words, or words that might be ambiguous in meaning. These words are good candidates to include in a phrase list feature.

Use phrase lists for rare, proprietary, and foreign words

LUIS may be unable to recognize rare and proprietary words, as well as foreign words (outside of the culture of the app), and therefore they should be added to a phrase list feature. This phrase list should be marked non-exchangeable, to indicate that the set of rare words form a class that LUIS should learn to recognize, but they are not synonyms or exchangeable with each other.

> **NOTE**
> A phrase list feature is not an instruction to LUIS to perform strict matching or always label all terms in the phrase list exactly the same. It is simply a hint. For example, you could have a phrase list that indicates that "Patti" and "Selma" are names, but LUIS can still use contextual information to recognize that they mean something different in "make a reservation for 2 at patti's diner for dinner" and "give me driving directions to selma, georgia".

When to use phrase lists instead of list entities

- When you use a phrase list, LUIS can still take context into account and generalize to identify items that are similar to, but not an exact match as items in a list. If you need your LUIS app to be able to generalize and identify new items in a category, it's better to use a phrase list.
- In contrast, a list entity explicitly defines every value an entity can take, and only identifies values those that match exactly. A list entity may be appropriate for an app in which all instances of an entity are known and don't change often, like the food items on a restaurant menu that changes infrequently. In a system in which you want to be able to recognize new instances of an entity, like a meeting scheduler that should recognize the names of new contacts, or an inventory app that should recognize new products, it's better to use another type of entity and then use phrase list features to help guide LUIS to recognize examples of the entity.

How to use patterns

The regular expression, or *regex*, in a pattern feature provides a hint to LUIS that helps it see the difference between words that match the regex and words that don't.

For example, a pattern can help recognize a `KnowledgeBaseArticle` entity if the regex matches the terms in the entity, like the regex `"kb\d+"` for identifying knowledge base article IDs like `kb8732827` or `kb23737`. In addition, you also need to enter some utterances and label the entity. For example, label the utterance `"look for article kb22716"` so that `"kb22716"` is tagged as a `KnowledgeBaseArticle` entity.

> **NOTE**
> A pattern feature is not an instruction to LUIS to perform strict matching or label all terms that match the expression exactly the same. It is simply a hint. For example, if you use a pattern to tell LUIS that airport codes in your travel app consist of three letters, you shouldn't expect (and don't want) LUIS to always label every three-letter word as an airport code.

Next steps

See Add Features to learn more about how to add features to your LUIS app.

Localization support in LUIS apps

6/27/2017 • 1 min to read • Edit Online

This article describes considerations for designing LUIS apps in multiple languages.

You choose the culture when you start creating your LUIS app, and it cannot be modified once the application is created.

LUIS understands utterances in the following languages. Support for prebuilt entities varies. See Prebuilt entities in LUIS for details.

LOCALE	LANGUAGE	PREBUILT ENTITY SUPPORT
en-US	American English	□□
fr-CA	Canadian French	-
fr-FR	French (France)	□□
it-IT	Italian	□□
nl-NL	Dutch	-
de-DE	German	□□
es-ES	Spanish (Spain)	□□
es-MX	Spanish (Mexico)	-
pt-BR	Portuguese (Brazil)	□□
ja-JP	Japanese	□□
ko-KR	Korean	-
zh-CN	Chinese	□□

Chinese support notes

- In the zh-cn culture, LUIS expects the simplified Chinese character set (not the traditional character set).
- The names of intents, entities, features, and regular expressions may be in Chinese or Roman characters.
- When writing regular expressions in Chinese, do not insert whitespace between Chinese characters.

Rare or foreign words in an application

In the en-us culture, LUIS can learn to distinguish most English words, including slang. In the zh-cn culture, LUIS can learn to distinguish most Chinese characters. If you use a rare word (en-us) or character (zh-cn), and you see that LUIS seems unable to distinguish that word or character, you can add that word or character to a phrase-list feature. For example, words outside of the culture of the application -- that is, foreign words -- should be added to a phrase-list feature. This phrase list should be marked non-exchangeable, to indicate that the set of rare words form a class

that LUIS should learn to recognize, but they are not synonyms or exchangable with each other.

Tokenization

In normal LUIS use, you don't need to worry about tokenization, but one place where tokenization is important is when manually adding labels to an exported application's JSON file. See the section on importing and exporting an application for details.

To perform machine learning, LUIS breaks an utterance into tokens. A token is the smallest unit that can be labeled in an entity.

How tokenization is done depends on the application's culture:

- **English, French, Italian, Brazilian Portuguese, and Spanish:** token breaks are inserted at any whitespace, and around any punctuation.
- **Korean & Chinese:** token breaks are inserted before and after any character, and at any whitespace, and around any punctuation.

Plan your LUIS app

6/27/2017 • 2 min to read • Edit Online

It is important to plan your app before you start creating it in LUIS. Prepare an outline or schema of the possible intents and entities that are relevant to the domain-specific topic of your application.

Identify your domain

A LUIS apps is usually centered around a domain-specific topic. For example, you may have an travel app that performs booking of tickets, flights, hotels, and rental cars. Another app may provide content related to exercising, tracking fitness efforts and setting goals.

> **TIP**
> LUIS offers prebuilt domains for many common scenarios. Check to see if you can use a prebuilt domain as a starting point for your app.

Identify your intents

You should think about the intents that are important to your application's task. Let's take the example of a travel app, with functions to book a flight and check the weather at the user's destination. You can define the "BookFlight" and "GetWeather" intents for these actions. In a more complex app with more functions, you will have more intents, and you should define them carefully so as to not be too specific. For example, "BookFlight" and "BookHotel" may need to be separate intents, but "BookInternationalFlight" and "BookDomesticFlight" may be too similar.

> **NOTE**
> It is a best practice to use only as many intents as you need to perform the functions of your app. If you define too many intents, it becomes harder for LUIS to classify utterances correctly. If you define too few, they may be so general as to be overlapping.

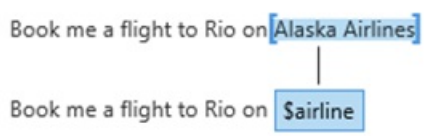
Identify your entities

To book a flight, you need some information like the destination, date, airline, ticket category and travel class. You can add these as entities because they are important for accomplishing an intent.

When you determine which entities to use in your app, keep in mind that there are different types of entities for capturing relationships between types of objects. Entities in LUIS provides more detail about the different types.

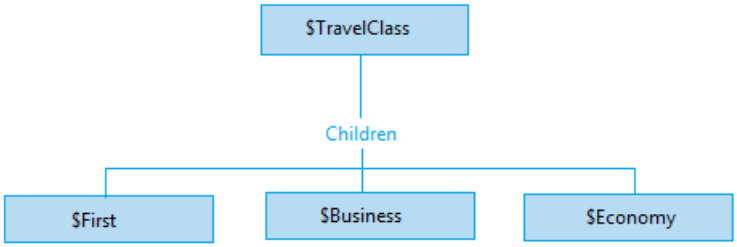
Simple entity

A simple entity describes a single concept.



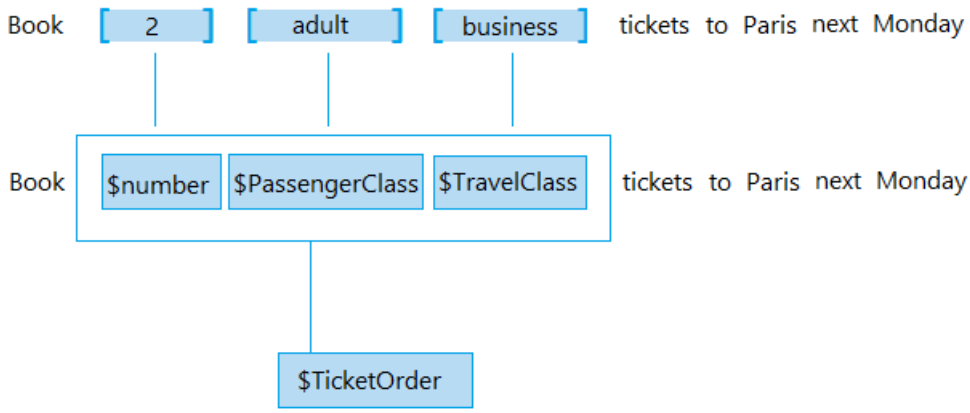
Hierarchical entity

A hierarchical entity represents a category and its members.



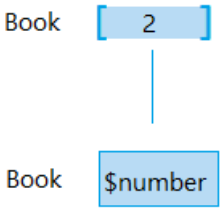
Composite entity

A composite entity is made up of other entities that form parts of a whole.



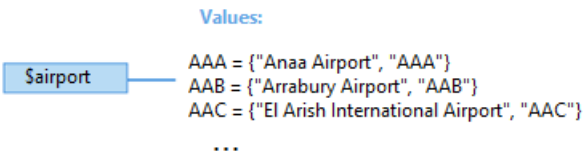
Prebuilt entity

LUIS provides prebuilt entities for common types like `Number`, which you can use for the number of tickets in a ticket order.



List entity

A list entity is an explicitly specified list of values. Each value consists of one or more synonyms. In a travel app you might choose to create a list entity to represent airport names.



Next steps

- See Create your first Language Understanding Intelligent Services (LUIS) app for a quick walkthrough of how to create a LUIS app.

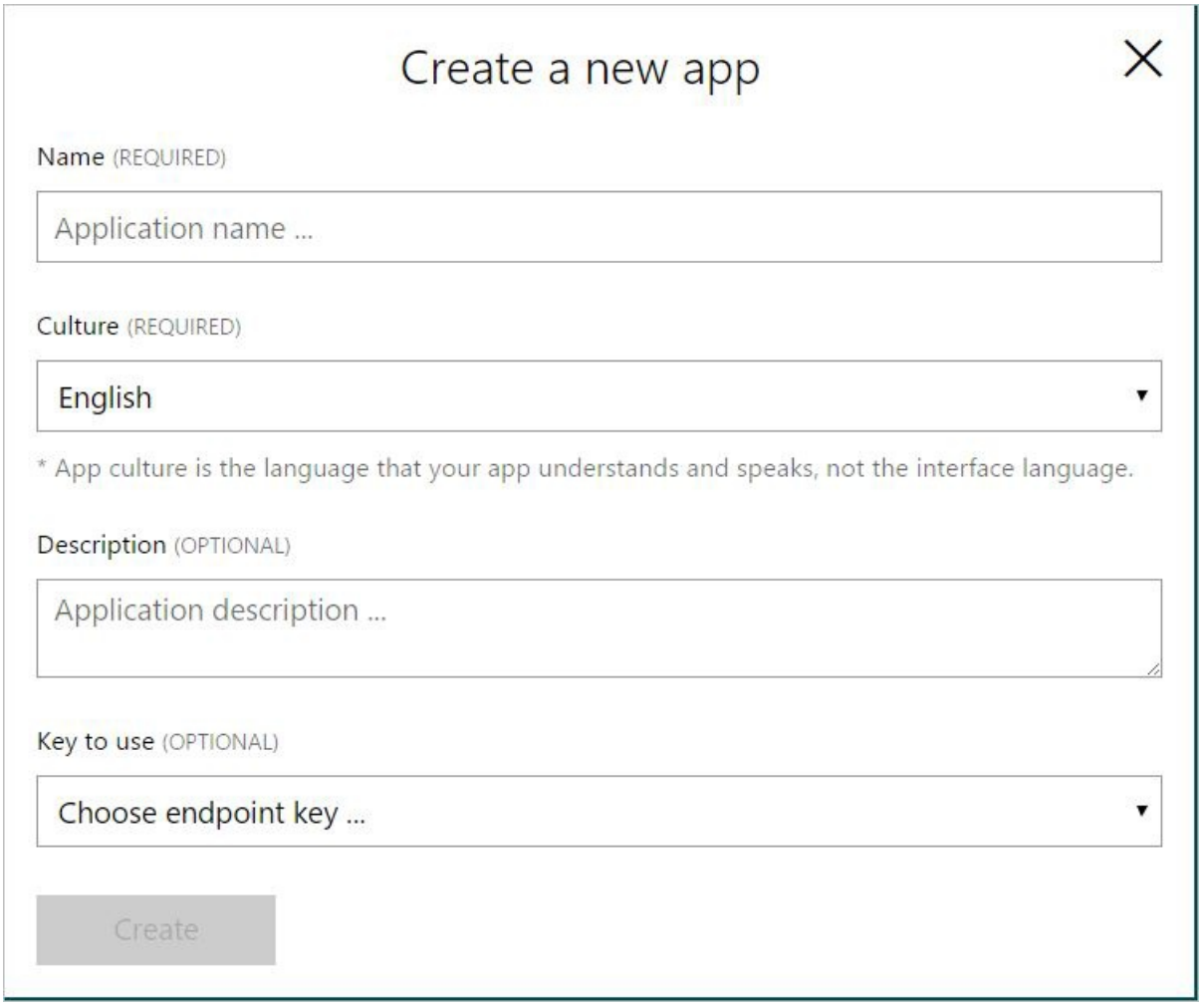
Create a New App

6/27/2017 • 1 min to read • Edit Online

You can create and manage your applications on **My Apps** page. You can always access this page by clicking **My Apps** on the top navigation bar of LUIS web page. This article shows you how to create a LUIS app and use it as an example application to work on throughout the LUIS help topics.

To create a new app:

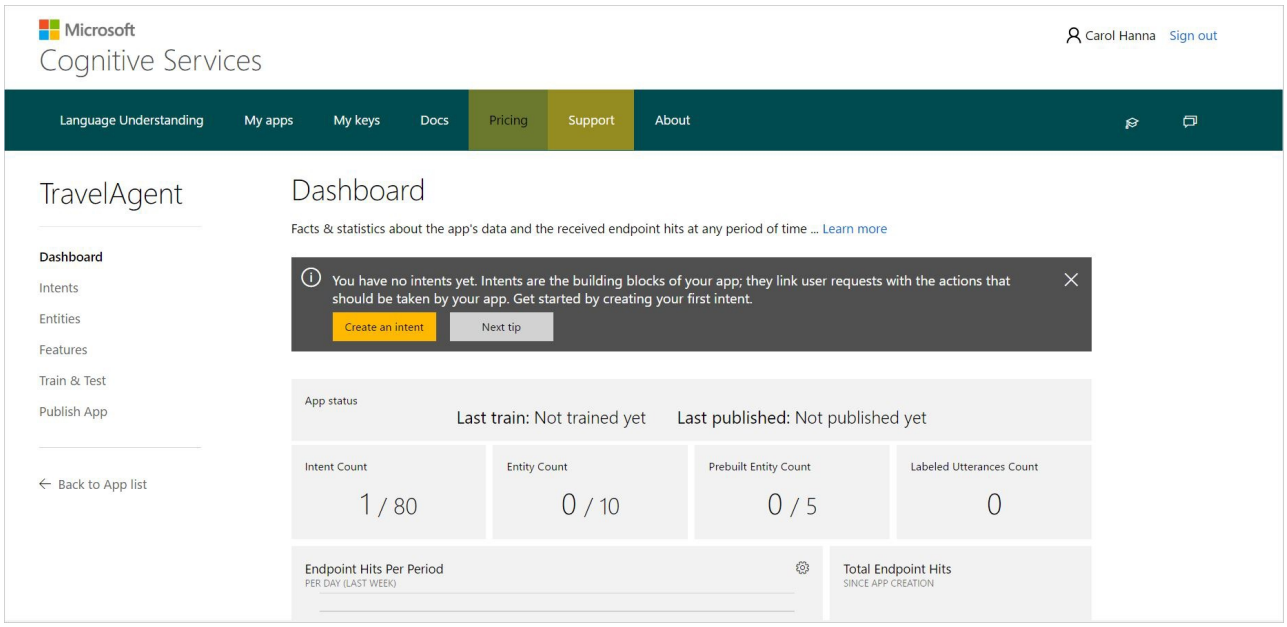
1. On **My Apps** page, click **New App**.
2. In the dialog box, name your application "TravelAgent".



3. Choose your application culture (for TravelAgent app, we'll choose English), and then click **Create**.

> **NOTE**
> The culture cannot be changed once the application is created.

LUIS creates the TravelAgent app and opens its main page which looks like the following screen. Use the navigation links in the left panel to move through your app pages to define data and work on your app.



Next steps

Your first task in the app is to add intents. For more info on how to add intents, see Add intents.

Add Intents

6/27/2017 • 1 min to read • Edit Online

An intent represents a task or action the user wants to perform. It is a purpose or goal expressed in a user's input, or utterances. Intents match user requests with the actions that should be taken by your app, so you must add intents to help your app understand user requests and respond to them.

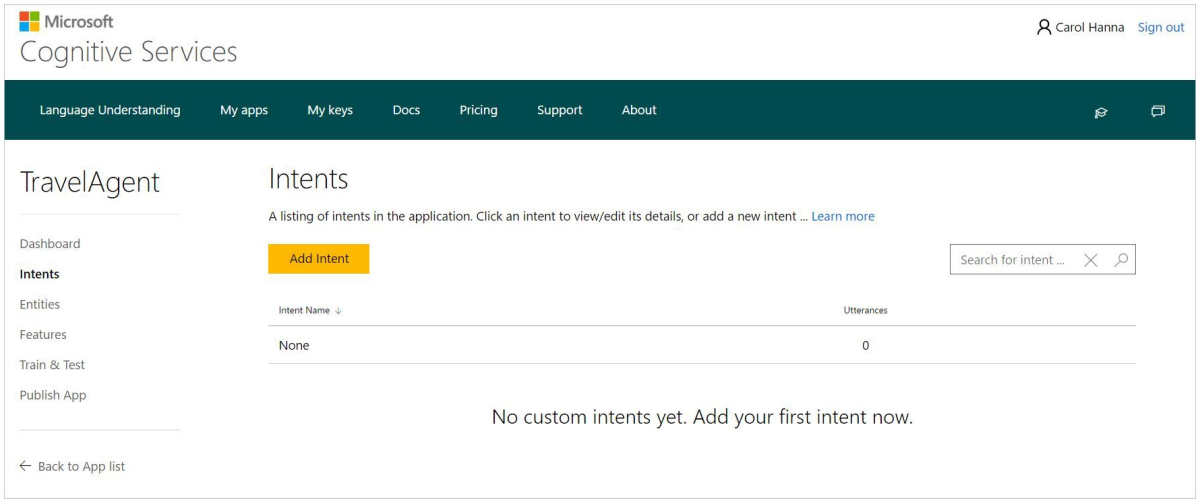
All applications come with the predefined intent, **None**. You should teach it to recognize user statements that are irrelevant to the app. For example, if a user says "Get me a great cookie recipe" in a travel agent app, label that utterance with the **None** intent.

You can add up to **80** intents in a single LUIS app. You add and manage your intents from the **Intents** page that is accessed by clicking **Intents** in your application's left panel.

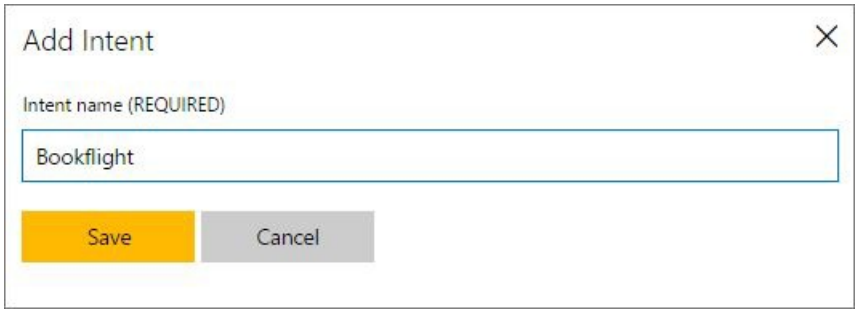
The following procedure demonstrates how to add the "Bookflight" intent in the TravelAgent app.

To add an intent:

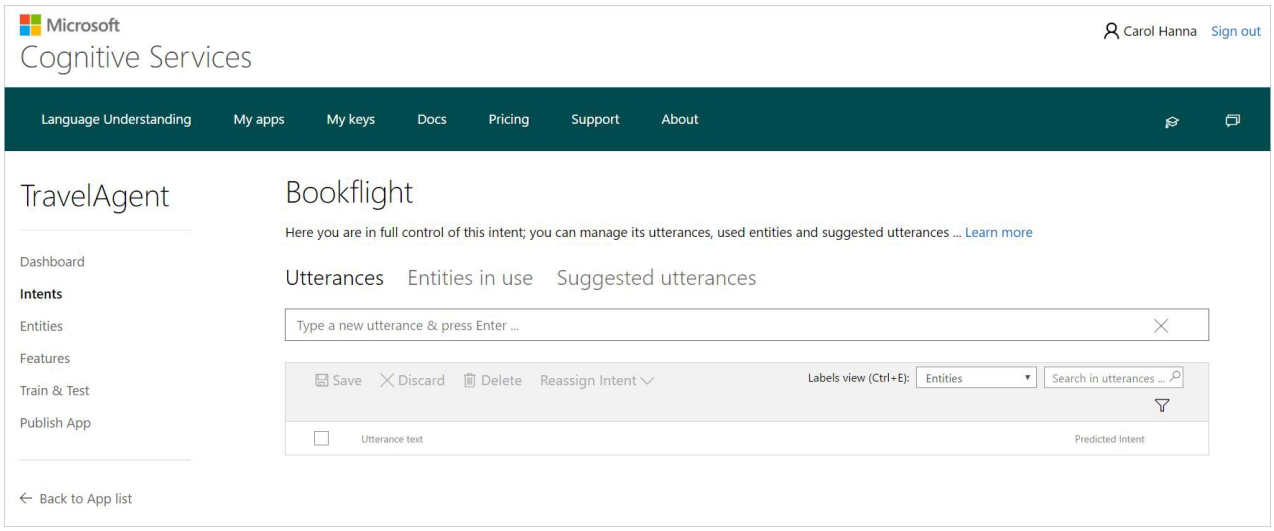
1. Open your app (for example, TravelAgent) by clicking its name on **My Apps** page, and then click **Intents** in the left panel.
2. On the **Intents** page, click **Add intent**.



3. In the **Add Intent** dialog box, type the intent name "BookFlight" and click **Save**.

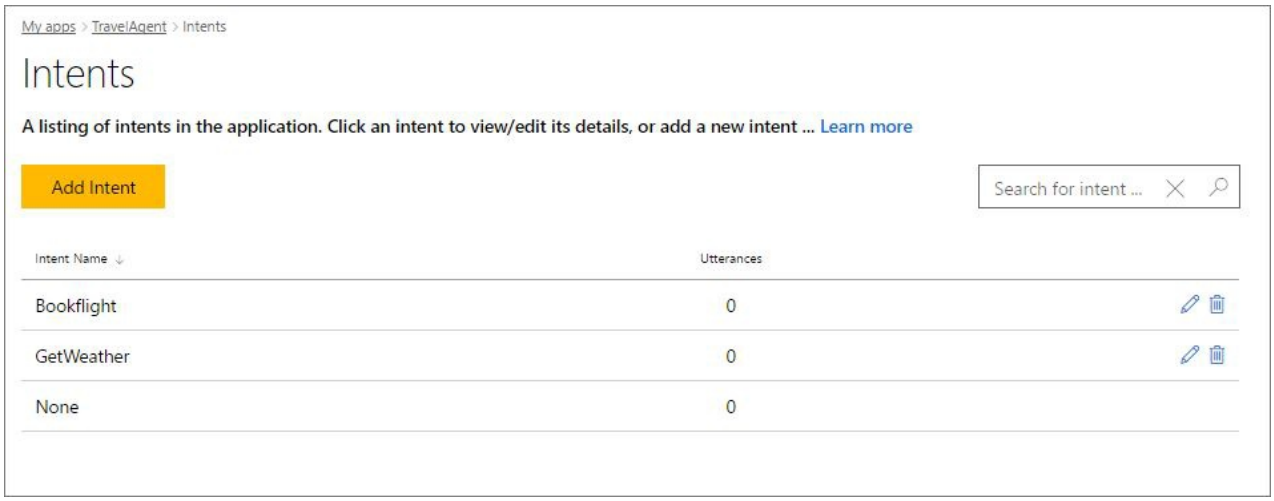


This takes you directly to the intent details page of the newly added intent "Bookflight", like the following screenshot, to add utterances for this intent. For instructions on adding utterances, see Add example utterances.



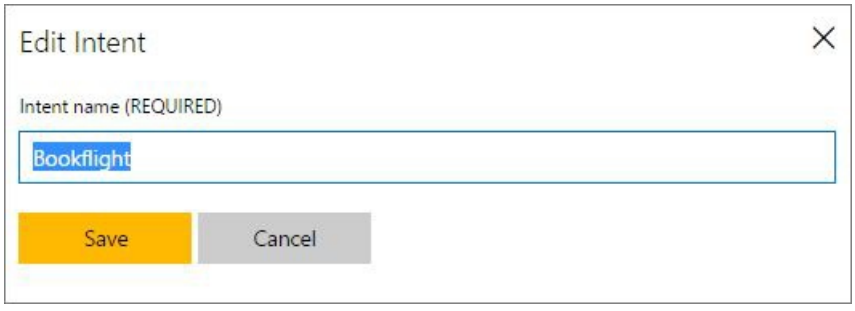
Manage your intents

You can view a list of all your intents and manage them on the **Intents** page, where you can add new intents, rename and delete existing ones, or access intent details for editing.



To rename an intent:

1. On the **Intents** page, click the Rename icon  next to the intent you want to rename.
2. In the **Edit Intent** dialog box, edit the intent name and click **Save**.



To delete an intent:

- On the **Intents** page, click the trash bin icon  next to the intent you want to delete.

To access intent details for editing:

- On the **Intents** page, click the intent name which you want to access its details.

Next steps

After adding intents to your app, now your next task is to start adding example utterances for the intents you've added. For instructions, see Add example utterances.

Add example utterances

6/27/2017 • 6 min to read • Edit Online

Utterances are sentences representing examples of user queries or commands that your application is expected to receive and interpret.

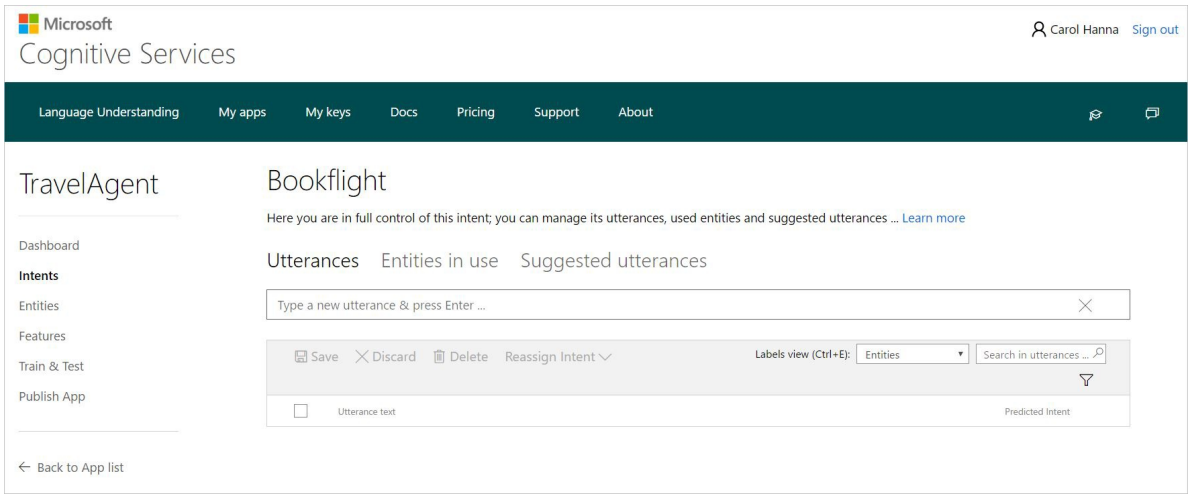
To train LUIS, you need to add example utterances for each intent in your app. LUIS learns from these utterances and your app is able to generalize and understand similar contexts. By constantly adding more utterances and labeling them, you are enhancing your application's language learning experience.

For each intent, add example utterances that trigger this intent and include as many utterance variations as you expect users to say. The more relevant and diverse examples you add to the intent, the better intent prediction you get from your app. For example, the utterance "book me a flight to Paris" may have variations like as "Reserve me a flight to Paris", "book me a ticket to Paris", "Get me a ticket to Paris", "Fly me to Paris", and "Take me on a flight to Paris".

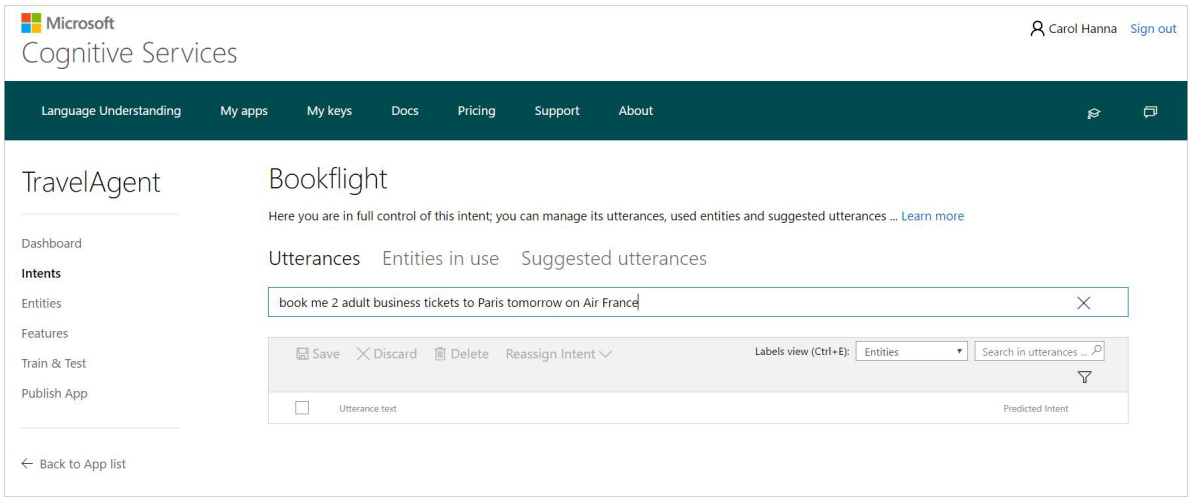
Utterances are added to an intent on the **Utterances** tab of the intent page. The following steps describe how to add example utterances to an intent. In this example we use the "BookFlight" intent in the TravelAgent app.

To add an utterance:

1. Open the TravelAgent app by clicking its name on **My Apps** page, and then click **Intents** in the left panel.
2. On the **Intents** page, click the intent name "BookFlight" to open its details page, with **Utterances** as the current tab, like the screen below.



3. Type `book me 2 adult business tickets to Paris tomorrow on Air France` as a new utterance in the text box, and then press Enter. Note that LUIS converts all utterances to lower case.



4. Repeat the previous step to add more example utterances.
5. Click **Save** to save the added utterances in the utterances list.

Utterances are added to the utterances list in the current intent. To delete one or more utterances from the list, select them and click **Delete**.

Label utterances

After adding utterances, your next step is to label them. Utterances are labeled in terms of intents and entities.

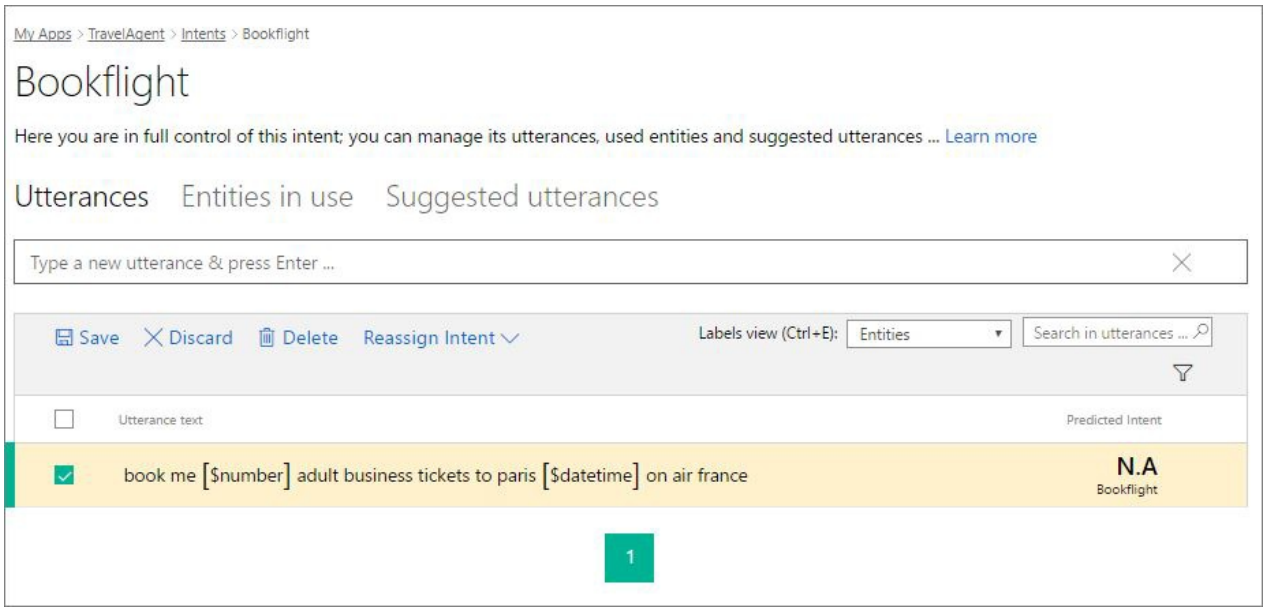
Intent label

Adding an utterance in an intent page means that it is labeled under this intent. That's how an utterance gets an intent label. You can change the intent label of one or more utterances by moving them to another intent. To do this, select the utterances, click **Reassign Intent**, and then select the intent where you want to move them.

Entity label

There are different types of entities: custom entities (which include simple, hierarchical and composite entities) and prebuilt entities. You only need to label custom entities, **because prebuilt entities are detected and labeled automatically by your app.**

For example, look at the utterance `book me 2 adult business tickets to Paris tomorrow on Air France` that you've just added to the "Bookflight" intent in the TravelAgent app. Before you start labeling entities in this utterance, if you have already added `number` and `datetime` as prebuilt entities, you'll notice that "2" and "tomorrow" were automatically detected as prebuilt entities, where "2" is labeled as `number` and "tomorrow" as `datetime`. This looks like the following screenshot.



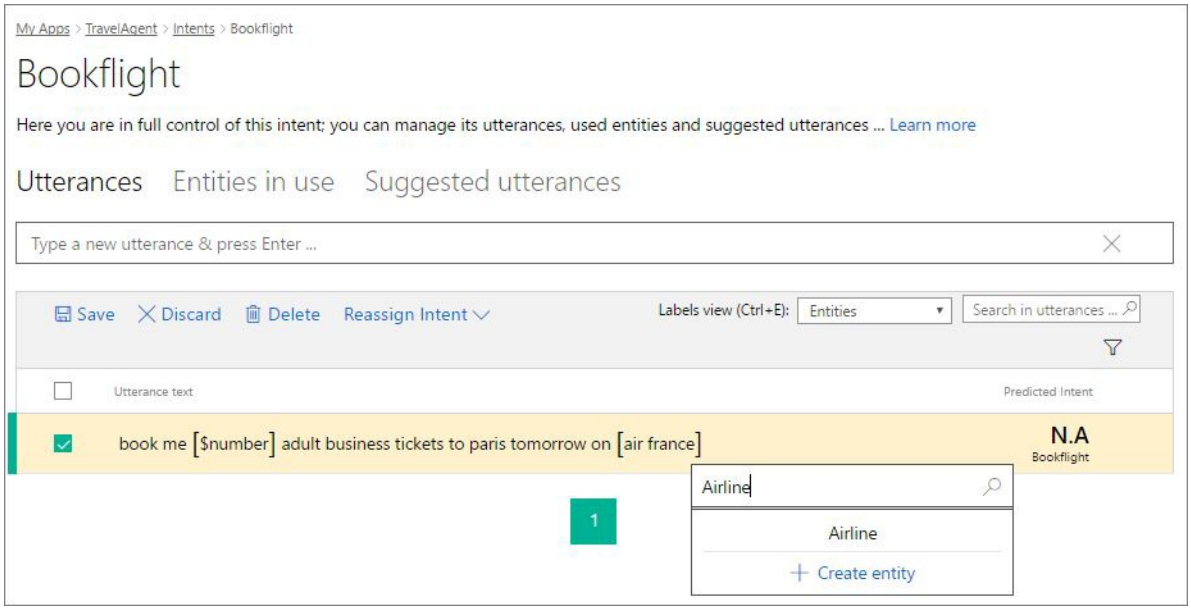
To Learn more about prebuilt entities and how to add them, see Add entities.

In the following procedure, you label custom entities (simple, hierarchical and composite entities) in the utterance `book me 2 adult business tickets to Paris tomorrow on Air France`.

1. Select "Air France" in the utterance mentioned above to label it as entity.

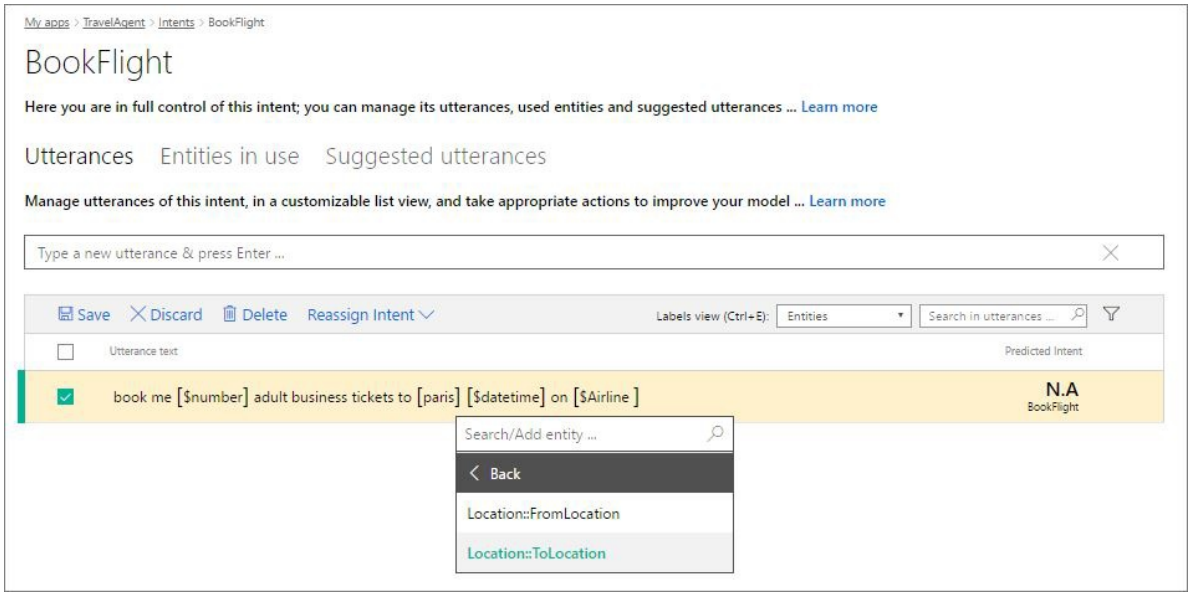
> **NOTE**
>
> When selecting words to label them as entities:
>
> - For a single word, just click it.
> - For a set of two or more words, click at the beginning and then at the end of the set.

2. In the entity drop-down box that appears, you can either click an existing entity (if available) to select it, or add a new entity by typing its name in the text box and clicking **Create entity**. Now, create the simple entity "Airline". Type "Airline" in the text box and then click **Create entity**.



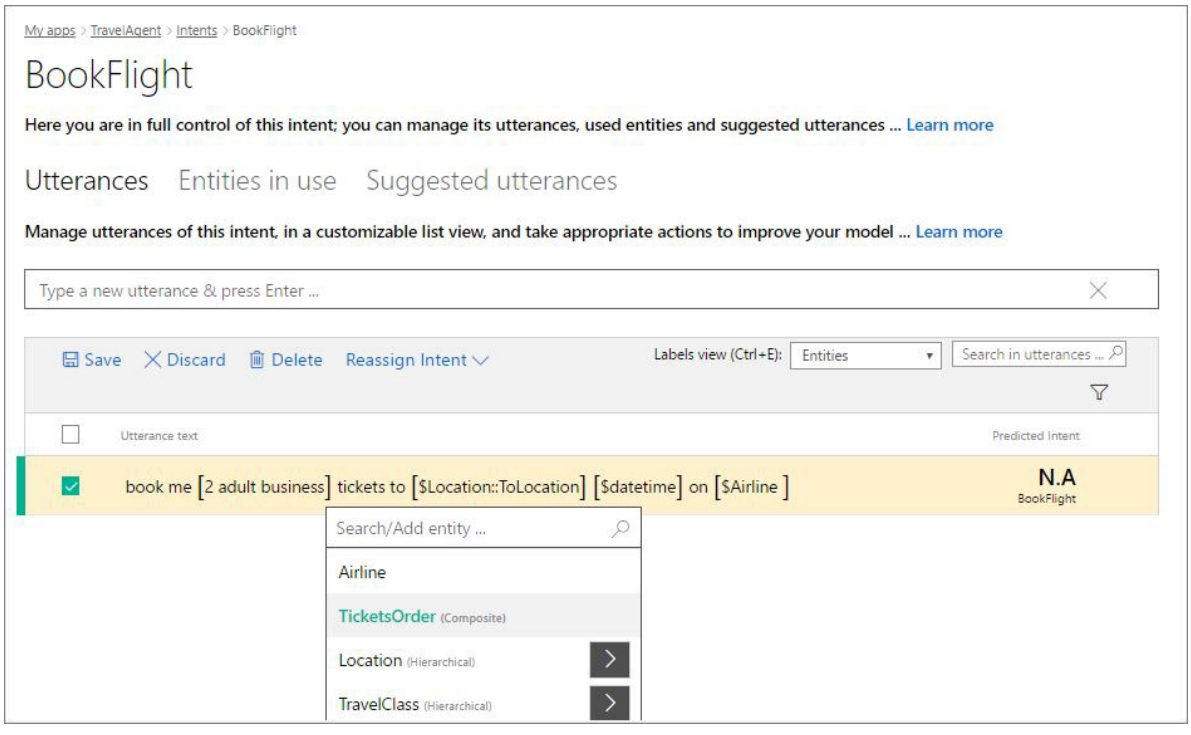
> **NOTE**
>
> This way is used to create a simple entity on the spot (while labeling utterances). Hierarchical and composite entities can only be created from the **Entities** page. For more instructions, see Add entities.

3. Click "Paris" in the same utterance, then click "ToLocation" in the entity drop-down box as the entity label. "ToLocation" is a hierarchical entity that must be added on the **Entities** page. To learn more about hierarchical entities and how to add them, see Add entities.



4. Click "2" (labeled as "number") and then click **Remove label** in the drop-down box. We remove this label as we do not want "2" to be interpreted individually, but to be part in a composite entity that we're going to label now.

5. Select the phrase "2 adult business" by clicking at the beginning and at the end of the phrase, then click "TicketsOrder" in the drop-down box. "TicketsOrder" is a composite entity that must be added on the **Entities** page. To learn more about composite entities and how to add them, see Add entities.



6. Click **Save**.

To remove an entity label:

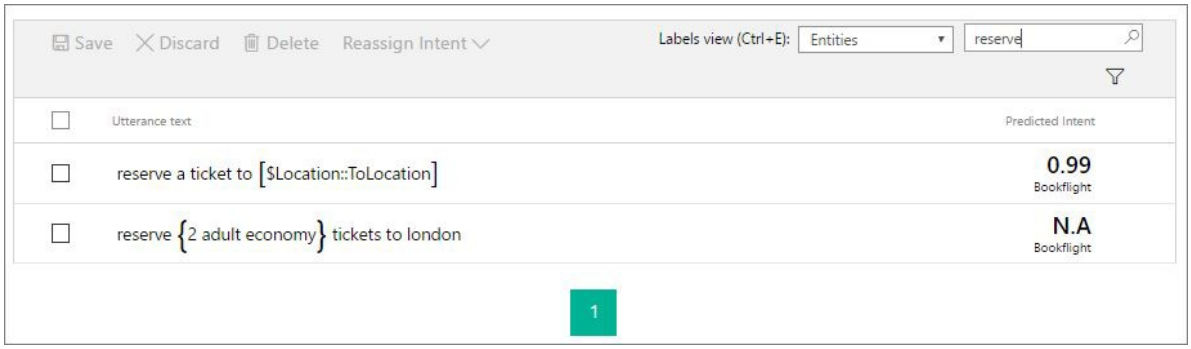
- Click the entity you want to remove its label and click **Remove label** in the entity drop-down box that appears. Then, click **Save** to save this change.

Search in utterances

Searching allows you to find utterances that contain a specific text (words/phrases). For example, sometimes you will notice an error that involves a particular word, and may want to find all the examples including it.

To search in utterances:

- Type the search text in the search box at the top right corner of the utterances list and press Enter. The utterances list will be updated to display only the utterances including your search text. For example, in the following screenshot, only the utterances which contain the search word "reserve" is displayed.



To cancel the search and restore your full list of utterances, delete the search text you've just typed.

Filter utterances

When you have a large number of utterances, it is useful to filter utterances in order to limit their view based on one or more filtering criteria.

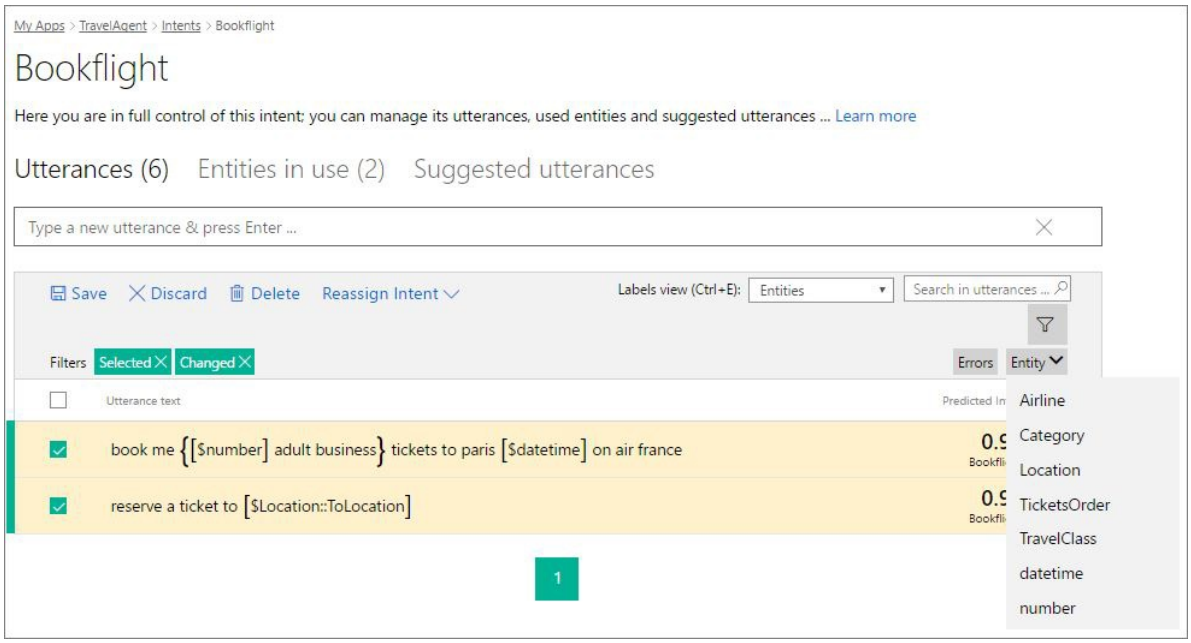
You can apply one or more filters on utterances, as needed. These are the available filters that you can use:

- **Selected:** displays only the currently selected utterances.
- **Changed:** displays only utterances which contain unsaved changes.
- **Errors:** displays only utterances which contain errors.
- **Entity:** displays only utterances that contain a specific entity.

> **NOTE**
> Utterances which contain unsaved changes are highlighted in light yellow. You can click **Save** to save changes or **Discard** to discard changes.

To apply filter(s):

1. Click the filter button , at the top right corner of the utterances list, to display all filters.
2. Click on the filter(s) that you want to apply on utterances. For the **Entity** filter, select the entity by which you want to filter utterances.



The applied filters appear as green buttons at the top left corner of the utterances list.

- To clear an applied filter, click its button at the top left corner.
- To clear all applied filters, click all their corresponding buttons, or just click the filter button  .

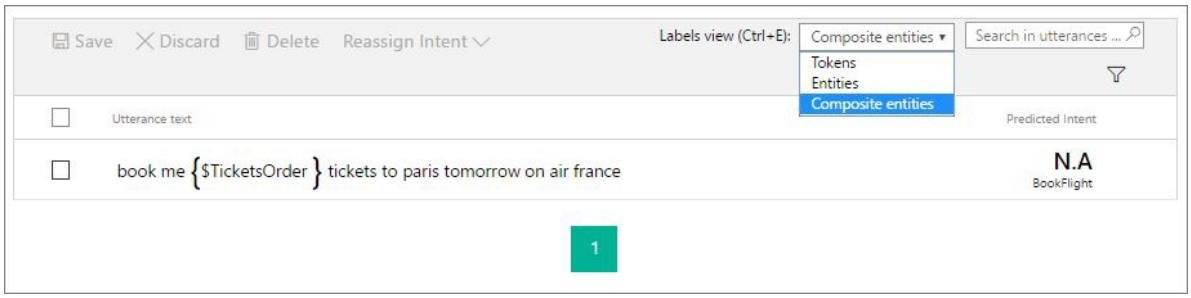
Choose labels view in utterances

You can control how you see the words labeled as entities in the utterances by selecting one of the available views for labeled entities. At the top of the utterances list, select a view from the **Labels view** list. These are the available views:

- **Entities:** Shows entity-labeled words in tagged format (entity labels), enclosed in square brackets, with only composite entities displayed as normal text between curly brackets.

- **Tokens:** Shows all entity-labeled words in text format (normal text), enclosed in square brackets except composite entities in curly brackets.

- **Composite entities:** Shows only the words labeled as composite entities in tagged format (entity labels), enclosed in curly brackets.



You can press `Ctrl+E` to quickly switch between views.

Add entities

6/27/2017 • 9 min to read • Edit Online

Entities are key data in your application's domain. An entity represents a class including a collection of similar objects (places, things, people, events or concepts). Entities describe information relevant to the intent, and sometimes they are essential for your app to perform its task.

For example, a News Search app may include entities such as "topic", "source", "keyword" and "publishing date", which are key data to search for news. In a travel booking app, the "location", "date", "airline", "travel class" and "tickets" are key information for flight booking (relevant to the "Bookflight" intent), that you can add as entities.

You do not need to create entities for every concept in your app, but only for those required for the app to take action. You can add up to **30** entities in a single LUIS app.

You can add, edit or delete entities in your app through the **Entities list** on the **Entities** page. Luis offers many types of entities; prebuilt entities, custom machine learned entities and list entities.

Add prebuilt entities

LUIS provides a set of prebuilt (system-defined) entities, covering many examples of the most common knowledge concepts such as date, age, temperature, percentage, dimension, cardinal and ordinal numbers, etc.

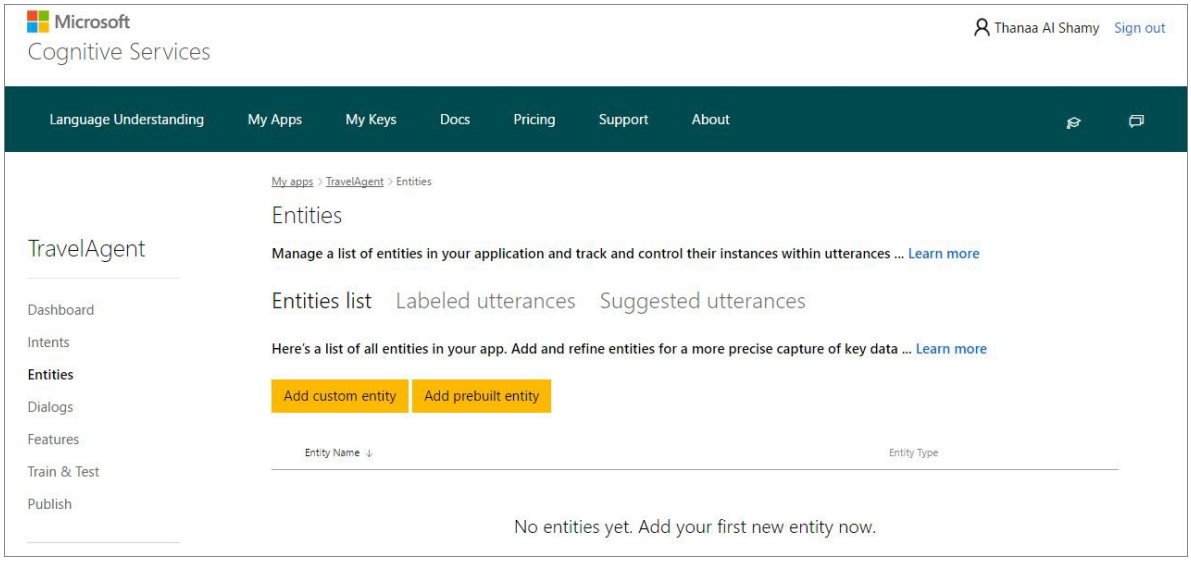
For example, the TravelAgent app may receive a request like "Book me a flight to Boston on May 4". This will require your app to understand date and time words in order to interpret the request properly. Rather than creating entities for such concepts from scratch, you can enable a ready-made prebuilt entity called "datetime".

As another example, a travel booking request like "Book me the first flight to Boston" might be sent to your app. This requires understanding of ordinal words (e.g. first, second). Therefore, you must add a prebuilt entity called "ordinal", for your app to recognize ordinal numbers.

For a full list of prebuilt entities and their use, see Prebuilt Entities List.

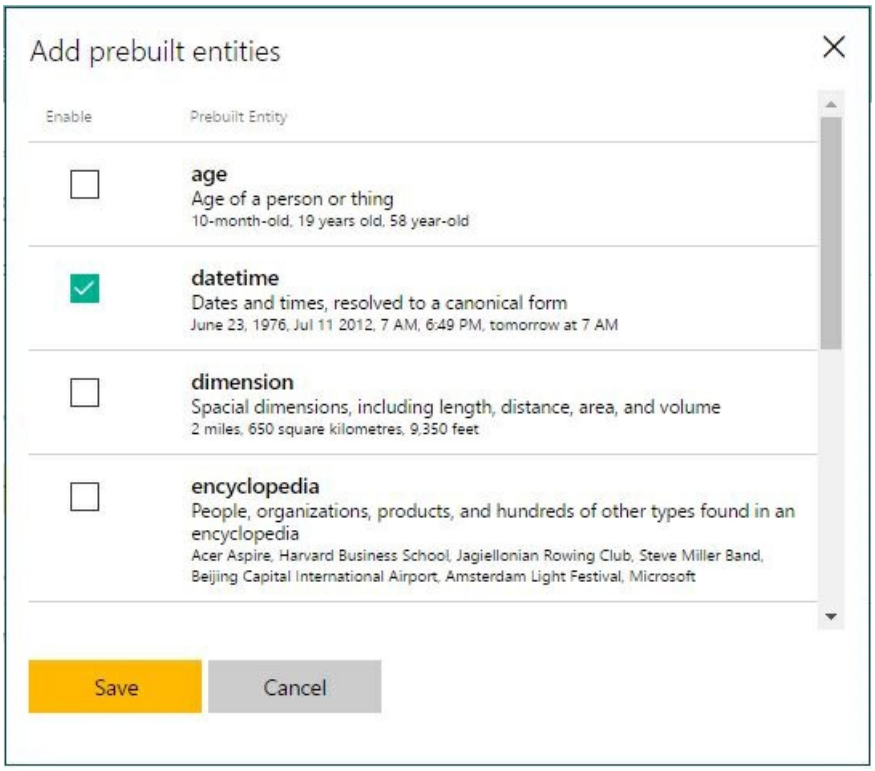
To add a prebuilt entity:

1. Open the TravelAgent app by clicking its name on **My Apps** page, and then click **Entities** in the left panel.
2. On the **Entities** page, click **Add prebuilt entity**.



3. In **Add prebuilt entities** dialog box, click the prebuilt entity you want to add (e.g. "datetime"), and then

click **Save**.



Custom entities

Custom entities are the entities you create in your app. These are the available types of custom entities:

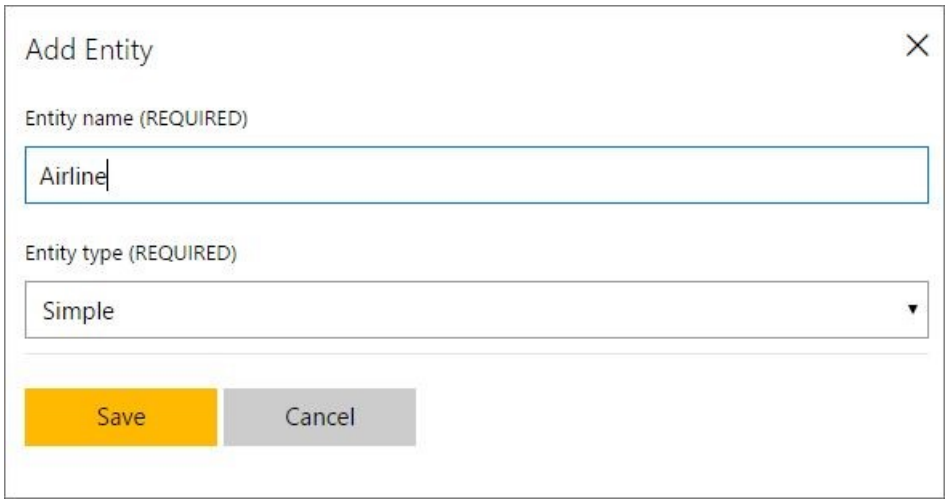
- **Simple:** a generic entity.
- **Hierarchical:** a parent including children (sub-types) which are dependent on the parent.
- **Composite:** a compound of two or more separate entities combined together forming a composite and treated as a single entity.
- **List:** a customized list of entity values, to be used as keywords or identifiers to recognize the entity within utterances.

Add simple entities

A simple entity is a generic entity that describes a single concept. In the example of the TravelAgent app, a user may say "Book me a flight to London tomorrow on British Airways", where "British Airways" is the name of an airline company. In order to capture the notion of airline names, let's create the entity "Airline".

To add a simple entity:

1. Open the TravelAgent app by clicking its name on **My Apps** page, and then click **Entities** in the left panel.
2. On the **Entities** page, click **Add custom entity**.
3. In the **Add Entity** dialog box, type "Airline" in the **Entity name** box, select **Simple** from the **Entity type** list, and then click **Save**.



Add hierarchical entities

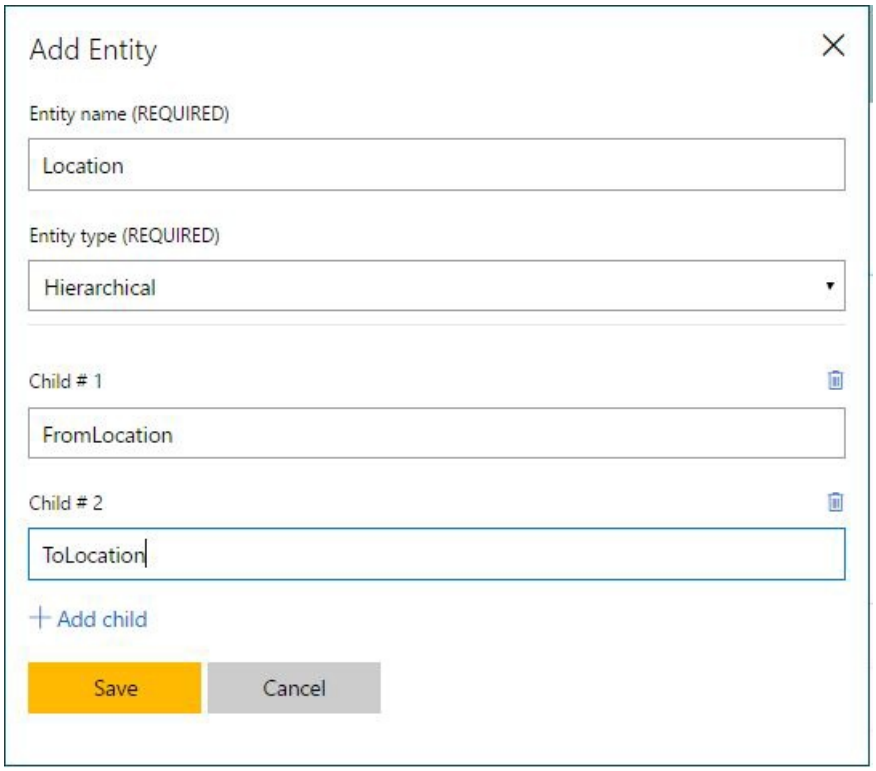
You can define relationships between entities based on hereditary hierarchical patterns, where the generic entity acts as the parent and the children are sub-types under the parent, and they share the same characteristics. For example, in the TravelAgent app, you can add three hierarchical entities:

- "Location", including the entity children "FromLocation" and "ToLocation", representing source and destination locations.
- "TravelClass", including the travel classes as children ("first", "business" and "economy")
- "Category", including ticket categories ("adult", "child" and "infant").

Do the following steps to add hierarchical entities and make sure to add the children at the same time you are creating the parent entity. You can add up to 10 entity children for each parent.

To add a hierarchical entity:

1. Open the TravelAgent app by clicking its name on **My Apps** page, and then click **Entities** in the left panel.
2. On the **Entities** page, click **Add custom entity**.
3. In the **Add Entity** dialog box, type "Location" in the **Entity name** box, and then select **Hierarchical** from the **Entity type** list.



4. Click **Add Child**, and then type "FromLocation" in **Child #1** box.
5. Click **Add Child**, and then type "ToLocation" in **Child #2** box.

> **NOTE**
> To delete a child (in case of a mistake), click the trash bin icon next to it.

6. Click **Save**.

Add composite entities

You can also define relationships between entities by creating “composite entities”. A composite entity is created by combining two or more existing entities (simple or hierarchical) and treating them as one entity. Unlike a hierarchical entity, the composite entity and the children forming it are not in a parent-child relationship. They are independent of each other and they do not need to share common characteristics. The composite pattern enables your app to identify entities, not only individually, but also in groups.

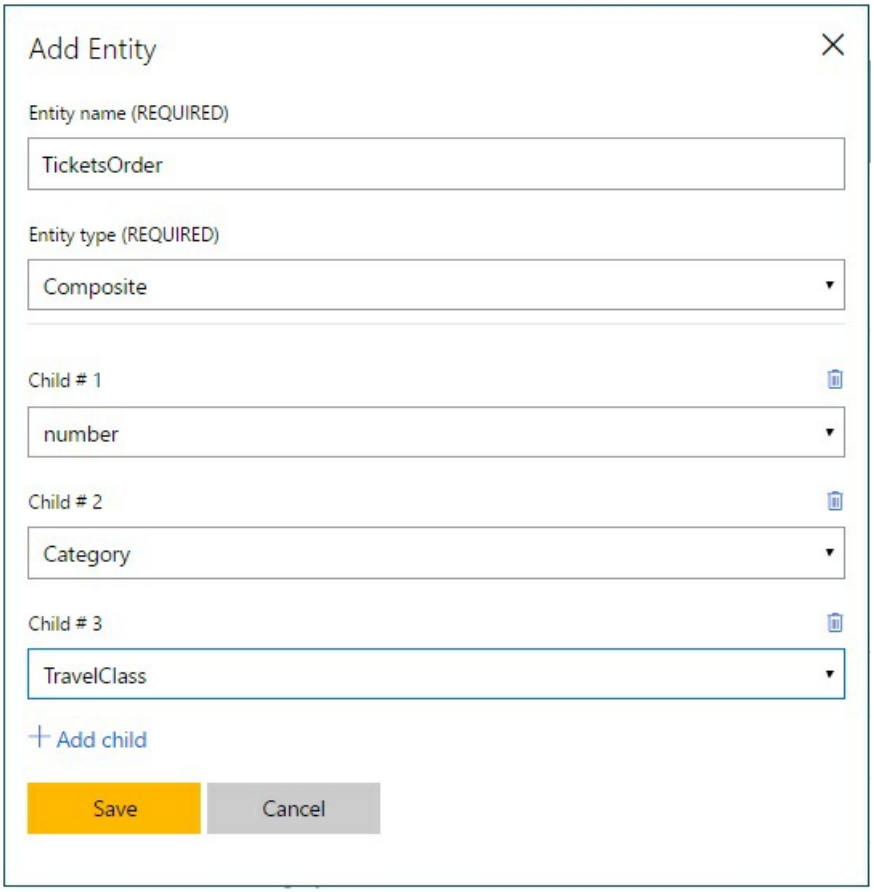
In the TravelAgent app example, a user may say “Book 2 adult business tickets to Paris next Monday”. In this example, we can create a composite entity called “TicketsOrder”, including three children entities: “number”, “category” and "class" which describe the tickets to be booked. Before creating a composite entity, you must first add the entities forming it, if they do not already exist.

To add the entities forming the composite:

1. Add the prebuilt entity “number”. For instructions, see the Add Prebuilt Entities section above.
2. Add the hierarchical entity "Category", including the sub-types: “adult”, “child” and “infant”, and "TravelClass" including "first", "business" and "economy". For more instructions, see the Hierarchical Entities section above.

To add the composite entity:

1. Open the TravelAgent app by clicking its name on **My Apps** page and click **Entities** in the app's left panel.
2. On the **Entities** page, click **Add custom entity**.
3. In the **Add Entity** dialog box, type "TicketsOrder" in the **Entity name** box, and then select **Composite** from the **Entity type** list.
4. Click **Add Child** to add a new child.
5. In **Child #1**, select the entity "number" from the list.
6. In **Child #2**, select the parent entity "Category" from the list.
7. In **Child #3**, select the parent entity "TravelClass" from the list.



8. Click **Save**.

> **NOTE**
> To delete a child (in case of a mistake), click the trash button next to it.

List entities

A list entity is an entity that is defined by a list of all its values. This entity type is identified in utterances, not by active learning of context, but by the direct matching of utterance text to the defined values.

In some use cases, you may have a fixed list of definite values for an entity. In this case, you can create a list entity including these values. For each value, you define a standard form "Canonical Form" and variant forms "Synonyms". Your app will identify this entity in an utterance if the utterance includes an exact match of any of the entity values (canonical forms/synonyms), so the list entity will be predicted as existing in the utterance, with a prediction score of 100%.

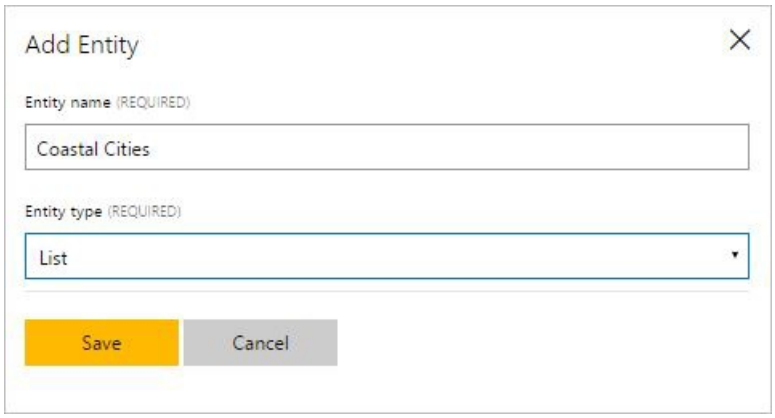
For example, in the TravelAgent app, we can create a list entity named “Coastal Cities”; including values such as “Barcelona, Venice, Miami, etc.” as canonical forms. For each canonical form, you can add synonyms. For example, synonyms of “Barcelona” can be “Capital City of Catalonia”, “BCN”, “Barna” and “Second Spanish City”.

List entity values can be added individually, or collectively by importing them as a JSON file. Furthermore, if your app culture is English, LUIS can propose some relevant values which you can add to your list. The following procedures explain how to add a list entity, and how to add its values by different methods.

To add a list entity:

1. Open the TravelAgent app by clicking its name on **My Apps** page and click **Entities** in the app's left panel.
2. On the **Entities** page, click **Add custom entity**.
3. In the **Add Entity** dialog box, type "Coastal Cities" in the **Entity name** box and select **List** as the **Entity**

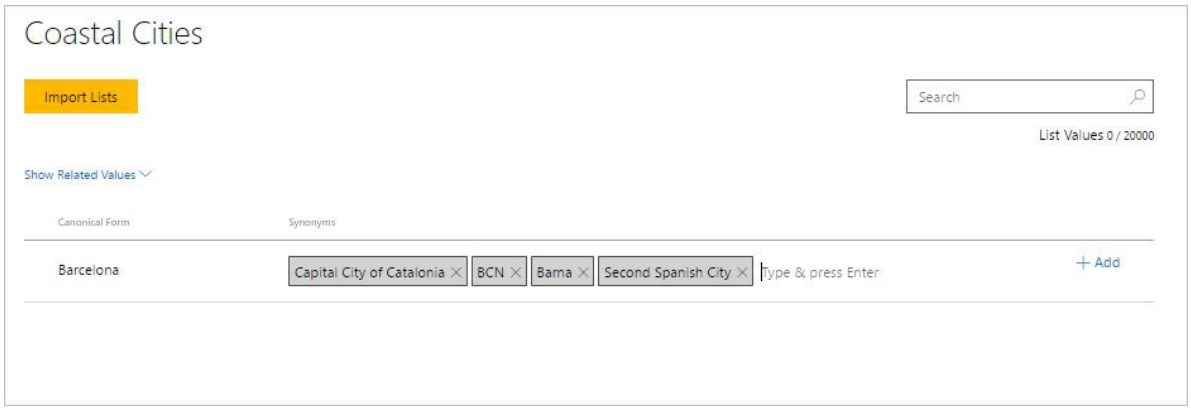
type.



4. Click **Save**. The list entity "Coastal Cities" will be added and its details page will be displayed where you should define its values.

To add list entity values individually:

1. On the "Coastal Cities" list entity page, type a standard form for each value in the **Canonical Form** column (e.g. Barcelona), and type other forms of it in **Synonyms** (e.g. "BCN", "Barna", etc.). After typing each synonym, press Enter.
2. When you finish typing all synonyms, click **Add**.



3. Repeat the above two steps to add more values to the entity list.

To import a list of values as a JSON file:

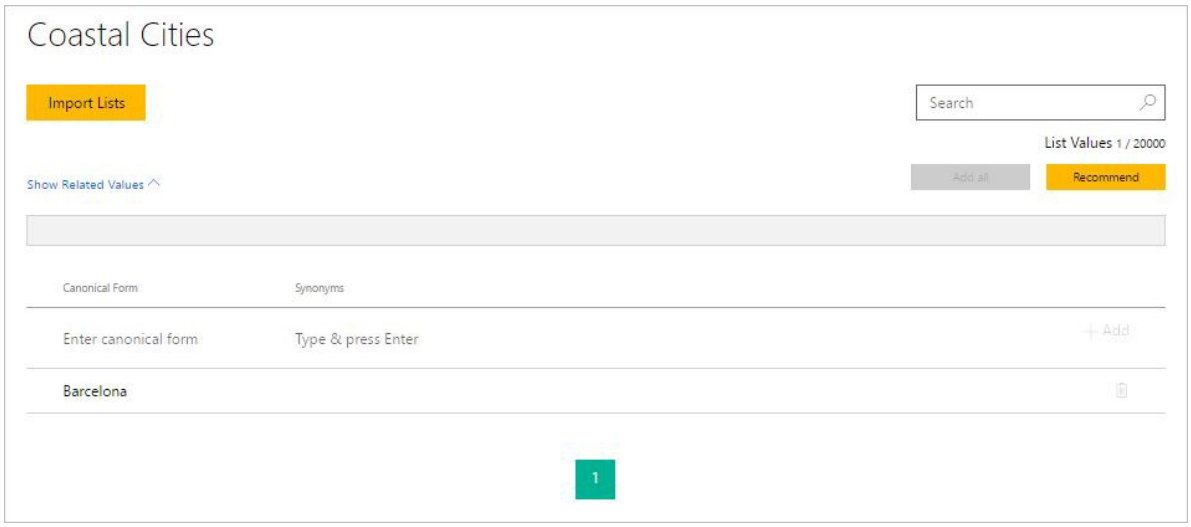
1. On the "Coastal Cities" list entity page, click **Import Lists**.
2. In **Import New Entries** dialog box, click **Choose File** and select the JSON file including the list.

3. To learn about the supported list syntax in JSON, click **Learn about supported list syntax** to expand the dialog and display an example of allowed syntax. To collapse the dialog and hide syntax, click it again.

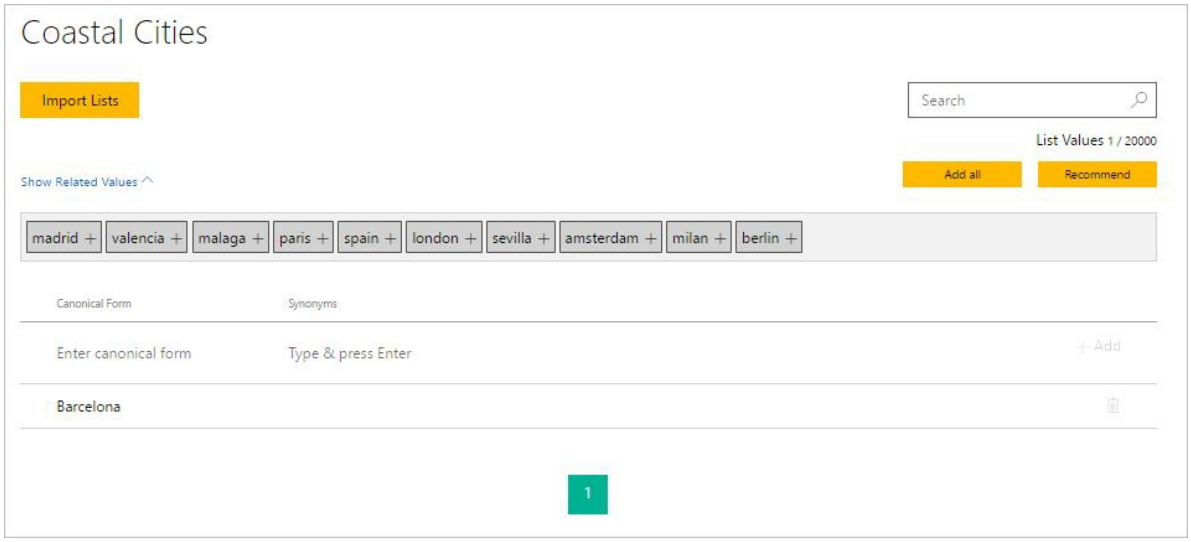
4. Click **Import**.

To get a list of proposed values (only for English culture apps):

1. On the "Coastal Cities" list entity page, with at least one canonical form added (e.g. Barcelona), click **Show Related Values**.



2. Click **Recommend**. You'll get a number of proposed values that are semantically related to the added canonical form (Barcelona in this example). The following screenshot shows the proposed values.



3. Click a value to add it to your list as a canonical form, or click **Add All** to add all values.

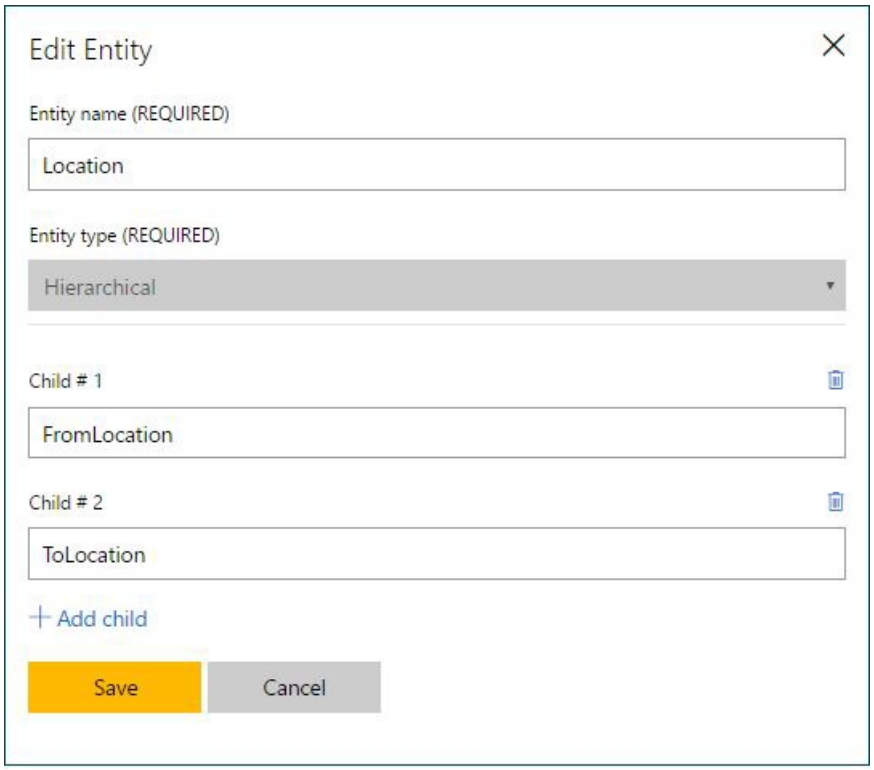
> **NOTE**
>
> - The proposed values are added as canonical forms not synonyms; synonyms should be typed manually.
> - To delete an added canonical form, click the trash bin button corresponding to it.

Edit or delete entities

You can edit or delete entities from the **Entities list** on the **Entities** page of your app.

To edit an entity:

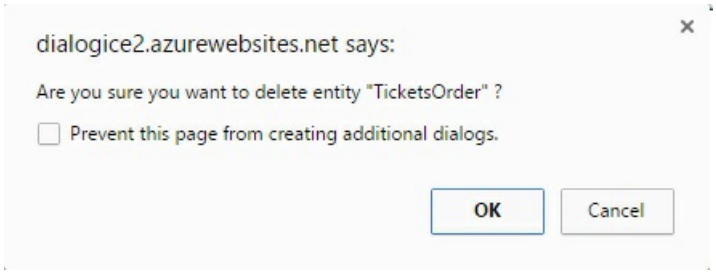
1. On the **Entities** page, click the entity in the **Entities list**.
2. In the **Edit Entity** dialog box, you can edit the entity name and children names, or add more children (for hierarchical/composite entities), but the entity type is not editable.



3. Click **Save**.

To delete an entity:

- In the **Entities list**, click the trash bin icon next to the entity you want to delete. Then, click **OK** in the confirmation message to confirm deletion.

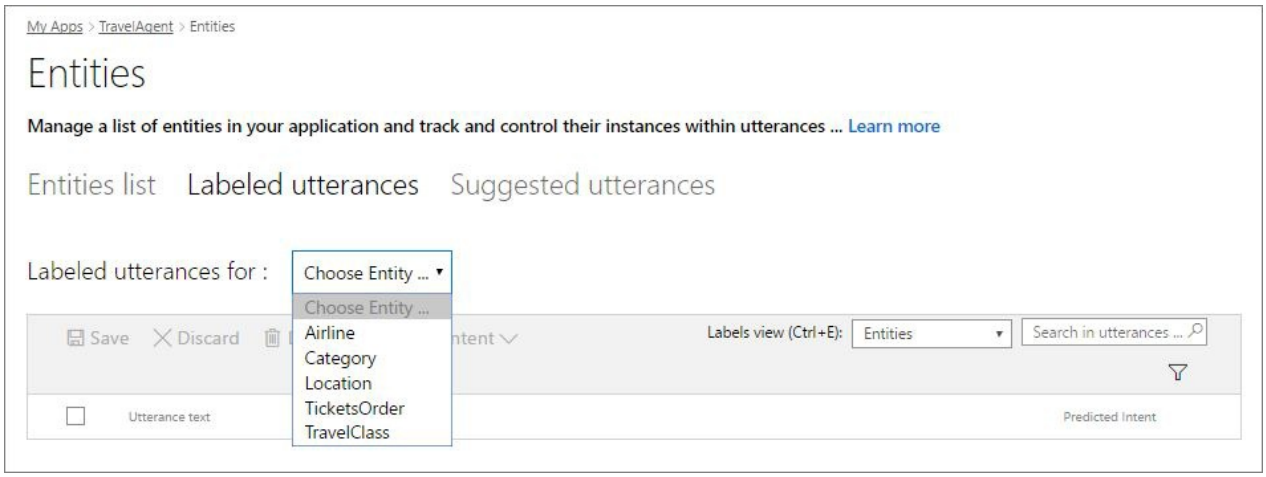


NOTE

- Deleting a hierarchical entity deletes all its children entities.
- Deleting a composite entity deletes only the composite and breaks the composite relationship, but doesn't delete the entities forming it.

Review labeled utterances for entities

To review the labeled utterances that contain a specific entity, click the **Labeled Utterances** tab on the **Entities** page, and choose the entity for which you want to display all labeled utterances. You can modify entity labels in labeled utterances, if required, and then click **Save**.



Next steps

Now that you have added intents, utterances and entities, you have a basic LUIS app ready to be trained and tested for publishing. For more information on how to train and test your app, click here.

Use features to improve your LUIS app's performance

6/27/2017 • 5 min to read • Edit Online

You can add features to your LUIS app to improve its performance. Features help LUIS recognize both intents and entities, by providing hints to LUIS that certain words and phrases are part of a category or follow a pattern. When your LUIS app has difficulty identifying an entity, adding a feature and retraining the LUIS app can often help improve the detection of related intents and entities.

- **Phrase list** features contain some or all of an entity's potential values. If LUIS learns how to recognize one member of the category, it can treat the others similarly.
- **Pattern features** help your LUIS app easily recognize regular patterns that are frequently used in your application's domain, such as the pattern of flight numbers in a travel app or product codes in a shopping app.

> **NOTE**
> Features don't define entities. They simply provide examples or patterns to help LUIS recognize entities and related intents.

Phrase list features

You can create a *phrase list* including a group of values (words or phrases) that belong to the same class and must be treated similarly (for example, names of cities or products). What LUIS learns about one of them will be automatically applied to the others as well.

For example, in a travel agent app, you can create a phrase list named "Cities" that contains the values London, Paris, and Cairo. If you label one of these values as an entity, LUIS will learn to recognize the others.

> **NOTE**
> Phrase list features show LUIS some examples of an entity's potential values. The phrase list does not have to include all members of a category, although it may.

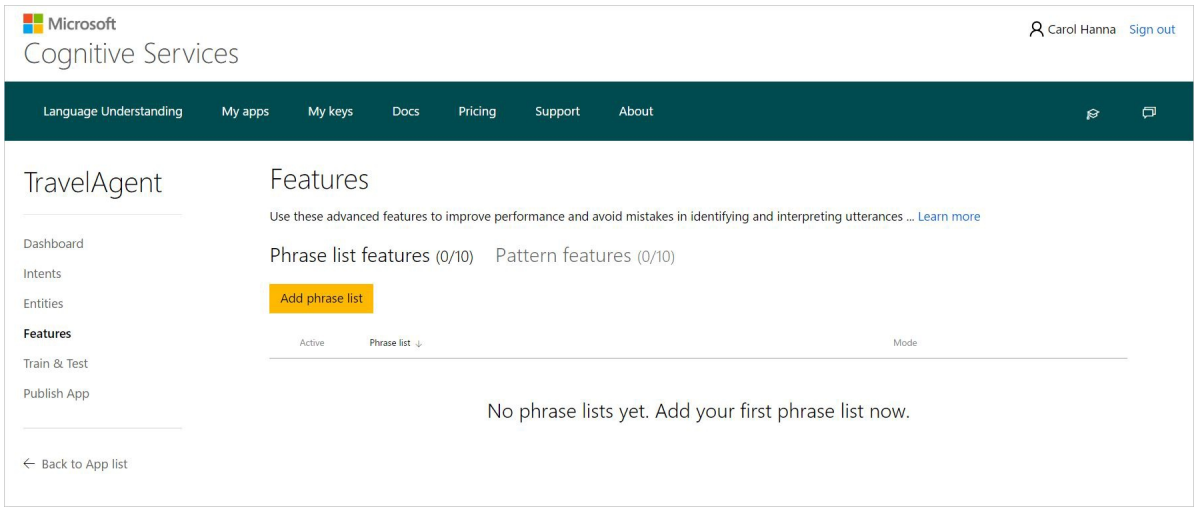
The maximum length of a phrase list is 5000 items. You may have a maximum of 10 phrase lists per LUIS app.

Use phrase lists for rare, proprietary and foreign words

LUIS may be unable to recognize rare and proprietary words, as well as foreign words (outside of the culture of the app), and therefore they should be added to a phrase list feature. This phrase list should be marked non-exchangeable, to indicate that the set of rare words form a class that LUIS should learn to recognize, but they are not synonyms or exchangable with each other.

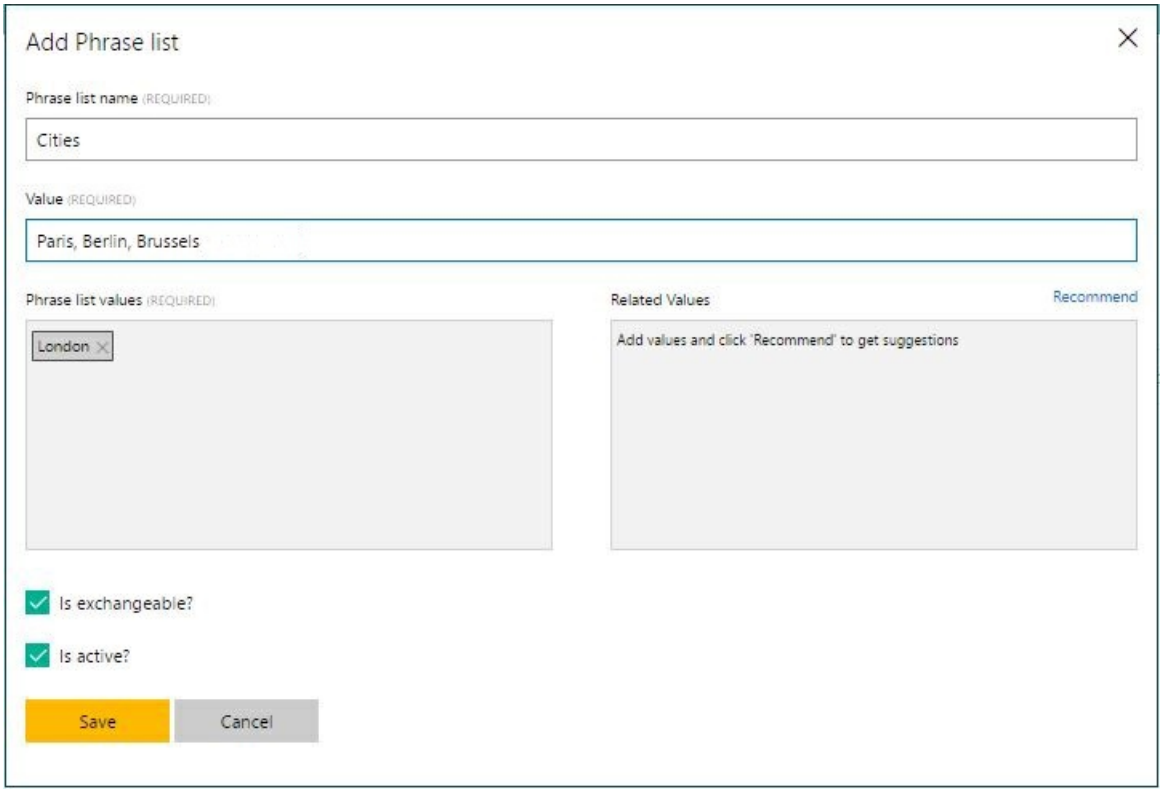
How to add a phrase list

1. Open your app by clicking its name on **My Apps** page, and then click **Features** in your app's left panel.
2. On the **Features** page, click **Add phrase list**.

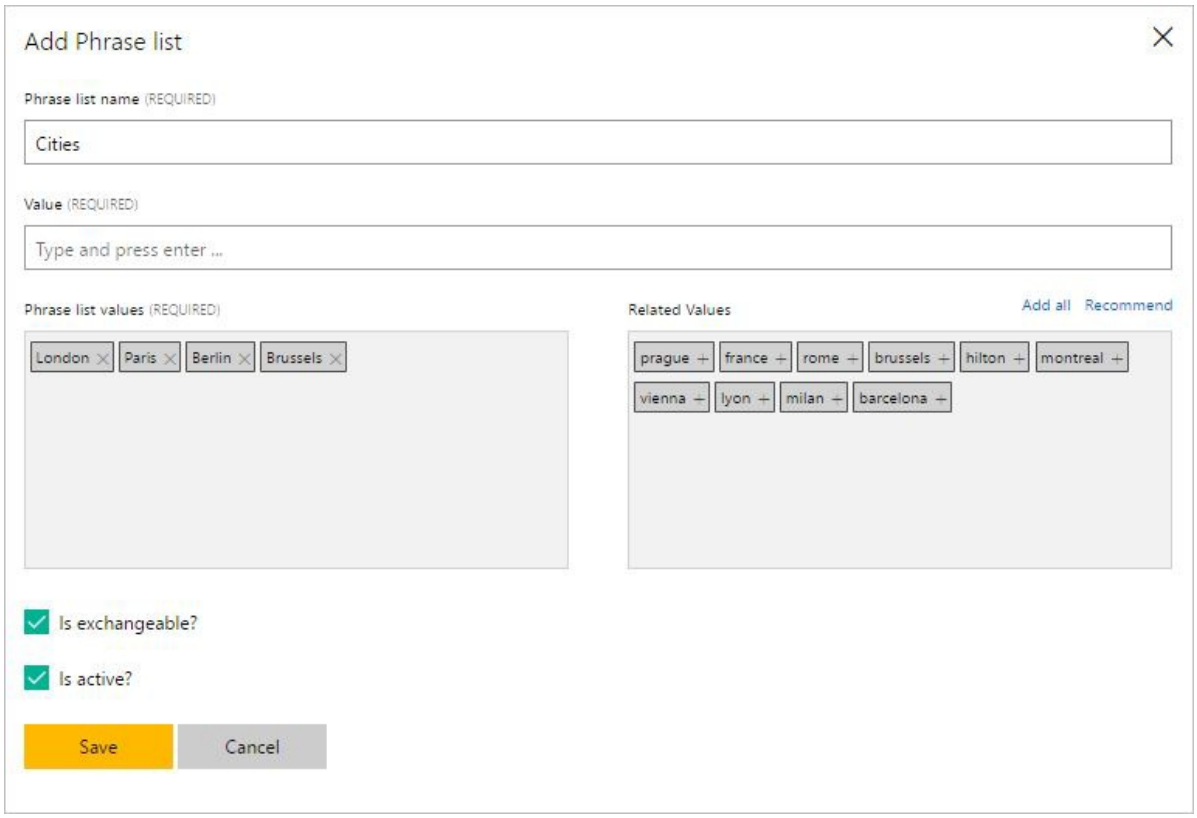


3. In the **Add Phrase List** dialog box, type "Cities" as the name of the phrase list in the **Phrase list name** box.

4. In the **Value** box, type the values of the phrase list. You can type one value at a time, or a set of values separated by commas (e.g. London, Paris, Berlin, Brussels, etc), and then press Enter.



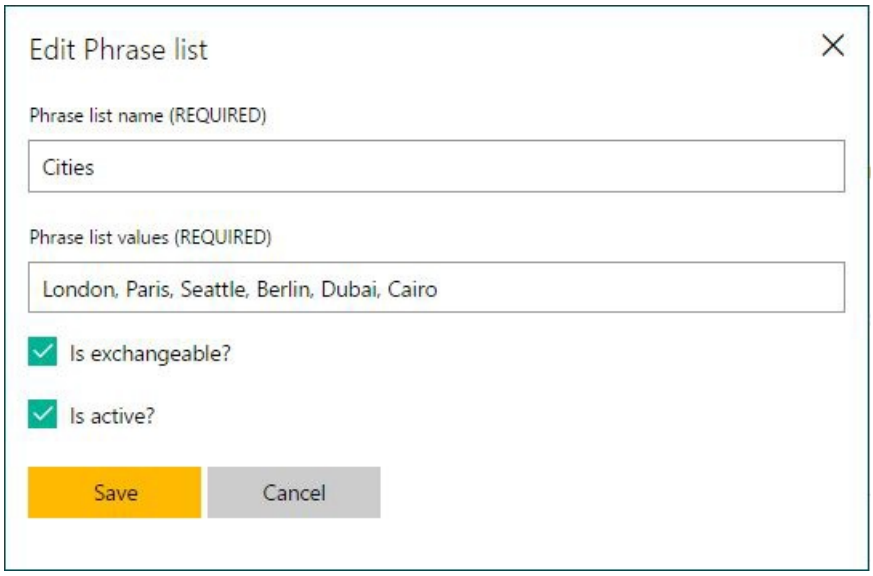
5. If your app culture is English, LUIS can propose some related values to add to your phrase list. Click **Recommend** to get a group of proposed values that are semantically related to the added value(s). You can click any of the proposed values to add it, or click **Add All** to add them all.



6. Click **Is exchangeable** if the added phrase list values are alternatives that can be used interchangeably.
7. Click **Is active** if you want this phrase list to be active (i.e. applicable and used) in your app.
8. Click **Save**. The phrase list will be added to phrase list features on the **Features** page.

To edit a phrase list:

- Click the name of the phrase list (e.g. Cities) on the **Features** page (the previous screenshot). In the **Edit Phrase List** dialog box that opens, make any required editing changes and then click **Save**.



To delete a phrase list:

- Click the trash bin button  next to the phrase list name on the **Features** page.

Pattern features

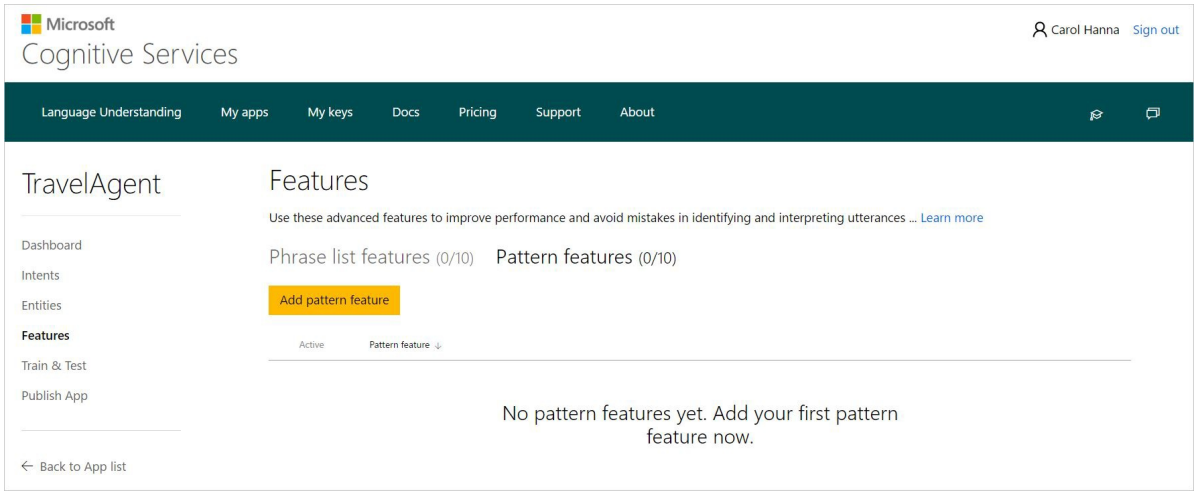
Pattern features define a regular expression to help LUIS recognize regular patterns that are frequently used in your application's domain, such as the pattern of flight numbers in a travel app or product codes in a shopping app.

This will help LUIS easily recognize the string of the defined pattern in utterances, and thus classify it correctly. For example, in a travel app, flight numbers might follow a regular pattern of two letters followed by three digits.

> [!NOTE:] A pattern feature isn't meant to guarantee exact matching every time an utterance matches regular expression. It's just a hint to LUIS that an entity takes a certain form. LUIS still takes context into account when labeling entities that match a pattern.

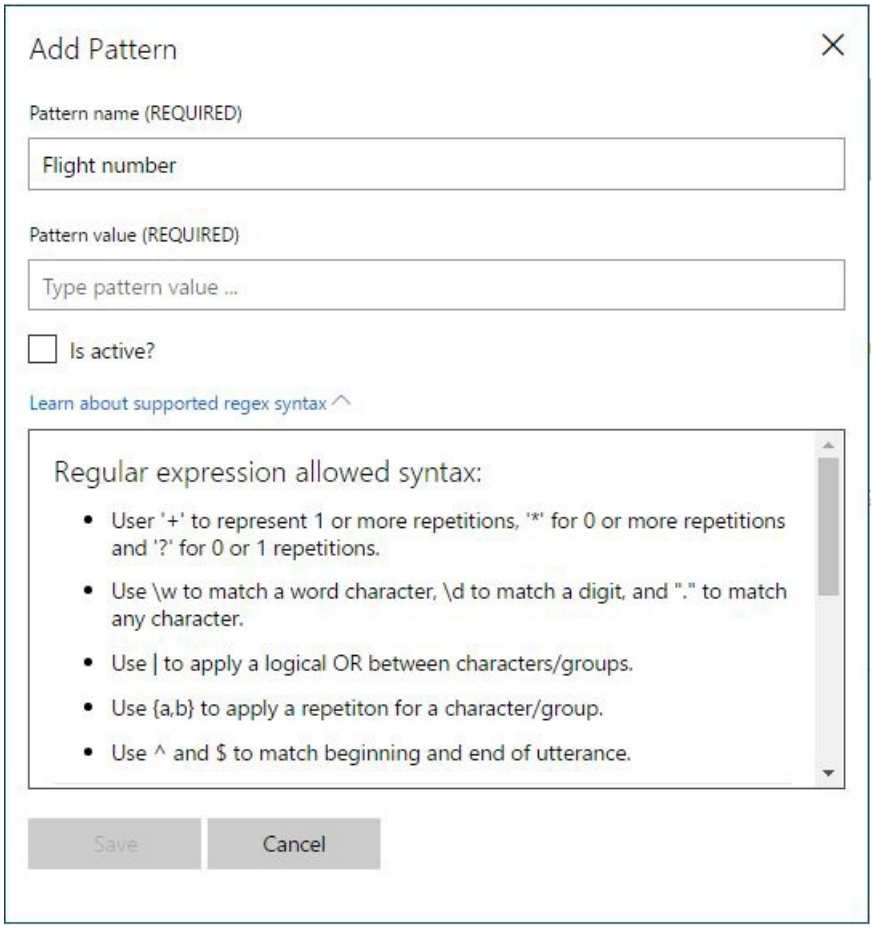
How to add a pattern

1. Open your app by clicking its name on **My Apps** page, and then click **Features** in your app's left panel.
2. On the **Features** page, click the **Pattern Features** tab, and then click **Add Pattern Feature**.

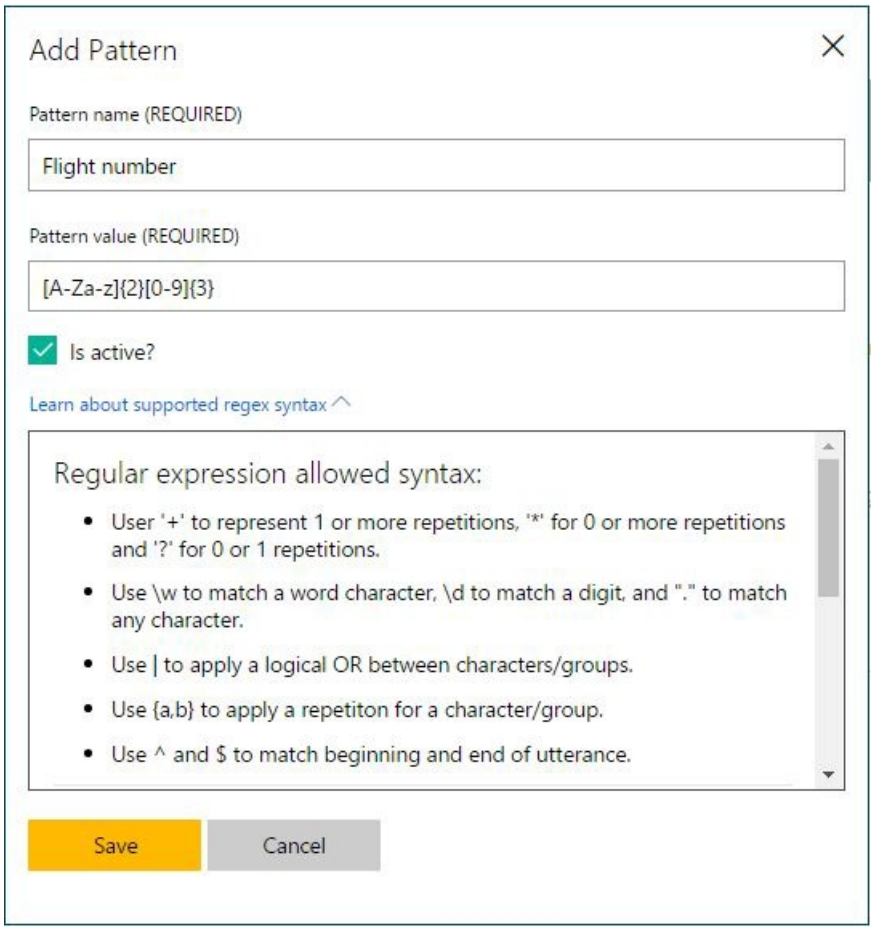


3. In the **Add Pattern** dialog box, type "Flight number" in the **Pattern name** text box.
4. To learn about the supported regex syntax, click **learn about supported regex syntax** to expand the

dialog and display it, as in the screen below. To collapse the dialog and hide syntax, click it again.



5. In the **Pattern value** text box, type [A-Za-z]{2}[0-9]{3} as the value of the flight number pattern.



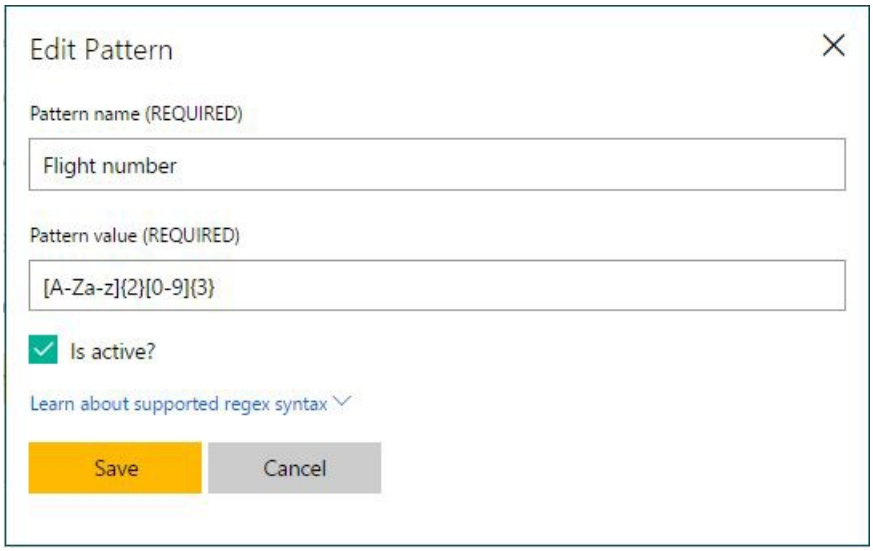
6. Click **Is active** if you want this pattern to be active (i.e. applicable and used) in your app. A feature is active by

default.

7. Click **Save**. The pattern will be added to pattern features on the **Features** page.

To edit a pattern:

- Click the pattern name in the list of pattern features. In the **Edit Pattern** dialog box that opens, make the required editing changes and then click **Save**.



To delete a pattern:

- Click the trash bin button  next to the pattern name in the list of pattern features.

Next steps

After adding a pattern, train and test the app again to see if performance improves.

Train and test your app

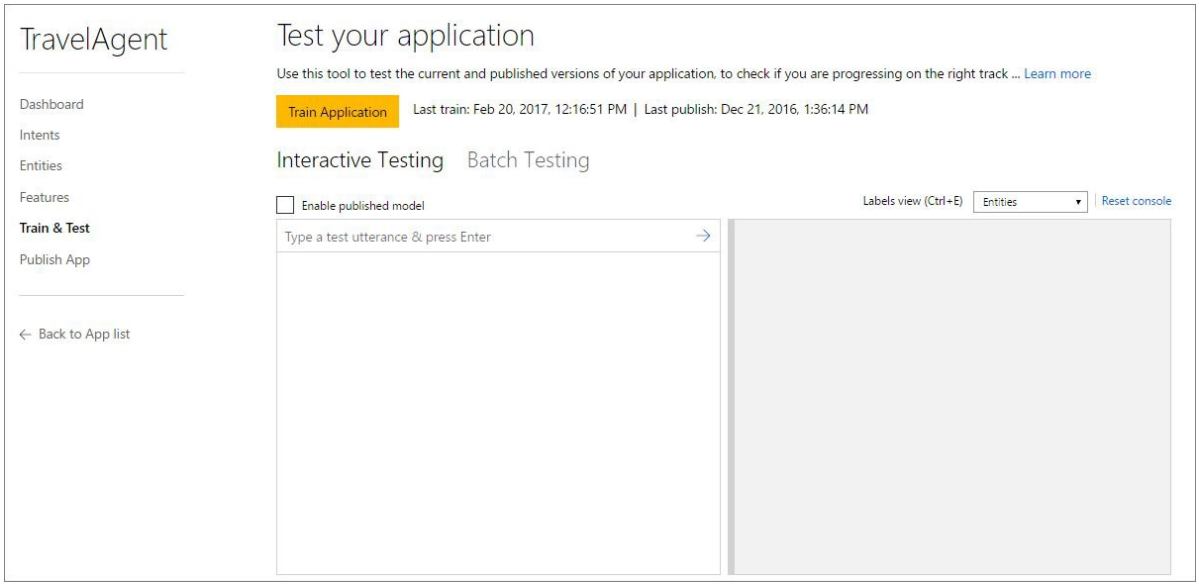
6/27/2017 • 8 min to read • Edit Online

Train to teach your app

You should continuously work on your application to refine it and improve its language understanding. Whenever you make updates by adding, editing or deleting entities, intents or utterances in your current model, you'll need to train your app before testing and publishing. When you "train" a model, LUIS generalizes from the examples you have labeled, and builds model to recognize the relevant intents and entities in the future, thus improving its classification accuracy.

To train your current model:

1. Access your app by clicking its name on **My Apps** page.
2. In your app, click **Train & Test** in the left panel.
3. On the **Test App** page, click **Train Application** to train the current model on the latest updates.



> **NOTE**
> If you have one or more intents in your app which do not contain example utterances, you'll not be able to train your app until you add utterances for all your intents. For more information, see Add example utterances.

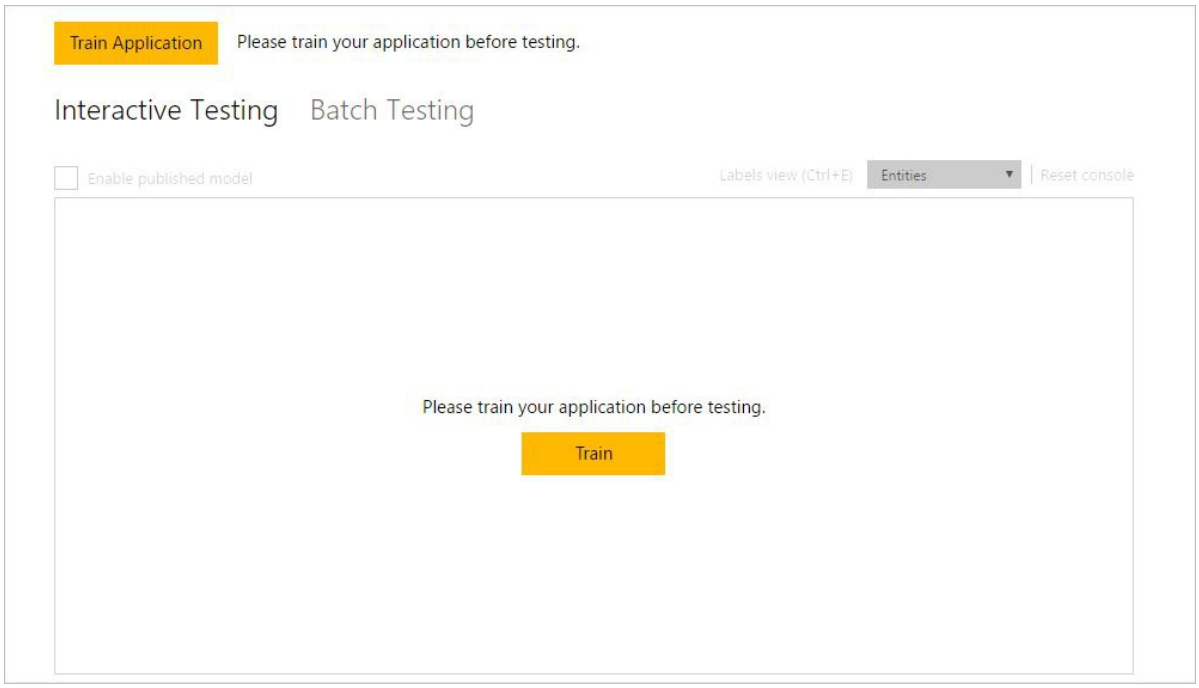
Test your app

LUIS provides two types of testing; interactive testing and batch testing. You can use any of them by using the corresponding tab on the **Test App** page.

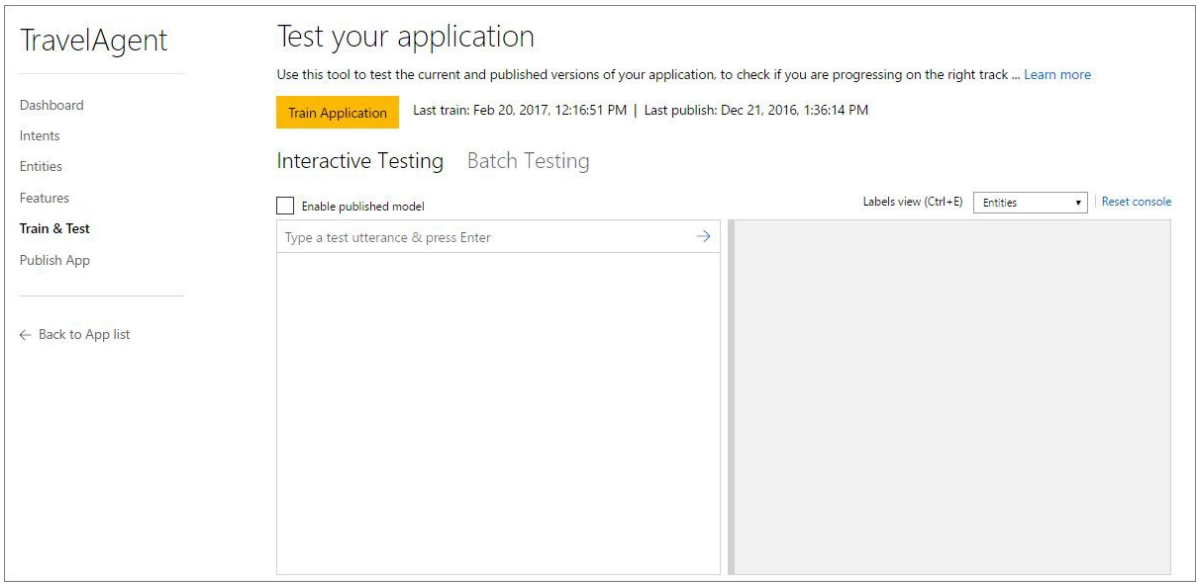
To access the Test App page:

1. Access your app by clicking its name on **My Apps** page,
2. Click **Train & Test** in your application's left panel to access the **Test App** page.

- If you haven't already trained your current model on recent updates, then your test page will look like this screenshot:



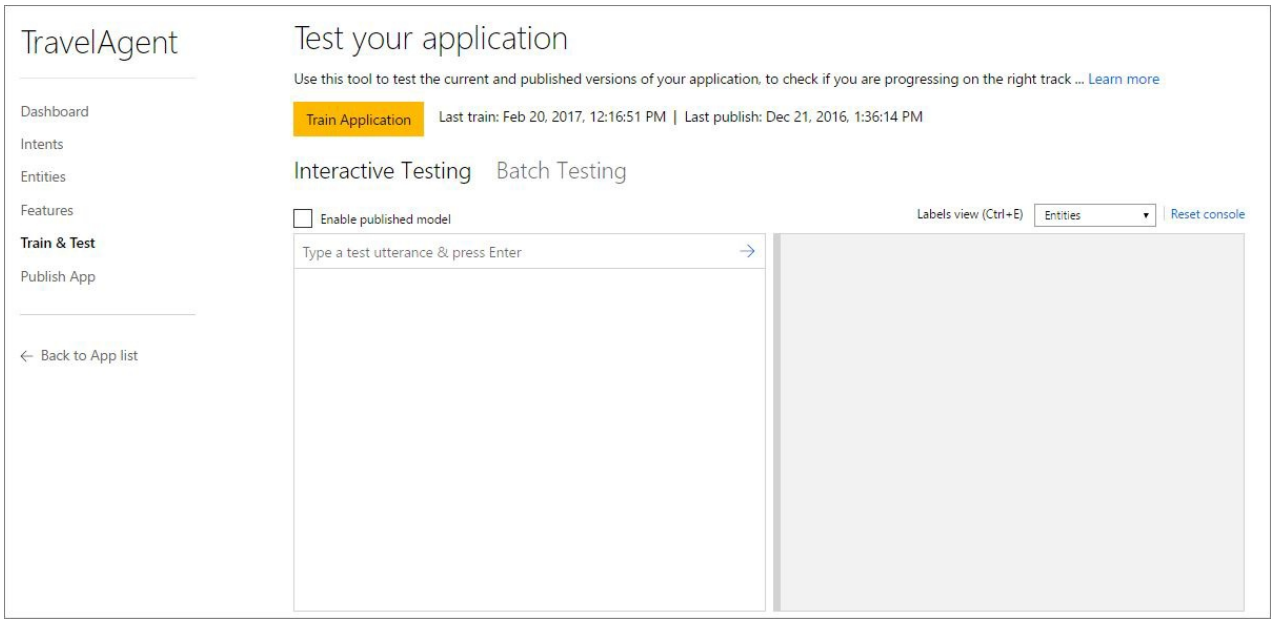
- If your model is trained, your test page will look like this screenshot:



Interactive Testing

Interactive testing enables you to test both the current and published versions of your app and compare their results in one screen. Interactive testing runs by default on the current trained model only. For a published model, interactive testing is disabled and needs your action to enable it, because it is counted in hits and will be deducted from your key balance.

The **Interactive Testing** tab is divided into two sections (as in the screenshot):

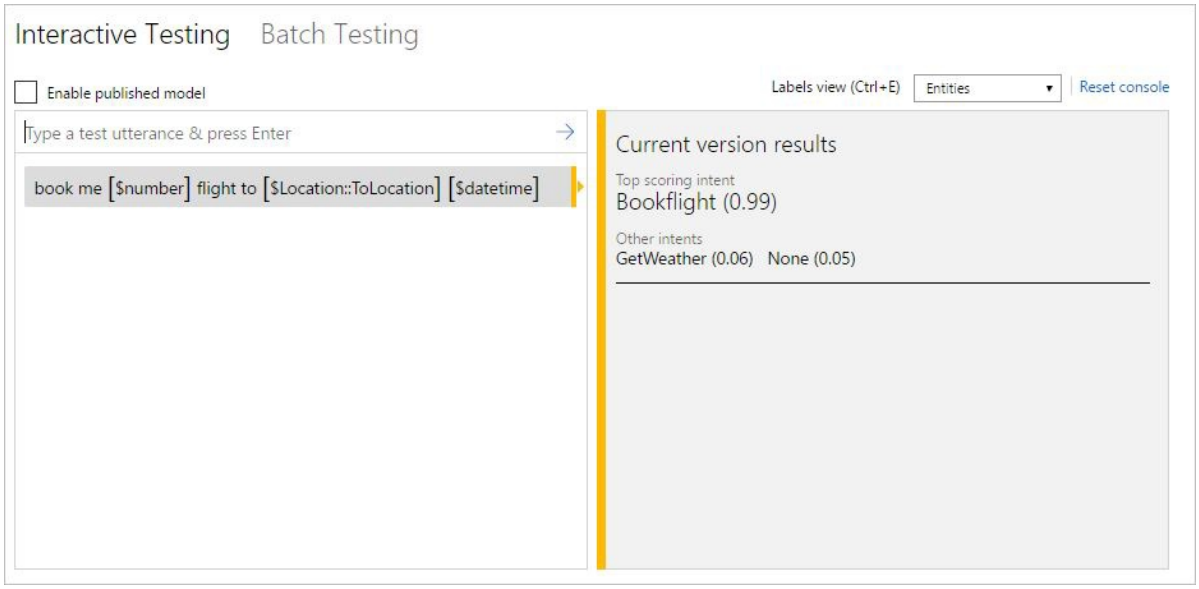


- **The test view**, on the left side of the screen, where you can type the test utterance in the text box and press Enter to submit it to your app.
- **The result view**, on the right side of the screen, where your LUIS app returns the test result, which is the predicted interpretation of the utterance.

In an interactive test, you submit individual test utterances and view the returned result for each utterance separately.

To perform interactive testing on the current model:

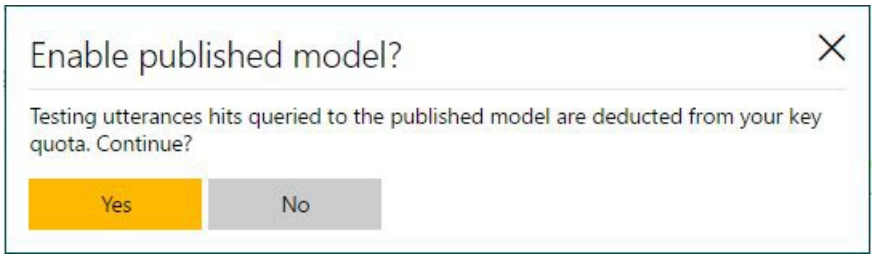
- On the **Test App** page, **Interactive Testing** tab, type "book me a flight to Boston tomorrow" as your test utterance in the text box and press Enter. You'll get the following result:



  The testing result includes the top scoring intent identified in the utterance, with its certainty score, as well as other intents existing in your model with their certainty scores. The identified entities will also be displayed within the utterance and you can control their view by selecting your preferred view from the **Labels view** list at the top of the test console.

To perform interactive testing on current and published models:

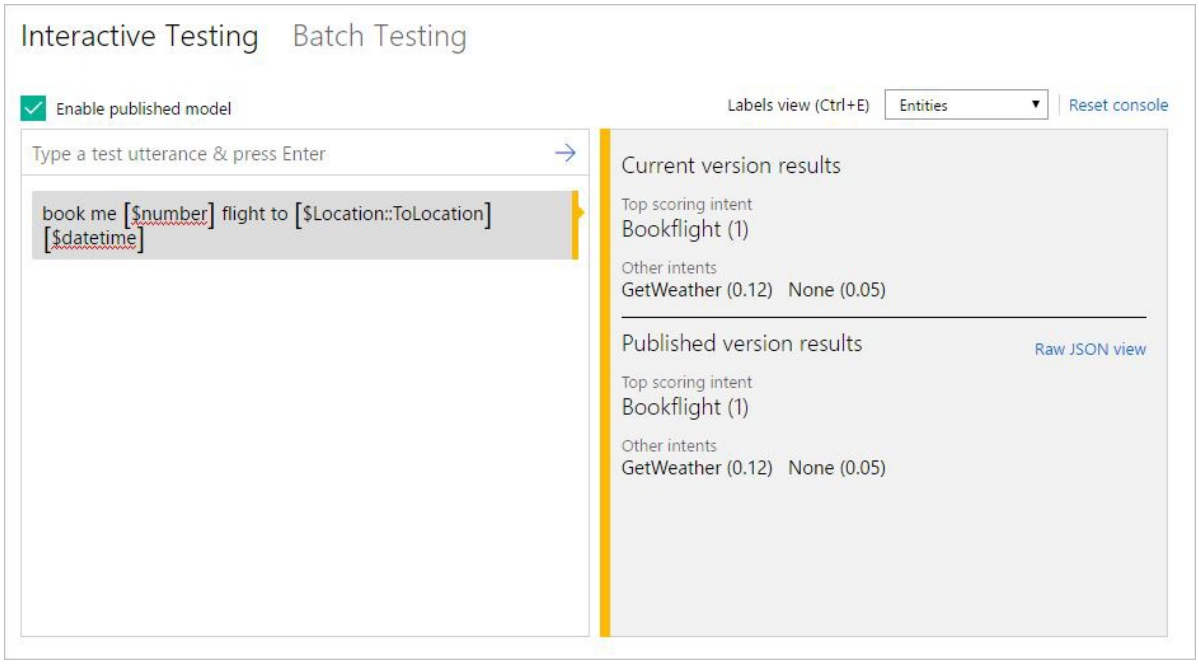
1. On the **Test App** page, **Interactive Testing** tab, click **Enable published model** check box and then click **Yes** in the following confirmation message:



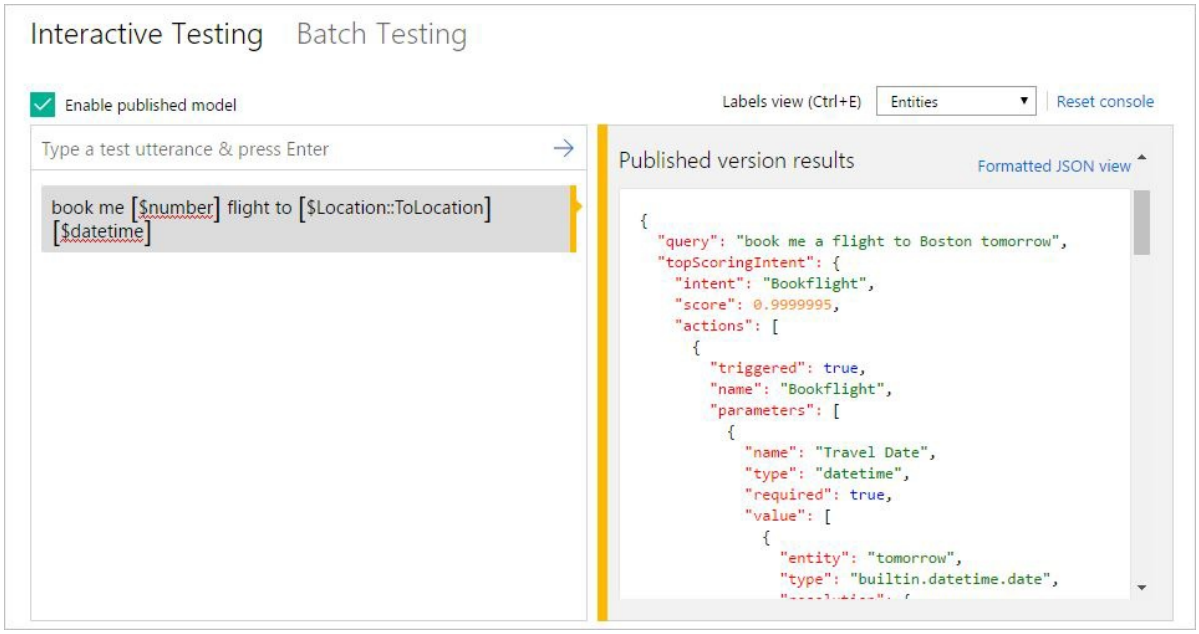
NOTE

If you do not have a published version of your application, the **Enable published model** check box will be disabled.

2. Type "book me a flight to Boston tomorrow" as your test utterance and press Enter. The result view on the right side will be split horizontally into two parts (as in the following screenshot) to display results of the test utterance in both the current and published models.

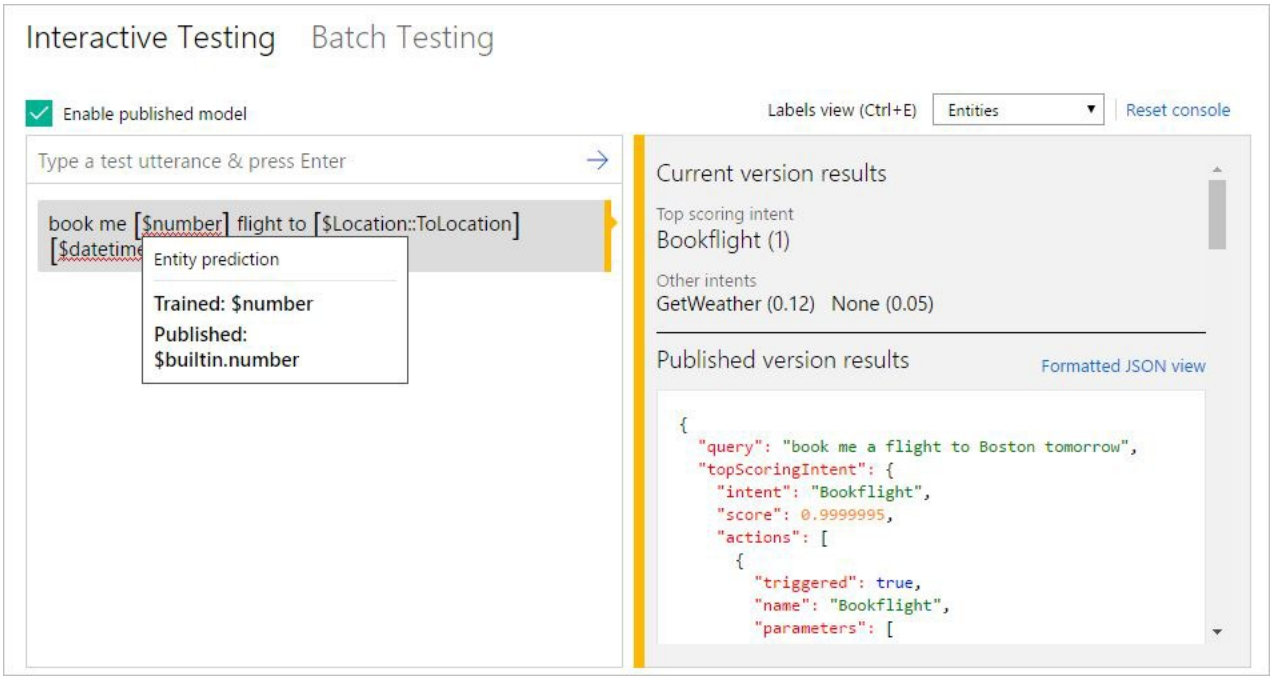


3. To view the test result of your published app in JSON format, click **Raw JSON view**. This will look like the following screenshot.



In case of interactive testing on both trained and published models together, an entity may have a different

prediction in each model. In the test result, this entity will be distinguished by a red underline. If you hover over the underlined entity, you can view the entity prediction in both trained and published models.



NOTE

About the interactive testing console:

- You can type as many test utterances as you want in the test view; only one utterance at a time.
- The result view shows the result of the latest utterance.
- To review the result of a previous utterance, just click it in the test view and its result will be displayed on the right.
- To clear all the entered test utterances and their results from the test console, click **Reset Console** on the top right corner of the console.

Batch Testing

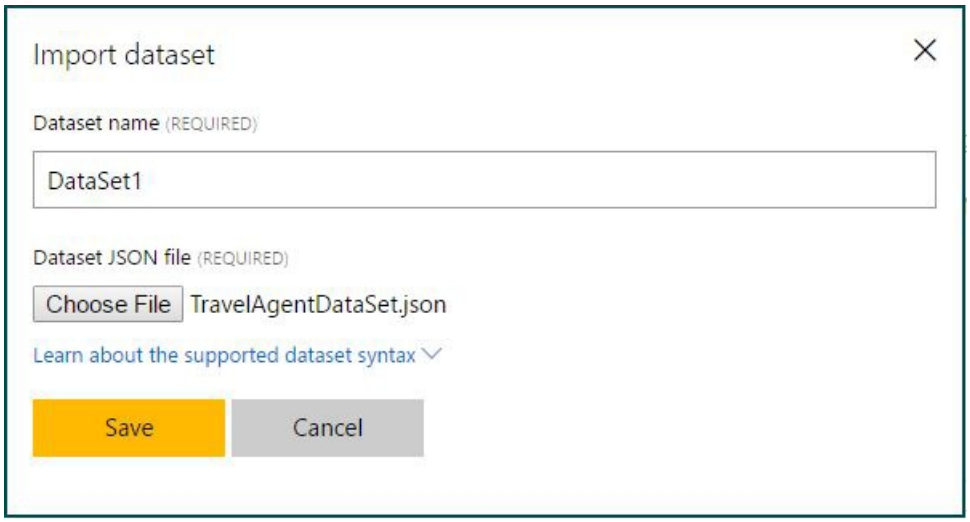
Batch testing enables you to run a comprehensive test on your current trained model to measure its performance in language understanding. In batch testing, you submit a large number of test utterances collectively in a batch file, known as a *dataset*. The dataset file should be written in JSON format and contains a maximum of 1000 utterances. All you need to do is to import this file to your app and run it to perform the test. Your LUIS app will return the result, enabling you to access detailed analysis of all utterances included in the batch.

You can import up to 10 dataset files to a single LUIS app. It is recommended that the utterances included in the dataset should be different from the example utterances you previously added while building your app.

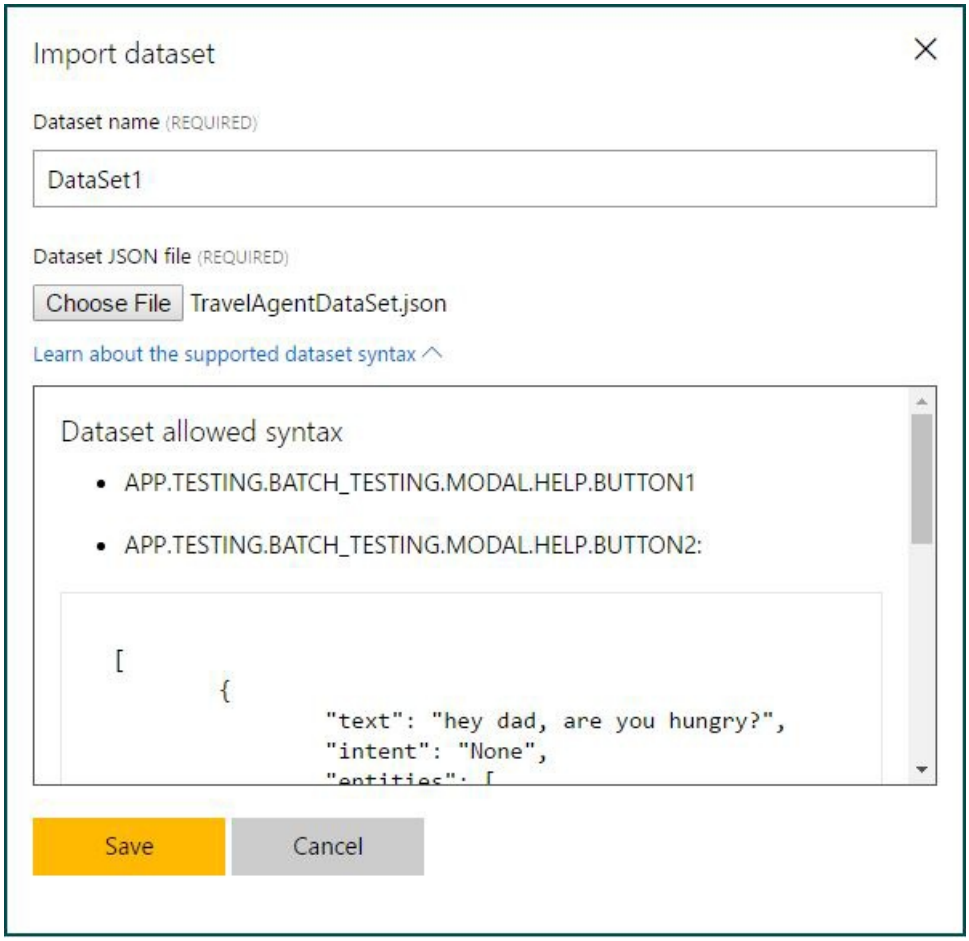
The following procedures will guide you on how to import a dataset file, run a batch test on your current trained app using the imported dataset, and finally to access the test results in a detailed visualized view.

To import a dataset file:

1. On the **Test App** page, click **Batch Testing**, and then click **Import dataset**. The **Import dataset** dialog box appears.



2. In **Dataset name**, type a name for your dataset file (For example "DataSet1").
3. To learn more about the supported syntax for dataset files to be imported, click **learn about the supported dataset syntax link**. The **Import dataset** dialog box will be expanded displaying the allowed syntax. To collapse the dialog and hide syntax, just click the link again.



4. Click **Choose File** to choose the dataset file you want to import, and then click **Save**. The dataset file will be added.

5. To rename, delete or download the imported dataset, you can use these buttons respectively: **Rename Dataset**, **Delete Dataset** and **Download Dataset JSON**.

To run a batch test on your trained app:

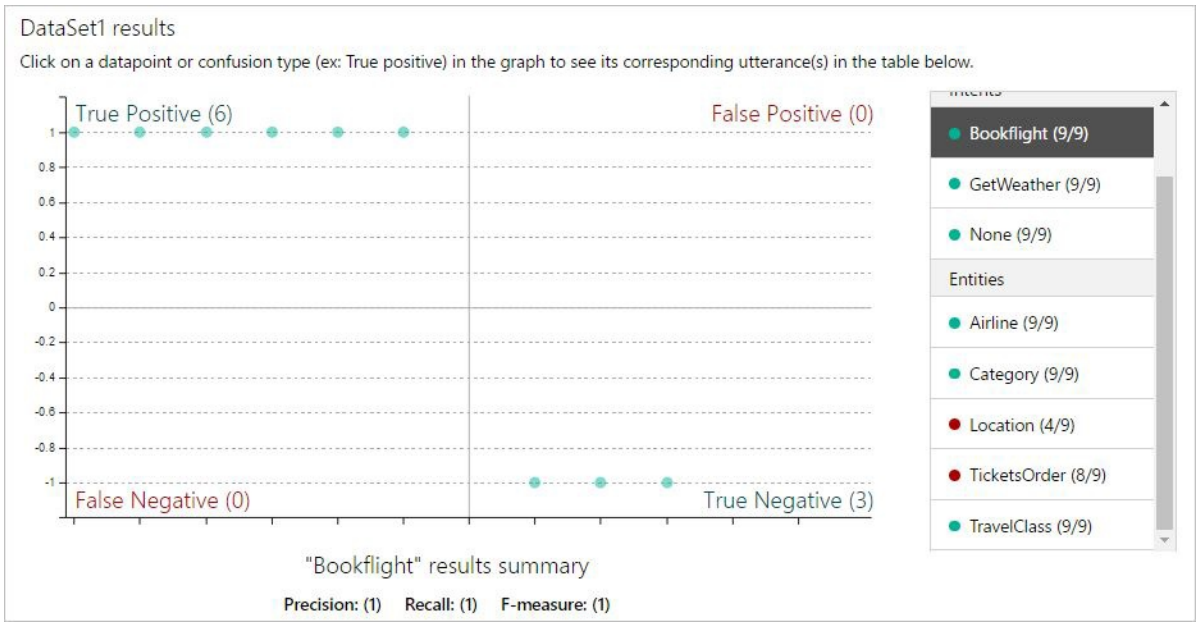
- Click **Test** next to the dataset you've just imported. Soon, the test result of the dataset will be displayed.

  In the above screenshot:

  - **Status** of the dataset shows whether or not the dataset result contains errors. In the above example, an error sign is displayed indicating that there are errors in one or more utterances. If the test result contains no errors, a green sign will be displayed instead.
  - **Utterance Count** is the total number of utterances included in the dataset file.
  - **Last Test Date** is the date of the latest test run for this dataset.
  - **Last Test Success** displays the percentage of correct predictions resulting from the test.

To access test result details in a visualized view:

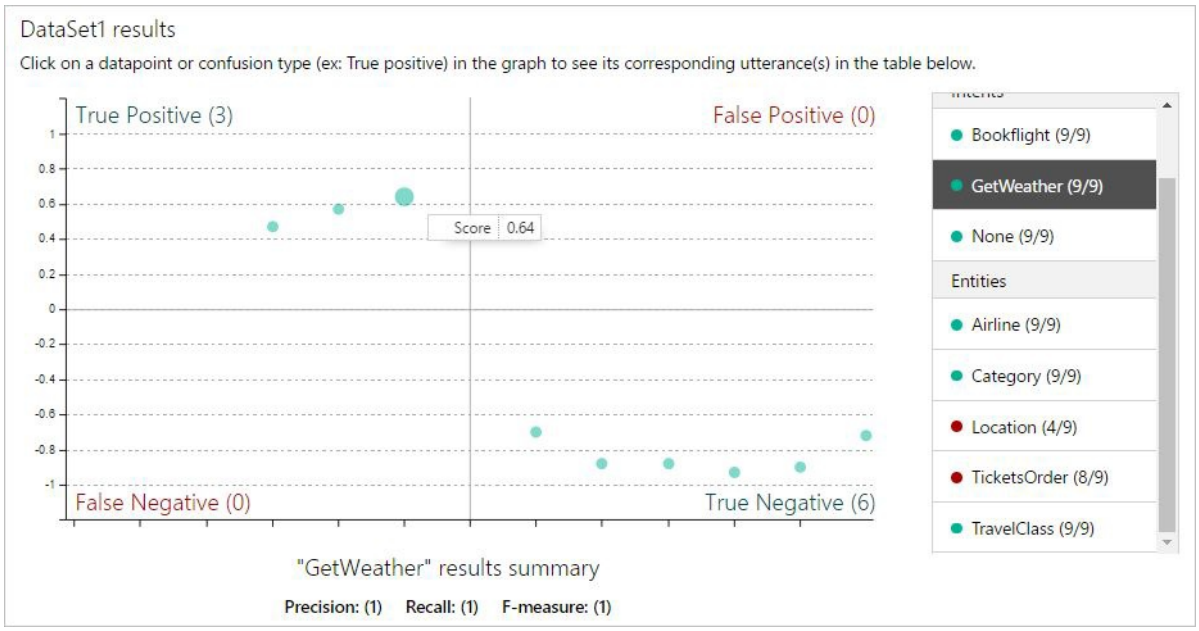
1. Click the **See results** link that appears as a result of running the test (see the above screenshot). A scatter graph (confusion matrix) is displayed, where the data points represent the utterances in the dataset. Green points indicate correct prediction and red ones indicate incorrect prediction.



NOTE

The filtering panel on the right side of the screen displays a list of all intents and entities in the app, with a green point for intents/entities which were predicted correctly in all dataset utterances, and a red one for those with errors. Also, for each intent/entity, you can see the number of correct predictions out of the total utterances. For example, in the above screenshot, the entity "Location (4/9)" has 4 correct predictions out of 9, so it has 5 errors.

2. To filter the view by a specific intent/entity, click on your target intent/entity in the filtering panel. The data points and their distribution will be updated according to your selection. For example, the following screenshot displays results for the "GetWeather" intent.



NOTE

Hovering over a data point shows the certainty score of its prediction.

The graph contains 4 sections representing the possible cases of your application's prediction:

- **True Positive (TP):** The data points in this section represent utterances in which your app correctly predicted the existence of the target intent/entity.

- **True Negative (TN):** The data points in this section represent utterances in which your app correctly predicted the absence of the target intent/entity.
- **False Positive (FP):** The data points in this section represent utterances in which your app incorrectly predicted the existence of the target intent/entity.
- **False Negative (FN):** The data points in this section represent utterances in which your app incorrectly predicted the absence of the target intent/entity.

  This means that data points on the **False Positive** & **False Negative** sections indicate errors, which should be investigated. On the other hand, if all data points are on the **True Positive** and **True Negative** sections, then your application's performance is perfect on this dataset.

3. Click a data point to retrieve its corresponding utterance in the utterances table at the bottom of the page. To display all utterances in a section, click the section title (e.g. True Positive, False Negative, ..etc.)

   For example, the following screenshot shows the results for the "None" intent when one of its data points is clicked, so the utterance "weather forecast for tomorrow" is displayed. This utterance falls under the **True Negative** section as your app correctly predicted that the "None" intent is not present in this utterance.

Thus, a batch test helps you view the performance of each intent and entity in your current trained model on a specific set of utterances. This helps you take appropriate actions, when required, to improve performance, such as adding more example utterances to an intent if your app frequently fails to identify it.

Label Suggested Utterances

6/27/2017 • 3 min to read • Edit Online

The breakthrough feature of LUIS is active learning. Once your application is deployed and traffic starts to flow into the system, LUIS uses active learning to improve itself. In the active learning process, LUIS examines all the utterances that have been sent to it, and calls to your attention the ones that it would like you to label. LUIS identifies the utterances that it is relatively unsure of and asks you to label them. Suggested utterances are the utterances that your LUIS app suggests for labeling. If you label these utterances, this will give your application the biggest boost in performance.

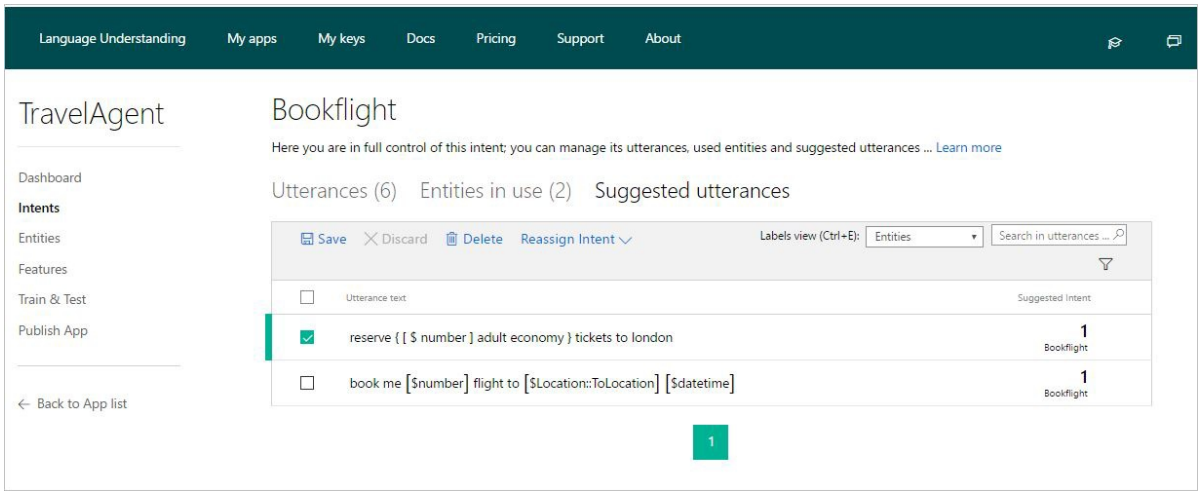
View suggested utterances

Suggested utterances are taken from end-user queries on the application's HTTP endpoint. If your app is not published or has not received hits yet, you will not have any suggested utterances. Also, you will not get suggested utterances for an intent/entity if no endpoint hits are received on this intent/entity.

You can view suggested utterances per intent or entity, under the **Suggested Utterances** tab on the intent page or on the **Entities** page.

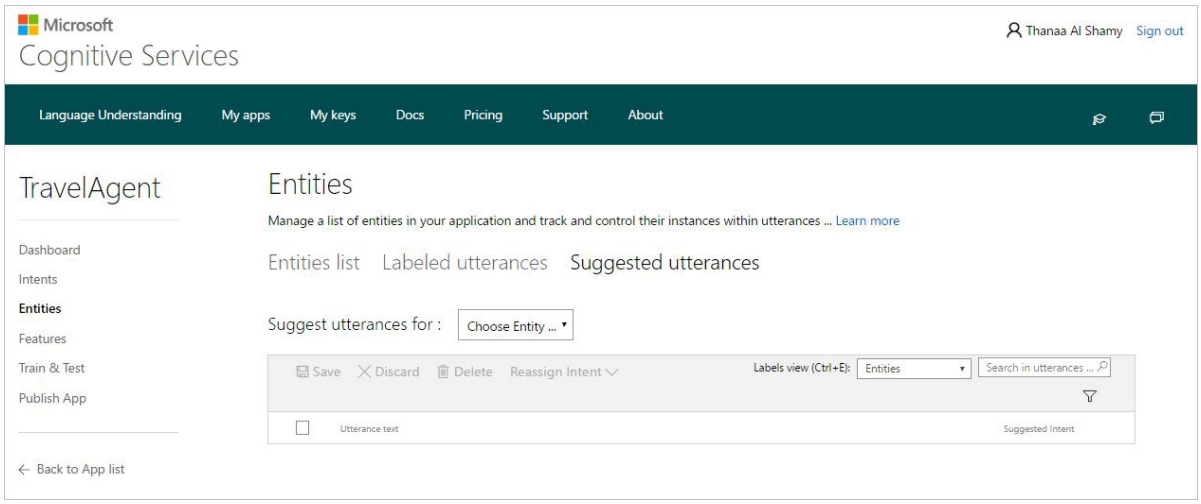
To view suggested utterances per intent:

1. Open your app (e.g. TravelAgent) by clicking its name on **My Apps** page, and then click **Intents** on the app's left panel.
2. On the **Intents** page, click the intent you want to view its suggested utterances (e.g. "Bookflight").
3. On the "Bookflight" intent page, click **Suggested Utterances**. The list of suggested utterances will be displayed. This will look like the following screenshot.

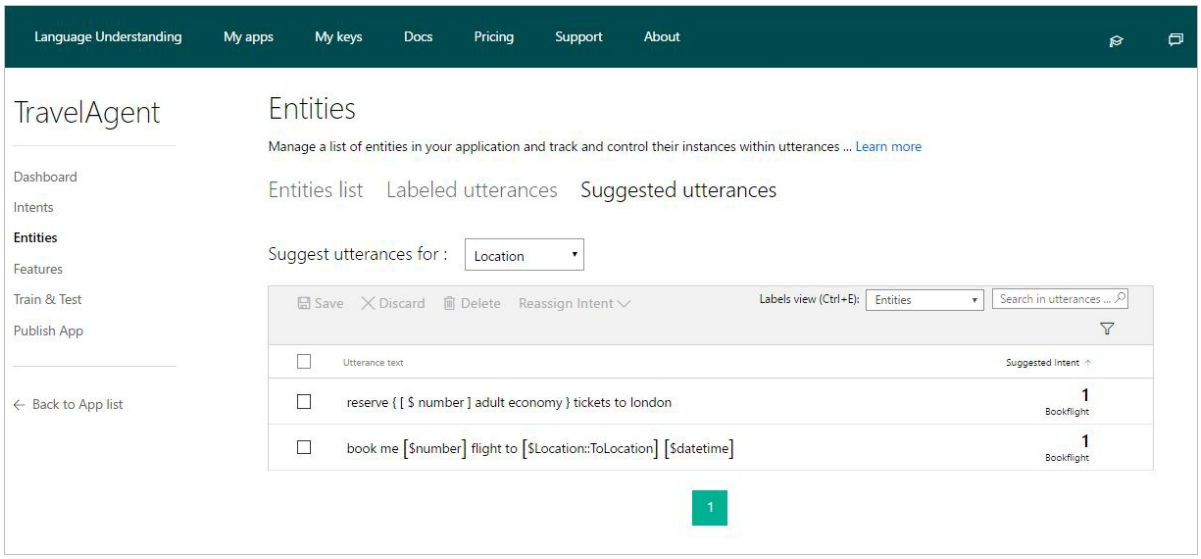


To view suggested utterances per entity:

1. Open your app (e.g. TravelAgent) by clicking its name on **My Apps** page, and then click **Entities** on the app's left panel.
2. On the **Entities** page, click **Suggested Utterances**.



3. Choose the entity you want to view its suggested utterances. The list of suggested utterances will be displayed. This will look like the following screenshot.



Suggested utterances per intent/entity are listed under the **Suggested Utterances** tab. For each suggested utterance, the most likely intent and its score (according to your app's prediction) are displayed. To sort the suggested utterances based on their prediction score, in ascending or descending order, you can click the **Suggested Intent** column header.

To control how you see the words classified as entities in suggested utterances, select a view from the **Labels View** list at the top of the suggested utterances list. You can also press `Ctrl+E`to quickly switch between views. These are the available views:

- **Tokens:** show entity-classified words in text format.
- **Entities:** show entity-classified words in a tagged format (entity labels enclosed in square brackets). This is the default view.
- **Composite entities:** show the words classified as composite entities in their entity labels.

To filter suggested utterances, click the filter button  to display all filters, and then click the filter(s) that you want to apply. For the **Entity** filter, select the entity by which you want to filter the suggested utterances.

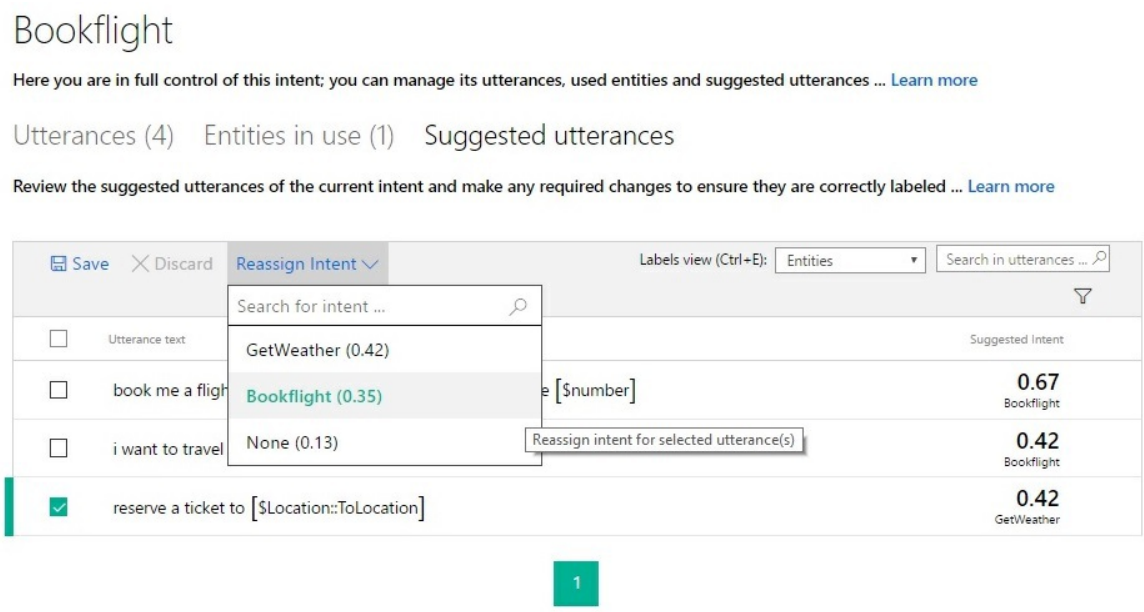
Label suggested utterances

After reviewing the utterances and their predictions, you are advised to take action to label them. You can accept the prediction if it is correct and save the utterance as is, or make any required changes to ensure they are correctly labeled. For example, you can change the intent of a suggested utterance (reassign intent), label

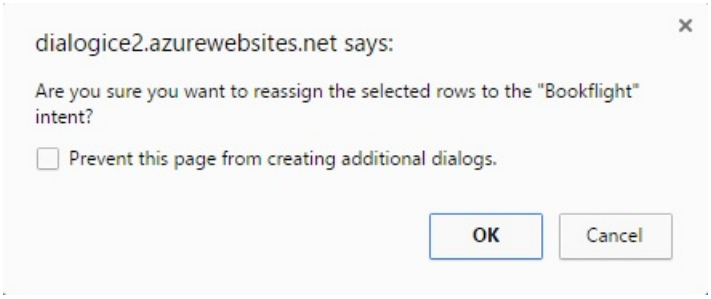
unlabeled entities, correct mis-labeled ones or even remove their labels.

The following are the possible cases you may have, along with the actions you can take in each case in order to label the suggested utterances (using examples from the TravelAgent app):

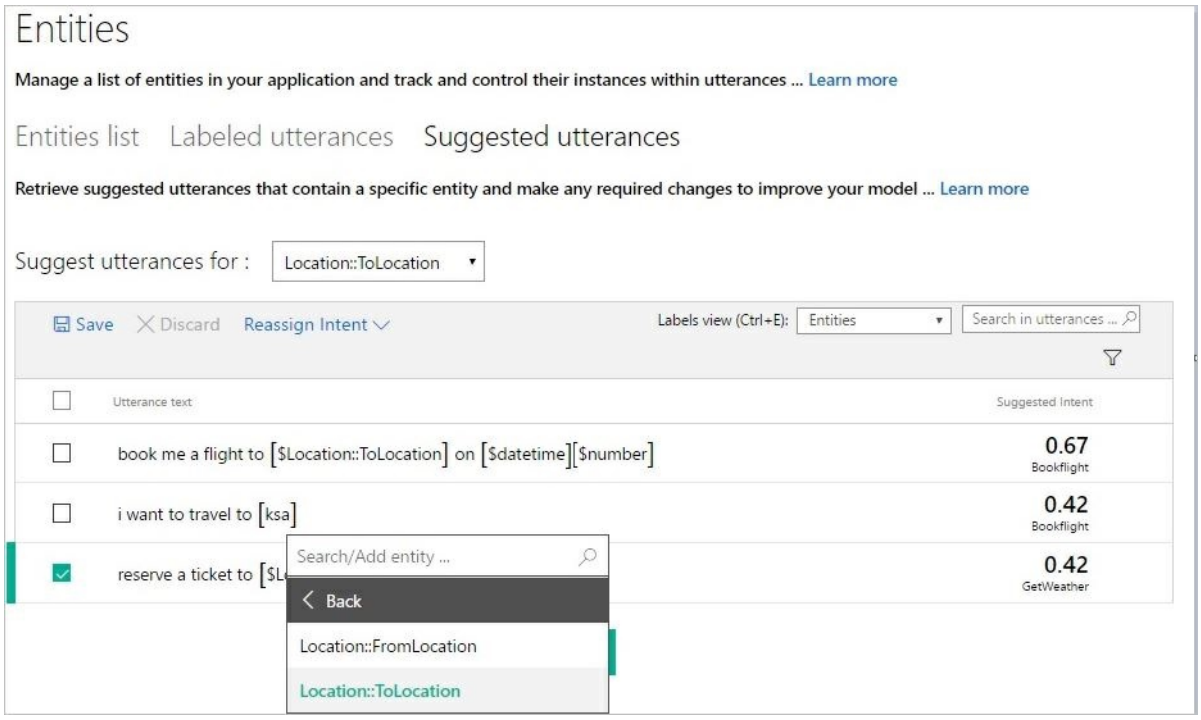
- If the predicted intent is correct and entities are correctly detected and labeled, then just select the utterance and click **Save** without making any changes to accept and save the utterance.
- If the predicted intent is incorrect, you need to change it. For example, in the screenshot below note that the predicted intent in the last suggested utterance "reserve a ticket to London" is "GetWeather", which is incorrect. Click **Reassign Intent** and choose the correct intent "Bookflight" from the list.



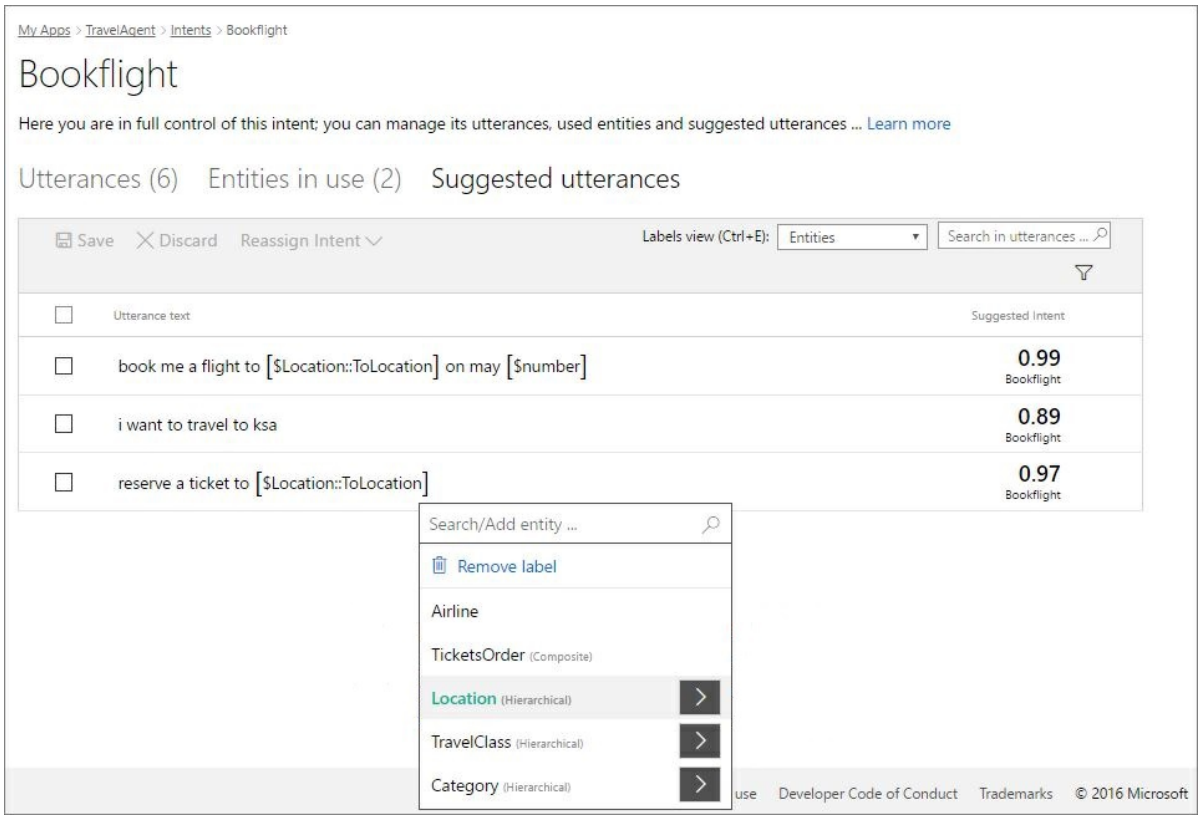
You will get a confirmation message. Click **OK** to confirm this action, and then click **Save** to save your changes.



- If a phrase is unlabeled, click it and select an entity label from the list. For example, "KSA" in the screenshot below is unlabeled. Click it and select "To Location" as its entity label and then click **Save**.



- If an entity is mislabeled, click it and then you can either select the correct label or click **Remove Label** to remove its label. Then, click **Save**.



Next steps

To test how performance improves after you label suggested utterances, you can access the test console by clicking **Train & Test** in the left panel. For instructions on how to test your app using the test console, see Train and test your app.

Publish your app

6/27/2017 • 2 min to read • Edit Online

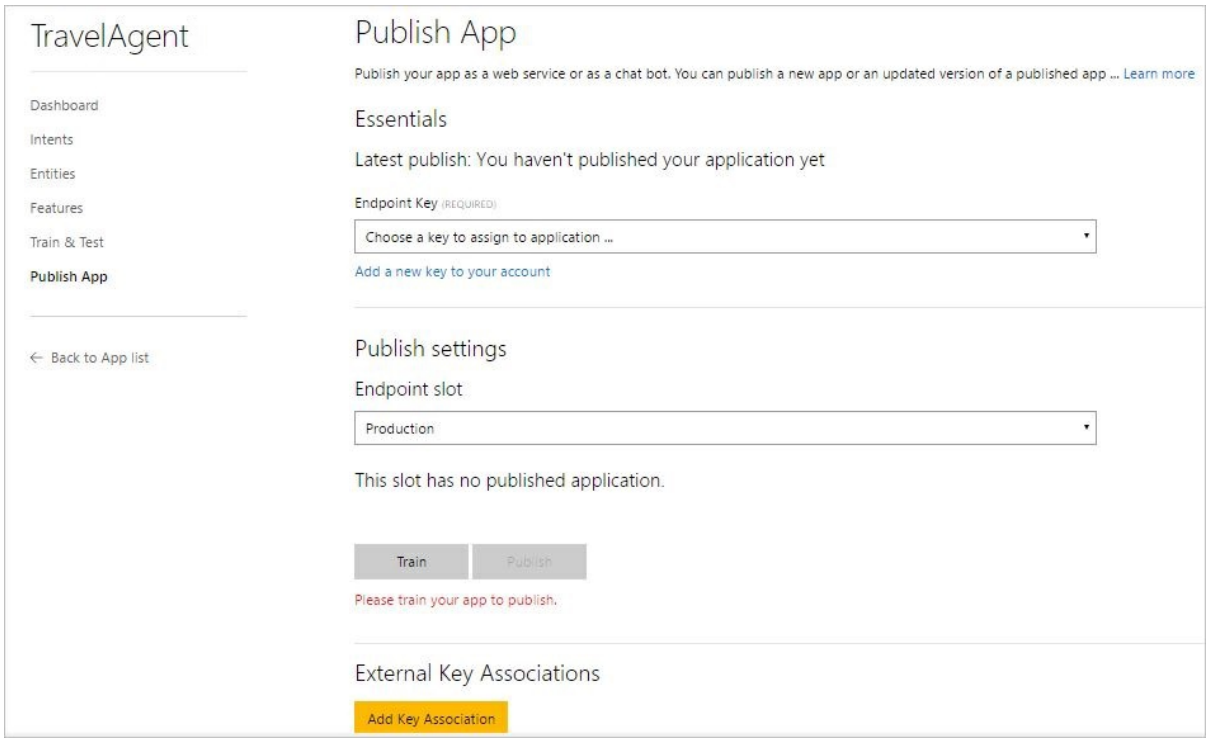
When you finish building and testing your app, you can publish it as a web service on Azure and get an HTTP endpoint that you can use from your client application.

It is optional but recommended to test your app before publishing it. For instructions, see Train and test your app.

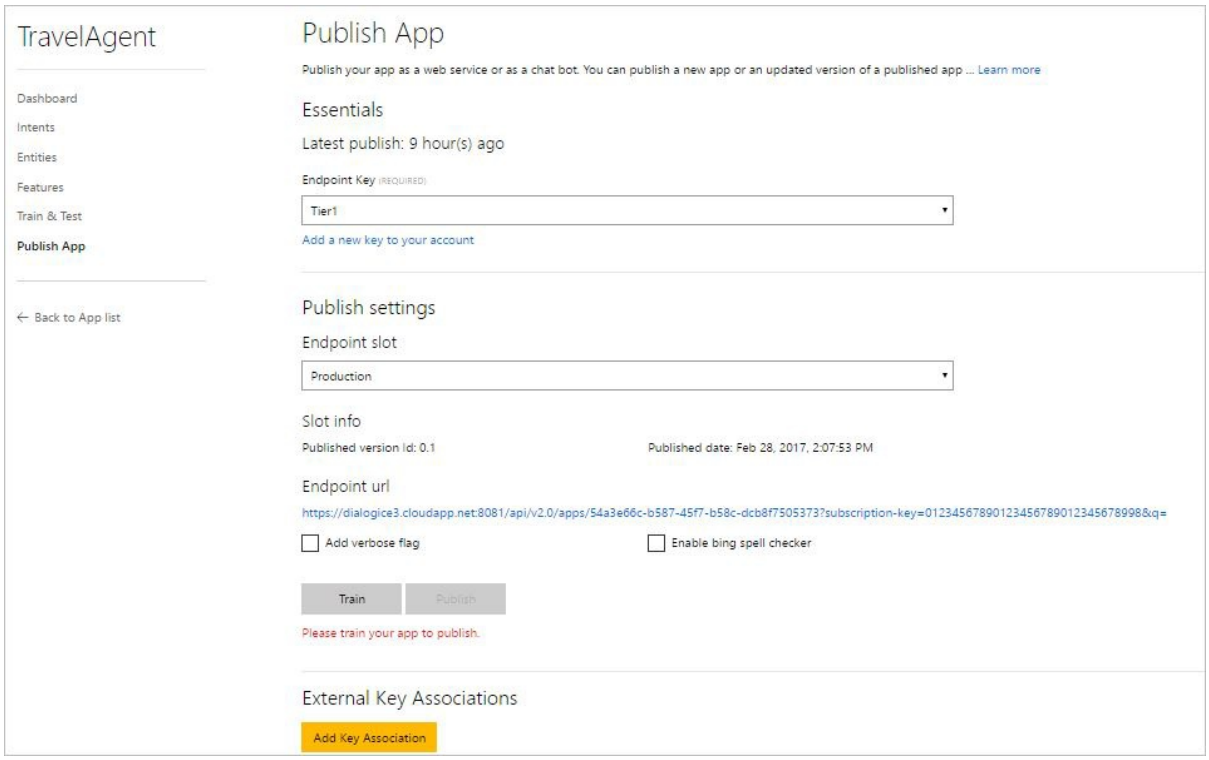
You can either publish your app directly to the **Production Slot** where end users can access and use your model, or you can publish to a **Staging Slot** where you can iteratively test your app to validate changes before publishing to the production slot.

Publish your app to an HTTP endpoint:

1. Open your app (for example, TravelAgent) by clicking its name on **My Apps** page, and then click **Publish App** in the left panel to access the **Publish App** page. The screenshot below shows the Publish page as it appears if you haven't published your app yet.

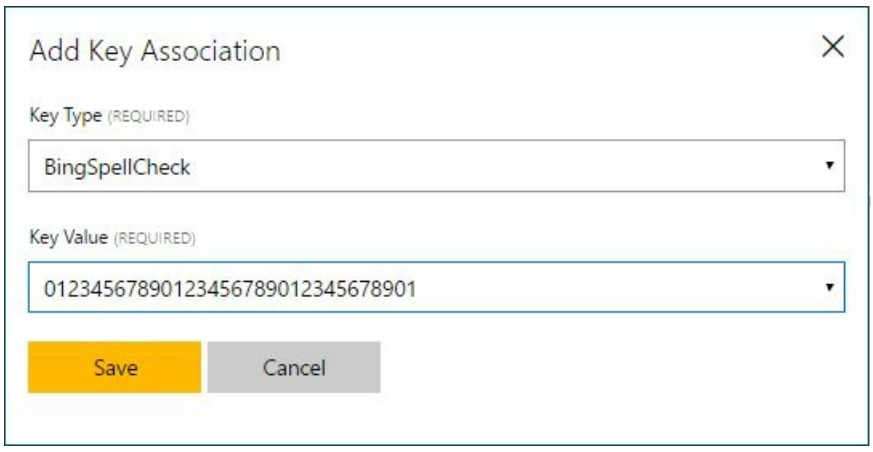


If you have previously published this app, this page looks like the following screenshot:

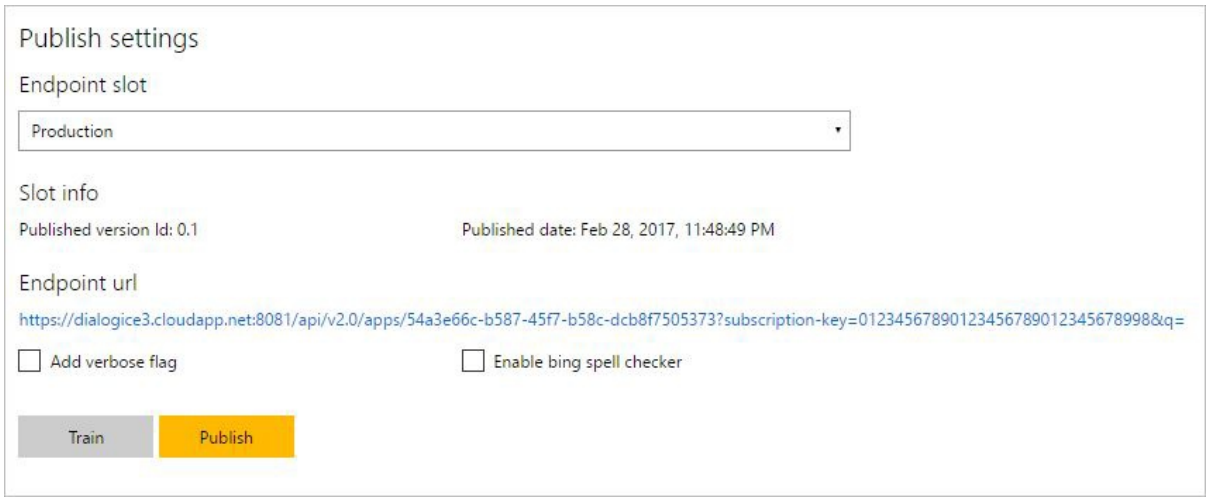


2. From the **Endpoint Key** list, select an existing endpoint key to assign to the app, or click **Add a new key to your account** to add a new one. For more information on how to create and add endpoint keys to your account, see Manage your keys. For pricing information, see Cognitive Services Pricing.
3. Choose whether to publish to the **Production** or to **Staging** slot by selecting the corresponding value from the **Endpoint Slot** list.
4. If you will use an external service with your LUIS app (for example, Bing Spell Check):

    - Click **Add Key Association** to assign the external service key to the app by selecting the key type and key value in the following dialog box.
    - Click **Enable Bing Spell Checker** check box.



5. If you want the JSON response of your published app to include all intents defined in your app and their prediction scores, click **Add Verbose Flag** checkbox. Otherwise, it will include only the top scoring intent.
6. Click **Train** if the app wasn't trained already.
7. Click **Publish**. The endpoint URL of your published app is displayed.



> **NOTE**
>
> If the **Publish** button is disabled, then either your app does not have an assigned an endpoint key, or you have not trained your app yet.

If you want to test your published endpoint in a browser using the generated URL, you can click the URL to open it in your browser, then set the URL parameter "&q" to your test query (for example: "&q=Book me a flight to Boston on May 4"), and then press Enter. You will get the JSON response of your HTTP endpoint.


```
{
  "query": "Book me a flight to Boston on May 4",
  "topScoringIntent": {
    "intent": "Bookflight",
    "score": 0.610224
  },
  "entities": [
    {
      "entity": "may 4",
      "type": "builtin.datetime.date",
      "startIndex": 30,
      "endIndex": 34,
      "resolution": {
        "date": "XXXX-05-04"
      }
    },
    {
      "entity": "4",
      "type": "builtin.number",
      "startIndex": 34,
      "endIndex": 34,
      "score": 0.9979208
    }
  ]
}
```


Next steps

To test how your published app works, you can access the test console by clicking **Train & Test** in the left panel. For instructions on how to test your app using the test console, see Train and test your app.

Prebuilt entities

7/18/2017 • 18 min to read • Edit Online

LUIS includes a set of prebuilt entities. When a prebuilt entity is included in your application, its predictions are included in your published application and can be used in the LUIS web UI to label utterances. The behavior of prebuilt entities **cannot** be modified. Unless otherwise noted, prebuilt entities are available in all LUIS application locales (cultures). The following table shows the prebuilt entities that are supported for each culture.

> **NOTE**
>
> **builtin.datetime** is deprecated. It is replaced by **builtin.datetimeV2**, which provides recognition of date and time ranges, as well as improved recognition of ambiguous dates and times.

PRE-BUILT ENTITY	EN-US	FR-FR	IT-IT	ES-ES	ZH-CN	DE-DE	PT-BR	JA-JP	KO-KR
DatetimeV2	□□	-	-	-	□□	-	-	-	-
Datetime	□□	□□	□□	□□	□□	□□	□□	□□	-
Number	□□	□□	□□	□□	□□	□□	□□	□□	-
Ordinal	□□	□□	□□	□□	□□	□□	□□	□□	-
Percentage	□□	□□	□□	□□	□□	□□	□□	□□	-
Temperature	□□	□□	□□	□□	□□	□□	□□	□□	-
Dimension	□□	□□	□□	□□	□□	□□	□□	□□	-
Money	□□	□□	□□	□□	□□	□□	□□	□□	-
Age	□□	□□	□□	□□	□□	□□	□□	□□	-
Geography	□□	-	-	-	-	-	-	-	-
Encyclopedia	□□	-	-	-	-	-	-	-	-
URL	□□	-	-	-	-	-	-	-	-
Email	□□	-	-	-	-	-	-	-	-

PRE-BUILT ENTITY	EN-US	FR-FR	IT-IT	ES-ES	ZH-CN	DE-DE	PT-BR	JA-JP	KO-KR
Phone number	□□	-	-	-	-	-	-	-	-

Examples of prebuilt entities

The following table lists prebuilt entities with example utterances and their return values.

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
builtin.number	ten	`{ "type": "builtin.number", "entity": "ten" }`
builtin.number	3.1415	`{ "type": "builtin.number", "entity": "3 . 1415" }`
builtin.ordinal	first	`{ "type": "builtin.ordinal", "entity": "first" }`
builtin.ordinal	10th	`{ "type": "builtin.ordinal", "entity": "10th" }`
builtin.temperature	10 degrees celcius	`{ "type": "builtin.temperature", "entity": "10 degrees celcius" }`
builtin.temperature	78 F	`{ "type": "builtin.temperature", "entity": "78 f" }`
builtin.dimension	2 miles	`{ "type": "builtin.dimension", "entity": "2 miles" }`
builtin.dimension	650 square kilometers	`{ "type": "builtin.dimension", "entity": "650 square kilometers" }`
builtin.money	1000.00 US dollars	`{ "type": "builtin.money", "entity": "1000.00 us dollars" }`
builtin.money	$ 67.5 B	`{ "type": "builtin.money", "entity": "$ 67.5" }`
builtin.age	100 year old	`{ "type": "builtin.age", "entity": "100 year old" }`
builtin.age	19 years old	`{ "type": "builtin.age", "entity": "19 years old" }`
builtin.percentage	The stock price increase by 7 $ this year	`{ "type": "builtin.percentage", "entity": "7 %" }`
builtin.datetimeV2	See builtin.datetimeV2	See builtin.datetimeV2
builtin.datetime	See builtin.datetime	See builtin.datetime

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
builtin.geography	See separate table	See separate table below
builtin.encyclopedia	See separate table	See separate table below

The last 3 built-in entity types listed in the table above encompass multiple subtypes. These are covered later in this article.

builtin.number resolution

There are many ways in which numeric values are used to quantify, express, and describe pieces of information, with more possibilities than the examples listed. LUIS interprets the variations in user utterances and returns consistent numeric values.

UTTERANCE	ENTITY	RESOLUTION
one thousand times	"one thousand"	"1000"
1,000 people	"1,000"	"1000"
1/2 cup	"1 / 2"	"0.5"
one half the amount	"one half"	"0.5"
one hundred fifty orders	"one hundred fifty"	"150"
one hundred and fifty books	"one hundred and fifty"	"150"
a grade of one point five	"one point five"	"1.5"
buy two dozen eggs	"two dozen"	"24"

LUIS includes the recognized value of a **builtin.number** entity in the `resolution` field of the JSON response it returns.

The following example shows a JSON response from LUIS, that includes the resolution of the value 24, for the utterance "two dozen".

```
{
 "query": "order two dozen eggs",
 "topScoringIntent": {
  "intent": "OrderFood",
  "score": 0.105443209
 },
 "intents": [
  {
   "intent": "None",
   "score": 0.105443209
  },
  {
   "intent": "OrderFood",
   "score": 0.9468431361
  },
  {
   "intent": "Help",
   "score": 0.000399122015
  },
 ],
 "entities": [
  {
   "entity": "two dozen",
   "type": "builtin.number",
   "startIndex": 6,
   "endIndex": 14,
   "resolution": {
    "value": "24"
   }
  }
 ]
}
```


Ordinal, percentage and currency resolution

The **builtin.ordinal**, **builtin.percentage**, and **builtin.currency** entities also provide resolution to a value.

Percentage resolution

The following example shows the resolution of the **builtin.percentage** entity.

```
{
 "query": "set a trigger when my stock goes up 2%",
 "topScoringIntent": {
  "intent": "SetTrigger",
  "score": 0.971157849
 },
 "intents": [
  {
   "intent": "SetTrigger",
   "score": 0.971157849
  },
  {
   "intent": "None",
   "score": 0.07398871
  },
  {
   "intent": "Help",
   "score": 2.57078386E-06
  }
 ],
 "entities": [
  {
   "entity": "2",
   "type": "builtin.number",
   "startIndex": 36,
   "endIndex": 36,
   "resolution": {
    "value": "2"
   }
  },
  {
   "entity": "2%",
   "type": "builtin.percentage",
   "startIndex": 36,
   "endIndex": 37,
   "resolution": {
    "value": "2%"
   }
  }
 ]
}
```


Ordinal resolution

The following example shows the resolution of the **builtin.ordinal** entity.

```
{
 "query": "Order the second option",
 "topScoringIntent": {
  "intent": "OrderFood",
  "score": 0.9993253
 },
 "intents": [
  {
   "intent": "OrderFood",
   "score": 0.9993253
  },
  {
   "intent": "None",
   "score": 0.05046708
  }
 ],
 "entities": [
  {
   "entity": "second",
   "type": "builtin.ordinal",
   "startIndex": 10,
   "endIndex": 15,
   "resolution": {
    "value": "2"
   }
  }
 ]
}
```


Currency resolution

The following example shows the resolution of the **builtin.currency** entity.

```
{
 "query": "search for items under $10.99",
 "topScoringIntent": {
  "intent": "SearchForItems",
  "score": 0.926173568
 },
 "intents": [
  {
   "intent": "SearchForItems",
   "score": 0.926173568
  },
  {
   "intent": "None",
   "score": 0.07376878
  }
 ],
 "entities": [
  {
   "entity": "10.99",
   "type": "builtin.number",
   "startIndex": 24,
   "endIndex": 28,
   "resolution": {
    "value": "10.99"
   }
  },
  {
   "entity": "$10.99",
   "type": "builtin.currency",
   "startIndex": 23,
   "endIndex": 28,
   "resolution": {
    "unit": "Dollar",
    "value": "10.99"
   }
  }
 ]
}
```


builtin.datetimeV2

The **builtin.datetimeV2** prebuilt entity automatically recognizes dates, times, and ranges of dates and times. This entity also resolves dates, times and date ranges to values in a standardized format for client programs to consume. If an utterance contains a date or time that isn't fully specified, both past and future values are included in the resolution.

> **NOTE**
>
> **builtin.datetimeV2** is available only in the en-us and zh-cn locales.

EXAMPLE	PROPERTY DESCRIPTIONS
The following is an example of a JSON response containing a builtin.datetimeV2 entity, of type `datetime`. For examples of other types of datetimeV2 entities, see Subtypes of datetimeV2.	entity: **string**. Text extracted from the utterance, that represents a date, time, date range or time range.

```
"entities": [
 {
  "entity": "8am on may 2nd 2017",
  "type": "builtin.datetimeV2.datetime",
  "startIndex": 15,
  "endIndex": 30,
  "resolution": {
   "values": [
    {
     "timex": "2017-05-02T08",
     "type": "datetime",
     "value": "2017-05-02 08:00:00"
    }
   ]
  }
 }
]
```

type	**string**. One of the following subtypes of datetimeV2: • builtin.datetimeV2.datetime • builtin.datetimeV2.date • builtin.datetimeV2.time • builtin.datetimeV2.daterange • builtin.datetimeV2.datetimerange • builtin.datetimeV2.duration • builtin.datetimeV2.set
startIndex	**int**. The index in the utterance at which the entity begins.
endIndex	**int**. The index in the utterance at which the entity ends.
resolution	Contains a `values` array that has one, two, or four values. • The array has one element if the date or time in the utterance is fully specified and unambiguous. • The array has two elements if the date or date range is ambiguous as to year, or a time or time range is ambiguous as to AM or PM. In the case of an ambiguous date, `values` contains the most recent past and most immediate future instances of the date. See Ambiguous dates for more examples. In the case of an ambiguous time, `values` contains both the AM and PM times. • The array has four elements if the utterance contains both a date or date range that is ambiguous as to year, and a time or time range that is ambiguous as to AM or PM. For example, 3:00 April 3rd. Each element of `values` may

contain the following fields:

timex	time, date, or date range expressed in TIMEX format that follows the ISO 8601 standard as well as using the TIMEX3 attributes for annotation using the TimeML language. This annotation is described in the TIMEX guidelines.
type	The subtype, which can be one of the following: datetime, date, time, daterange, datetimerange, duration, set.
value	**Optional.** A datetime object in the Format yyyy:MM:dd (date), HH:mm:ss (time) yyyy:MM:dd HH:mm:ss (datetime). If `type` is `duration`, the value is the number of seconds (duration) Only used if `type` is `datetime` or `date`, `time`, or `duration`.

		start	A value representing the start of a time or date range, in the same format as `value`. Only used if `type` is `daterange` or `datetimerange`.
	end		A value representing the end of a time or date range, in the same format as `value`. Only used if `type` is `daterange` or `datetimerange`.

Ambiguous dates

If it's unclear from an utterance whether a date refers to that date in the past or the future, LUIS provides both the most immediate past and future instances of that date. One case of this is an utterance that includes the month and date, but not the year. If today's date precedes the date in the utterance in the current year, the most immediate past instance of that date is in the previous year. Otherwise the most immediate past date is in the current year.

For example, given the utterance "May 2nd":

- If today's date is May 3rd 2017, LUIS provides both "2017-05-02" and "2018-05-02" as values.
- If today's date is May 1st 2017, LUIS provides both "2016-05-02" and "2017-05-02" as values.

The following example shows the resolution of the entity "may 2nd" provided that today's date is a date between May 2nd 2017 and May 1st 2018. Fields containing `X` in the `timex` field represent parts of the date that are not explicitly specified in the utterance.


```
"entities": [
  {
    "entity": "may 2nd",
    "type": "builtin.datetimeV2.date",
    "startIndex": 0,
    "endIndex": 6,
    "resolution": {
      "values": [
        {
          "timex": "XXXX-05-02",
          "type": "date",
          "value": "2017-05-02"
        },
        {
          "timex": "XXXX-05-02",
          "type": "date",
          "value": "2018-05-02"
        }
      ]
    }
  }
]
```


Date range resolution examples

The datetimeV2 entity can recognize date and time ranges. The `start` and `end` fields specify the beginning and end of the range. For the utterance "May 2nd to May 5th", LUIS provides **daterange** values for both the current year and the following year. In the `timex` field, the `XXXX` values represent the year that is not explicitly specified in the utterance, and `P3D` indicates that the time period is 3 days long.


```
"entities": [
  {
   "entity": "may 2nd to may 5th",
   "type": "builtin.datetimeV2.daterange",
   "startIndex": 0,
   "endIndex": 17,
   "resolution": {
    "values": [
     {
      "timex": "(XXXX-05-02,XXXX-05-05,P3D)",
      "type": "daterange",
      "start": "2017-05-02",
      "end": "2017-05-05"
     },
     {
      "timex": "(XXXX-05-02,XXXX-05-05,P3D)",
      "type": "daterange",
      "start": "2018-05-02",
      "end": "2018-05-05"
     }
    ]
   }
  }
]
```


The following example shows how LUIS uses **datetimeV2** to resolve the utterance "Tuesday to Thursday". In this example the current date is June 19th. Note that LUIS includes **daterange** values for both of the date ranges that precede and follow the current date.


```
"entities": [
  {
   "entity": "tuesday to thursday",
   "type": "builtin.datetimeV2.daterange",
   "startIndex": 0,
   "endIndex": 19,
   "resolution": {
    "values": [
     {
      "timex": "(XXXX-WXX-2,XXXX-WXX-4,P2D)",
      "type": "daterange",
      "start": "2017-06-13",
      "end": "2017-06-15"
     },
     {
      "timex": "(XXXX-WXX-2,XXXX-WXX-4,P2D)",
      "type": "daterange",
      "start": "2017-06-20",
      "end": "2017-06-22"
     }
    ]
   }
  }
]
```


Subtypes of datetimeV2

The **builtin.datetimeV2** prebuilt entity has the following subtypes, and examples of each are provided in the table that follows:

- builtin.datetimeV2.date
- builtin.datetimeV2.time
- builtin.datetimeV2.daterange
- builtin.datetimeV2.datetimerange
- builtin.datetimeV2.duration
- builtin.datetimeV2.set

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
builtin.datetimeV2.date	tomorrow	`{ "type": "builtin.datetimeV2.date", "entity": "tomorrow", "resolution": {"values": [{"timex": "2017-06-21","type": "date", "value": "2017-06-21"}]} }`
builtin.datetimeV2.date	january 10 2009	`{ "type": "builtin.datetimeV2.date", "entity": "january 10 2009", "resolution": { "values": [ {"timex": "2009-01-10", "type": "date", "value": "2009-01-10" }] } }`
builtin.datetimeV2.date	monday	`{ "entity": "monday", "type": "builtin.datetimeV2.date", "resolution": { "values": [{ "timex": "XXXX-WXX-1", "type": "date", "value": "2017-06-19" }, {"timex": "XXXX-WXX-1","type": "date", "value": "2017-06-26"}]} }`
builtin.datetimeV2.daterange	next week	`{ "entity": "next week", "type": "builtin.datetime.dateV2.daterange", "resolution": { "values": [{ "timex": "2017-W27", "type": "daterange", "start": "2017-06-26", "end": "2017-07-03"}] } }`
builtin.datetimeV2.date	next monday	`{ "entity": "next monday", "type": "builtin.datetimeV2.date", "resolution": { "values": [{ "timex": "2017-06-26", "type": "date", "value": "2017-06-26" }] } }`
builtin.datetimeV2.time	3 : 00	`{ "type": "builtin.datetimeV2.time", "entity": "3 : 00", "resolution": { "values": [{ "timex": "T03:00", "type": "time", "value": "03:00:00" }, { "timex": "T15:00", "type": "time", "value": "15:00:00" }]} }`
builtin.datetimeV2.time	4 pm	`{ "type": "builtin.datetimeV2.time", "entity": "4 pm", "resolution": { "values": [{"timex": "T16", "type": "time", "value": "16:00:00"}] } }`

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
builtin.datetimeV2.datetimerange	tomorrow morning	`{ "entity": "tomorrow morning", "type": "builtin.datetimev2.datetimerange", "resolution": { "values": [{"timex": "2017-06-21TMO","type": "datetimerange", "start": "2017-06-21 08:00:00", "end": "2017-06-21 12:00:00"}]} }`
builtin.datetimeV2.datetimerange	tonight	`{ "entity": "tonight", "type": "builtin.datetimeV2.datetimerange", "resolution": { "values": [{"timex": "2017-06-20TNI","type": "datetimerange", "start": "2017-06-20 20:00:00", "end": "2017-06-20 23:59:59"}]} }`
builtin.datetimeV2.duration	for 3 hours	`{ "type": "builtin.datetimeV2.duration", "entity": "3 hours", "resolution": { "values": [{ "timex": "PT3H", "type": "duration", "value": "10800"}] } }`
builtin.datetimeV2.duration	30 minutes long	`{ "type": "builtin.datetimeV2.duration", "entity": "30 minutes", "resolution": { "values": [{ "timex": "PT30M", "type": "duration", "value": "1800"}] } }`
builtin.datetimeV2.duration	all day	`{ "type": "builtin.datetimeV2.duration", "entity": "all day", "resolution": { "values": [{ "timex": "P1D", "type": "duration", "value": "86400"}] } }`
builtin.datetimeV2.set	daily	`{ "type": "builtin.datetimeV2.set", "entity": "daily", "resolution": { "values": [{ "timex": "P1D", "type": "set", "value": "not resolved"}]} }`
builtin.datetimeV2.set	every tuesday	`{ "entity": "every tuesday", "type": "builtin.datetimeV2.set", "resolution": { "values": [{ "timex": "XXXX-WXX-2", "type": "set", "value": "not resolved"}]} }`
builtin.datetimeV2.set	every week	`{ "entity": "every week", "type": "builtin.datetimeV2.set", "resolution": {"time": "XXXX-WXX"} }`

builtin.datetime

The **builtin.datetime** prebuilt entity is deprecated and replaced by builtin.datetimeV2.

To replace **builtin.datetime** with **builtin.datetimeV2** in your LUIS app, do the following:

1. Open the **Entities** pane of the LUIS web interface.
2. Delete the **datetime** prebuilt entity.

3. Click **Add prebuilt entity**
4. Select **datetimeV2** and click **Save**.

The following table provides a comparison of datetime and datetimeV2. In the examples, the current date is 2017-06-20.

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
builtin.datetime.date	tomorrow	`{ "type": "builtin.datetime.date", "entity": "tomorrow", "resolution": {"date": "2017-06-21"} }`
builtin.datetimeV2.date	tomorrow	`{ "type": "builtin.datetimeV2.date", "entity": "tomorrow", "resolution": {"values": [{"timex": "2017-06-21","type": "date", "value": "2017-06-21"}]} }`
builtin.datetime.date	january 10 2009	`{ "type": "builtin.datetime.date", "entity": "january 10 2009", "resolution": {"date": "2009-01-10"} }`
builtin.datetimeV2.date	january 10 2009	`{ "type": "builtin.datetimeV2.date", "entity": "january 10 2009", "resolution": { "values": [ {"timex": "2009-01-10", "type": "date", "value": "2009-01-10" }] } }`
builtin.datetime.date	monday	`{ "entity": "monday", "type": "builtin.datetime.date", "resolution": {"date": "XXXX-WXX-1"} }`
builtin.datetimeV2.date	monday	`{ "entity": "monday", "type": "builtin.datetimeV2.date", "resolution": { "values": [{ "timex": "XXXX-WXX-1", "type": "date", "value": "2017-06-19" }, {"timex": "XXXX-WXX-1","type": "date", "value": "2017-06-26"}]} }`
builtin.datetime.date	next week	`{ "entity": "next week", "type": "builtin.datetime.date", "resolution": {"date": "2017-W26"} }`
builtin.datetimeV2.daterange	next week	`{ "entity": "next week", "type": "builtin.datetime.dateV2.daterange", "resolution": { "values": [{ "timex": "2017-W27", "type": "daterange", "start": "2017-06-26", "end": "2017-07-03"}] } }`
builtin.datetime.date	next monday	`{ "entity": "next monday", "type": "builtin.datetime.date", "resolution": {"date": "2017-06-26"} }`

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
builtin.datetimeV2.date	next monday	`{ "entity": "next monday", "type": "builtin.datetimeV2.date", "resolution": { "values": [{ "timex": "2017-06-26", "type": "date", "value": "2017-06-26" }] } }`
builtin.datetime.time	3 : 00	`{ "type": "builtin.datetime.time", "entity": "3 : 00", "resolution": {"comment": "ampm", "time": "T03:00"} }`
builtin.datetimeV2.time	3 : 00	`{ "type": "builtin.datetimeV2.time", "entity": "3 : 00", "resolution": { "values": [{ "timex": "T03:00", "type": "time", "value": "03:00:00" }, { "timex": "T15:00", "type": "time", "value": "15:00:00" }]} }`
builtin.datetime.time	4 pm	`{ "type": "builtin.datetime.time", "entity": "4 pm", "resolution": {"time": "T16"} }`
builtin.datetimeV2.time	4 pm	`{ "type": "builtin.datetimeV2.time", "entity": "4 pm", "resolution": { "values": [{"timex": "T16", "type": "time", "value": "16:00:00"}] } }`
builtin.datetime.time	tomorrow morning	`{ "entity": "tomorrow morning", "type": "builtin.datetime.time", "resolution": {"time": "2015-08-15TMO"} }`
builtin.datetimeV2.datetimerange	tomorrow morning	`{ "entity": "tomorrow morning", "type": "builtin.datetimev2.datetimerange", "resolution": { "values": [{"timex": "2017-06-21TMO","type": "datetimerange", "start": "2017-06-21 08:00:00", "end": "2017-06-21 12:00:00"}]} }`
builtin.datetime.time	tonight	`{ "entity": "tonight", "type": "builtin.datetime.time", "resolution": {"time": "2015-08-14TNI"} }`
builtin.datetimeV2.datetimerange	tonight	`{ "entity": "tonight", "type": "builtin.datetimeV2.datetimerange", "resolution": { "values": [{"timex": "2017-06-20TNI","type": "datetimerange", "start": "2017-06-20 20:00:00", "end": "2017-06-20 23:59:59"}]} }`
builtin.datetime.duration	for 3 hours	`{ "type": "builtin.datetime.duration", "entity": "3 hours", "resolution": {"duration": "PT3H"} }`

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
builtin.datetimeV2.duration	for 3 hours	`{ "type": "builtin.datetimeV2.duration", "entity": "3 hours", "resolution": { "values": [{ "timex": "PT3H", "type": "duration", "value": "10800"}] } }`
builtin.datetime.duration	30 minutes long	`{ "type": "builtin.datetime.duration", "entity": "30 minutes", "resolution": {"duration": "PT30M"} }`
builtin.datetimeV2.duration	30 minutes long	`{ "type": "builtin.datetimeV2.duration", "entity": "30 minutes", "resolution": { "values": [{ "timex": "PT30M", "type": "duration", "value": "1800"}] } }`
builtin.datetime.duration	all day	`{ "type": "builtin.datetime.duration", "entity": "all day", "resolution": {"duration": "P1D"} }`
builtin.datetimeV2.duration	all day	`{ "type": "builtin.datetimeV2.duration", "entity": "all day", "resolution": { "values": [{ "timex": "P1D", "type": "duration", "value": "86400"}] } }`
builtin.datetime.set	daily	`{ "type": "builtin.datetime.set", "entity": "daily", "resolution": "set": {"XXXX-XX-XX"} }`
builtin.datetimeV2.set	daily	`{ "type": "builtin.datetimeV2.set", "entity": "daily", "resolution": { "values": [{ "timex": "P1D", "type": "set", "value": "not resolved"}]} }`
builtin.datetime.set	every tuesday	`{ "entity": "every tuesday", "type": "builtin.datetime.set", "resolution": {"time": "XXXX-WXX-2"} }`
builtin.datetimeV2.set	every tuesday	`{ "entity": "every tuesday", "type": "builtin.datetimeV2.set", "resolution": { "values": [{ "timex": "XXXX-WXX-2", "type": "set", "value": "not resolved"}]} }`
builtin.datetime.set	every week	`{ "entity": "every week", "type": "builtin.datetime.set", "resolution": {"time": "XXXX-WXX"} }`
builtin.datetimeV2.set	every week	`{ "entity": "every week", "type": "builtin.datetimeV2.set", "resolution": {"time": "XXXX-WXX"} }`

builtin.geography

> **NOTE**
>
> **builtin.geography** is available only in the en-us locale.

The builtin.geography built-in entity type has 3 sub-types:

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
builtin.geography.city	seattle	`{ "type": "builtin.geography.city", "entity": "seattle" }`
builtin.geography.city	paris	`{ "type": "builtin.geography.city", "entity": "paris" }`
builtin.geography.country	australia	`{ "type": "builtin.geography.country", "entity": "australia" }`
builtin.geography.country	japan	`{ "type": "builtin.geography.country", "entity": "japan" }`
builtin.geography.pointOfInterest	amazon river	`{ "type": "builtin.geography.pointOfInterest", "entity": "amazon river" }`
builtin.geography.pointOfInterest	sahara desert	`{ "type": "builtin.geography.pointOfInterest", "entity": "sahara desert" }`

builtin.encyclopedia

> **NOTE**
>
> **builtin.encyclopedia** is available only in the en-US locale.

The builtin.encyclopedia built-in entity includes over 100 sub-types, listed below. In addition, encyclopedia entities often map to multiple types. For example, the query Ronald Reagan yields:

```
{
    "entity": "ronald reagan",
    "type": "builtin.encyclopedia.people.person"
  },
  {
    "entity": "ronald reagan",
    "type": "builtin.encyclopedia.film.actor"
  },
  {
    "entity": "ronald reagan",
    "type": "builtin.encyclopedia.government.us_president"
  },
  {
    "entity": "ronald reagan",
    "type": "builtin.encyclopedia.book.author"
  }
```

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.people.person	builtin.encyclopedia.people.person	bryan adams
builtin.encyclopedia.people.person	builtin.encyclopedia.film.producer	walt disney
builtin.encyclopedia.people.person	builtin.encyclopedia.film.cinematographer	adam greenberg
builtin.encyclopedia.people.person	builtin.encyclopedia.royalty.monarch	elizabeth ii
builtin.encyclopedia.people.person	builtin.encyclopedia.film.director	steven spielberg
builtin.encyclopedia.people.person	builtin.encyclopedia.film.writer	alfred hitchcock
builtin.encyclopedia.people.person	builtin.encyclopedia.film.actor	robert de niro
builtin.encyclopedia.people.person	builtin.encyclopedia.martial_arts.martial_artist	bruce lee
builtin.encyclopedia.people.person	builtin.encyclopedia.architecture.architect	james gallier
builtin.encyclopedia.people.person	builtin.encyclopedia.geography.mountaineer	jean couzy
builtin.encyclopedia.people.person	builtin.encyclopedia.celebrities.celebrity	angelina jolie
builtin.encyclopedia.people.person	builtin.encyclopedia.music.musician	bob dylan
builtin.encyclopedia.people.person	builtin.encyclopedia.soccer.player	diego maradona
builtin.encyclopedia.people.person	builtin.encyclopedia.baseball.player	babe ruth
builtin.encyclopedia.people.person	builtin.encyclopedia.basketball.player	heiko schaffartzik
builtin.encyclopedia.people.person	builtin.encyclopedia.olympics.athlete	andre agassi
builtin.encyclopedia.people.person	builtin.encyclopedia.basketball.coach	bob huggins
builtin.encyclopedia.people.person	builtin.encyclopedia.american_football.coach	james franklin
builtin.encyclopedia.people.person	builtin.encyclopedia.cricket.coach	andy flower
builtin.encyclopedia.people.person	builtin.encyclopedia.ice_hockey.coach	david quinn
builtin.encyclopedia.people.person	builtin.encyclopedia.ice_hockey.player	vincent lecavalier
builtin.encyclopedia.people.person	builtin.encyclopedia.government.politician	harold nicolson

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.people.person	builtin.encyclopedia.government.us_president	barack obama
builtin.encyclopedia.people.person	builtin.encyclopedia.government.us_vice_president	dick cheney
builtin.encyclopedia.organization.organization	builtin.encyclopedia.organization.organization	united nations
builtin.encyclopedia.organization.organization	builtin.encyclopedia.sports.league	american league
builtin.encyclopedia.organization.organization	builtin.encyclopedia.ice_hockey.conference	western hockey league
builtin.encyclopedia.organization.organization	builtin.encyclopedia.baseball.division	american league east
builtin.encyclopedia.organization.organization	builtin.encyclopedia.baseball.league	major league baseball
builtin.encyclopedia.organization.organization	builtin.encyclopedia.basketball.conference	national basketball league
builtin.encyclopedia.organization.organization	builtin.encyclopedia.basketball.division	pacific division
builtin.encyclopedia.organization.organization	builtin.encyclopedia.soccer.league	premier league
builtin.encyclopedia.organization.organization	builtin.encyclopedia.american_football.division	afc north
builtin.encyclopedia.organization.organization	builtin.encyclopedia.broadcast.broadcast	nebraska educational telecommunications
builtin.encyclopedia.organization.organization	builtin.encyclopedia.broadcast.tv_station	abc
builtin.encyclopedia.organization.organization	builtin.encyclopedia.broadcast.tv_channel	cnbc world
builtin.encyclopedia.organization.organization	builtin.encyclopedia.broadcast.radio_station	bbc radio 1
builtin.encyclopedia.organization.organization	builtin.encyclopedia.business.operation	bank of china
builtin.encyclopedia.organization.organization	builtin.encyclopedia.music.record_label	pixar
builtin.encyclopedia.organization.organization	builtin.encyclopedia.aviation.airline	air france

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.organization.organization	builtin.encyclopedia.automotive.company	general motors
builtin.encyclopedia.organization.organization	builtin.encyclopedia.music.musical_instrument_company	gibson guitar corporation
builtin.encyclopedia.organization.organization	builtin.encyclopedia.tv.network	cartoon network
builtin.encyclopedia.organization.organization	builtin.encyclopedia.education.educational_institution	cornwall hill college
builtin.encyclopedia.organization.organization	builtin.encyclopedia.education.school	boston arts academy
builtin.encyclopedia.organization.organization	builtin.encyclopedia.education.university	johns hopkins university
builtin.encyclopedia.organization.organization	builtin.encyclopedia.sports.team	united states national handball team
builtin.encyclopedia.organization.organization	builtin.encyclopedia.basketball.team	chicago bulls
builtin.encyclopedia.organization.organization	builtin.encyclopedia.sports.professional_sports_team	boston celtics
builtin.encyclopedia.organization.organization	builtin.encyclopedia.cricket.team	mumbai indians
builtin.encyclopedia.organization.organization	builtin.encyclopedia.baseball.team	houston astros
builtin.encyclopedia.organization.organization	builtin.encyclopedia.american_football.team	green bay packers
builtin.encyclopedia.organization.organization	builtin.encyclopedia.ice_hockey.team	hamilton bulldogs
builtin.encyclopedia.organization.organization	builtin.encyclopedia.soccer.team	fc bayern munich
builtin.encyclopedia.organization.organization	builtin.encyclopedia.government.political_party	pertubuhan kebangsaan melayu singapura
builtin.encyclopedia.time.event	builtin.encyclopedia.time.event	1740 batavia massacre
builtin.encyclopedia.time.event	builtin.encyclopedia.sports.championship_event	super bowl xxxix
builtin.encyclopedia.time.event	builtin.encyclopedia.award.competition	eurovision song contest 2003
builtin.encyclopedia.tv.series_episode	builtin.encyclopedia.tv.series_episode	the magnificent seven

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.tv.series_episode	builtin.encyclopedia.tv.multipart_tv_episode	the deadly assassin
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.commerce.consumer_product	nokia lumia 620
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.music.album	dance pool
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.automotive.model	pontiac fiero
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.computer.computer	toshiba satellite
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.computer.web_browser	internet explorer
builtin.encyclopedia.commerce.brand	builtin.encyclopedia.commerce.brand	diet coke
builtin.encyclopedia.commerce.brand	builtin.encyclopedia.automotive.make	chrysler
builtin.encyclopedia.music.artist	builtin.encyclopedia.music.artist	michael jackson
builtin.encyclopedia.music.artist	builtin.encyclopedia.music.group	the yardbirds
builtin.encyclopedia.music.music_video	builtin.encyclopedia.music.music_video	the beatles anthology
builtin.encyclopedia.theater.play	builtin.encyclopedia.theater.play	camelot
builtin.encyclopedia.sports.fight_song	builtin.encyclopedia.sports.fight_song	the cougar song
builtin.encyclopedia.film.series	builtin.encyclopedia.film.series	the twilight saga
builtin.encyclopedia.tv.program	builtin.encyclopedia.tv.program	late night with david letterman
builtin.encyclopedia.radio.radio_program	builtin.encyclopedia.radio.radio_program	grand ole opry
builtin.encyclopedia.film.film	builtin.encyclopedia.film.film	alice in wonderland
builtin.encyclopedia.cricket.tournament	builtin.encyclopedia.cricket.tournament	cricket world cup
builtin.encyclopedia.government.government	builtin.encyclopedia.government.government	european commission
builtin.encyclopedia.sports.team_owner	builtin.encyclopedia.sports.team_owner	bob castellini
builtin.encyclopedia.music.genre	builtin.encyclopedia.music.genre	eastern europe
builtin.encyclopedia.ice_hockey.division	builtin.encyclopedia.ice_hockey.division	hockeyallsvenskan

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.architecture.style	builtin.encyclopedia.architecture.style	spanish colonial revival architecture
builtin.encyclopedia.broadcast.producer	builtin.encyclopedia.broadcast.producer	columbia tristar television
builtin.encyclopedia.book.author	builtin.encyclopedia.book.author	adam maxwell
builtin.encyclopedia.religion.founding_figur	builtin.encyclopedia.religion.founding_figur	gautama buddha
builtin.encyclopedia.martial_arts.martial_art	builtin.encyclopedia.martial_arts.martial_art	american kenpo
builtin.encyclopedia.sports.school	builtin.encyclopedia.sports.school	yale university
builtin.encyclopedia.business.product_line	builtin.encyclopedia.business.product_line	canon powershot
builtin.encyclopedia.internet.website	builtin.encyclopedia.internet.website	bing
builtin.encyclopedia.time.holiday	builtin.encyclopedia.time.holiday	easter
builtin.encyclopedia.food.candy_bar	builtin.encyclopedia.food.candy_bar	cadbury dairy milk
builtin.encyclopedia.finance.stock_exchange	builtin.encyclopedia.finance.stock_exchange	tokyo stock exchange
builtin.encyclopedia.film.festival	builtin.encyclopedia.film.festival	berlin international film festival

Use prebuilt domains in LUIS apps

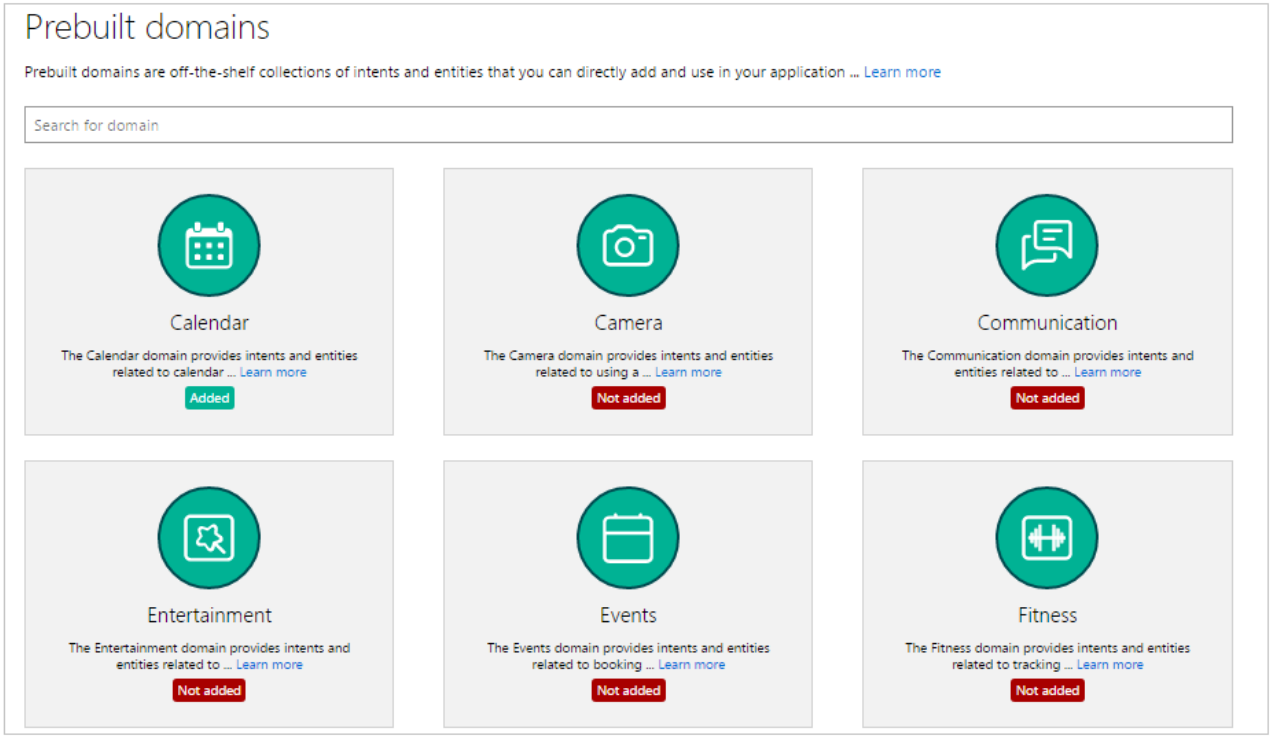
6/27/2017 • 2 min to read • Edit Online

LUIS provides *prebuilt domains*, which are prebuilt sets of intents and entities that work together for domains or common categories of apps. The prebuilt domains have been pre-trained and are ready for use. The intents and entities in a prebuilt domain are fully customizable once you've added them to your app - you can train them with utterances from your system so they work for your users. You can use an entire prebuilt domain as a starting point for customization, or just borrow a few intents or entities from a domain for your application.

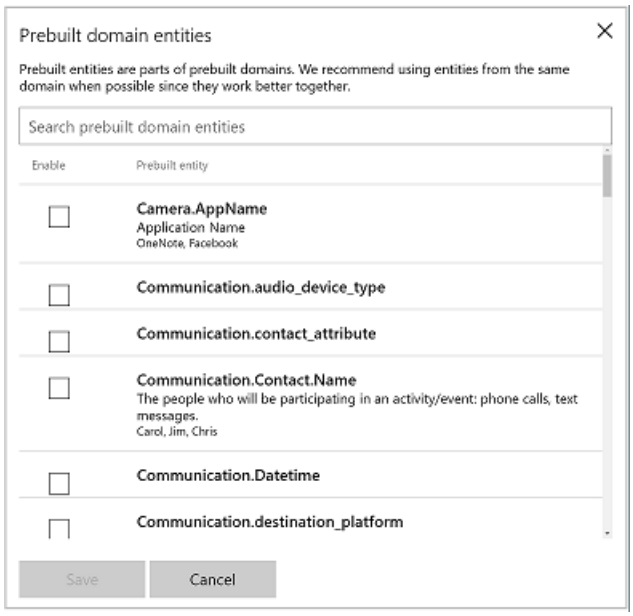
LUIS offers 20 prebuilt domains. They include but are not limited to:

PREBUILT DOMAIN	DESCRIPTION
Camera	Taking pictures and recording videos.
Communication	Sending messages and making phone calls.
Entertainment	Handling queries related to music, movies, and TV.
Places	Handling queries related to places like businesses, institutions, restaurants, public spaces and addresses.
Utilities	Handling requests that are common in many domains, like "help", "repeat", "start over".

Browse the **Prebuilt domains** tab to see other prebuilt domains you can use in your app. Click on the tile for a domain to add it to your app, or click on "learn more" in its tile to learn about its intents and entities.



Within a domain, look for individual intents and entities that you want to use.



Designing LUIS apps from prebuilt domains

When using prebuilt domains in your LUIS app, you can customize an entire prebuilt domain, or just start with a few of its intents and entities.

Customizing an entire prebuilt domain

Prebuilt domains are designed to be general and contain many intents and entities, that you can choose from to customize an app to your needs. If you start from customizing an entire prebuilt domain, delete the intents and entities that your app doesn't need to use. You can also add some intents or entities to the set that the prebuilt domain already provides. For example, if you are using the **Events** prebuilt domain for a sports event app, you may want to add entities for sports teams or tournaments. When you start providing utterances to LUIS, include terms that are specific to your app. LUIS learns to recognize them and tailors the prebuilt domain's intents and entities to your app's needs.

> **TIP**
>
> The intents and entities in a prebuilt domain work best together. It's better to combine intents and entities from the same domain when possible.
>
> - A best practice is to use related intents from the same domain. For example, if you are customizing the `MakeReservation` intent in the **Places** domain, then select the `Cancel` intent from the **Places** domain instead of the Cancel intent in the **Events** or **Utilities** domains.

Changing the behavior of a prebuilt domain intent

You might find that a prebuilt domain contains an intent that is similar to an intent you want to have in your LUIS app but you want it to behave differently. For example, the **Places** prebuilt domain provides an `MakeReservation` intent for making a restaurant reservation, but you want your app to use that intent to make hotel reservations. In that case, you can modify the behavior of that intent by providing utterances to LUIS about making hotel reservations and labeling them using the `MakeReservation` intent, so then LUIS can be retrained to recognize the `MakeReservation` intent in a request to book a hotel.

> **TIP**
>
> Check out the Utilities domain for prebuilt intents that you can customize for use in any domain. For example, you can add `Utilities.Repeat` to your app and train it recognize whatever actions user might want to repeat in your application.

Cortana Prebuilt App

6/27/2017 • 20 min to read • Edit Online

IMPORTANT

We recommend that you use the prebuilt domains, instead of the Cortana prebuilt app. For example, instead of **builtin.intent.calendar.create_calendar_entry**, use **Calendar.Add** from the **Calendar** prebuilt domain. The prebuilt domains provide these advantages:

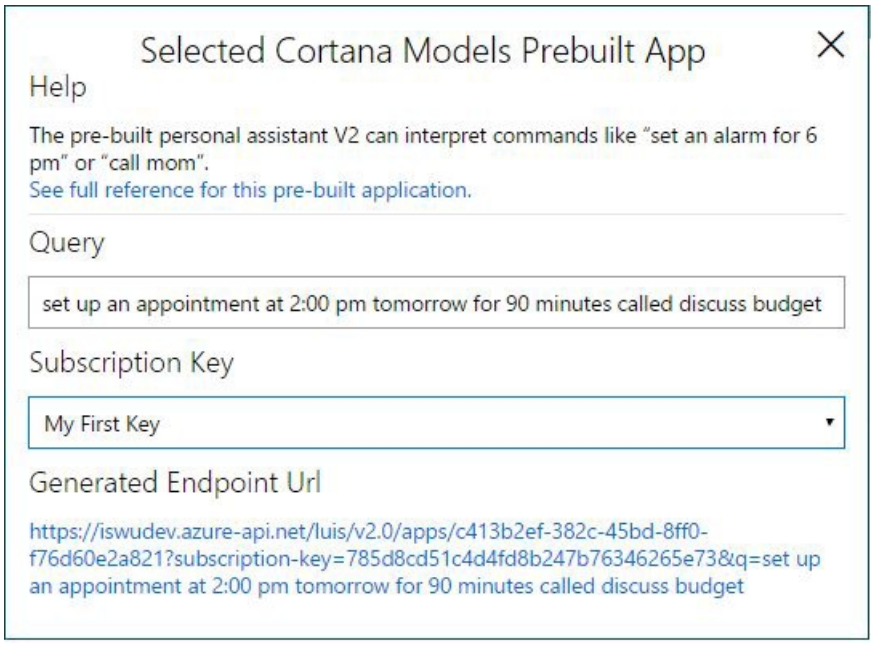
- They provide packages of prebuilt and pretrained intents and entities that are designed to work well with each other. You can integrate a prebuilt domain directly into your app. For example, if you're building a fitness tracker, you can add the **Fitness** domain and have an entire set of intents and entities for tracking fitness activities, including intents for tracking weight and meal planning, remaining time or distance, and saving fitness activity notes.
- The prebuilt domain intents are customizable. For example, if you want to provide reviews of hotels, you can train and customize the **Places.GetReviews** intent from the **Places** domain to recognize requests for hotel reviews.
- The prebuilt domains are extensible. For example, if you want to use the **Places** prebuilt domain in a bot that searches for restaurants, and need an intent for getting the type of cuisine, you can build and train a **Places.GetCuisine** intent.

In addition to allowing you to build your own applications, LUIS also provides intents and entities from the Microsoft Cortana personal assistant as a prebuilt app. This is an "as-is" application. The intents and entities in this application cannot be edited or integrated into other LUIS apps. If you'd like your client to have access to both this prebuilt application and your own LUIS application, then your client has to reference both LUIS apps.

The pre-built personal assistant app is available in these cultures (locales): English, French, Italian, Spanish, and Chinese.

Use the Cortana Prebuilt App

1. On **My Apps** page, click **Cortana prebuilt apps** and select your language, English for example. The following dialog box appears:



2. In **Query**, type the utterance you want to interpret. For example, type "set up an appointment at 2:00 pm tomorrow for 90 minutes called discuss budget"

3. From the **Subscription Key** list, select the subscription key to be used for this endpoint hit to Cortana app.
4. Click the generated endpoint URL to access the endpoint and get the result of the query. The screenshot below shows the result returned in JSON format for the example utterance: "set up an appointment at 2:00 pm tomorrow for 90 minutes called discuss budget"


```
{
  "query": "set up an appointment at 2:00 pm tomorrow for 90 minutes called discuss budget",
  "intents": [
    {
      "intent": "builtin.intent.calendar.create_calendar_entry"
    }
  ],
  "entities": [
    {
      "entity": "2:00 pm",
      "type": "builtin.calendar.start_time",
      "resolution": {
        "resolution_type": "builtin.datetime.time",
        "time": "T14:00"
      }
    },
    {
      "entity": "tomorrow",
      "type": "builtin.calendar.start_date",
      "resolution": {
        "date": "2017-02-20",
        "resolution_type": "builtin.datetime.date"
      }
    },
    {
      "entity": "90 minutes",
      "type": "builtin.calendar.duration",
      "resolution": {
        "duration": "PT90M",
        "resolution_type": "builtin.datetime.duration"
      }
    },
    {
      "entity": "discuss budget",
      "type": "builtin.calendar.title"
    }
  ]
}
```


Cortana prebuilt intents

The pre-built personal assistant application can identify the following intents:

INTENT	EXAMPLE UTTERANCES
builtin.intent.alarm.alarm_other	update my 7:30 alarm to be eight o'clock change my alarm from 8am to 9am
builtin.intent.alarm.delete_alarm	delete an alarm delete my alarm "wake up"
builtin.intent.alarm.find_alarm	what time is my wake-up alarm set for? is my wake-up alarm on?
builtin.intent.alarm.set_alarm	turn on my wake up alarm can you set an alarm for 12 called take antibiotics?
builtin.intent.alarm.snooze	snooze alarm for 5 minutes snooze alarm
builtin.intent.alarm.time_remaining	how much longer do i have until "wake-up"? how much time until my next alarm?

INTENT	EXAMPLE UTTERANCES
builtin.intent.alarm.turn_off_alarm	turn off my 7am alarm turn off my wake up alarm
builtin.intent.calendar.change_calendar_entry	change an appointment move my meeting from today to tomorrow
builtin.intent.calendar.check_availability	is tim busy on friday? is brian free on saturday
builtin.intent.calendar.connect_to_meeting	connect to a meeting join the meeting online
builtin.intent.calendar.create_calendar_entry	i need to schedule an appointment with tony for friday make an appointment with lisa at 2pm on sunday
builtin.intent.calendar.delete_calendar_entry	delete my appointment remove the meeting at 3 pm tomorrow from my calendar
builtin.intent.calendar.find_calendar_entry	show my calendar display my weekly calendar
builtin.intent.calendar.find_calendar_when	when is my next meeting with jon? what time is my appointment with larry on monday?
builtin.intent.calendar.find_calendar_where	show me the location of my 6 o'clock meeting where is that meeting with jon?
builtin.intent.calendar.find_calendar_who	who is this meeting with? who else is invited?
builtin.intent.calendar.find_calendar_why	what are the details of my 11 o'clock meeting? what is that meeting with jon about?
builtin.intent.calendar.find_duration	how long is my next meeting how many minutes long is my meeting this afternoon
builtin.intent.calendar.time_remaining	how much time until my next appointment? how much more time do i have before the meeting?
builtin.intent.communication.add_contact	save this number and put the name as jose put jason in my contacts list
builtin.intent.communication.answer_phone	answer answer the call
builtin.intent.communication.assign_nickname	edit nickname for nickolas add a nickname for john doe
builtin.intent.communication.call_voice_mail	voice mail call voicemail
builtin.intent.communication.find_contact	show me my contacts find contacts

INTENT	EXAMPLE UTTERANCES
builtin.intent.communication.forwarding_off	stop forwarding my calls switch off call forwarding
builtin.intent.communication.forwarding_on	set call forwarding to send calls to home phone forward calls to home phone
builtin.intent.communication.ignore_incoming	do not answer that call not now, i'm busy
builtin.intent.communication.ignore_with_message	don't answer that call but send a message instead ignore and send a text back
builtin.intent.communication.make_call	call bob and john call mom and dad
builtin.intent.communication.press_key	dial star press the 1 2 3
builtin.intent.communication.read_aloud	read text what did she say in the message
builtin.intent.communication.redial	redial my last call redial
builtin.intent.communication.send_email	email to mike waters mike that dinner last week was splendid send an email to bob
builtin.intent.communication.send_text	send text to bob and john message
builtin.intent.communication.speakerphone_off	take me off speaker turn off speakerphone
builtin.intent.communication.speakerphone_on	speakerphone mode put speakerphone on
builtin.intent.mystuff.find_attachment	find the document john emailed me yesterday find the doc from john
builtin.intent.mystuff.find_my_stuff	i want to edit my shopping list from yesterday find my chemistry notes from september
builtin.intent.mystuff.search_messages	open message messages
builtin.intent.mystuff.transform_my_stuff	share my shopping list with my husband delete my shopping list
builtin.intent.ondevice.open_setting	open cortana settings jump to notifications
builtin.intent.ondevice.pause	turn off music music off

INTENT	EXAMPLE UTTERANCES
builtin.intent.ondevice.play_music	i want to hear owner of a lonely heart play some gospel music
builtin.intent.ondevice.recognize_song	tell me what this song is analyze and retrieve title of song
builtin.intent.ondevice.repeat	repeat this track play this song again
builtin.intent.ondevice.resume	restart music start music again
builtin.intent.ondevice.skip_back	back up a track previous song
builtin.intent.ondevice.skip_forward	next song skip ahead a track
builtin.intent.ondevice.turn_off_setting	wifi off disable airplane mode
builtin.intent.ondevice.turn_on_setting	wifi on turn on bluetooth
builtin.intent.places.add_favorite_place	add this address to my favorites save this location to my favorites
builtin.intent.places.book_public_transportation	buy a train ticket book a tram pass
builtin.intent.places.book_taxi	can you find me a ride at this hour? find a taxi
builtin.intent.places.check_area_traffic	what's the traffic like on 520 how is the traffic on i-5
builtin.intent.places.check_into_place	check into joya check in here
builtin.intent.places.check_route_traffic	show me the traffic on the way to kirkland how is the traffic to seattle?
builtin.intent.places.find_place	where do i live give me the top 3 japanese restaurants
builtin.intent.places.get_address	show me the address of guu on robson street what is the address of the nearest starbucks?
builtin.intent.places.get_coupon	find deals on tvs in my area discounts in mountain view
builtin.intent.places.get_distance	how far away is holiday inn? what's the distance to tahoe

INTENT	EXAMPLE UTTERANCES
builtin.intent.places.get_hours	what are bar del corso's hours on mondays? when is the library open?
builtin.intent.places.get_menu	show me applebee's menu what's on the menu at sizzler
builtin.intent.places.get_phone_number	give the number for home depot what is the phone number of the nearest starbucks?
builtin.intent.places.get_price_range	how much does dinner at red lobster cost how expensive is purple cafe
builtin.intent.places.get_reviews	show me reviews for local hardware stores i want to see restaurant reviews
builtin.intent.places.get_route	give me directions to
builtin.intent.places.get_star_rating	how many stars does the french laundry have? is the aquarium in monterrey good?
builtin.intent.places.get_transportation_schedule	what time does the san francisco ferry leave? when is the latest train to seattle?
builtin.intent.places.get_travel_time	what's the driving time to denver from sf how long will it take to get to san francisco from here
builtin.intent.places.make_call	call dr smith in bellevue call the home depot on 1st street
builtin.intent.places.rate_place	give a rating for this restaurant rate il fornaio 5 stars on yelp
builtin.intent.places.show_map	what's my current location what's my location
builtin.intent.places.takes_reservations	is it possible to make a reservation at the olive garden does the art gallery accept reservations
builtin.intent.reminder.change_reminder	change a reminder move my picture day reminder up 30 minutes
builtin.intent.reminder.create_single_reminder	remind me to wake up at 7 am remind me to go trick or treating with luca at 4:40pm
builtin.intent.reminder.delete_reminder	delete this reminder delete my picture reminder
builtin.intent.reminder.find_reminder	do i have any reminders set up for today do i have a reminder set for luca's party
builtin.intent.reminder.read_aloud	read reminder out loud read that reminder

INTENT	EXAMPLE UTTERANCES
builtin.intent.reminder.snooze	snooze that reminder until tomorrow snooze this reminder
builtin.intent.reminder.turn_off_reminder	turn off reminder dismiss airport pick up reminder
builtin.intent.weather.change_temperature_unit	change from fahrenheit to kelvin change from fahrenheit to celsius
builtin.intent.weather.check_weather	show me the forecast for my upcoming trip to seattle how will the weather be this weekend
builtin.intent.weather.check_weather_facts	what is the weather like in hawaii in december? what was the temperature this time last year?
builtin.intent.weather.compare_weather	give me a comparison between the temperature high and lows of dallas and houston, tx how does the weather compare to slc and nyc
builtin.intent.weather.get_frequent_locations	give me my most frequent location show most often stops
builtin.intent.weather.get_weather_advisory	weather warnings show weather advisory for sacramento
builtin.intent.weather.get_weather_maps	display a weather map show me weather maps for africa
builtin.intent.weather.question_weather	will it be foggy tomorrow morning? will i need to shovel snow this weekend?
builtin.intent.weather.show_weather_progression	show local weather radar begin radar
builtin.intent.none	how old are you open camera

Prebuilt entities

Here are some examples of entities the prebuilt personal assistant application can identify:

ENTITY	EXAMPLE OF ENTITY IN UTTERANCE
builtin.alarm.alarm_state	turn `off` my wake up alarm is my wake up alarm `on`
builtin.alarm.duration	snooze for `10 minutes` snooze alarm for `5 minutes`
builtin.alarm.start_date	set an alarm for `monday` at 7 am set an alarm for `tomorrow` at noon

ENTITY	EXAMPLE OF ENTITY IN UTTERANCE
builtin.alarm.start_time	create an alarm for `30 minutes` set the alarm to go off `in 20 minutes`
builtin.alarm.title	is my `wake up` alarm on can you set an alarm for quarter to 12 monday to friday called `take antibiotics`
builtin.calendar.absolute_location	create an appointment for tomorrow at `123 main street` the meeting will take place in `cincinnati` on the 5th of june
builtin.calendar.contact_name	put a marketing meeting on my calendar and be sure that `joe` is there i want to set up a lunch at il fornaio and invite `paul`
builtin.calendar.destination_calendar	add this to my `work` schedule put this on my `personal` calendar
builtin.calendar.duration	set up an appointment for `an hour` at 6 tonight book a `2 hour` meeting with joe
builtin.calendar.end_date	create a calendar entry called vacation from tomorrow until `next monday` block my time as busy until `monday, october 5th`
builtin.calendar.end_time	the meeting ends at `5:30 PM` schedule it from 11 to `noon`
builtin.calendar.implicit_location	cancel the appointment at the dmv change the location of miles' birthday to `poppy restaurant`
builtin.calendar.move_earlier_time	push the meeting forward `an hour` move the dentist's appointment up `30 minutes`
builtin.calendar.move_later_time	move my dentist appointment `30 minutes` push the meeting out `an hour`
builtin.calendar.original_start_date	reschedule my appointment at the barber from 'tuesday' to wednesday move my meeting with ken from `monday` to tuesday
builtin.calendar.original_start_time	reschedule my meeting from `2:00` to 3 change my dentist appointment from `3:30` to 4
builtin.calendar.start_date	what time does my party start on `flag day` schedule lunch for the `friday after next` at noon
builtin.calendar.start_time	i want to schedule it for `this morning` i want to schedule it in the `morning`
builtin.calendar.title	`vet appointment` `dentist` tuesday

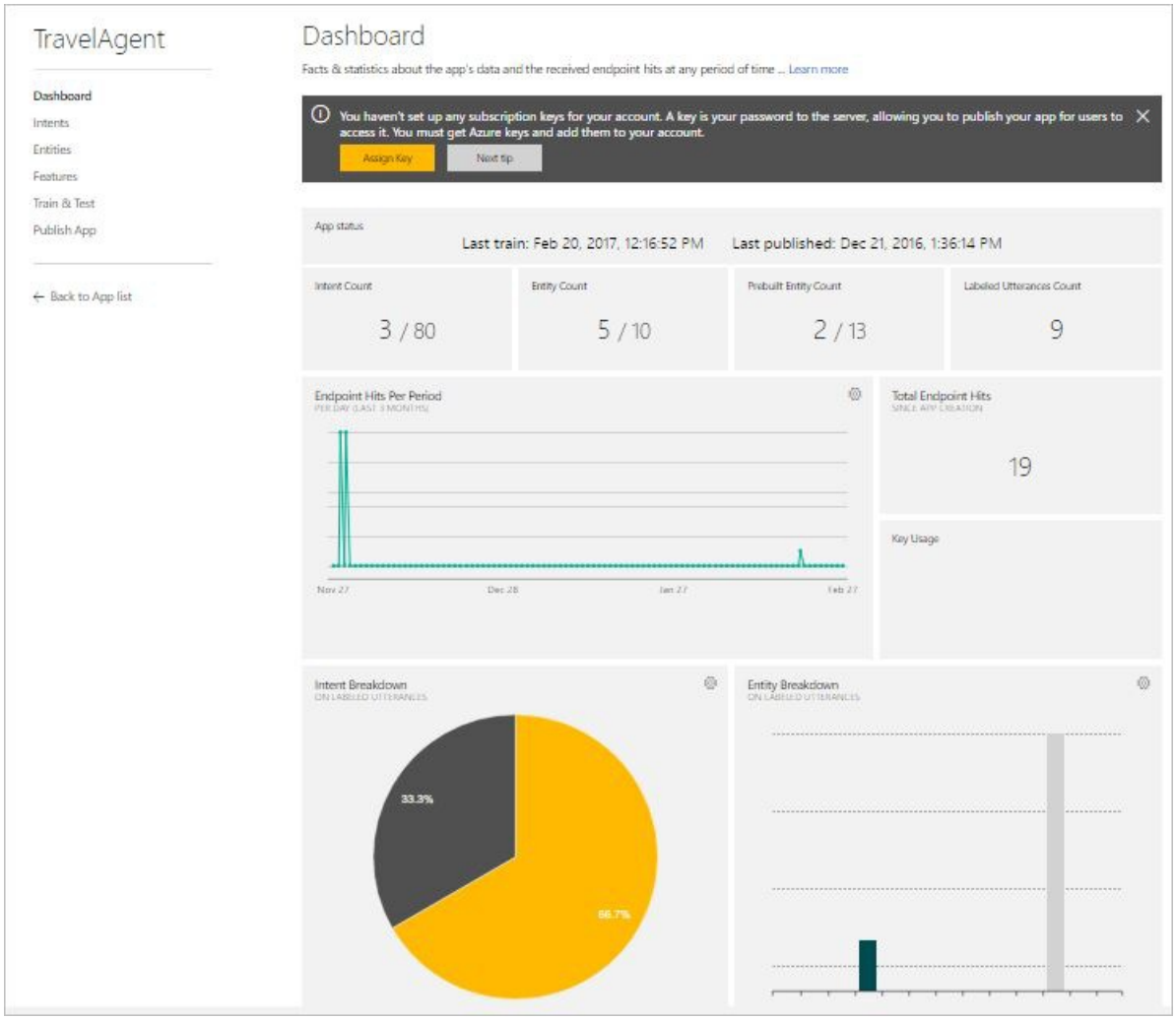
ENTITY	EXAMPLE OF ENTITY IN UTTERANCE
builtin.communication.audio_device_type	make the call using `bluetooth` call using my `headset`
builtin.communication.contact_name	text `bob jones`
builtin.communication.destination_platform	call dave in `london` call his `work line`
builtin.communication.from_relationship_name	show calls from my `daughter` read the email from `mom`
builtin.communication.key	dial `star` press the `hash` key
builtin.communication.message	email carly to say `i'm running late` please text angus smith `good luck on your exam`
builtin.communication.message_category	new email marked for `follow up` new email marked `high priority`
builtin.communication.message_type	send an `email` read my `text` messages aloud
builtin.communication.phone_number	i want to dial `1-800-328-9459` call `555-555-5555`
builtin.communication.relationship_name	text my `husband` email `family`
builtin.communication.slot_attribute	change the `recipient` change the `text`

Application Dashboard

6/27/2017 • 3 min to read • Edit Online

The app dashboard is a visualized reporting tool which enables you to monitor your app at a single glance. The **Dashboard** is the main page that is displayed when you open an app by clicking the application name on **My Apps** page. Also, you can access the **Dashboard** page from inside your app by clicking **Dashboard** in the application's left panel.

The **Dashboard** page gives you an overview of the app and displays significant data compiled from multiple app pages. Some data are analyzed and visualized by graphs and charts to help you get more insight into the app. The dashboard incorporates the latest updates in the app up to the moment and reflects any model changes. The screenshot below shows the **Dashboard** page.



At the top of the **Dashboard** page, a contextual notification bar constantly displays notifications to update you on the required or recommended actions appropriate for the current state of your app. It also provides useful tips and alerts as needed. Below is a detailed description of the data reported on the **Dashboard** page.

App Status

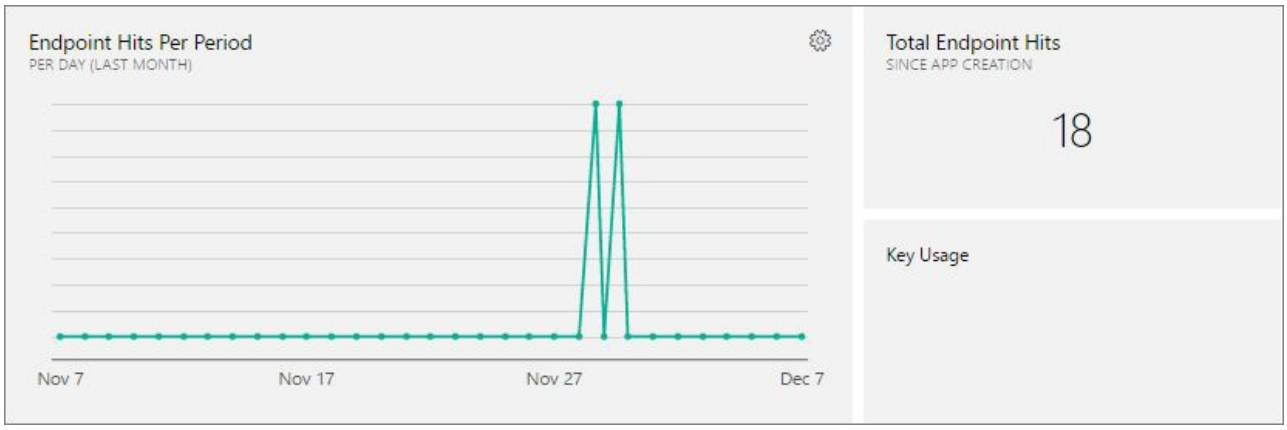
The dashboard displays the application's training and publishing status, including the date and time when the app was last trained and published.

Model Data Statistics

The dashboard displays the total numbers of intents, entities & labeled utterances existing in the app.

Endpoint Hits

The dashboard displays the total endpoint hits received to the app and enables you to display hits within a period that you specify.

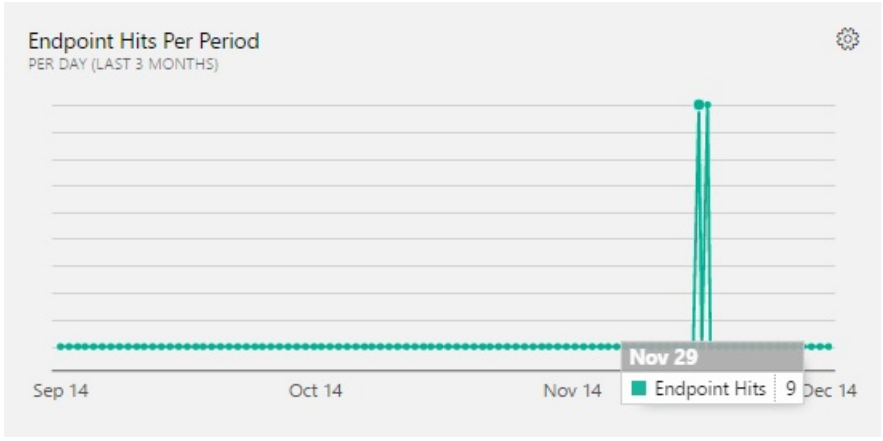


Total endpoint hits

The total number of endpoint hits received to your app since app creation up to the current date.

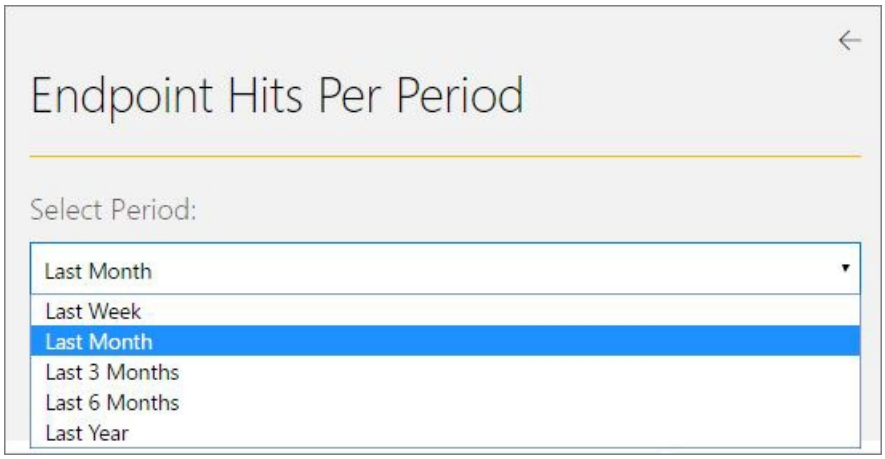
Endpoint hits per period

The number of hits received within a past period, displayed per day. A visualized line chart shows the period span from the calculated start date up to the current date (end date). The points between the start and end dates represent the days falling in this period. Hover your mouse pointer over each point to see the hits count in each day within the period. The screenshot below shows the line chart.



To select a period to view its hits on the chart:

1. Click **Additional Settings**  to access the periods list. You can select periods ranging from one week up to one year beforehand.



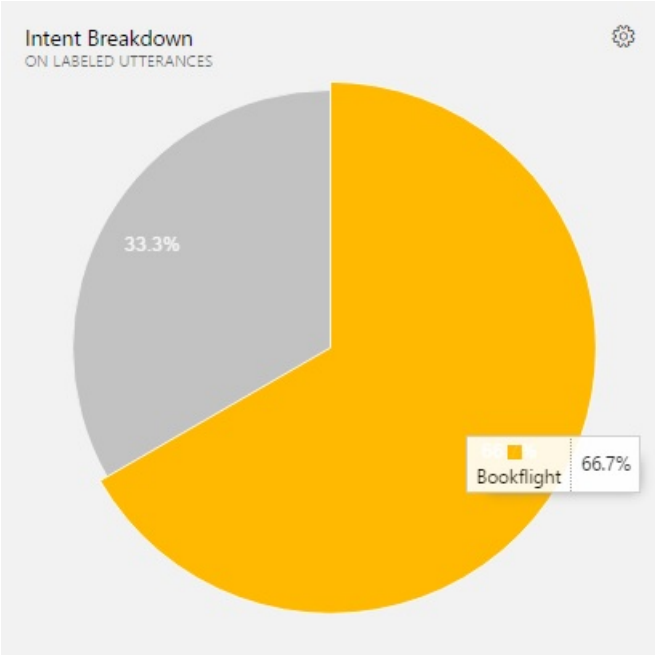
2. Select a period from the list and then click the back arrow to display its hits on the chart.

Key usage

The number of hits consumed from the application's subscription key. For more details about subscription keys, see Manage your keys.

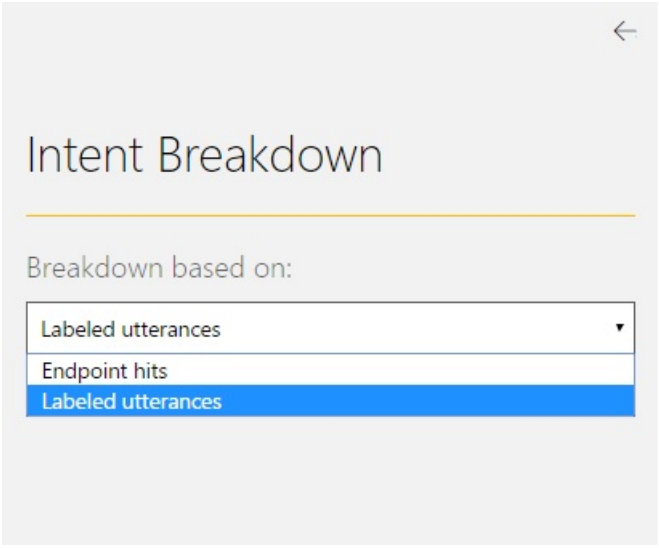
Intent Breakdown

The dashboard displays a breakdown of intents based on labeled utterances or endpoint hits. It visualizes the distribution of intents across labeled utterances/endpoint hits, so that you can see the relative importance of each intent in the app. The intent breakdown is visualized by a pie chart with the slices representing intents. When you hover your mouse pointer over a slice, you'll see the intent name and the percentage it represents of the total count of labeled utterances/endpoint hits.



To control whether the breakdown is based on labeled utterances or endpoint hits:

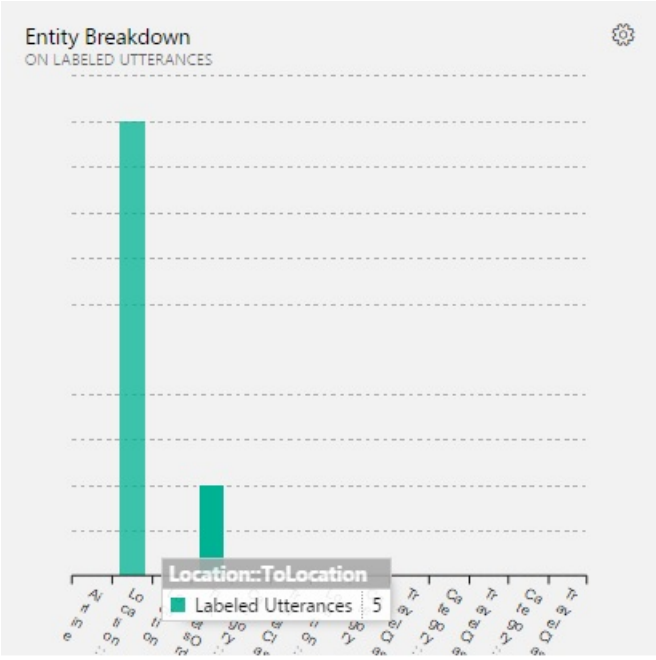
1. Click **Additional Settings** to access the list as in the screenshot below.



2. Select a value from the list and then click the back arrow to display the chart accordingly.

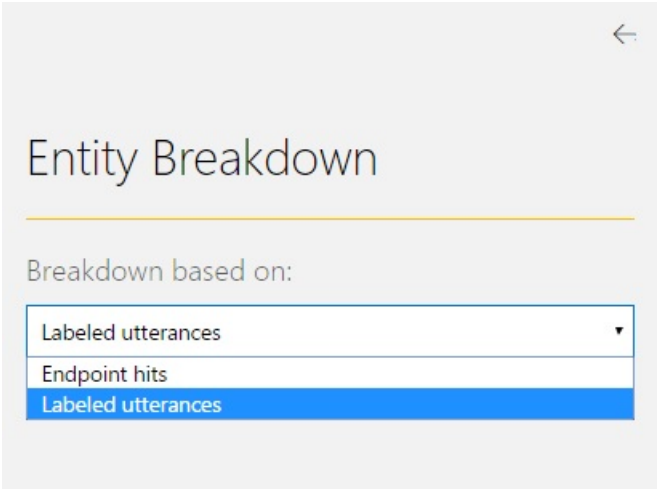
Entity Breakdown

The dashboard displays a breakdown of entities based on labeled utterances or endpoint hits. It visualizes the distribution of entities across labeled utterances/endpoint hits, showing the usage of each entity in labeled utterances/endpoint hits compared to the other entities. The entity breakdown is visualized by a column chart where entities are displayed along the horizontal axis while their count in labeled utterances/endpoint hits along the vertical axis. When you hover your mouse pointer over a rectangular bar, you'll see the entity name and its count (number of occurrences) in labeled utterances/endpoint hits.



To control whether the breakdown is based on labeled utterances or endpoint hits:

1. Click **Additional Settings** to access the list as in the screenshot below.



2. Select a value from the list and then click the back arrow  to display the chart accordingly.

Manage your keys

6/27/2017 • 4 min to read • Edit Online

A key is your passport to the server allowing you to publish your app to be used by end users. LUIS has three different types of keys:

- **Programmatic API Key:** Created automatically for LUIS account and it's free. It enables you to author and edit your application using the LUIS Programmatic APIs. Click here for a complete API Reference.
- **Endpoint Key(s):** You need to buy it from the Microsoft Azure portal. It is essential for publishing your app and accessing your HTTP endpoint. This key reflects your quota of endpoint hits based on the usage plan you specified while creating the key. See Cognitive Services Pricing for pricing information.
- **External Key(s):** You need to buy an external key only if you want to use any external services with LUIS.

The process of creating and using endpoint and external keys involves the following tasks in the same order:

1. Create a key on the Azure portal.
2. Add the key to your LUIS account (on the **My Keys** page).
3. Assign the key to your app on the **Publish** page.

Regions and keys

The region to which you publish your LUIS app must correspond to the region or location you specify in the Azure portal when you create a key. To publish a LUIS app to more than one region, you need at least one key per region. LUIS apps created on https://www.luis.ai can be published to endpoints in the following regions:

AZURE REGION	ENDPOINT URL FORMAT
West US	https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY
East US	https://eastus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY
West Central US	https://westcentralus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY
Southeast Asia	https://southeastasia.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY

To publish to the European regions, you can create LUIS apps at https://eu.luis.ai.

> **NOTE**
> LUIS apps created at https://eu.luis.ai don't automatically migrate to https://www.luis.ai. You will need to export and then import the LUIS app in order to migrate it.

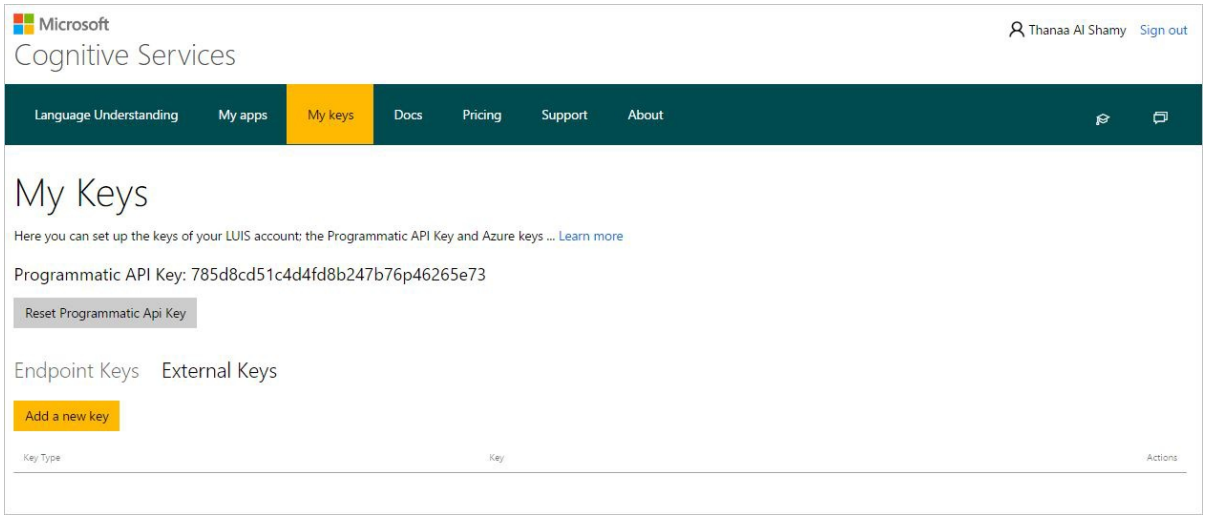
AZURE REGION	ENDPOINT URL FORMAT
West Europe	https://westeurope.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY

Reset your Programmatic API key

You can reset your Programmatic API key to get a new one generated for your account.

To reset a Programmatic API key:

1. Click **My Keys** on LUIS top navigation bar to access **My Keys** page.



2. At the top of **My Keys** page, you can see your current Programmatic API key. Click **Reset Programmatic API Key**. The new key will be generated replacing the existing one.

Create and use endpoint keys for your apps

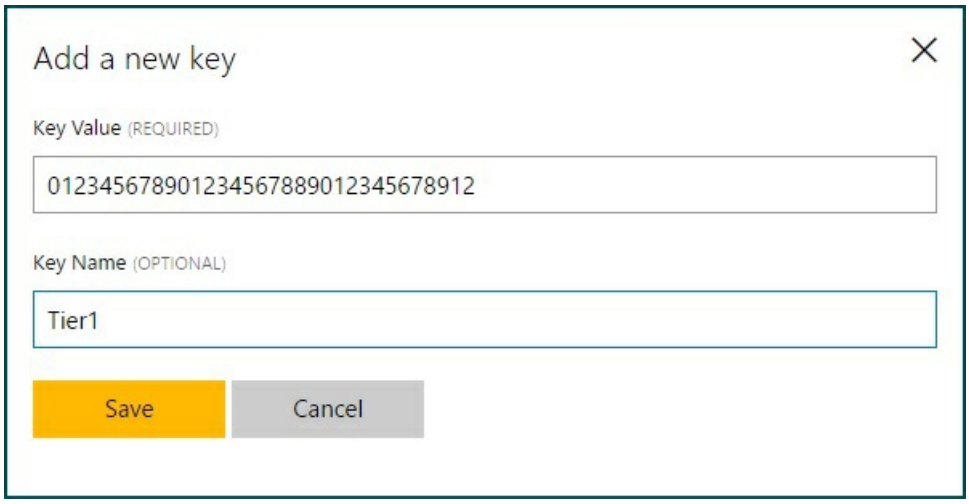
You can create as many endpoint keys as you need for your LUIS apps on Azure portal. These keys will be used for publishing your apps to the Web.

To create an endpoint key on Azure portal:

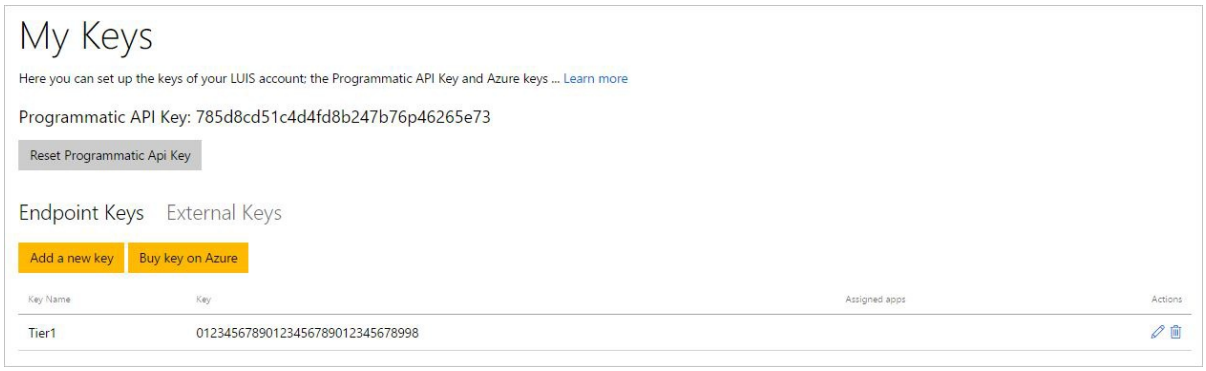
1. Click **My Keys** on LUIS top navigation bar to access **My Keys** page.
2. On **My Keys** page, **Endpoint Keys** tab, click **Buy Key on Azure**. This will take you directly to Microsoft Azure portal. You can create one or more keys per account by subscribing to one or more subscription tiers.
3. For further instructions, see Creating a subscription key using Azure.

To add the endpoint key to your LUIS account:

1. On **My Keys** page, **Endpoint Keys** tab, click **Add a new key**.



2. Copy the key you created in Azure portal in the previous procedure and paste it in the **Key Value** text box.
3. You can optionally type a name for the key in **Key Name**, for example "Tier1" , and then click **Save**. The key will be added to the keys list.



In the list of added keys, you can edit a key name, or delete a key (if no longer needed). To do this, click the edit  or delete button  corresponding to that key in the list.

To assign the endpoint key to your app:

1. Access the **Publish** page by clicking **Publish App** on the left panel.
2. From the **Endpoint Key** list, select the key that you want to assign to the app.

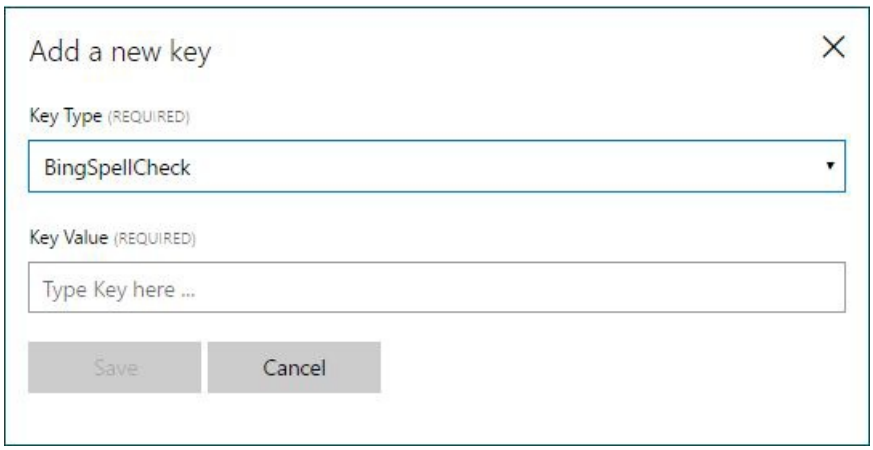
> **NOTE**
> Whenever you assign a key to the app, an updated endpoint URL will be generated. Remember to use the updated endpoint URL in your code.

Create and use external keys

External keys are the keys required for any external services that you want to use with your LUIS app to enhance the language understanding experience. One of these services, which is currently available, is Bing Spell Check. It can be used with LUIS to detect and correct any spelling mistakes in end-user queries submitted to your application's endpoint.

To create and add an external key to your LUIS account:

1. On **My Keys** page, **External Keys**, click **Add a new key**. The following dialog box appears.



2. From the **Key Type** list, select "BingSpellCheck".
3. Copy the key you created on Azure and paste it in **Key Value**, and then click **Save**. The key will be added to the keys list.

To assign the external key to your app:

1. Access the **Publish** page by clicking **Publish App** on the left panel.
2. Click **Add Key Association** 

.
3. Select Bing Spell Check as the key type from the **Key Type** list, and select from the **Key Value** list the external key that you want to assign to the app.

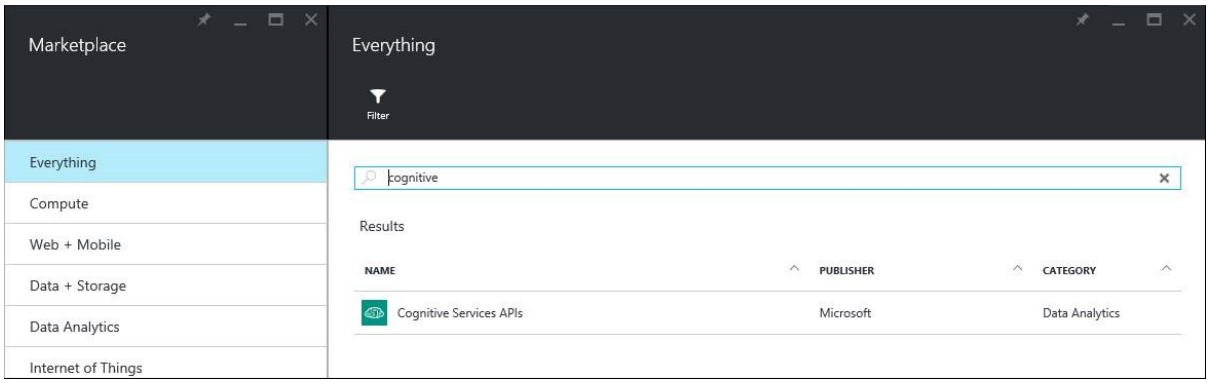
Creating Subscription Keys Via Azure

6/27/2017 • 1 min to read • Edit Online

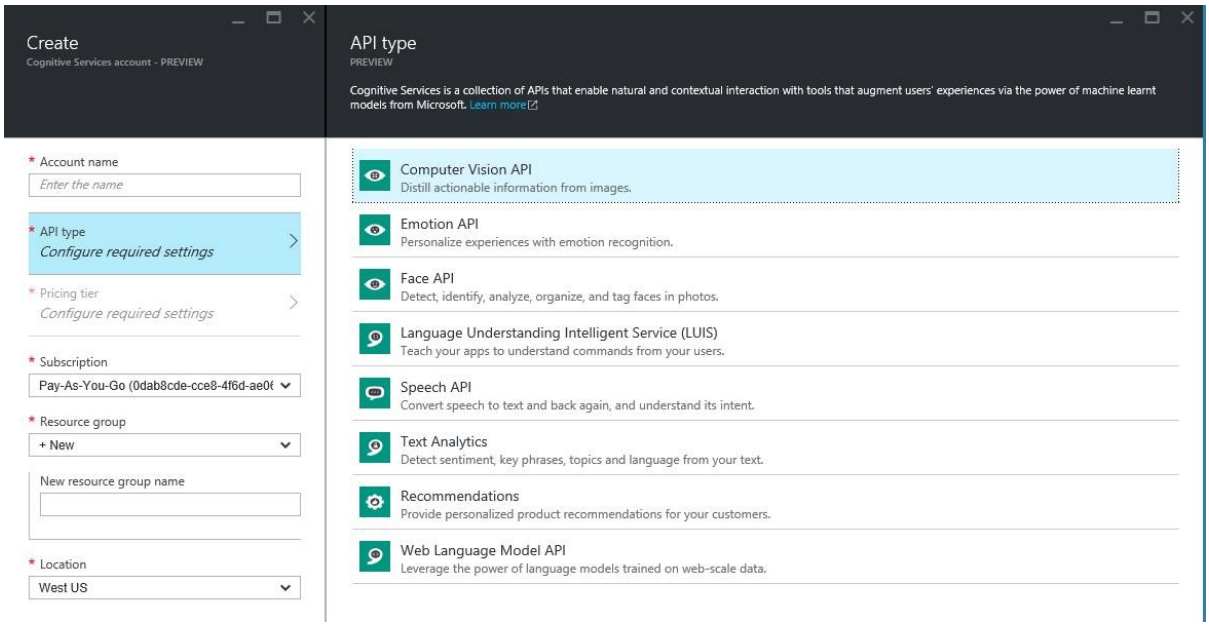
For unlimited traffic to your HTTP endpoint, you must create a metered key for your account on LUIS. Metered keys provide unlimited traffic to your endpoint following a payment plan. See Cognitive Services Pricing for pricing information.

To create your key, follow these steps:

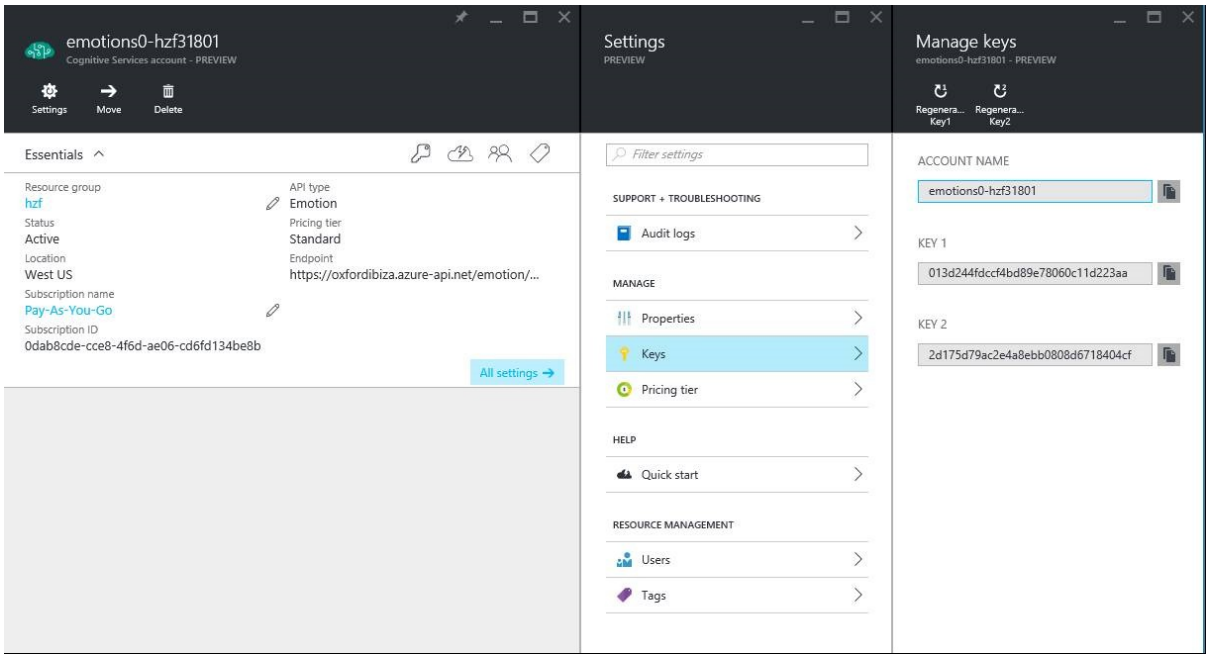
1. Sign in to the **Microsoft Azure Portal**
2. Click the green **+** sign in the upper left-hand panel and search for "Cognitive Services" in the marketplace, then click on **Cognitive Services APIs** and follow the **create experience** to create an API account you are interested in.



3. Choose the API you are interested in by clicking on the **API Type** to create a subscription. In this case, choose the **Language Understanding Intelligent Service (LUIS)** option. Configure the subscription with settings including account name, pricing tiers, etc.



4. Once you have created the API account, you can view the keys generated in the **Settings->Keys** blade. These can be tested in your existing application or by following the LUIS documentation to create a new application.



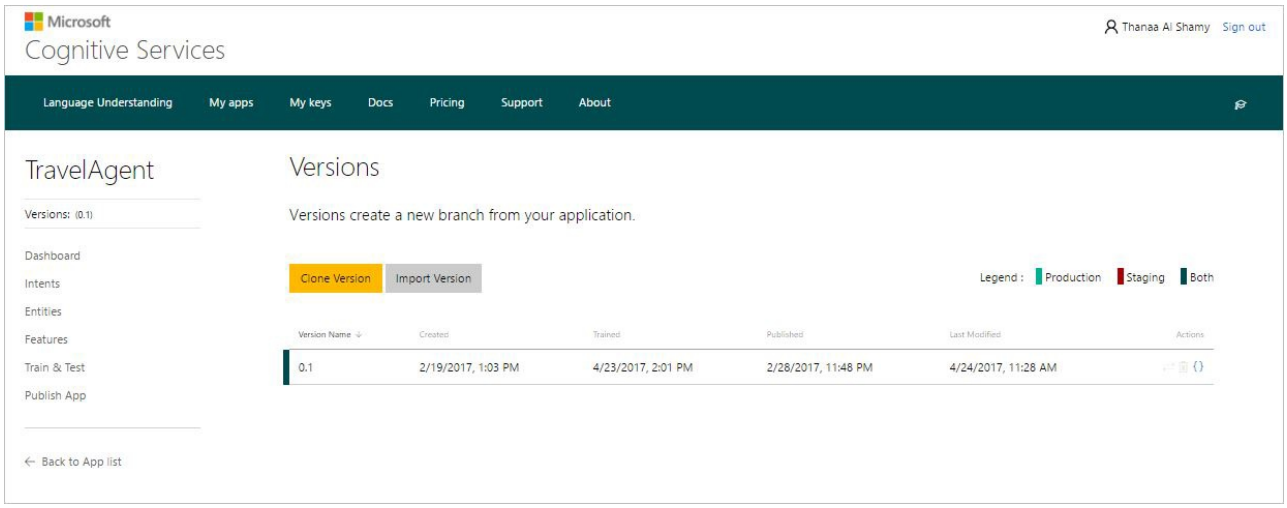
When you have created your key, you can add it to your account via the application settings/configuration dialog, found in any application.

Manage Versions

6/27/2017 • 2 min to read • Edit Online

You can create and manage different versions of your applications. When an app is first created, the default initial version is (0.1).

To work with versions, open your app (e.g. TravelAgent app) by clicking its name on **My Apps** page, and then click **Versions** in the application's left panel to access the **Versions** page.



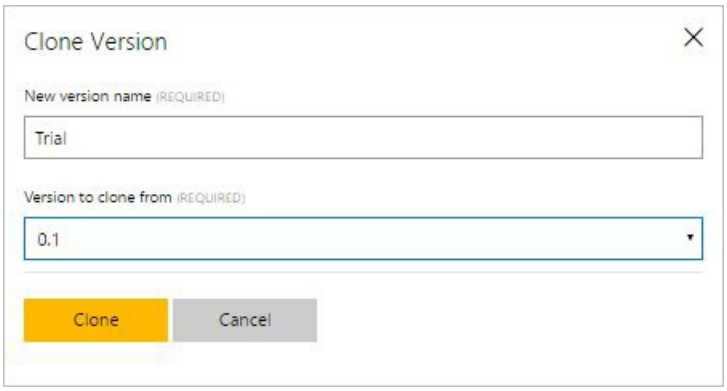
The following procedures will guide you through the actions you need to do while working with versions.

Clone a version

You can clone a version to create a copy of an existing version and save it as a new version. You may need to do this to use the same content of the existing version as a starting point for the new version and make updates to it, while keeping the source version unchanged.

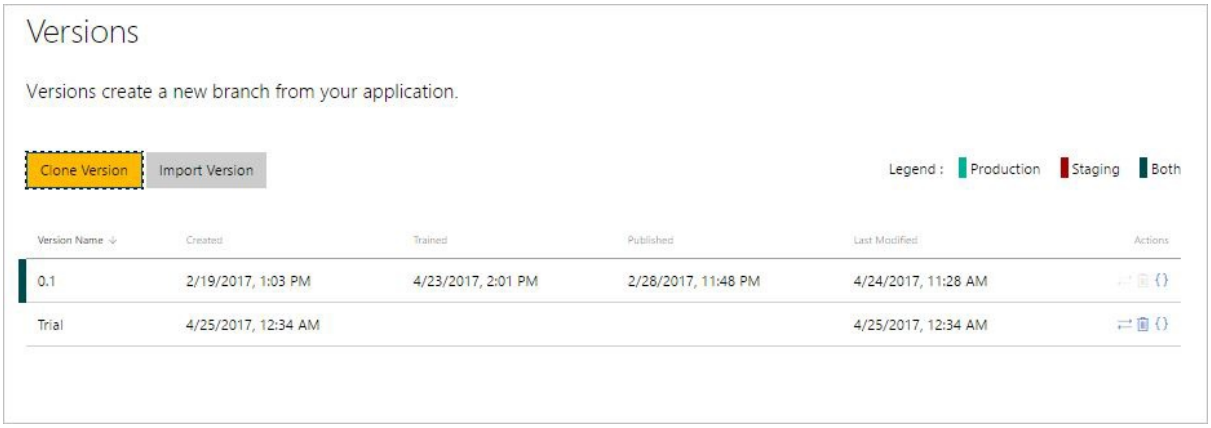
To clone a version:

1. On the **Versions** page, click **Clone Version**.
2. In the **Clone Version** dialog box, type a name for the new version in the **New Version Name** box (e.g. Trial), select the source version from **Version to clone from**, and then click **Clone**.



> **NOTE**
> Version name cannot be longer than 10 alphanumeric characters.

A new version with the specified name will be created and added to the list.



NOTE

As shown in the above screenshot, a published version is associated with a colored mark, indicating the slot where it has been published: Production slot (green), Staging slot (red) and Both slots (black). Also, the training and publishing dates will be displayed for each published version.

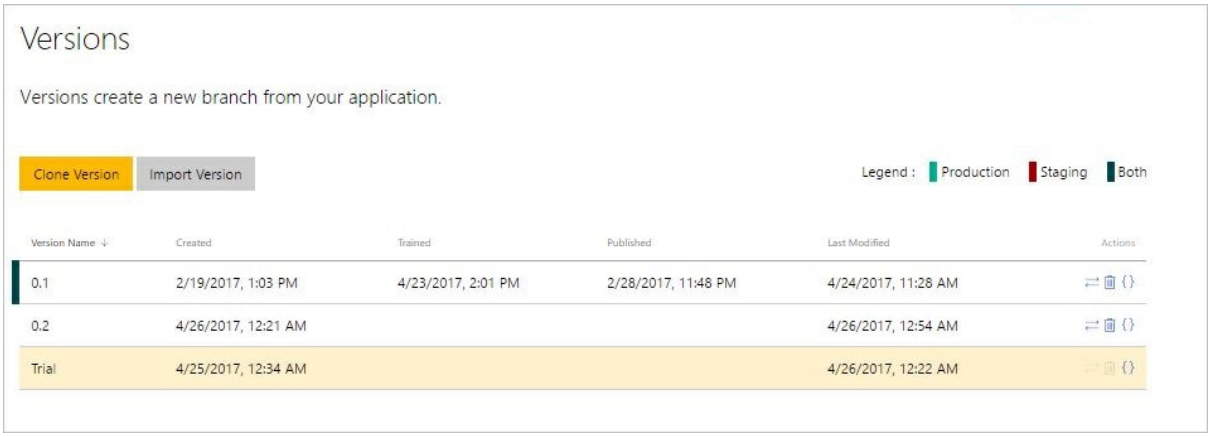
Set a version as active

To set a version as active means to make it the current version to work on and edit. You'll need to set a version as active to access its data, make updates, as well as to test and publish it.

The initial version (0.1) is the default active version unless you set another version as active. The name of the currently active version is displayed in the left panel under the app name.

To set a version as active:

1. On the **Versions** page, click the **Set as Active Version** button , corresponding to the version you want to set as active.
2. In the confirmation message, click **Yes** to confirm this action. The version will be set as active. The active version is highlighted by a light pink color, as shown in the following screenshot.



Import/export a version

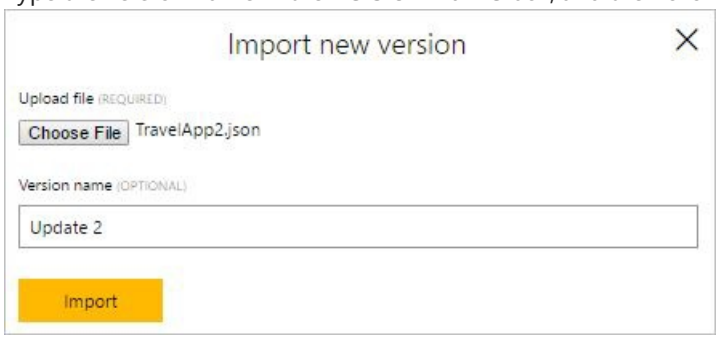
You can import/export a version as a JSON file.

To import a version:

1. On the **Versions** page, click **Import Version**.
2. In the **Import New Version** dialog box, click **Choose File** to choose the JSON file of the version you want to

import.

3. Type the version name in the **Version Name** box, and then click **Import**.



To export a version:

1. On the **Versions** page, click the Export Version button {}, corresponding to the version you want to export.
2. In the **Save As** dialog box, choose the location where you want to save the exported app version, and click **Save**.

Delete a version

You can delete versions, but you have to keep at least one version of the app. You can delete all versions except the active version.

To delete a version:

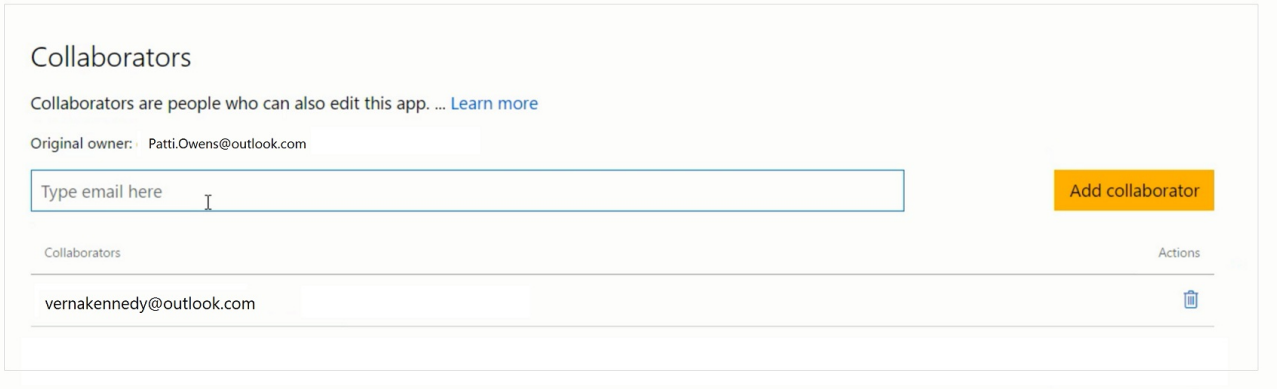
1. Click the **Delete Version** button , corresponding to the version you want to delete.
2. In the confirmation message, click **Yes** to confirm deletion.

Collaborate with other contributors on Language Understanding Intelligent Service (LUIS) apps

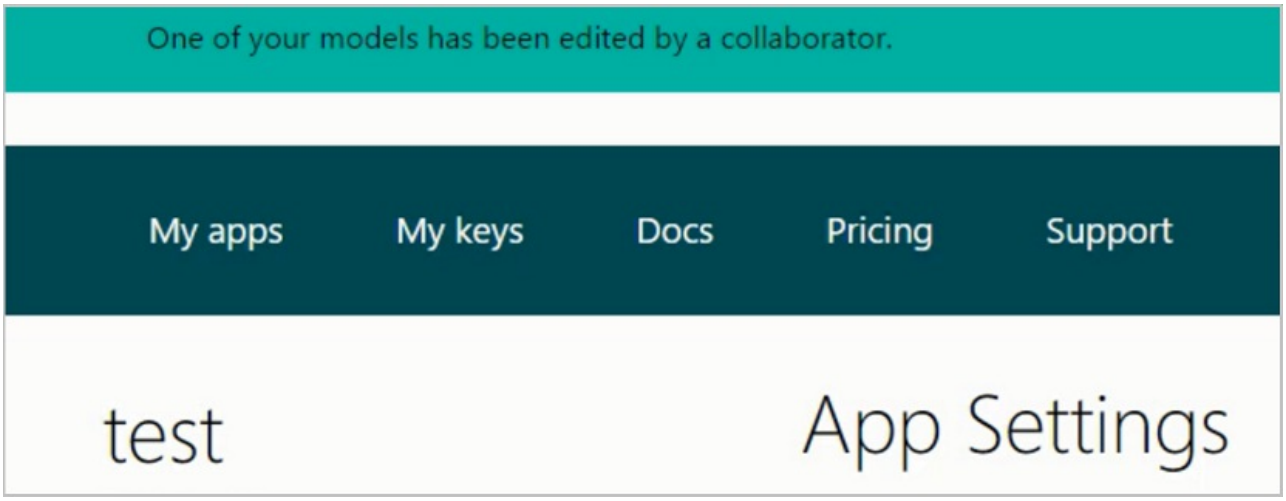
6/27/2017 • 1 min to read • Edit Online

You can collaborate with others and work on your LUIS app at the same time.

To allow collaborators to edit your LUIS app, in the **Collaborators** section of in your **My Apps** page, enter the email of the collaborator and click **Add collaborator**.



- Collaborators can sign in and edit your LUIS app as the same time as you. If a collaborator edits the LUIS app by adding an intent or entity or labeling an utterance, you'll see a notication.



NOTE

Collaborators cannot add other collaborators.

Language Understanding Intelligent Services (LUIS) Frequently Asked Questions

7/19/2017 • 6 min to read • Edit Online

This article contains answers to some frequently asked questions about LUIS.

I want to build a LUIS app with more than 80 intents. What should I do?

First, consider whether your system is using too many intents. Intents that are too similar can make it more difficult for LUIS to distinguish between them. Intents should be varied enough to capture the main tasks the user is asking for, but they don't need to capture every path your code takes. For example, BookFlight and BookHotel may be separate intents in a travel app, but BookInternationalFlight and BookDomesticFlight are too similar, and if your system needs to distinguish them, use entities or other logic rather than intents.

If you cannot use fewer intents, divide your intents into multiple LUIS apps, by grouping together related intents. One best practice is to group related intents together if you're using multiple apps for your system. For example, if you're developing an office assistant that has over 80 intents, but 20 intents relate to scheduling meetings, 20 intents are about reminders, 20 intents are about getting information about colleagues, and 20 intents are for sending email, you can put the intent for each of those categories in a separate LUIS app.

When your system receives an utterance, you can use a variety of techniques to determine how to direct user utterances to LUIS apps:

- Create a top-level LUIS app to determine the category of utterance, and then use the result to send the utterance to the LUIS app for that category.
- Do some preprocessing on the utterance, like matching on regular expressions, to determine which LUIS app or set of apps receives it.

Consider the following tradeoffs when deciding which approach you use with multiple LUIS apps:

- **Saving suggested utterances for training** Your LUIS apps get a performance boost when you label the user utterances it receives, especially the suggested utterances that LUIS is relatively unsure of. Any LUIS app that doesn't receive an utterance won't have the benefit of learning from it.
- **Calling LUIS apps in parallel instead of in series** It is a common to design a system to reduce to the number of REST API calls that happen in series to improve responsiveness. If you send the utterance to multiple LUIS apps and pick the intent with the highest score, you can call them in parallel by sending all the requests asynchronously. If you call a top-level LUIS app to determine a category, and then use the result to send the utterance to another LUIS app, the LUIS calls happen in series.

I want to build an app in LUIS with more than 30 entities. What should I do?

You might need to use hierarchical and composite entities. Hierarchical entities reflect the relationship between entities that share characteristics or are members of a category. The child entities are all members of their parent's category. For example, a hierarchical entity named PlaneTicketClass may have the child entities EconomyClass and FirstClass. The hierarchy spans only one level of depth.

Composite entities represent parts of a whole. For example, a composite entity named PlaneTicketOrder may have child entities Airline, Destination, DepartureCity, DepartureDate, and PlaneTicketClass. You build a composite entity from pre-existing simple entities, children of hierarchical entities or prebuilt entities. LUIS is limited to 10 parent

entities with up to 10 children for each parent entity (composite or hierarchical).

LUIS also provides the list entity type that is not machine learned but allows users to specify a fixed set entities with a given set of values.

What is the best way to start on building my app in LUIS?

The best way to build you app is through an incremental process. You could start by defining the schema of your app (intents and entities). For every intent and entity model, you can provide a few dozen labels. Train and publish your app to get an endpoint. Then, upload 100-200 unlabeled utterances to your app. You can use the suggestion feature to leverage LUIS intelligence in selecting the most informative utterances to label. You can select the intent or entity you want to improve and label the utterances suggested by LUIS. Labeling a few hundred utterance should result in a decent accuracy for intents. Entities might need more examples to converge.

What is a good practice to model the intents of my app? Should I create more specific or more generic intents?

Choose intents that are not so general as to be overlapping, but not so specific that it makes it difficult for LUIS to distinguish between similar intents. Creating discriminative specific intents is one of LUIS modeling best practices.

Is it important to train the None intent?

Yes, it is good that to train your **None** intent with more utterances as you add more labels to other intents. A good ratio is to add 1 or 2 labels to **None** for every 10 labels added to an intent. This boosts the discriminative power of LUIS.

How can I deal with spelling mistakes in utterances?

You have one of two options:

1. Pass your utterances by a spell checker before sending them to the LUIS endpoint.
2. Label utterances that have spelling mistakes that are as diverse as possible, so that LUIS can learn proper spelling as well as typos.

The second option takes more labeling effort while the first might be easier.

I see some errors in the batch testing pane for some of the models in my app. How can I address this problem?

This is an indication that there is some discrepancy between your labels and the predictions from your models. You need to do one or both of the following:

1. Add more labels to help LUIS make the discrimination among intents better.
2. Add phrase list feature(s) to introduce domain-specific vocabulary to help LUIS learn faster.

I have an app in one language and would like to create a parallel app in another language. What is the easiest way to do so?

1. Export your app.
2. Translate the labeled utterances in the JSON of the exported app to the target language.
3. You might need to change the names of the intents and entities or leave them as they are.
4. Import the app afterwards to have an LUIS app in the target language

How can I delete data from LUIS?

- If you delete an utterance from your LUIS app, it is removed from the LUIS web service and not available for export.
- If you delete an account, all apps and their utterances are deleted. Data is retained on the servers for 60 days before permanent deletion.
- You can turn off the logging of user utterances by setting `log=false` in the URL when your client application queries LUIS. However, note that this will disable your LUIS app's ability to suggest utterances or improve performance based on user queries. If you set `log=false` due to data privacy concerns be aware that you won't be able to download a record of user utterances from LUIS or use those utterances to improve your app.

Next steps

- You can find many answers in the Stack Overflow questions tagged with LUIS.
- Another resource is the MSDN LUIS Forum