Table of Contents

LUIS FAQ

Pricing

Stack Overflow

Learn about Language Understanding Intelligent Service (LUIS)

11/3/2017 • 4 min to read • Edit Online

Language Understanding Intelligent Service (LUIS) allows your application to understand what a person wants in their own words. LUIS uses machine learning to allow developers to build applications that can receive user input in natural language and extract meaning from it. A client application that converses with the user can pass user input to a LUIS app and receive relevant, detailed information back.

Several Microsoft technologies work with LUIS:

- Bot Framework allows a chat bot to talk with a user via text input.
- Bing Speech API converts spoken language requests into text. Once converted to text, LUIS processes the requests.

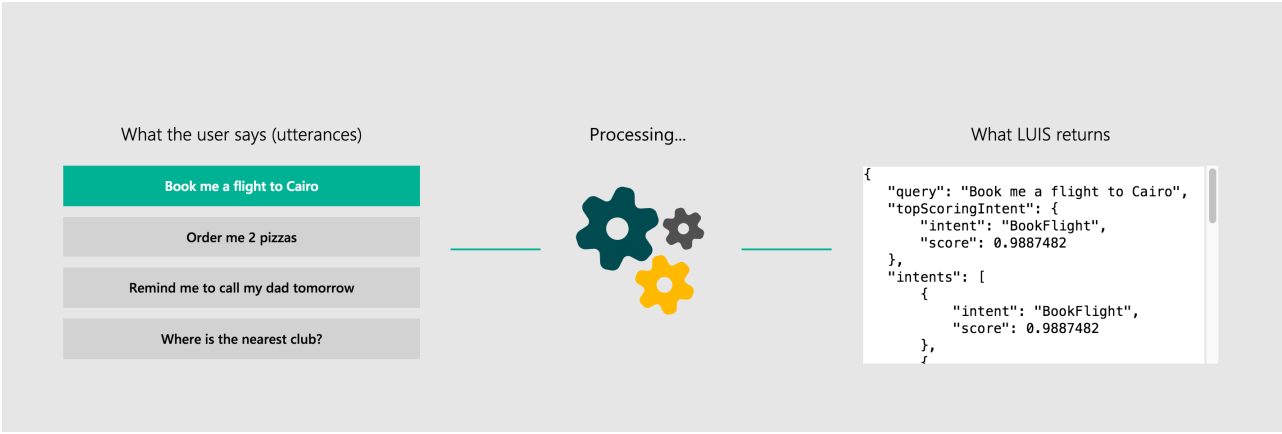
What is a LUIS app?

A LUIS app is a domain-specific language model designed by you and tailored to your needs. You can start with a pre-built domain model, build your own, or blend pieces of a pre-built domain with your own custom information.

A model starts with a list of general user intentions such as "Book Flight" or "Contact Help Desk." Once the intentions are identified, you supply example phrases called utterances for the intents. Then you label the utterances with any specific details you want LUIS to pull out of the utterance.

Pre-built domain models include all these pieces for you and are a great way to start using LUIS quickly.

After the model is designed, trained, and published, it is ready to receive and process utterances. The LUIS app receives the utterance as an HTTP request and responds with extracted user intentions. Your client application sends the utterance and receives LUIS's evaluation as a JSON object. Your app can then take appropriate action.



Key LUIS concepts

- **Intents** An intent represents actions the user wants to perform. The intent 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 and name intents that correspond to these actions. A travel app may define an intent named "BookFlight."
- **Utterances** An utterance is text input from the user that your app needs to understand. It may be a sentence, like "Book 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.

- **Entities** An entity represents detailed information that is relevant in the utterance. For example, in the utterance "Book a ticket to Paris", "Paris" is a location. By recognizing and labeling the entities that are mentioned in the user's utterance, LUIS helps you choose the specific action to take to answer a user's request.

INTENT	SAMPLE USER UTTERANCE	ENTITIES
BookFlight	"Book a flight to **Seattle**?"	Seattle
StoreHoursAndLocation	"When does your store **open**?"	open
ScheduleMeeting	"Schedule a meeting at **1pm** with **Bob** in Distribution"	1pm, Bob

Accessing LUIS

LUIS has two ways to build a model: the Authoring APIs and the LUIS.ai web app. Both methods give you and your collaborators control of your LUIS model definition. You can use either LUIS.ai or the Authoring APIs or a combination of both to build your model. This includes management of: models, versions, collaborators, external APIs, testing, and training.

Once your model is built and published, you pass the utterance to LUIS and receive the JSON object results with the Endpoint APIs.

> **NOTE**
> - The Authoring APIs and LUIS.ai use the programmatic key found in your LUIS.ai account page.
> - The Endpoint APIs use the LUIS subscription key found in the Azure portal.

Author your LUIS model

Begin your LUIS model with the intents your app will resolve. Intents are just names such as "BookFlight" or "OrderPizza."

After an intent is identified, you need sample utterances that you want LUIS to map to your intent such as "Buy a ticket to Seattle tomorrow." Then, label the parts of the utterance that are relevant to your app domain as entities and set a type such as date or location.

Generally, an **intent** is used to trigger an action and an **entity** is used as a parameter 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.

Identify Entities

Entity identification determines how successfully the end user gets the correct answer. LUIS provides several ways to identify and categorize entities.

- **Prebuilt Entities** LUIS has many prebuilt domain models which include intents, utterances, and prebuilt entities. You can use the prebuilt entities without having to use the intents and utterances of the prebuilt model. The prebuilt entities save you time.
- **Custom Entities** LUIS gives you several ways to identify your own custom entities including simple entities, composite entities, list entities, and hierarchical entities.
- **Regular Expressions and Phrase Features** LUIS provides features such as regular expression patterns and

phrase lists, which also help identify entities.

Improve performance using active learning

Once your application is published and real user utterances are entered, LUIS uses active learning to improve identification. In the active learning process, LUIS identifies the utterances that it is relatively unsure of. You can label them according to intent and entities, retrain, and republish.

This iterative process has tremendous advantages. LUIS knows what it is unsure of, and your help leads to the maximum improvement in system performance. LUIS learns quicker, and takes the minimum amount of your time and effort. LUIS is an active machine learning at its best.

Next steps

Create a new LUIS app.

Watch this short video tutorial on these steps.

Localization support in LUIS apps

11/9/2017 • 2 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.

LANGUAGE	LOCALE	PREBUILT ENTITY SUPPORT	NOTES
American English	`en-US`	✔	
Canadian French	`fr-CA`	-	
French (France)	`fr-FR`	✔	Support for the **datetimeV2** prebuilt entity is expected in late November 2017.
Italian	`it-IT`	✔	
Dutch	`nl-NL`	-	
German	`de-DE`	✔	Support for the **datetimeV2** prebuilt entity is expected in late December 2017.
Spanish (Spain)	`es-ES`	✔	
Spanish (Mexico)	`es-MX`	-	
Portuguese (Brazil)	`pt-BR`	✔	Support for the **datetimeV2** prebuilt entity is expected in late December 2017.
Japanese	`ja-JP`	✔	
Korean	`ko-KR`	-	
Chinese	`zh-CN`	✔	See support notes listed below

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.

Create your first LUIS app

11/6/2017 • 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.

This article shows you how to create a LUIS app that uses the Home.Automation prebuilt domain. The prebuilt domain provides intents and entities for a home automation system for turning lights and appliances on and off.

Before you begin

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

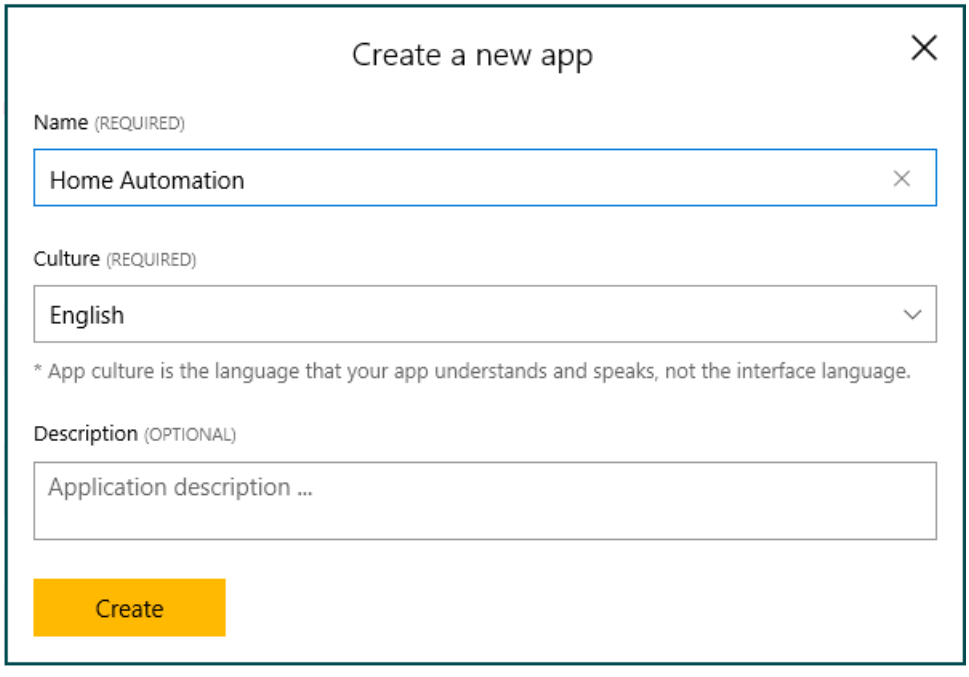
If you don't have an Azure subscription, create a free account before you begin.

For this article, you also need a LUIS.ai account in order to author your LUIS application.

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 LUIS web page.

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

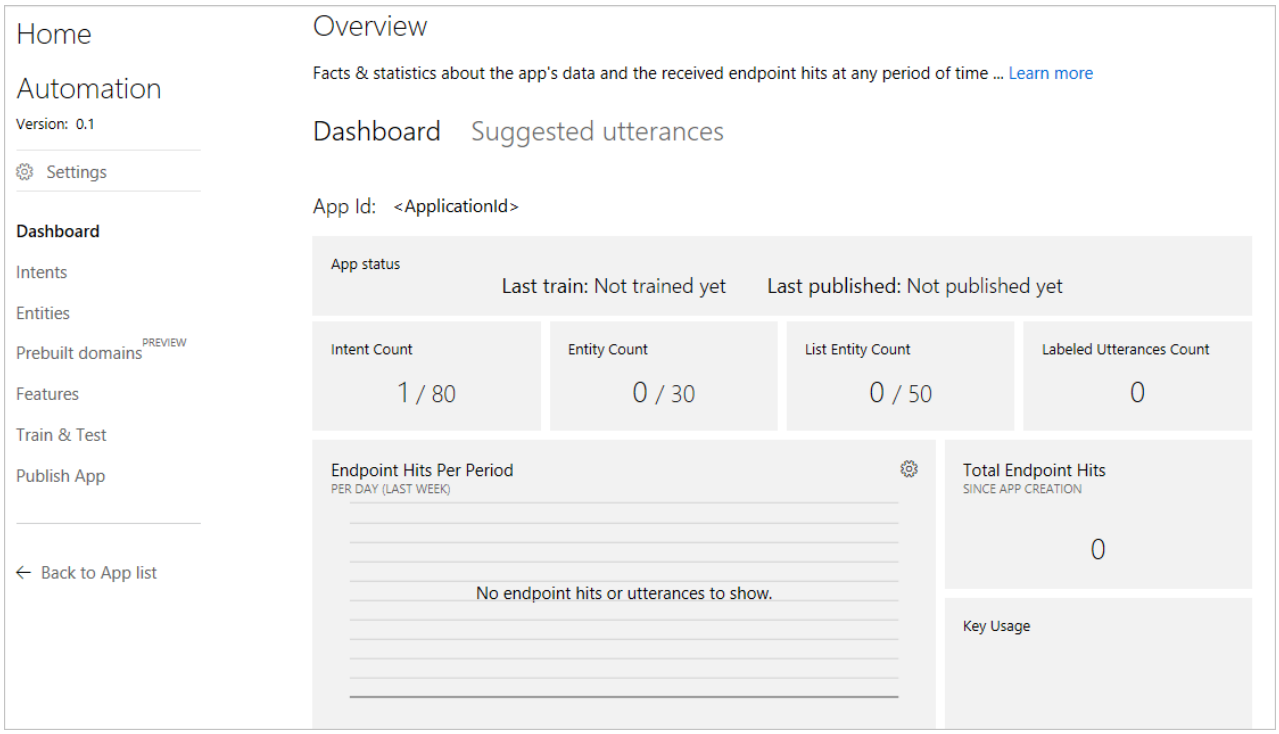


3. Choose your application culture (for this Home Automation app, we'll choose English), and then click **Create**.

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

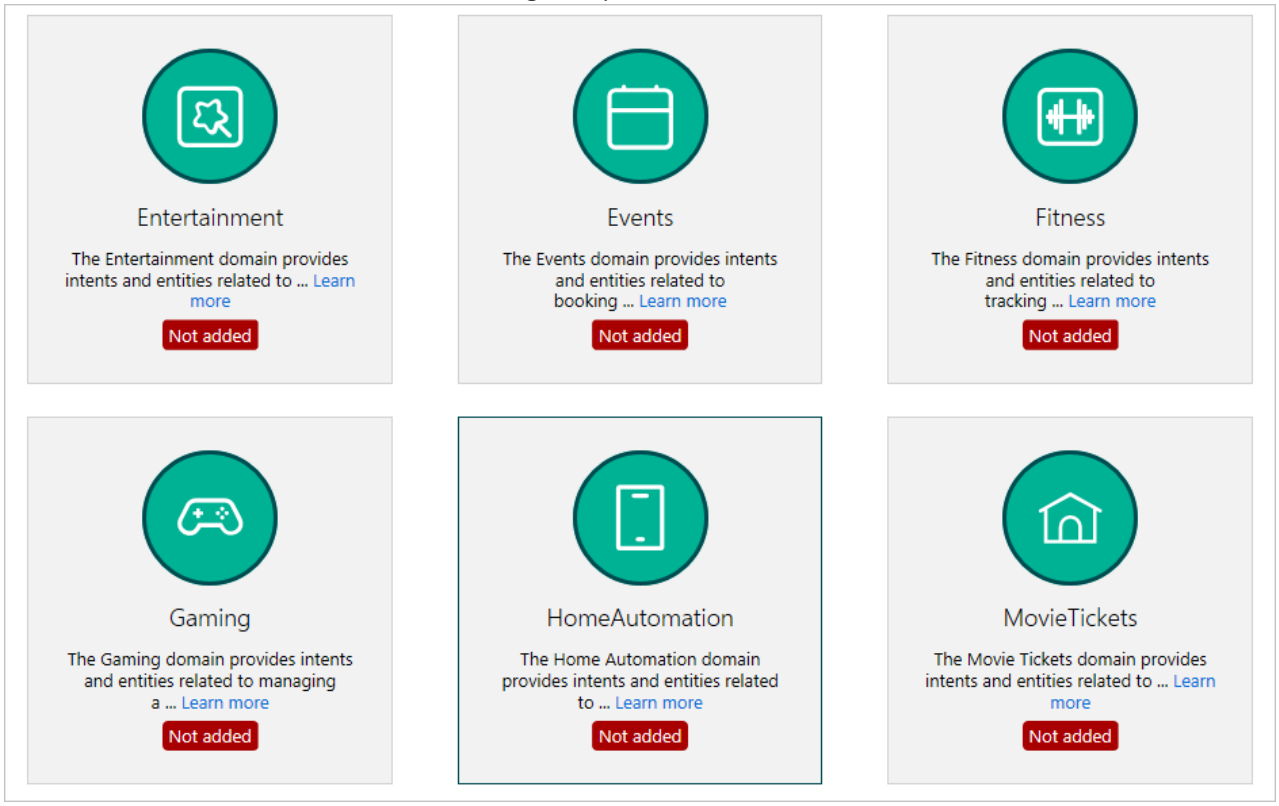
LUIS creates the Home Automation app and opens to the Dashboard. The application dashboard contains summary information about app usage.

You can explore your application using the links in the left panel.

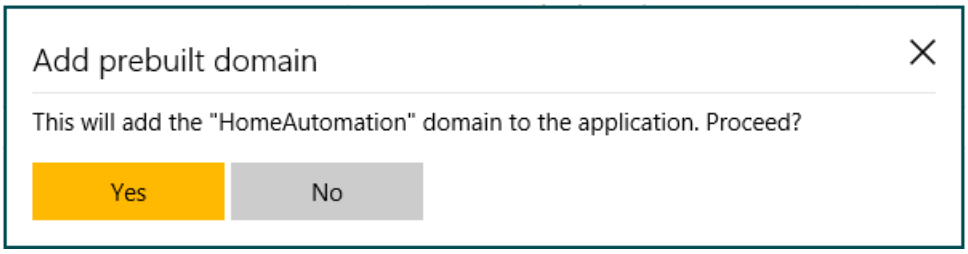


Add the Home Automation prebuilt domain

Click on **Prebuilt domains** in the left-side navigation pane. Then click on **HomeAutomation**.



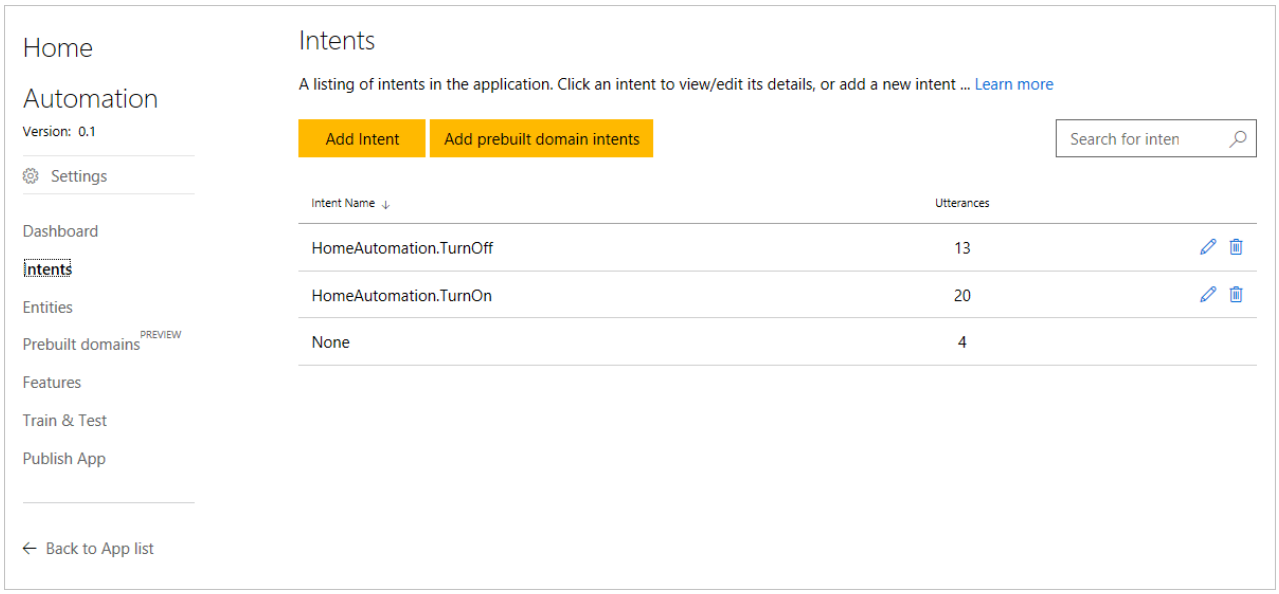
Click **Yes** when prompted to add the "HomeAutomation" domain to the app.



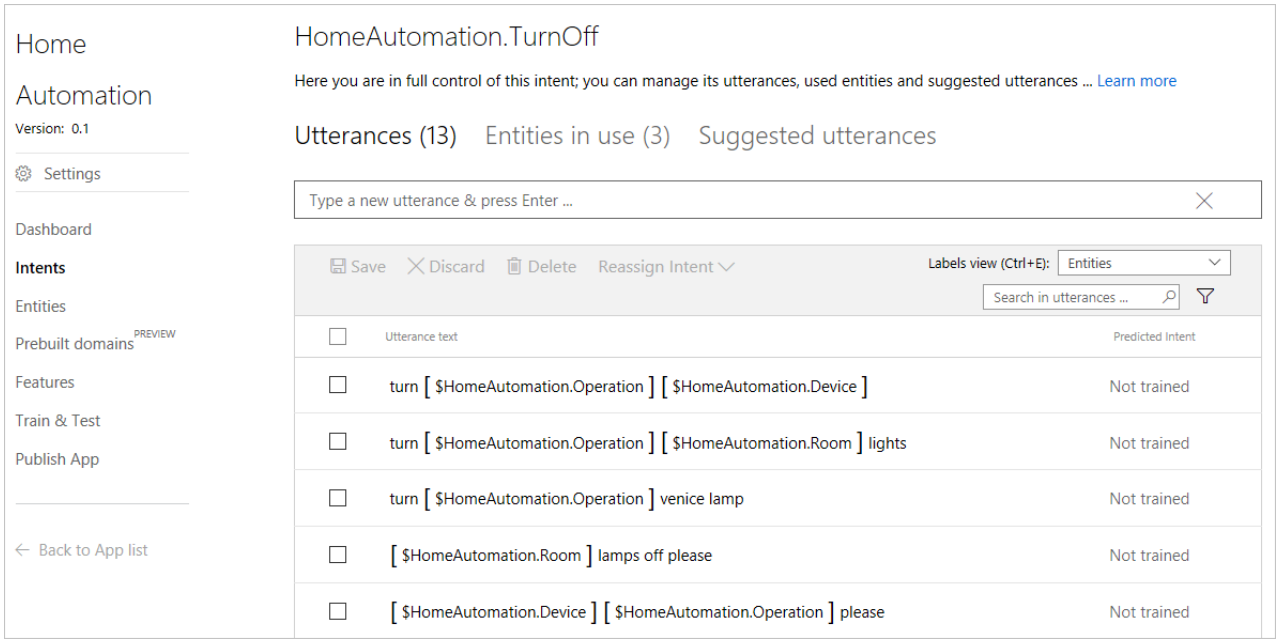
Take a look at the intents and entities

Click on **Intents** in the left-side navigation pane, and you can see that the HomeAutomation domain provides **HomeAutomation.TurnOff**, **HomeAutomation.TurnOn**, and **None** intents in your application. Each intent has sample utterances.

> **NOTE**
>
> **None** is an intent provided by all LUIS apps. You use it to handle utterances that don't correspond to functionality your app provides.



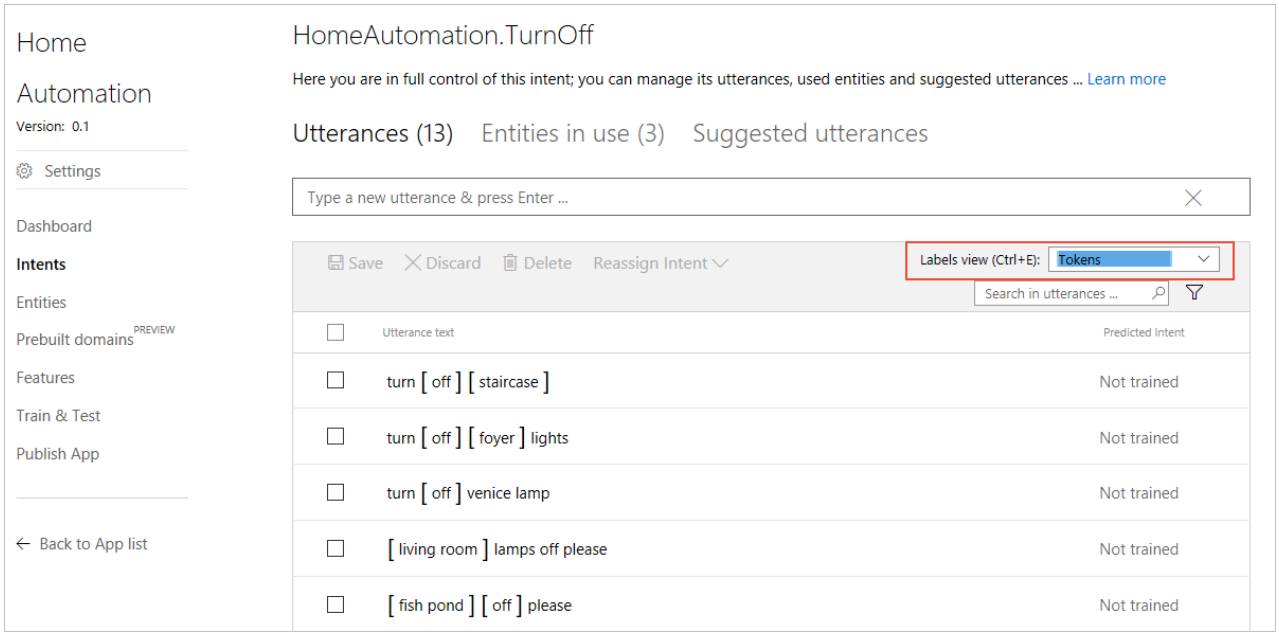
Click on the **HomeAutomation.TurnOff** intent. You can see that the intent contains a list of utterances which are labeled with entities.



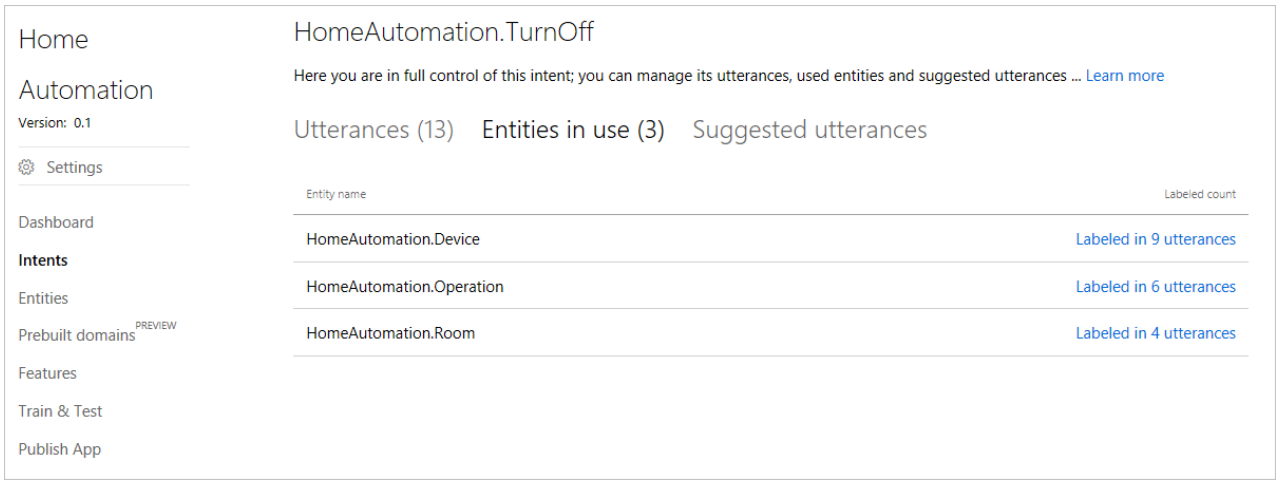
Click on the **Labels view** and select **tokens**. This shows the text tokens that make up each labeled entity, instead of the name of the entity type.

If you compare the same utterance in the tokens view and the entities view, you can see that some of the words of each utterance have already been labeled.

The first utterance is "turn off staircase." The word "off" has been labeled as the type of HomeAutomation.Operation. The word "staircase" has been labeled as the type of "HomeAutomation.Device."

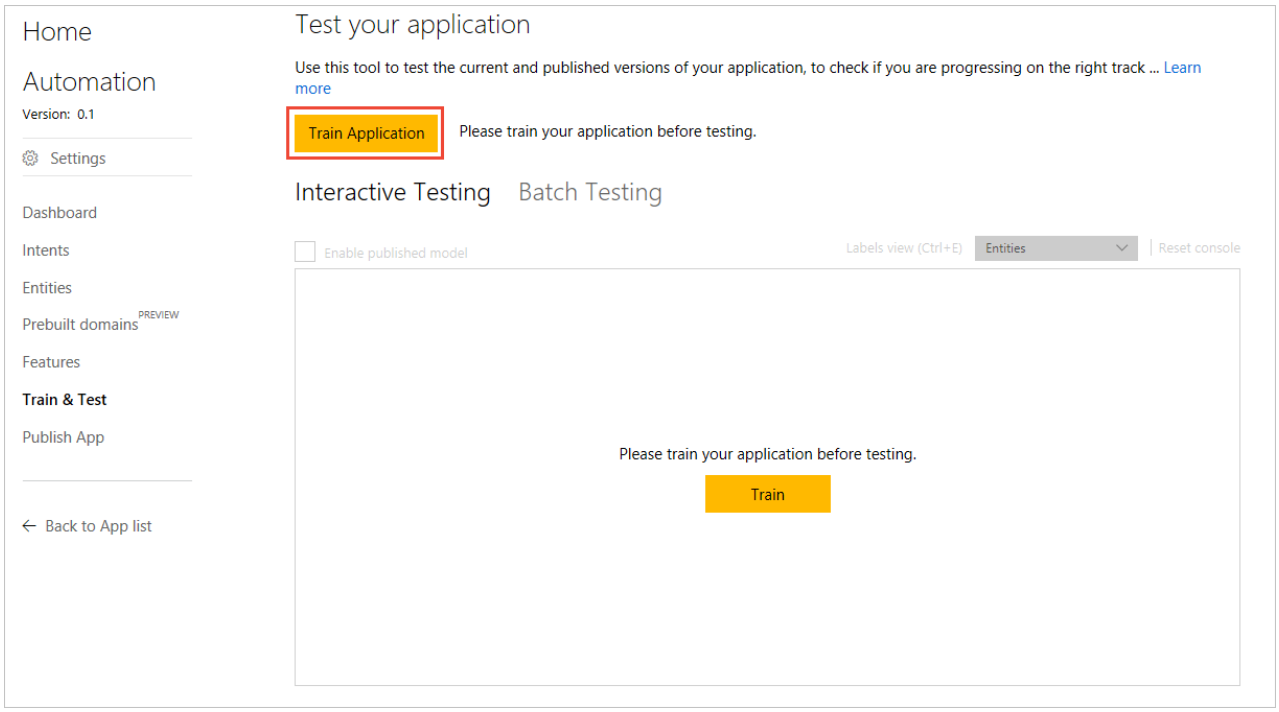


Click **Entities in use**. This shows the entities this app identifies in the utterances.



Train your app

Click on **Train & Test** in the left-side navigation, then click **Train application**.



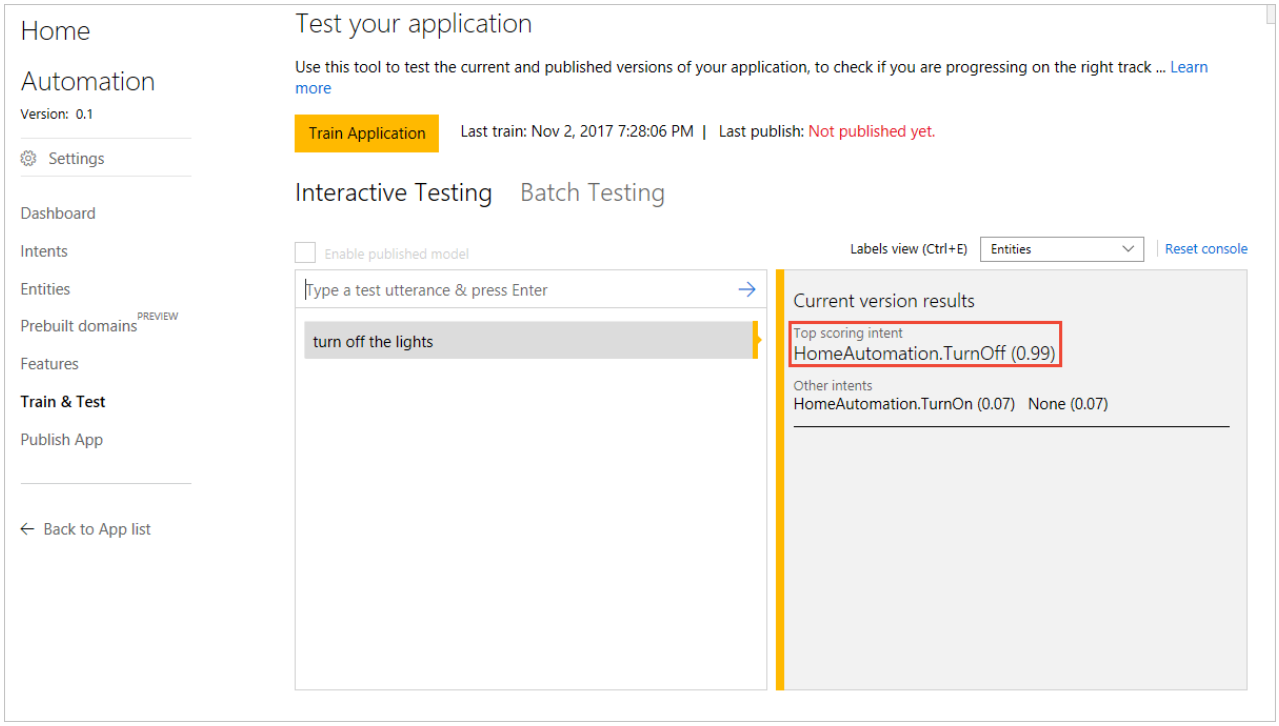
Test your app

Once you've trained your app, you can test it. Type a test utterance like "Turn off the lights" into the Interactive Testing pane, and press Enter.

```
Turn off the lights
```

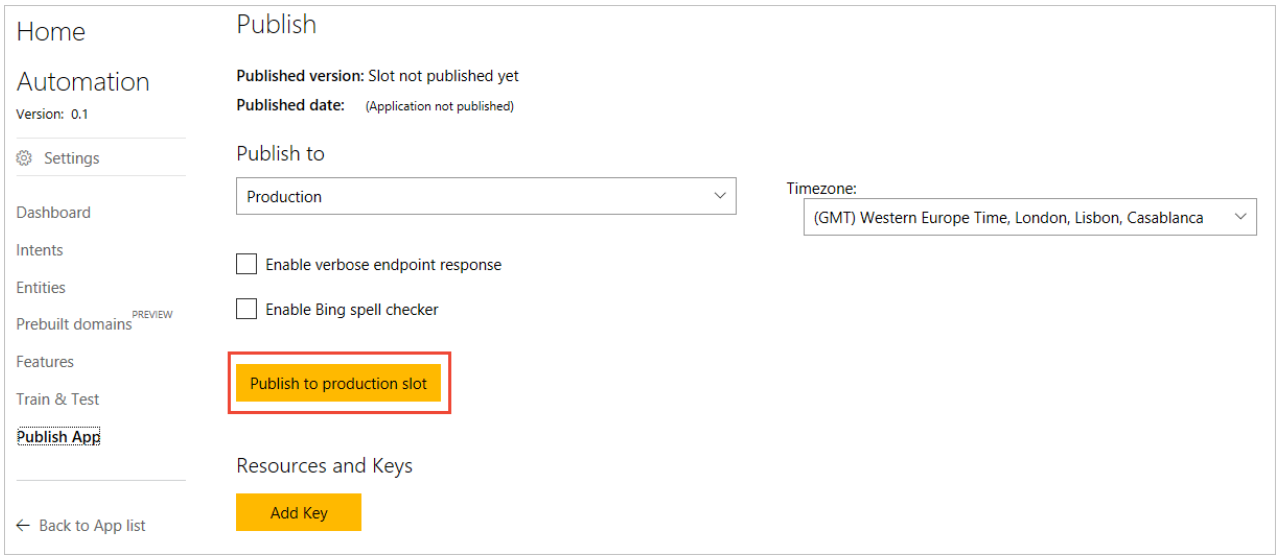
The results display the score associated with each intent. Check that the top scoring intent corresponds to the intent you expected for each test utterance.

In this example, "Turn off the lights" is correctly identified as the top scoring intent of "HomeAutomation.TurnOff."

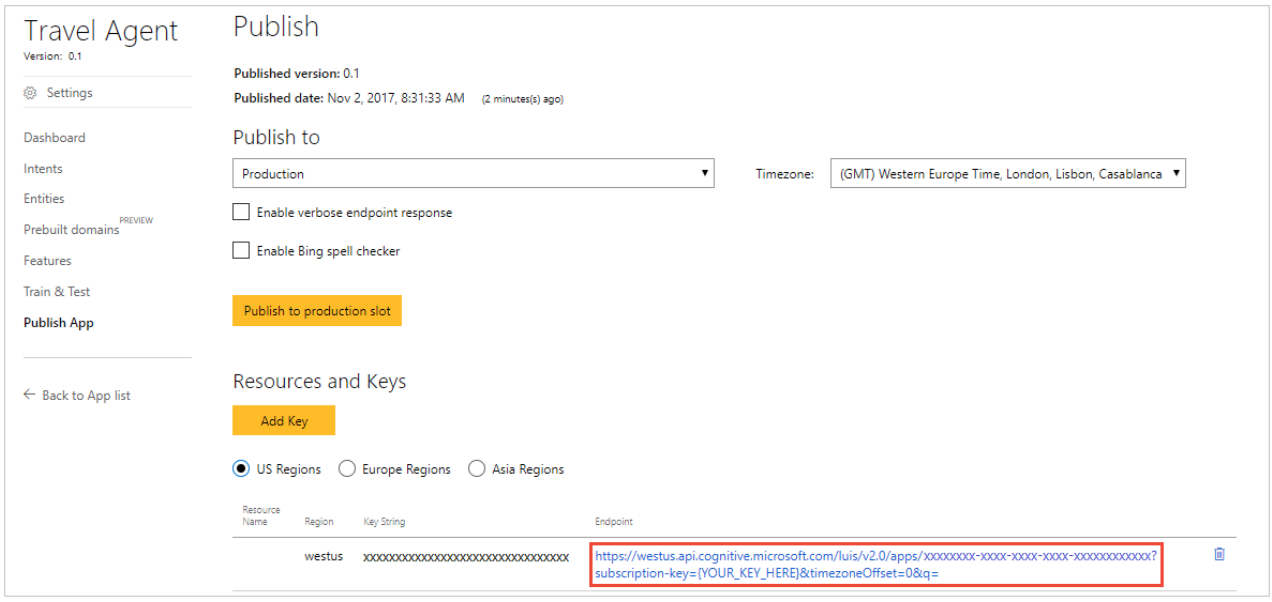


Publish your app

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



After you've successfully published, you can use the Endpoint URL that the **Publish App** page displays.



Use your app

You can test your published endpoint in a browser using the generated URL. Copy the URL, then replace the `{YOUR-KEY-HERE}` with one of the keys listed in the **Key String** column for the resource you want to use. To open this URL in your browser, set the URL parameter "&q" to your test query. For example, append `&q=turn off the living room light` to your URL, and then press Enter. The browser displays the JSON response of your HTTP endpoint.

Next steps

You can call the endpoint from code:

[Call a LUIS endpoint using code]

Call a LUIS app using C#

11/3/2017 • 2 min to read • Edit Online

This quickstart shows you how to call your Language Understanding Intelligent Service (LUIS) app in just a few minutes. When you're finished, you'll be able to use C# code to pass utterances to a LUIS endpoint and get results.

Before you begin

You need a Cognitive Services API key to make calls to the sample LUIS app we use in this walkthrough. To get an API key follow these steps:

1. 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.
2. Log in to the Azure portal at https://portal.azure.com.
3. Follow the steps in Creating Subscription Keys using Azure to get a key.
4. Go back to https://www.luis.ai and log in using your Azure account.

Understand what LUIS returns

To understand what a LUIS app returns, you can paste the URL of a sample LUIS app into a browser window. The sample app you'll use is an IoT app that detects whether the user wants to turn on or turn off lights.

1. The endpoint of the sample app is in this format:
   `https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951ca1c2?subscription-key=<YOUR_API_KEY>&verbose=false&q=turn%20on%20the%20bedroom%20light`
   Copy the URL and substitute your subscription key for the value of the `subscription-key` field.
2. Paste the URL into a browser window and press Enter. The browser displays a JSON result that indicates that LUIS detects the `HomeAutomation.TurnOn` intent and the `HomeAutomation.Room` entity with the value `bedroom`.

3. Change the value of the `q=` parameter in the URL to `turn off the living room light`, and press Enter. The result now indicates that the LUIS detected the `HomeAutomation.TurnOff` intent and the `HomeAutomation.Room` entity with value `living room`.

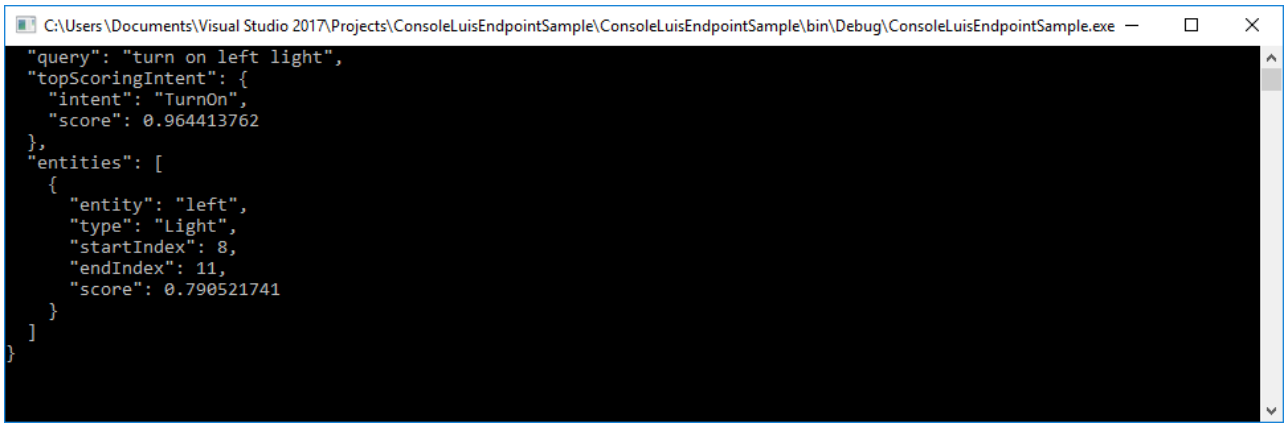
Consume a LUIS result using the Endpoint API with C#

You can use C# to access the same results you saw in the browser window in the previous step.

1. Create a new console application in Visual Studio. Copy the code that follows and save it into an .cs file:

```
using System;
using System.Net.Http;
using System.Web;

namespace ConsoleLuisEndpointSample
{
    class Program
    {
        static void Main(string[] args)
        {
            MakeRequest();
            Console.WriteLine("Hit ENTER to exit...");
            Console.ReadLine();
        }

        static async void MakeRequest()
        {
            var client = new HttpClient();
            var queryString = HttpUtility.ParseQueryString(string.Empty);

            // This app ID is for a public sample app that recognizes requests to turn on and turn off
lights
            var luisAppId = "df67dcdb-c37d-46af-88e1-8b97951ca1c2";
            var subscriptionKey = "YOUR_SUBSCRIPTION_KEY";

            // The request header contains your subscription key
            client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", subscriptionKey);

            // The "q" parameter contains the utterance to send to LUIS
            queryString["q"] = "turn on the left light";

            // These optional request parameters are set to their default values
            queryString["timezoneOffset"] = "0";
            queryString["verbose"] = "false";
            queryString["spellCheck"] = "false";
            queryString["staging"] = "false";

            var uri = "https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/" + luisAppId + "?" +
queryString;
            var response = await client.GetAsync(uri);

            var strResponseContent = await response.Content.ReadAsStringAsync();

            // Display the JSON result from LUIS
            Console.WriteLine(strResponseContent.ToString());
        }
    }
}
```

2. Replace the value of the `subscriptionKey` variable with your LUIS subscription key.
3. In the Visual Studio project, add a reference to **System.Web**.
4. Run the console application. It displays the same JSON that you saw earlier in the browser window.



Next steps

- See the LUIS Endpoint API reference to learn more about the parameters for calling your LUIS endpoint.

Call a LUIS app using Java

11/3/2017 • 2 min to read • Edit Online

This quickstart shows you how to call your Language Understanding Intelligent Service (LUIS) app in just a few minutes. When you're finished, you'll be able to use Java code to pass utterances to a LUIS endpoint and get results.

Before you begin

You need a Cognitive Services API key to make calls to the sample LUIS app we use in this walkthrough. To get an API key follow these steps:

1. 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.
2. Log in to the Azure portal at https://portal.azure.com.
3. Follow the steps in Creating Subscription Keys using Azure to get a key.
4. Go back to https://www.luis.ai and log in using your Azure account.

Understand what LUIS returns

To understand what a LUIS app returns, you can paste the URL of a sample LUIS app into a browser window. The sample app you'll use is an IoT app that detects whether the user wants to turn on or turn off lights.

1. The endpoint of the sample app is in this format:
   ```
   https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951ca1c2?subscription-key=<YOUR_API_KEY>&verbose=false&q=turn%20on%20the%20bedroom%20light
   ```
   Copy the URL and substitute your subscription key for the value of the `subscription-key` field.
2. Paste the URL into a browser window and press Enter. The browser displays a JSON result that indicates that LUIS detects the `HomeAutomation.TurnOn` intent and the `HomeAutomation.Room` entity with the value `bedroom`.

3. Change the value of the `q=` parameter in the URL to `turn off the living room light`, and press enter. The result now indicates that the LUIS detected the `HomeAutomation.TurnOff` intent and the `HomeAutomation.Room` entity with value `living room`.

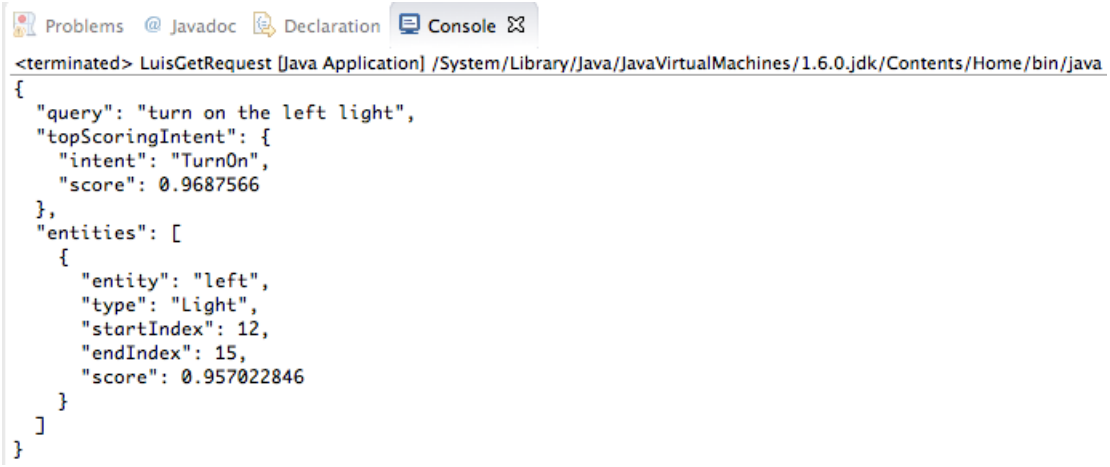
Consume a LUIS result using the Endpoint API with Java

You can use Java to access the same results you saw in the browser window in the previous step.

1. Copy the following code to create a class in your IDE:

```
// This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/httpcomponents-
client-ga/)
import java.net.URI;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class LuisGetRequest {

    public static void main(String[] args)
    {
        HttpClient httpclient = HttpClients.createDefault();

        try
        {

            // The ID of a public sample LUIS app that recognizes intents for turning on and off lights
            String AppId = "df67dcdb-c37d-46af-88e1-8b97951ca1c2";
            // Add your subscription key
            String SubscriptionKey = "YOUR-SUBSCRIPTION-KEY";

        URIBuilder builder =
            new URIBuilder("https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/" + AppId + "?");

            builder.setParameter("q", "turn on the left light");
            builder.setParameter("timezoneOffset", "0");
            builder.setParameter("verbose", "false");
            builder.setParameter("spellCheck", "false");
            builder.setParameter("staging", "false");

            URI uri = builder.build();
            HttpGet request = new HttpGet(uri);
            request.setHeader("Ocp-Apim-Subscription-Key", SubscriptionKey);

            HttpResponse response = httpclient.execute(request);
            HttpEntity entity = response.getEntity();


            if (entity != null)
            {
                System.out.println(EntityUtils.toString(entity));
            }
        }

        catch (Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
}
```

2. Replace the value of the `SubscriptionKey` variable with your LUIS subscription key.
3. In your IDE, add references to `httpclient` and `httpcore` libraries.
4. Run the console application. It displays the same JSON that you saw earlier in the browser window.



Next steps

- See the LUIS Endpoint API reference to learn more about the parameters for calling your LUIS endpoint.

Call a LUIS app using JavaScript

11/3/2017 • 2 min to read • Edit Online

This quickstart shows you how to call your Language Understanding Intelligent Service (LUIS) app in just a few minutes. When you're finished, you'll be able to use JavaScript code to pass utterances to a LUIS endpoint and get results.

Before you begin

You need a Cognitive Services API key to make calls to the sample LUIS app we use in this walkthrough. To get an API key follow these steps:

1. 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.
2. Log in to the Azure portal at https://portal.azure.com.
3. Follow the steps in Creating Subscription Keys using Azure to get a key.
4. Go back to https://www.luis.ai and log in using your Azure account.

Understand what LUIS returns

To understand what a LUIS app returns, you can paste the URL of a sample LUIS app into a browser window. The sample app you'll use is an IoT app that detects whether the user wants to turn on or turn off lights.

1. The endpoint of the sample app is in this format:
   `https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951ca1c2?subscription-key=<YOUR_API_KEY>&verbose=false&q=turn%20on%20the%20bedroom%20light`
   . Copy the URL and substitute your subscription key for the value of the `subscription-key` field.
2. Paste the URL into a browser window and press Enter. The browser displays a JSON result that indicates that LUIS detects the `HomeAutomation.TurnOn` intent and the `HomeAutomation.Room` entity with the value `bedroom`.

3. Change the value of the `q=` parameter in the URL to `turn off the living room light`, and press enter. The result now indicates that the LUIS detected the `HomeAutomation.TurnOff` intent and the `HomeAutomation.Room` entity with value `living room`.

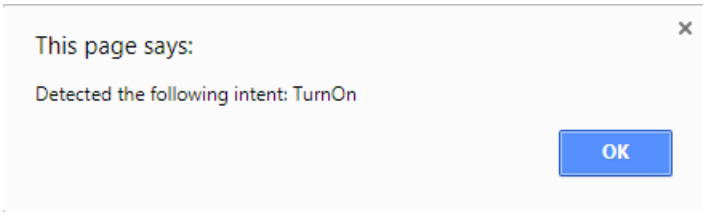
Consume a LUIS result using the Endpoint API with JavaScript

You can use JavaScript to access the same results you saw in the browser window in the previous step.

1. Copy the code that follows and save it into an HTML file:

```
<!DOCTYPE html>
<html>
<head>
    <title>JSSample</title>
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.0/jquery.min.js"></script>
</head>
<body>

<script type="text/javascript">
    $(function() {
        var params = {
            // These are optional request parameters. They are set to their default values.
            "timezoneOffset": "0",
            "verbose": "false",
            "spellCheck": "false",
            "staging": "false",
        };

        $.ajax({
            url: "https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-
8b97951ca1c2?q=turn%20on%20the%20left%20light" + $.param(params),
            beforeSend: function(xhrObj){
                // Request headers
                xhrObj.setRequestHeader("Ocp-Apim-Subscription-Key","YOUR-SUBSCRIPTION-KEY");
            },
            type: "GET",
            // The request body may be empty for a GET request
            data: "{body}",
        })
        .done(function(data) {
            // Display a popup containing the top intent
            alert("Detected the following intent: " + data.topScoringIntent.intent);
        })
        .fail(function() {
            alert("error");
        });
    });
</script>
</body>
</html>
```

2. Replace `"YOUR SUBSCRIPTION KEY"` with your subscription key in this line of code:
   `xhrObj.setRequestHeader("Ocp-Apim-Subscription-Key","YOUR SUBSCRIPTION KEY");`
3. Open the file you saved using a web browser. An alert window should pop up that says `Detected the following intent: TurnOn`.



Next steps

- See the LUIS Endpoint API reference to learn more about the parameters for calling your LUIS endpoint.

Call a LUIS app using Node.js

11/3/2017 • 2 min to read • Edit Online

This quickstart shows you how to call your Language Understanding Intelligent Service (LUIS) app in just a few minutes. When you're finished, you'll be able to use Node.js code to pass utterances to a LUIS endpoint and get results.

Before you begin

You need a Cognitive Services API key to make calls to the sample LUIS app we use in this walkthrough. To get an API key follow these steps:

1. 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.
2. Log in to the Azure portal at https://portal.azure.com.
3. Follow the steps in Creating Subscription Keys using Azure to get a key.
4. Go back to https://www.luis.ai and log in using your Azure account.

Understand what LUIS returns

To understand what a LUIS app returns, you can paste the URL of a sample LUIS app into a browser window. The sample app you'll use is an IoT app that detects whether the user wants to turn on or turn off lights.

1. The endpoint of the sample app is in this format:
   ```
   https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951ca1c2?subscription-key=<YOUR_API_KEY>&verbose=false&q=turn%20on%20the%20bedroom%20light
   ```
   Copy the URL and substitute your subscription key for the value of the `subscription-key` field.
2. Paste the URL into a browser window and press Enter. The browser displays a JSON result that indicates that LUIS detects the `HomeAutomation.TurnOn` intent and the `HomeAutomation.Room` entity with the value `bedroom`.

3. Change the value of the `q=` parameter in the URL to `turn off the living room light`, and press Enter. The result now indicates that the LUIS detected the `HomeAutomation.TurnOff` intent and the `HomeAutomation.Room` entity with value `living room`.

Consume a LUIS result using the Endpoint API with Node.js

You can use Node.js to access the same results you saw in the browser window in the previous step.

1. Copy the following code snippet:

```
require('dotenv').config();

var request = require('request');
var querystring = require('querystring');

function getLuisIntent(utterance) {
    var endpoint =
        "https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/";

    // Set the LUIS_APP_ID environment variable
    // to df67dcdb-c37d-46af-88e1-8b97951ca1c2, which is the ID
    // of a public sample application.
    var luisAppId = process.env.LUIS_APP_ID;

    // Set the LUIS_SUBSCRIPTION_KEY environment variable
    // to the value of your Cognitive Services subscription key
    var queryParams = {
        "subscription-key": process.env.LUIS_SUBSCRIPTION_KEY,
        "timezoneOffset": "0",
        "verbose":  true,
        "q": utterance
    }

    var luisRequest =
        endpoint + luisAppId +
        '?' + querystring.stringify(queryParams);

    request(luisRequest,
        function (err,
            response, body) {
            if (err)
                console.log(err);
            else {
                var data = JSON.parse(body);

                console.log(`Query: ${data.query}`);
                console.log(`Top Intent: ${data.topScoringIntent.intent}`);
                console.log('Intents:');
                console.log(data.intents);
            }
        });
}

// Pass an utterance to the sample LUIS app
getLuisIntent('turn on the left light');
```

2. Set the `LUIS_APP_ID` environment variable as described in the code comments.
3. Set the `LUIS_SUBSCRIPTION_KEY` environment variable to your Cognitive Services subscription key.
4. Run the code. It displays the same values that you saw earlier in the browser window.

Next steps

- See the LUIS Endpoint API reference to learn more about the parameters for calling your LUIS endpoint.

Call a LUIS app using Python

11/3/2017 • 2 min to read • Edit Online

This quickstart shows you how to call your Language Understanding Intelligent Service (LUIS) app in just a few minutes. When you're finished, you'll be able to use Python code to pass utterances to a LUIS endpoint and get results.

Before you begin

You need a Cognitive Services API key to make calls to the sample LUIS app we use in this walkthrough. To get an API key follow these steps:

1. 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.
2. Log in to the Azure portal at https://portal.azure.com.
3. Follow the steps in Creating Subscription Keys using Azure to get a key.
4. Go back to https://www.luis.ai and log in using your Azure account.

Understand what LUIS returns

To understand what a LUIS app returns, you can paste the URL of a sample LUIS app into a browser window. The sample app you'll use is an IoT app that detects whether the user wants to turn on or turn off lights.

1. The endpoint of the sample app is in this format:
   ```
   https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951ca1c2?subscription-key=<YOUR_API_KEY>&verbose=false&q=turn%20on%20the%20bedroom%20light
   ```
   Copy the URL and substitute your subscription key for the value of the `subscription-key` field.
2. Paste the URL into a browser window and press Enter. The browser displays a JSON result that indicates that LUIS detects the `HomeAutomation.TurnOn` intent and the `HomeAutomation.Room` entity with the value `bedroom`.

3. Change the value of the `q=` parameter in the URL to `turn off the living room light`, and press enter. The result now indicates that the LUIS detected the `HomeAutomation.TurnOff` intent and the `HomeAutomation.Room` entity with value `living room`.

Consume a LUIS result using the Endpoint API with Python

You can use Python to access the same results you saw in the browser window in the previous step.

1. Copy one of the following code snippets:

```
############ Python 2.7 #############
import httplib, urllib, base64

headers = {
    # Request headers
    'Ocp-Apim-Subscription-Key': 'YOUR-SUBSCRIPTION-KEY',
}

params = urllib.urlencode({
    # Query parameter
    'q': 'turn on the left light',
    # Optional request parameters, set to default values
    'timezoneOffset': '0',
    'verbose': 'false',
    'spellCheck': 'false',
    'staging': 'false',
})

try:
    conn = httplib.HTTPSConnection('westus.api.cognitive.microsoft.com')
    conn.request("GET", "/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951ca1c2?%s" % params, "{body}",
headers)
    response = conn.getresponse()
    data = response.read()
    print(data)
    conn.close()
except Exception as e:
    print("[Errno {0}] {1}".format(e.errno, e.strerror))

####################################
```

```
########### Python 3.6 #############
import requests

headers = {
    # Request headers
    'Ocp-Apim-Subscription-Key': 'YOUR-SUBSCRIPTION-KEY',
}

params ={
    # Query parameter
    'q': 'turn on the left light',
    # Optional request parameters, set to default values
    'timezoneOffset': '0',
    'verbose': 'false',
    'spellCheck': 'false',
    'staging': 'false',
}

try:
    r = requests.get('https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-
8b97951ca1c2',headers=headers, params=params)
    print(r.json())

except Exception as e:
    print("[Errno {0}] {1}".format(e.errno, e.strerror))

####################################
```

2. Replace the value of the `Ocp-Apim-Subscription-Key` field with your LUIS subscription key.
3. Run the script. It displays the same JSON that you saw earlier in the browser window.

Next steps

- See the LUIS Endpoint API reference to learn more about the parameters for calling your LUIS endpoint.

Call a LUIS app using PHP

11/9/2017 • 2 min to read • Edit Online

This quickstart shows you how to call your Language Understanding Intelligent Service (LUIS) app in just a few minutes. When you're finished, you'll be able to use PHP code to pass utterances to a LUIS endpoint and get results.

Before you begin

You need a Cognitive Services API key to make calls to the sample LUIS app we use in this walkthrough. To get an API key follow these steps:

1. 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.
2. Log in to the Azure portal at https://portal.azure.com.
3. Follow the steps in Creating Subscription Keys using Azure to get a key.
4. Go back to https://www.luis.ai and log in using your Azure account.

Understand what LUIS returns

To understand what a LUIS app returns, you can paste the URL of a sample LUIS app into a browser window. The sample app you'll use is an IoT app that detects whether the user wants to turn on or turn off lights.

1. The endpoint of the sample app is in this format:
   ```
   https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951ca1c2?subscription-key=<YOUR_API_KEY>&verbose=false&q=turn%20on%20the%20bedroom%20light
   ```
   Copy the URL and substitute your subscription key for the value of the `subscription-key` field.
2. Paste the URL into a browser window and press Enter. The browser displays a JSON result that indicates that LUIS detects the `HomeAutomation.TurnOn` intent and the `HomeAutomation.Room` entity with the value `bedroom`.

3. Change the value of the `q=` parameter in the URL to `turn off the living room light`, and press enter. The result now indicates that the LUIS detected the `HomeAutomation.TurnOff` intent and the `HomeAutomation.Room` entity with value `living room`.

Consume a LUIS result using the Endpoint API with PHP

You can use PHP to access the same results you saw in the browser window in the previous step.

1. Copy the code that follows and save it into an HTML file:

```
<?php

// NOTE: Be sure to uncomment the following line in your php.ini file.
// ;extension=php_openssl.dll

// **********************************************
// *** Update or verify the following values. ***
// **********************************************

// The ID of a public sample LUIS app that recognizes intents for turning on and off lights
$appId = "df67dcdb-c37d-46af-88e1-8b97951ca1c2";

// Replace the subscriptionKey string value with your valid Azure Subscription key.
$subscriptionKey = "YOUR-SUBSCRIPTION-KEY";

// The endpoint URI below is for the West US region.
// If your subscription is in a different region, update accordingly.
$endpoint = "https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/";

// The LUIS query term
$term = "turn on the left light";

function LuisQuery($url, $app, $key, $query) {
    // Prepare HTTP request
    // NOTE: Use the key 'http' even if you are making an HTTPS request. See:
    // http://php.net/manual/en/function.stream-context-create.php
    $headers = "Ocp-Apim-Subscription-Key: $key\r\n";
    $options = array ( 'http' => array (
                           'header' => $headers,
                           'method' => 'GET',
                           'ignore_errors' => true));

    // build query string
    $qs = http_build_query( array (
        "q" => $query,
        "timezoneOffset" => 0,
        "verbose" => "false",
        "spellCheck" => "false",
        "staging" => "false"
      )
    );
```

```
    $url = $url . $app . "?" . $qs;
    print($url);

    // Perform the Web request and get the JSON response
    $context = stream_context_create($options);
    $result = file_get_contents($url, false, $context);

    return $result;
}

if (strlen($subscriptionKey) == 32) {

    print("LUIS Query: " . $term . "\n");

    $json = LuisQuery($endpoint, $appId, $subscriptionKey, $term);

    print("\nJSON Response:\n\n");
    print(json_encode(json_decode($json), JSON_PRETTY_PRINT));

} else {

    print("Invalid LUIS API subscription key!\n");
    print("Please paste yours into the source code.\n");

}
?>
```

2. Replace `"YOUR-SUBSCRIPTION-KEY"` with your subscription key in this line of code:
   `$subscriptionKey = "YOUR-SUBSCRIPTION-KEY";`

3. Run the PHP application. It displays the same JSON that you saw earlier in the browser window.

Next steps

LUIS Endpoint API reference

Call a LUIS app using Ruby

11/9/2017 • 2 min to read • Edit Online

This quickstart shows you how to call your Language Understanding Intelligent Service (LUIS) app in just a few minutes. When you're finished, you'll be able to use Ruby code to pass utterances to a LUIS endpoint and get results.

Before you begin

You need a Cognitive Services API key to make calls to the sample LUIS app we use in this walkthrough. To get an API key follow these steps:

1. 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.
2. Log in to the Azure portal at https://portal.azure.com.
3. Follow the steps in Creating Subscription Keys using Azure to get a key.
4. Go back to https://www.luis.ai and log in using your Azure account.

Understand what LUIS returns

To understand what a LUIS app returns, you can paste the URL of a sample LUIS app into a browser window. The sample app you'll use is an IoT app that detects whether the user wants to turn on or turn off lights.

1. The endpoint of the sample app is in this format:
   ```
   https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951ca1c2?subscription-key=<YOUR_API_KEY>&verbose=false&q=turn%20on%20the%20bedroom%20light
   ```
   Copy the URL and substitute your subscription key for the value of the `subscription-key` field.
2. Paste the URL into a browser window and press Enter. The browser displays a JSON result that indicates that LUIS detects the `HomeAutomation.TurnOn` intent and the `HomeAutomation.Room` entity with the value `bedroom`.

3. Change the value of the `q=` parameter in the URL to `turn off the living room light`, and press enter. The result now indicates that the LUIS detected the `HomeAutomation.TurnOff` intent and the `HomeAutomation.Room` entity with value `living room`.

Consume a LUIS result using the Endpoint API with Ruby

You can use Ruby to access the same results you saw in the browser window in the previous step.

1. Copy the code that follows and save it into an HTML file:

```
require 'net/https'
require 'uri'
require 'json'

# **********************************************
# *** Update or verify the following values. ***
# **********************************************

# The ID of a public sample LUIS app that recognizes intents for turning on and off lights
appId = "df67dcdb-c37d-46af-88e1-8b97951ca1c2"

# Replace the subscriptionKey string value with your valid Azure Subscription key.
subscriptionKey = "YOUR-SUBSCRIPTION-KEY"

# The endpoint URI below is for the West US region.
# If your subscription is in a different region, update accordingly.
host = "https://westus.api.cognitive.microsoft.com"
path = "/luis/v2.0/apps/"

# The LUIS query term
term = "turn on the left light"

if subscriptionKey.length != 32 then
    puts "Invalid LUIS API subscription key!"
    puts "Please paste yours into the source code."
    abort
end

qs = URI.encode_www_form(
    "q" => term,
    "timezoneOffset" => 0,
    "verbose" => false,
    "spellCheck" => false,
    "staging" => false
)
uri = URI(host + path + appId + "?" + qs)

puts
puts "LUIS query: " + term
puts
puts "Request URI: " + uri.to_s

request = Net::HTTP::Get.new(uri)
request["Ocp-Apim-Subscription-Key"] = subscriptionKey

response = Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do |http|
    http.request(request)
end

puts "\nJSON Response:\n\n"
puts JSON::pretty_generate(JSON(response.body))
```

2. Replace `"YOUR-SUBSCRIPTION-KEY"` with your subscription key in this line of code:
   `subscriptionKey = "YOUR-SUBSCRIPTION-KEY"`

3. Run the Ruby application. It displays the same JSON that you saw earlier in the browser window.

Next steps

LUIS Endpoint API reference

Add utterances to a LUIS app using Node.js

11/8/2017 • 6 min to read • Edit Online

This quickstart shows you how to programmatically add utterances to your Language Understanding Intelligent Service (LUIS) app and train LUIS.

Using the command line is a quick way to enter many utterances and train LUIS. You can also automate this task into a larger pipeline.

Refer to the Authoring API definitions for technical documentation for the add utterance, train, and training status APIs.

Prerequisites

- Latest **Node.js** with NPM.
- NPM dependencies for this quickstart: **request**, **request-promise**, **fs-extra**.
- **[Recommended]** Visual Studio Code for IntelliSense and debugging.
- Your LUIS **programmatic key**. You can find this key under Account Settings in https://www.luis.ai.
- Your existing LUIS **application ID**. The application ID is shown in the application dashboard. The LUIS application with the intents and entities used in the `utterances.json` file must exist prior to running the code in `add-utterances.js`. The code in this article will not create the intents and entities. It will only add the utterances for existing intents and entities.
- The **version ID** within the application that receives the utterances. The default ID is "0.1"
- Create a new file named `add-utterances.js` project in VSCode.

> **NOTE**
> The complete `add-utterances.js` file is available from the **LUIS-Samples** Github repository.

Write the Node.js code

Add the NPM dependencies to the file.

```
// NPM Dependencies
var rp = require('request-promise');
var fse = require('fs-extra');
var path = require('path');
```

Add the LUIS constants to the file. Copy the following code and change to your programmatic key, application ID, and version ID.

```
// To run this sample, change these constants.

// Programmatic key, available in luis.ai under Account Settings
const LUIS_programmaticKey = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
// ID of your LUIS app to which you want to add an utterance
const LUIS_appId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";
// The version number of your LUIS app
const LUIS_versionId = "0.1";
```

Add the name and location of the upload file containing your utterances.

```
// uploadFile is the file containing JSON for utterance(s) to add to the LUIS app.
// The contents of the file must be in this format described at: https://aka.ms/add-utterance-json-format
const uploadFile = "./utterances.json"
```

Add the variables that hold the command-line values.

```
// Variables holding command line arguments
var trainAfterAdd = false;
var requestTrainingStatus = false;


// Parse command line arguments:
// -train to train based on the utterances in uploadFile
// -status to get training status
if (process.argv.length >= 3) {
    if (process.argv[2] === "-train") {
        trainAfterAdd = true;
    } else if (process.argv[2] === "-status") {
        requestTrainingStatus = true;
    }
}
```

Add the function `sendUtteranceToApi` to send and receive HTTP calls.

```
// Send JSON as the body of the POST request to the API
var sendUtteranceToApi = async (options) => {
    try {

        var response;
        if (options.method === 'POST') {
            response = await rp.post(options);
        } else if (options.method === 'GET') {
            response = await rp.get(options);
        }

        return { request: options.body, response: response };

    } catch (err) {
        throw err;
    }
}
```

Add the configuration JSON object used by the `addUtterance` function.

```
// upload configuration
var configAddUtterance = {
    LUIS_subscriptionKey: LUIS_programmaticKey,
    LUIS_appId: LUIS_appId,
    LUIS_versionId: LUIS_versionId,
    inFile: path.join(__dirname, uploadFile),
    uri:
"https://westus.api.cognitive.microsoft.com/luis/api/v2.0/apps/{appId}/versions/{versionId}/examples".replace("
{appId}", LUIS_appId).replace("{versionId}", LUIS_versionId)
};
```

Add the function `addUtterance` manage the API request and response used by `SendUtteranceToApp`.

```
// Call add-utterance
var addUtterance = async (config) => {

    try {

        // Extract the JSON for the request body
        // The contents of the file to upload need to be in this format described in the comments above.
        var jsonUtterance = await fse.readJson(config.inFile);

        // Add an utterance
        var utterancePromise = sendUtteranceToApi({
            uri: config.uri,
            method: 'POST',
            headers: {
                'Ocp-Apim-Subscription-Key': config.LUIS_subscriptionKey
            },
            json: true,
            body: jsonUtterance
        });

        let results = await utterancePromise;
        let response = await fse.writeJson(config.inFile.replace('.json', '.results.json'), results);

        console.log("Add utterance done");

    } catch (err) {
        console.log(`Error adding utterance:  ${err.message} `);
        //throw err;
    }

}
```

Add the configuration JSON object used by the `train` function.

```
// training configuration
var configTrain = {
    LUIS_subscriptionKey: LUIS_programmaticKey,
    LUIS_appId: LUIS_appId,
    LUIS_versionId: LUIS_versionId,
    uri:
"https://westus.api.cognitive.microsoft.com/luis/api/v2.0/apps/{appId}/versions/{versionId}/train".replace("
{appId}", LUIS_appId).replace("{versionId}", LUIS_versionId),
    method: 'POST', // POST to request training, GET to get training status
};
```

Add the function `train` to start the training process.

```
// Call train
var train = async (config) => {

    try {

        var trainingPromise = sendUtteranceToApi({
            uri: config.uri,
            method: config.method, // Use POST to request training, GET to get training status
            headers: {
                'Ocp-Apim-Subscription-Key': config.LUIS_subscriptionKey
            },
            json: true,
            body: null      // The body can be empty for a training request
        });

        let results = await trainingPromise;

        if (config.method === 'POST') {
            let response = await fse.writeJson(path.join(__dirname, 'training-results.json'), results);
            console.log(`Training request sent. The status of the training request is:
${results.response.status}.`);
        } else if (config.method === 'GET') {
            let response = await fse.writeJson(path.join(__dirname, 'training-status-results.json'), results);
            console.log(`Training status saved to file. `);
        }

    } catch (err) {
        console.log(`Error in Training:  ${err.message} `);
        // throw err;
    }

}
```

Add the code that chooses which action to take (add utterance or train) based on the command-line variables.

```
// MAIN
if (trainAfterAdd) {
    // Add the utterance to the LUIS app and train
    addUtterance(configAddUtterance)
        .then(() => {
            console.log("Add utterance complete. About to request training.");
            configTrain.method = 'POST';
            return train(configTrain, false);
        }).then(() => {
            console.log("Training process complete. Requesting training status.");
            configTrain.method = 'GET';
            return train(configTrain, true);
        }).then(() => {
            console.log("process done");
        });
} else if (requestTrainingStatus) {
    // Get the training status
    configTrain.method = 'GET';
    train(configTrain)
        .then(() => {
            console.log("Requested training status.");
        });
}
else {
    // Add the utterance to the LUIS app without training it afterwards
    addUtterance(configAddUtterance)
        .then(() => {
            console.log("Add utterance complete.");
        });

}
```

Specify utterances to add

Create and edit the file `utterances.json` to specify the entities you want to add to the LUIS app. The intent and entities **must** already be in the LUIS app.

> **NOTE**
>
> The LUIS application with the intents and entities used in the `utterances.json` file must exist prior to running the code in `add-utterances.js`. The code in this article will not create the intents and entities. It will only add the utterances for existing intents and entities.

The `text` field contains the text of the utterance. The `intentName` field must correspond to the name of an intent in the LUIS app. The `entityLabels` field is required. If you don't want to label any entities, provide an empty list as shown in the following example.

If the entityLabels list is not empty, the `startCharIndex` and `endCharIndex` need to mark the entity referred to in the `entityName` field. Both indexes are zero-based counts meaning 6 in the top example refers to the "S" of Seattle and not the space before the capital S.

```
[
    {
        "text": "go to Seattle",
        "intentName": "BookFlight",
        "entityLabels": [
            {
                "entityName": "Location::LocationTo",
                "startCharIndex": 6,
                "endCharIndex": 12
            }
        ]
    },
    {
        "text": "book a flight",
        "intentName": "BookFlight",
        "entityLabels": []
    }
]
```

Add an utterance from the command-line

Run the application from a command-line with Node.js.

Calling add-utterance with no arguments adds an utterance to the app, without training it.

```
> node add-utterances.js
```

This sample creates a file with the `results.json` that contains the results from calling the add utterances API. The `response` field is in this format for utterances that was added. The `hasError` is false, indicating the utterance was added.

```
    "response": [
        {
            "value": {
                "UtteranceText": "go to seattle",
                "ExampleId": -5123383
            },
            "hasError": false
        },
        {
            "value": {
                "UtteranceText": "book a flight",
                "ExampleId": -169157
            },
            "hasError": false
        }
    ]
```

Add an utterance and train from the command-line

Call add-utterance with the `-train` argument to send a request to train, and subsequently request training status. The status is queued immediately after training begins. Status details are written to a file.

```
> node add-utterances.js -train
```

> **NOTE**
> Duplicate utterances aren't added again, but don't cause an error. The `response` contains the ID of the original utterance.

When you call the sample with the `-train` argument, it creates a `training-results.json` file indicating the request to train the LUIS app was successfully queued.

The following shows the result of a successful request to train:

```
{
    "request": null,
    "response": {
        "statusId": 9,
        "status": "Queued"
    }
}
```

After the request to train is queued, it can take a moment to complete training.

Get training status from the command line

Call the sample with the `-status` argument to check the training status and write status details to a file.

```
> node add-utterances.js -status
```

Next steps

Integrate LUIS with a bot

Add utterances to a LUIS app using Python

11/13/2017 • 5 min to read • Edit Online

This quickstart shows you how to programmatically add utterances to your Language Understanding Intelligent Service (LUIS) app and train LUIS.

Using the command line is a quick way to enter many utterances and train LUIS. You can also automate this task into a larger pipeline.

Refer to the Authoring API definitions for technical documentation for the add utterance, train, and training status APIs.

Prerequisites

- Python 3.6 or later.
- **[Recommended]** Visual Studio Code for IntelliSense and debugging.
- Your LUIS **programmatic key**. You can find this key under Account Settings in https://www.luis.ai.
- Your existing LUIS **application ID**. The application ID is shown in the application dashboard. The LUIS application with the intents and entities used in the `utterances.json` file must exist prior to running the code in `add-utterances.js`. The code in this article does not create the intents and entities. It only adds the utterances for existing intents and entities.
- The **version ID** within the application that receives the utterances. The default ID is "0.1"
- Create a new file named `add-utterances-3-6.py` project in VSCode.

> **NOTE**
> The complete `add-utterances-3-6.py` file is available from the **LUIS-Samples** Github repository.

Write the Python code

1. Copy the following code snippets:

```
############ Python 3.6 #############
# -*- coding: utf-8 -*-

import http.client, sys, os.path, json

# Programmatic key, available in luis.ai under Account Settings
LUIS_programmaticKey  = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

# ID of your LUIS app to which you want to add an utterance
LUIS_APP_ID      = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"

# The version number of your LUIS app
LUIS_APP_VERSION = "0.1"

# Update the host if your LUIS subscription is not in the West US region
LUIS_HOST       = "westus.api.cognitive.microsoft.com"

# uploadFile is the file containing JSON for utterance(s) to add to the LUIS app.
# The contents of the file must be in this format described at: https://aka.ms/add-utterance-json-format
UTTERANCE_FILE   = "./utterances.json"
RESULTS_FILE     = "./utterances.results.json"
```

```
# LUIS client class for adding and training utterances
class LUISClient:

    # endpoint method names
    TRAIN    = "train"
    EXAMPLES = "examples"

    # HTTP verbs
    GET  = "GET"
    POST = "POST"

    # Encoding
    UTF8 = "UTF8"

    # path template for LUIS endpoint URIs
    PATH     = "/luis/api/v2.0/apps/{app_id}/versions/{app_version}/"

    # default HTTP status information for when we haven't yet done a request
    http_status = 200
    reason = ""
    result = ""

    def __init__(self, host, app_id, app_version, key):
        if len(key) != 32:
            raise ValueError("LUIS subscription key not specified in " +
                             os.path.basename(__file__))
        if len(app_id) != 36:
            raise ValueError("LUIS application ID not specified in " +
                             os.path.basename(__file__))
        self.key = key
        self.host = host
        self.path = self.PATH.format(app_id=app_id, app_version=app_version)

    def call(self, luis_endpoint, method, data=""):
        path = self.path + luis_endpoint
        headers = {'Ocp-Apim-Subscription-Key': self.key}
        conn = http.client.HTTPSConnection(self.host)
        conn.request(method, path, data.encode(self.UTF8) or None, headers)
        response = conn.getresponse()
        self.result = json.dumps(json.loads(response.read().decode(self.UTF8)),
                                 indent=2)
        self.http_status = response.status
        self.reason = response.reason
        return self

    def add_utterances(self, filename=UTTERANCE_FILE):
        with open(filename, encoding=self.UTF8) as utterance:
            data = utterance.read()
        return self.call(self.EXAMPLES, self.POST, data)

    def train(self):
        return self.call(self.TRAIN, self.POST)

    def status(self):
        return self.call(self.TRAIN, self.GET)

    def write(self, filename=RESULTS_FILE):
        if self.result:
            with open(filename, "w", encoding=self.UTF8) as outfile:
                outfile.write(self.result)
        return self

    def print(self):
        if self.result:
            print(self.result)
        return self

    def raise_for_status(self):
        if 200 <= self.http_status < 300:
```

```
        if 200 <= self.http_status < 300:
            return self
        raise http.client.HTTPException("{} {}".format(
            self.http_status, self.reason))

if __name__ == "__main__":

    # uncomment a line below to simulate command line options
    # sys.argv.append("-train")
    # sys.argv.append("-status")

    luis = LUISClient(LUIS_HOST, LUIS_APP_ID, LUIS_APP_VERSION,
                      LUIS_programmaticKey)

    try:
        if len(sys.argv) > 1:
            option = sys.argv[1].lower().lstrip("-")
            if option == "train":
                print("Adding utterance(s).")
                luis.add_utterances()   .write().raise_for_status()
                print("Added utterance(s). Requesting training.")
                luis.train()            .write().raise_for_status()
                print("Requested training. Requesting training status.")
                luis.status()           .write().raise_for_status()
            elif option == "status":
                print("Requesting training status.")
                luis.status().write().raise_for_status()
        else:
            print("Adding utterance(s).")
            luis.add_utterances().write().raise_for_status()
    except Exception as ex:
        luis.print()    # JSON response may have more details
        print("{0.__name__}: {1}".format(type(ex), ex))
    else:
        print("Success: results in", RESULTS_FILE)
```

Specify utterances to add

Create and edit the file `utterances.json` to specify the entities you want to add to the LUIS app. The intent and entities **must** already be in the LUIS app.

> **NOTE**
>
> The LUIS application with the intents and entities used in the `utterances.json` file must exist prior to running the code in `add-utterances.js`. The code in this article does not create the intents and entities. It only adds the utterances for existing intents and entities.

The `text` field contains the text of the utterance. The `intentName` field must correspond to the name of an intent in the LUIS app. The `entityLabels` field is required. If you don't want to label any entities, provide an empty list as shown in the following example:

If the entityLabels list is not empty, the `startCharIndex` and `endCharIndex` need to mark the entity referred to in the `entityName` field. Both indexes are zero-based counts meaning 6 in the top example refers to the "S" of Seattle and not the space before the capital S.

```
[
    {
        "text": "go to Seattle",
        "intentName": "BookFlight",
        "entityLabels": [
            {
                "entityName": "Location::LocationTo",
                "startCharIndex": 6,
                "endCharIndex": 12
            }
        ]
    },
    {
        "text": "book a flight",
        "intentName": "BookFlight",
        "entityLabels": []
    }
]
```

Add an utterance from the command-line

Run the application from a command-line with Python 3.6.

Calling add-utterance with no arguments adds an utterance to the app, without training it.

```
> python add-utterances-3-6.py
```

This sample creates a file with the `results.json` that contains the results from calling the add utterances API. The `response` field is in this format for utterances that was added. The `hasError` is false, indicating the utterance was added.

```
    "response": [
        {
            "value": {
                "UtteranceText": "go to seattle",
                "ExampleId": -5123383
            },
            "hasError": false
        },
        {
            "value": {
                "UtteranceText": "book a flight",
                "ExampleId": -169157
            },
            "hasError": false
        }
    ]
```

Add an utterance and train from the command-line

Call add-utterance with the `-train` argument to send a request to train, and subsequently request training status. The status is queued immediately after training begins. Status details are written to a file.

```
> python add-utterances-3-6.py -train
```

> **NOTE**
> Duplicate utterances aren't added again, but don't cause an error. The `response` contains the ID of the original utterance.

When you call the sample with the `-train` argument, it creates a `training-results.json` file indicating the request to train the LUIS app was successfully queued.

The following shows the result of a successful request to train:

```
{
    "request": null,
    "response": {
        "statusId": 9,
        "status": "Queued"
    }
}
```

After the request to train is queued, it can take a moment to complete training.

Get training status from the command line

Call the sample with the `-status` argument to check the training status and write status details to a file.

```
> python add-utterances-3-6.py -status
```

Next steps

Integrate LUIS with a bot

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

10/18/2017 • 7 min to read • Edit Online

This tutorial walks you through connecting to a bot built with Bot Framework and integrated with a LUIS app.

When a user talks to your bot with a phrase such as "Search hotels in Seattle", the bot calls into your LUIS app with the LUIS endpoint URL. LUIS returns a response to the bot and the bot constructs a response to the user.

In the Bot Framework Emulator, you can see what is happening from the user's view as well as the HTTP request/response log in the log panel.

Prerequisites

The tutorial assumes you have the following before you begin the next steps:

- The latest version of Node.js installed
- An Azure LUIS subscription

Outline of steps

The steps in this tutorial are summarized here. More details will be given in the following sections for these steps.

- Download and install the Bot Framework emulator.
- Download the BotFramework-Samples repository. This has the LUIS app definition file as well as the LUIS sample bot.
- In luis.ai, create and publish a LUIS app from a file provided in BotFramework-Samples. Publishing the app gives you the app's LUIS endpoint. This will be the URL that the bot calls.
- In a code editor, edit the BotFramework-Samples LUIS bot's .env for the LUIS endpoint.
- From a command line/terminal, start the BotFramework-Samples LUIS bot. Note the port, such as 3978
- Start the Bot Framework Emulator. Enter the URL for the bot, such as http://localhost:3978/api/messages.
- Ask the bot: "Search hotels in Seattle"
- View the bot's response

> **NOTE**
>
> - If you choose to secure your bot or work with a bot that is not local to your computer, you need to create a Bot Framework developer account and register your bot. Learn more about the Bot Framework.
> - Bot Builder is used as an NPM dependency in the LUIS sample bot application. You don't have to do anything other than the typical "npm install" in order to get the dependency.

Download and install the Bot Emulator

The Bot Framework Emulator is available on GitHub. Download and install the correct version for your operating system. Note where the application is on your computer so you can start it in a later step.

Clone or download the BotFramework-Samples repository

BotBuilder-Samples is a GitHub repository with more samples beyond just the LUIS bot. The subfolder you need

for this tutorial is ./Node/Intelligence-LUIS.

Clone or download the repository to your computer. You edit and run the Node/Intelligence-LUIS sample found in this repository.

Create a new LUIS application from the application definition file

Create a luis.ai account and log in.

In order to get a new LUIS application set up for the bot, you need to import the **LuisBot.json** file found at ./BotBuilder-Samples/Node/Intelligence-LUIS folder. The file contains the application definition for the LUIS app the sample bot uses.

1. On the **My Apps** page of the LUIS web page, click **Import App**.
2. In the **Import new app** dialog box, click **Choose file** and upload the LuisBot.json file. 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.
3. Once the app is imported, you need to change the Assigned endpoint key on the Publish App page to your Azure LUIS subscription. Then publish the app to get the endpoint API URL. You need to copy the URL, you need to paste that URL as an environment variable in the next step.

Set the LUIS endpoint in the LUIS sample bot

In a code editor, set the LUIS_MODEL_URL environment variable and comment out the security variables.

1. Open the `.env` file and change `LUIS_MODEL_URL` to the LUIS endpoint 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`
3. Add the pound sign, '#', in front of the MICROSOFT_APP environment variables:
   `#MICROSOFT_APP_ID= #MICROSOFT_APP_PASSWORD=` These environment variables are for security to a remote bot so you do not need them.

Start the bot

From a command line, at the root of the LUIS sample bot, install the node dependencies and start the LUIS Sample.

Install the node dependencies:

```
> npm install
```

After the node modules are installed, start the bot.

```
> npm start
```

You should see a response such as the following response:

```
restify listening to http://[::]:3978
```

The port number is necessary for the next step.

Start the Bot Emulator

Start the Bot Framework emulator.

The Bot Emulator needs the sample bot's endpoint such as http://localhost:3978/api/messages. Enter that into the address bar at the top of the emulator. It asks you for the Bot's application ID and password. You don't need those values if the bot is local to your computer, so click on CONNECT.

If you see the following connection error in the log, the Bot Emulator cannot find your bot.

```
POST connect ECONNREFUSED 127.0.0.1:3978
```

If you see this successful HTTP response in the log, the Bot emulator connected to the LUIS sample bot:

```
POST 202 [conversationUpdate]
```

Ask the bot a question

In the bottom bar of the emulator, enter:

```
Search hotels in Seattle
```

The bot should respond with suggestions for hotels.

See the bot's response

In the left panel of the emulator, the user sees suggested hotels.

In the right panel, the HTTP conversation between the bot emulator and the LUIS sample bot is shown.

```
Log
[13:10:55] Emulator listening on http://[::]:61655
[13:10:55] ngrok not configured (only needed when connecting to remotely hosted bots)
[13:10:55] Connecting to bots hosted remotely
[13:10:55] Edit ngrok settings
[13:10:55] Checking for new version...
[13:10:56] Application is up to date.
[13:11:51] -> POST 202 [conversationUpdate]
[13:11:51] -> POST 202 [conversationUpdate]
[13:12:01] -> POST 202 [message] search for hotels in seattle
[13:12:01] <- GET 200 getPrivateConversationData
[13:12:01] <- GET 200 getUserData
[13:12:01] <- GET 200 getConversationData
[13:12:02] <- POST 200 setPrivateConversationData
[13:12:02] <- POST 200 Reply[message] Welcome to the Hotels finder! We are analyzing you...
[13:12:02] <- POST 200 Reply[message] Looking for hotels in seattle...
[13:12:02] <- POST 200 Reply[event] Debug Event
[13:12:03] <- POST 200 setPrivateConversationData
[13:12:03] <- POST 200 Reply[message] I found 5 hotels:
[13:12:03] <- POST 200 Reply[message] application/vnd.microsoft.card.hero
[13:12:03] <- POST 200 Reply[event] Debug Event
```

In the command line for the LUIS bot sample, you will see the following

```
Debugging with inspector protocol because Node.js v8.6.0 was detected.
node --inspect-brk=37939 app.js
Debugger listening on ws://127.0.0.1:37939/b905da3e-0ffb-4c54-bf5c-e5ca051682b8
restify listening to http://[::]:3978
WARN: ChatConnector: receive - emulator running without security enabled.
ChatConnector: message received.
WARN: ChatConnector: receive - emulator running without security enabled.
ChatConnector: message received.
WARN: ChatConnector: receive - emulator running without security enabled.
ChatConnector: message received.
UniversalBot("*") routing "find hotels in seattle" from "emulator"
Library("*")recognize() recognized: SearchHotels(1)
Library("*").findRoutes() explanation:
    GlobalAction(1)
Session.beginDialog(*:SearchHotels)
SearchHotels - waterfall() step 1 of 2
SearchHotels - Session.send()
SearchHotels - waterfall() step 2 of 2
SearchHotels - Session.send()
SearchHotels - Session.sendBatch() sending 2 message(s)
SearchHotels - Session.send()
SearchHotels - Session.send()
SearchHotels - Session.endDialog()
Session.sendBatch() sending 2 message(s)
WARN: ChatConnector: receive - emulator running without security enabled.
ChatConnector: message received.
UniversalBot("*") routing "find hotels in seattle" from "emulator"
Library("*")recognize() recognized: SearchHotels(1)
Library("*").findRoutes() explanation:
    GlobalAction(1)
Session.beginDialog(*:SearchHotels)
SearchHotels - waterfall() step 1 of 2
SearchHotels - Session.send()
SearchHotels - waterfall() step 2 of 2
```

See the LUIS response while running the bot

The Bot Builder SDK uses the LUIS endpoint API you set in the **.env** file when creating the recognizer. After creating the recognizer, it is set on the **bot** object.

```
// app.js - register LUIS endpoint API
var recognizer = new builder.LuisRecognizer(process.env.LUIS_MODEL_URL);
bot.recognizer(recognizer);
```

If you set a breakpoint in the **app.js file** in the first function of **bot.dialog**, you can watch the LUIS response for cityEntity and airportEntity.

```
// app.js - set breakpoint to watch the LUIS response
var cityEntity = builder.EntityRecognizer.findEntity(args.intent.entities, 'builtin.geography.city');
var airportEntity = builder.EntityRecognizer.findEntity(args.intent.entities, 'AirportCode');
```

Using the sample request "Search for hotel in Seattle," LUIS response with a filled cityEntity and an null airportEntity.

```
// cityEntity viewed in debug
{
    entity: "seattle", type: "builtin.geography.city", startIndex: 15, endIndex: 21, score: 0.9239899}
    endIndex: 21
    entity: "seattle"
    score: 0.9239899
    startIndex: 15
    type: "builtin.geography.city"
    __proto__: Object {__defineGetter__: , __defineSetter__: , hasOwnProperty: , …
}
```

The score of .92 is a high probability for the entity **Seattle** found with the built-in type **geography.city**. That is enough information to search for hotels in the city of Seattle.

Next steps

- You can learn more about this specific sample.
- 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.

Integrate LUIS with a bot using the Bot Builder SDK for C#

10/25/2017 • 6 min to read • Edit Online

This tutorial walks you through connecting to a bot built with Bot Framework integrated with a LUIS app.

When a user talks to your bot with a phrase such as "Search hotels in Seattle", the bot calls into your LUIS app with the LUIS endpoint URL. LUIS returns a response to the bot and the bot constructs a response to the user.

In the Bot Framework Emulator, you can see what is happening from the user's view as well as the HTTP request/response log in the log panel.

Prerequisites

The tutorial assumes you have the following before you begin the next steps:

- The latest version of Visual Studio installed
- An Azure LUIS subscription

Outline of steps

The steps in this tutorial are summarized here. More details will be given in the following sections for these steps.

- Download and install the Bot Framework emulator.
- Download the BotBuilder-Samples repository. This has the LUIS app definition file as well as the LUIS sample bot.
- In luis.ai, create and publish a LUIS app from a file provided in BotBuilder-Samples. Publishing the app gives you the app's LUIS endpoint. This will be the URL that the bot calls.
- In Visual Studio, edit the BotBuilder-Samples LUIS bot's RootLuisDialog.cs for the LUIS endpoint.
- In Visual Studio, start the BotBuilder-Samples LUIS bot. Note the port, such as 3979
- Start the Bot Framework Emulator. Enter the URL for the bot, such as http://localhost:3979/api/messages.
- Ask the bot: "Search hotels in Seattle"
- View the bot's response

> **NOTE**
>
> - If you choose to secure your bot or work with a bot that is not local to your computer, you need to create a Bot Framework developer account and register your bot. Learn more about the Bot Framework.
> - Bot Builder is used as an NuGet dependency in the LUIS sample bot application. You don't have to do anything in order to get the dependency. It is already included for you.

Download and install the Bot Emulator

The Bot Framework Emulator is available on GitHub. Download and install the correct version for your operating system. Note where the application is on your computer so you can start it in a later step.

Clone or download the BotBuilder-Samples repository

BotBuilder-Samples is a GitHub repository with more samples beyond just the LUIS bot. The subfolder you need for

this tutorial is ./CSharp/Intelligence-LUIS.

Clone or download the repository to your computer. You edit and run the CSharp/Intelligence-LUIS sample found in this repository.

Create a new LUIS Application from the application definition file

Create a luis.ai account and log in.

In order to get a new LUIS application set up for the bot, you need to import the **LuisBot.json** file found at ./BotBuilder-Samples/CSharp/Intelligence-LUIS folder. The file contains the application definition for the LUIS app the sample bot uses.

1. On the **My Apps** page of the LUIS web page, click **Import App**.
2. In the **Import new app** dialog box, click **Choose file** and upload the LuisBot.json file. 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.
3. Once the app is imported, you need to change the Assigned endpoint key on the Publish App page to your Azure LUIS subscription. Then publish the app. Copy your endpoint URL which contains your LUIS app ID and your LUIS subscription ID.

The URL looks something like the following URL:

```
# LUIS endpoint URL
https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/a0be191c-ebe0-4153-a4c3-d880b9c5d753?subscription-key=a237d6bc86cd4562bf67b09dff44d2e6&timezoneOffset=0&verbose=true&q=
```

The number after the /apps/ is your app ID. The number in the query string for subscription-key is your subscription ID. Using the above example, the app ID is:

```
a0be191c-ebe0-4153-a4c3-d880b9c5d753
```

and the subscription ID is:

```
a237d6bc86cd4562bf67b09dff44d2e6
```

and the domain is the base URL:

```
https://westus.api.cognitive.microsoft.com
```

Set the LUIS endpoint in the LUIS sample bot

In Visual Studio, set your LUIS application credentials. Open the `RootLuisDialog.cs` file and change the `LuisModel` attribute settings for the app ID, subscription ID, and domain:

```
[LuisModel("a0be191c-ebe0-4153-a4c3-d880b9c5d753", "a237d6bc86cd4562bf67b09dff44d2e6", domain: "westus.api.cognitive.microsoft.com")]
[Serializable]
public class RootLuisDialog : LuisDialog<object>
```

Start the bot

From Visual Studio, start the LUIS bot sample. A browser window will open including the URL with a port number. The port number, such as 3979, is necessary for the next step.

Start the Bot Emulator

Start the Bot Framework emulator.

The Bot Emulator needs the sample bot's endpoint such as http://localhost:3979/api/messages. Enter that into the address bar at the top of the emulator. It asks you for the Bot's application ID and password. You don't need those values if the bot is local to your computer, so click on CONNECT.

If you see the following connection error in the log, the Bot Emulator cannot find your bot.

```
POST connect ECONNREFUSED 127.0.0.1:3979
```

If you see this successful HTTP response in the log, the Bot emulator connected to the LUIS sample bot:

```
POST 202 [conversationUpdate]
```

Ask the bot a question

In the bottom bar of the emulator, enter:

```
Search hotels in Seattle
```

The bot should respond with suggestions for hotels.

See the bot's response

In the left panel of the emulator, the user sees suggested hotels.

In the right panel, the HTTP conversation between the bot emulator and the LUIS sample bot is shown.

```
Log
[13:10:55] Emulator listening on http://[::]:61655
[13:10:55] ngrok not configured (only needed when connecting to remotely hosted bots)
[13:10:55] Connecting to bots hosted remotely
[13:10:55] Edit ngrok settings
[13:10:55] Checking for new version...
[13:10:56] Application is up to date.
[13:11:51] -> POST 202 [conversationUpdate]
[13:11:51] -> POST 202 [conversationUpdate]
[13:12:01] -> POST 202 [message] search for hotels in seattle
[13:12:01] <- GET 200 getPrivateConversationData
[13:12:01] <- GET 200 getUserData
[13:12:01] <- GET 200 getConversationData
[13:12:02] <- POST 200 setPrivateConversationData
[13:12:02] <- POST 200 Reply[message] Welcome to the Hotels finder! We are analyzing you...
[13:12:02] <- POST 200 Reply[message] Looking for hotels in seattle...
[13:12:02] <- POST 200 Reply[event] Debug Event
[13:12:03] <- POST 200 setPrivateConversationData
[13:12:03] <- POST 200 Reply[message] I found 5 hotels:
[13:12:03] <- POST 200 Reply[message] application/vnd.microsoft.card.hero
[13:12:03] <- POST 200 Reply[event] Debug Event
```

See the LUIS response while running the bot

The Bot Builder SDK uses the LUIS endpoint API you set in the **RootLuisDialog.cs** file in the class level attribute **LuisModel**. The LuisModel is defined as:

```
public LuisModelAttribute(string modelID, string subscriptionKey, LuisApiVersion apiVersion =
LuisApiVersion.V2, string domain = null, bool log = true, bool spellCheck = false, bool staging = false, bool
verbose = false);
```

Set a breakpoint in each of the methods decorated with your LUIS intents of Help, None, ShowHotelsReviews, and SearchHotels. And enter your query again: "Search hotels in Seattle".

When the breakpoint stops in the **SearchHotels** intent's method of **Search**, you can inspect what LUIS returned with the result parameter. The result parameter includes **TopScoringIntent** which in this case is "SearchHotels" as well as all entities and Intents found.

Next steps

- You can learn more about this specific sample.
- 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.

Build a LUIS app programmatically using Node.js

11/13/2017 • 13 min to read • Edit Online

LUIS provides a programmatic API that does everything that the UI at https://www.luis.ai does. This can save time when you might have a lot of preexisting data and it'd be faster to create a LUIS app programmatically than by entering information by hand.

Prerequisites

- Log in to www.luis.ai and find your Programmatic Key in Account Settings. You use this key to call the Authoring API.
- If you don't have an Azure subscription, create a free account before you begin.
- This tutorial starts with a CSV for a hypothetical company's log files of user requests. Download it here.
- Install the latest Node.js with NPM. Download it from here.
- **[Recommended]** Visual Studio Code for IntelliSense and debugging, download it from here for free.

Map preexisting data to intents and entities

Even if you have a system that wasn't created with LUIS in mind, if it contains textual data that maps to different things users want to do, you might be able to come up with a mapping from the existing categories of user input to intents in LUIS. If you can identify important words or phrases in what the users said, these might map to entities.

Open the `IoT.csv` file. It contains a log of user queries to a hypothetical home automation service, including how they were categorized, what the user said, and some columns with useful information pulled out of them.

	RequestType	Utterance	Operation	Device	Room	timestamp
2	TurnOn	Turn on the lights	on	lights		11/09/2017 04:01 PM
3	TurnOn	Turn the heat on	on	heat		11/09/2017 03:49 PM
4	TurnOn	Switch on the kitchen fan	on	fan	kitchen	11/09/2017 03:29 PM
5	TurnOff	Turn off bedroom lights	off	lights	bedroom	11/09/2017 03:27 PM
6	TurnOff	Turn off air conditioning	off	air conditioning		11/09/2017 03:22 PM
7	TurnOff	kill the lights		lights		11/09/2017 02:49 PM
8	Dim	dim the lights	dim	lights		11/09/2017 02:43 PM
9	Other	hi how are you				11/09/2017 02:39 PM
10	Other	answer the phone		phone		11/09/2017 01:19 PM
11	Other	are you there				11/09/2017 01:08 PM
12	Other	help				11/09/2017 12:40 PM
13	Other	testing the circuit				11/09/2017 12:39 PM

You'll see that the **RequestType** column could be intents, the **Request** column shows an example utterance, and the other fields could be entities if they actually occur in the utterance. Because we have intents, entities, and example utterances, we have what we need to create a sample app.

Steps to generate a LUIS app from non-LUIS data

To generate a new LUIS app from the source file, first you parse the data from the CSV file and convert this data to a format that you can upload to LUIS using the Authoring API. From the parsed data, you gather information on what intents and entities are there. Then you make API calls to create the app, and add intents and entities that were gathered from the parsed data. Once you have created the LUIS app, you can add the example utterances from the parsed data. You can see this flow in the last part of the code below. Copy or download this code and save it in

`index.js`.

```
var path = require('path');

const parse = require('./_parse');
const createApp = require('./_create');
const addEntities = require('./_entities');
const addIntents = require('./_intents');
const upload = require('./_upload');

// Change these values
const LUIS_programmaticKey = "YOUR_PROGRAMMATIC_KEY";
const LUIS_appName = "Sample App";
const LUIS_appCulture = "en-us";
const LUIS_versionId = "0.1";

// NOTE: final output of add-utterances api named utterances.upload.json
const downloadFile = "./IoT.csv";
const uploadFile = "./utterances.json"

// The app ID is returned from LUIS when your app is created
var LUIS_appId = ""; // default app ID
var intents = [];
var entities = [];


/* add utterances parameters */
var configAddUtterances = {
    LUIS_subscriptionKey: LUIS_programmaticKey,
    LUIS_appId: LUIS_appId,
    LUIS_versionId: LUIS_versionId,
    inFile: path.join(__dirname, uploadFile),
    batchSize: 100,
    uri: "https://westus.api.cognitive.microsoft.com/luis/api/v2.0/apps/{appId}/versions/{versionId}/examples"
};

/* create app parameters */
var configCreateApp = {
    LUIS_subscriptionKey: LUIS_programmaticKey,
    LUIS_versionId: LUIS_versionId,
    appName: LUIS_appName,
    culture: LUIS_appCulture,
    uri: "https://westus.api.cognitive.microsoft.com/luis/api/v2.0/apps/"
};

/* add intents parameters */
var configAddIntents = {
    LUIS_subscriptionKey: LUIS_programmaticKey,
    LUIS_appId: LUIS_appId,
    LUIS_versionId: LUIS_versionId,
    intentList: intents,
    uri: "https://westus.api.cognitive.microsoft.com/luis/api/v2.0/apps/{appId}/versions/{versionId}/intents"
};

/* add entities parameters */
var configAddEntities = {
    LUIS_subscriptionKey: LUIS_programmaticKey,
    LUIS_appId: LUIS_appId,
    LUIS_versionId: LUIS_versionId,
    entityList: intents,
    uri: "https://westus.api.cognitive.microsoft.com/luis/api/v2.0/apps/{appId}/versions/{versionId}/entities"
};

/* input and output files for parsing CSV */
var configParse = {
    inFile: path.join(__dirname, downloadFile),
    outFile: path.join(__dirname, uploadFile)
};
```

```
// Parse CSV
parse(configParse)
    .then((model) => {
        // Save intent and entity names from parse
        intents = model.intents;
        entities = model.entities;
        // Create the LUIS app
        return createApp(configCreateApp);

    }).then((appId) => {
        // Add intents
        LUIS_appId = appId;
        configAddIntents.LUIS_appId = appId;
        configAddIntents.intentList = intents;
        return addIntents(configAddIntents);

    }).then(() => {
        // Add entities
        configAddEntities.LUIS_appId = LUIS_appId;
        configAddEntities.entityList = entities;
        return addEntities(configAddEntities);

    }).then(() => {
        // Add example utterances to the intents in the app
        configAddUtterances.LUIS_appId = LUIS_appId;
        return upload(configAddUtterances);

    }).catch(err => {
        console.log(err.message);
    });
```

Parse the CSV

The column entries that contain the utterances in the CSV have to be parsed into a JSON format that LUIS can understand. This JSON format must contain an `intentName` field that identifies the intent of the utterance. It must also contain an `entityLabels` field, which can be empty if there are no entities in the utterance.

For example, the entry for "Turn on the lights" maps to this JSON:

```
        {
            "text": "Turn on the lights",
            "intentName": "TurnOn",
            "entityLabels": [
                {
                    "entityName": "Operation",
                    "startCharIndex": 5,
                    "endCharIndex": 6
                },
                {
                    "entityName": "Device",
                    "startCharIndex": 12,
                    "endCharIndex": 17
                }
            ]
        }
```

In this example, the `intentName` comes from the user request under the **Request** column heading in the CSV file, and the `entityName` comes from the other columns with key information. For example, if there's an entry for **Operation** or **Device**, and that string also occurs in the actual request, then it can be labeled as an entity. The following code demonstrates this parsing process. You can copy or download it and save it to `_parse.js`.

```
// node 7.x
// built with streams for larger files

const fse = require('fs-extra');
const path = require('path');
const lineReader = require('line-reader');
const babyparse = require('babyparse');
const Promise = require('bluebird');

const intent_column = 0;
const utterance_column = 1;
var entityNames = [];

var eachLine = Promise.promisify(lineReader.eachLine);

function listOfIntents(intents) {
    return intents.reduce(function (a, d) {
        if (a.indexOf(d.intentName) === -1) {
            a.push(d.intentName);
        }
        return a;
    }, []);

}

function listOfEntities(utterances) {
    return utterances.reduce(function (a, d) {
        d.entityLabels.forEach(function(entityLabel) {
            if (a.indexOf(entityLabel.entityName) === -1) {
                a.push(entityLabel.entityName);
            }
        }, this);
        return a;
    }, []);
}

var utterance = function (rowAsString) {

    let json = {
        "text": "",
        "intentName": "",
        "entityLabels": [],
    };

    if (!rowAsString) return json;

    let dataRow = babyparse.parse(rowAsString);
    // Get intent name and utterance text
    json.intentName = dataRow.data[0][intent_column];
    json.text = dataRow.data[0][utterance_column];
    // For each column heading that may be an entity, search for the element in this column in the utterance.
    entityNames.forEach(function (entityName) {
        entityToFind = dataRow.data[0][entityName.column];
        if (entityToFind != "") {
            strInd = json.text.indexOf(entityToFind);
            if (strInd > -1) {
                let entityLabel = {
                    "entityName": entityName.name,
                    "startCharIndex": strInd,
                    "endCharIndex": strInd + entityToFind.length - 1
                }
                json.entityLabels.push(entityLabel);
            }
        }
    }, this);
    return json;

};
```

```
const convert = async (config) => {

    try {

        var i = 0;

        // get inFile stream
        inFileStream = await fse.createReadStream(config.inFile, 'utf-8')

        // create out file
        var myOutFile = await fse.createWriteStream(config.outFile, 'utf-8');
        var utterances = [];

        // read 1 line at a time
        return eachLine(inFileStream, (line) => {

            // skip first line with headers
            if (i++ == 0) {

                // csv to baby parser object
                let dataRow = babyparse.parse(line);

                // populate entityType list
                var index = 0;
                dataRow.data[0].forEach(function (element) {
                    if ((index != intent_column) && (index != utterance_column)) {
                        entityNames.push({ name: element, column: index });
                    }
                    index++;
                }, this);

                return;
            }

            // transform utterance from csv to json
            utterances.push(utterance(line));

        }).then(() => {
            console.log("intents: " + JSON.stringify(listOfIntents(utterances)));
            console.log("entities: " + JSON.stringify(listOfEntities(utterances)));
            myOutFile.write(JSON.stringify({ "converted_date": new Date().toLocaleString(), "utterances":
utterances }));
            myOutFile.end();
            console.log("parse done");
            console.log("JSON file should contain utterances. Next step is to create an app with the intents
and entities it found.");

            var model =
            {
                intents: listOfIntents(utterances),
                entities: listOfEntities(utterances)
            }
            return model;

        });

    } catch (err) {
        throw err;
    }

}

module.exports = convert;
```

Create the LUIS app

Once the data has been parsed into JSON, we need a LUIS app to add it to. The following code creates the LUIS app. Copy or download it, and save it into `_create.js`.

```
// node 7.x
// uses async/await - promises

var rp = require('request-promise');
var fse = require('fs-extra');
var path = require('path');



// main function to call
// Call Apps_Create
var createApp = async (config) => {

        try {

            // JSON for the request body
            // { "name": MyAppName, "culture": "en-us"}
            var jsonBody = {
                "name": config.appName,
                "culture": config.culture
            };

            // Create a LUIS app
            var createAppPromise = callCreateApp({
                uri: config.uri,
                method: 'POST',
                headers: {
                    'Ocp-Apim-Subscription-Key': config.LUIS_subscriptionKey
                },
                json: true,
                body: jsonBody
            });

            let results = await createAppPromise;

            // Create app returns an app ID
            let appId = results.response;
            console.log(`Called createApp, created app with ID ${appId}`);
            return appId;


        } catch (err) {
            console.log(`Error creating app:  ${err.message} `);
            throw err;
        }

    }

// Send JSON as the body of the POST request to the API
var callCreateApp = async (options) => {
    try {

        var response;
        if (options.method === 'POST') {
            response = await rp.post(options);
        } else if (options.method === 'GET') { // TODO: There's no GET for create app
            response = await rp.get(options);
        }
        // response from successful create should be the new app ID
        return { response };

    } catch (err) {
        throw err;
    }
}
```

```
}

module.exports = createApp;
```

Add intents

Once you have an app, you need to intents to it. The following code creates the LUIS app. Copy or download it, and save it into `_intents.js`.

```
var rp = require('request-promise');
var fse = require('fs-extra');
var path = require('path');
var request = require('requestretry');

// time delay between requests
const delayMS = 1000;

// retry recount
const maxRetry = 5;

// retry request if error or 429 received
var retryStrategy = function (err, response, body) {
    let shouldRetry = err || (response.statusCode === 429);
    if (shouldRetry) console.log("retrying add intent...");
    return shouldRetry;
}

// Call add-intents
var addIntents = async (config) => {
    var intentPromises = [];
    config.uri = config.uri.replace("{appId}", config.LUIS_appId).replace("{versionId}",
config.LUIS_versionId);

    config.intentList.forEach(function (intent) {
        config.intentName = intent;
        try {

            // JSON for the request body
            var jsonBody = {
                "name": config.intentName,
            };

            // Create an intent
            var addIntentPromise = callAddIntent({
                // uri: config.uri,
                url: config.uri,
                fullResponse: false,
                method: 'POST',
                headers: {
                    'Ocp-Apim-Subscription-Key': config.LUIS_subscriptionKey
                },
                json: true,
                body: jsonBody,
                maxAttempts: maxRetry,
                retryDelay: delayMS,
                retryStrategy: retryStrategy
            });
            intentPromises.push(addIntentPromise);

            console.log(`Called addIntents for intent named ${intent}.`);

        } catch (err) {
            console.log(`Error in addIntents:  ${err.message} `);

        }
```

```
    }, this);

    let results = await Promise.all(intentPromises);
    console.log(`Results of all promises = ${JSON.stringify(results)}`);
    let response = results;


}

// Send JSON as the body of the POST request to the API
var callAddIntent = async (options) => {
    try {

        var response;
        response = await request(options);
        return { response: response };

    } catch (err) {
        console.log(`Error in callAddIntent:  ${err.message} `);
    }
}

module.exports = addIntents;
```

Add entities

The following code adds the entities to the LUIS app. Copy or download it, and save it into `_entities.js`.

```
// node 7.x
// uses async/await - promises

const request = require("requestretry");
var rp = require('request-promise');
var fse = require('fs-extra');
var path = require('path');

// time delay between requests
const delayMS = 1000;

// retry recount
const maxRetry = 5;

// retry request if error or 429 received
var retryStrategy = function (err, response, body) {
    let shouldRetry = err || (response.statusCode === 429);
    if (shouldRetry) console.log("retrying add entity...");
    return shouldRetry;
}

// main function to call
// Call add-entities
var addEntities = async (config) => {
    var entityPromises = [];
    config.uri = config.uri.replace("{appId}", config.LUIS_appId).replace("{versionId}",
config.LUIS_versionId);

    config.entityList.forEach(function (entity) {
        try {
            config.entityName = entity;
            // JSON for the request body
            // { "name": MyEntityName}
            var jsonBody = {
                "name": config.entityName,
            };

            // Create an app
```

```
            var addEntityPromise = callAddEntity({
                url: config.uri,
                fullResponse: false,
                method: 'POST',
                headers: {
                    'Ocp-Apim-Subscription-Key': config.LUIS_subscriptionKey
                },
                json: true,
                body: jsonBody,
                maxAttempts: maxRetry,
                retryDelay: delayMS,
                retryStrategy: retryStrategy
            });
            entityPromises.push(addEntityPromise);

            console.log(`called addEntity for entity named ${entity}.`);

        } catch (err) {
            console.log(`Error in addEntities:  ${err.message} `);
            //throw err;
        }
    }, this);
    let results = await Promise.all(entityPromises);
    console.log(`Results of all promises = ${JSON.stringify(results)}`);
    let response = results;// await fse.writeJson(createResults.json, results);


}

// Send JSON as the body of the POST request to the API
var callAddEntity = async (options) => {
    try {

        var response;
        response = await request(options);
        return { response: response };

    } catch (err) {
        console.log(`error in callAddEntity: ${err.message}`);
    }
}

module.exports = addEntities;
```

Add utterances

Once the entities and intents have been defined in the LUIS app, you can add the utterances. The following code uses the Utterances_AddBatch API which allows you to add up to 100 utterances at a time. Copy or download it, and save it into `_upload.js`.

```
// node 7.x
// uses async/await - promises

var rp = require('request-promise');
var fse = require('fs-extra');
var path = require('path');
var request = require('requestretry');

// time delay between requests
const delayMS = 500;

// retry recount
const maxRetry = 5;

// retry request if error or 429 received
```

```
var retryStrategy = function (err, response, body) {
    let shouldRetry = err || (response.statusCode === 429);
    if (shouldRetry) console.log("retrying add examples...");
    return shouldRetry;
}

// main function to call
var upload = async (config) => {

    try{

        // read in utterances
        var entireBatch = await fse.readJson(config.inFile);

        // break items into pages to fit max batch size
        var pages = getPagesForBatch(entireBatch.utterances, config.batchSize);

        var uploadPromises = [];

        // load up promise array
        pages.forEach(page => {
            config.uri =
"https://westus.api.cognitive.microsoft.com/luis/api/v2.0/apps/{appId}/versions/{versionId}/examples".replace("
{appId}", config.LUIS_appId).replace("{versionId}", config.LUIS_versionId)
            var pagePromise = sendBatchToApi({
                url: config.uri,
                fullResponse: false,
                method: 'POST',
                headers: {
                    'Ocp-Apim-Subscription-Key': config.LUIS_subscriptionKey
                },
                json: true,
                body: page,
                maxAttempts: maxRetry,
                retryDelay: delayMS,
                retryStrategy: retryStrategy
            });

            uploadPromises.push(pagePromise);
        })

        //execute promise array

        let results =  await Promise.all(uploadPromises)
        console.log(`\n\nResults of all promises = ${JSON.stringify(results)}`);
        let response = await fse.writeJson(config.inFile.replace('.json','.upload.json'),results);

        console.log("upload done");

    } catch(err){
        throw err;
    }

}
// turn whole batch into pages batch
// because API can only deal with N items in batch
var getPagesForBatch = (batch, maxItems) => {

    try{
        var pages = [];
        var currentPage = 0;

        var pageCount = (batch.length % maxItems == 0) ? Math.round(batch.length / maxItems) :
Math.round((batch.length / maxItems) + 1);

        for (let i = 0;i<pageCount;i++){

            var currentStart = currentPage * maxItems;
            var currentEnd = currentStart + maxItems;
```

```
            var pagedBatch = batch.slice(currentStart,currentEnd);

            var j = 0;
            pagedBatch.forEach(item=>{
                item.ExampleId = j++;
            });

            pages.push(pagedBatch);

            currentPage++;
        }
        return pages;
    }catch(err){
        throw(err);
    }
}

// send json batch as post.body to API
var sendBatchToApi = async (options) => {
    try {

        var response = await request(options);
        //return {page: options.body, response:response};
        return {response:response};
    }catch(err){
        throw err;
    }
}

module.exports = upload;
```

Run the code

Install Node.js dependencies

Install the Node.js dependencies from NPM in the terminal/command line.

```
> npm install
```

Change Configuration Settings

In order to use this application, you need to change the values in the index.js file to your own subscription key, and provide the name you want the app to have. You can also set the app's culture or change the version number.

Open the index.js file, and change these values at the top of the file.

```
// Change these values
const LUIS_programmaticKey = "YOUR_PROGRAMMATIC_KEY";
const LUIS_appName = "Sample App";
const LUIS_appCulture = "en-us";
const LUIS_versionId = "0.1";
```

Run the script

Run the script from a terminal/command line with Node.js.

```
> node index.js
```

or

```
> npm start
```

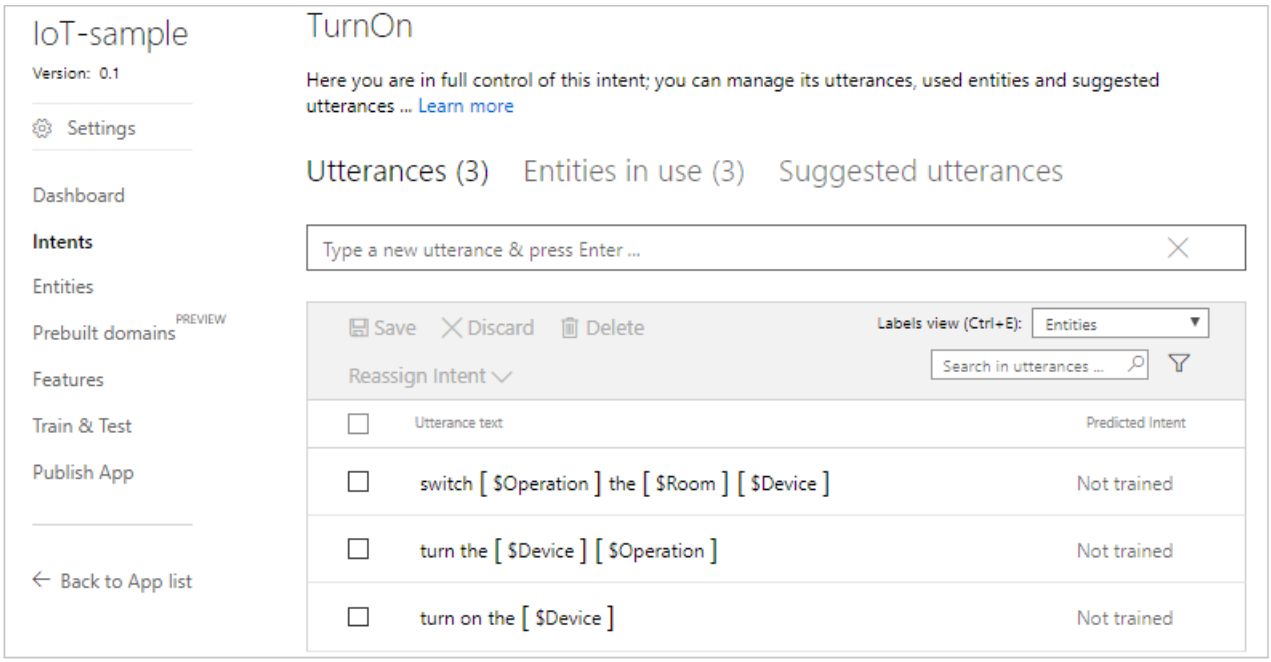
Application progress

While the application is running, the command line shows progress. The command line output includes the format of the responses from LUIS.

```
> node index.js
intents: ["TurnOn","TurnOff","Dim","Other"]
entities: ["Operation","Device","Room"]
parse done
JSON file should contain utterances. Next step is to create an app with the intents and entities it found.
Called createApp, created app with ID 314b306c-0033-4e09-92ab-94fe5ed158a2
Called addIntents for intent named TurnOn.
Called addIntents for intent named TurnOff.
Called addIntents for intent named Dim.
Called addIntents for intent named Other.
Results of all calls to addIntent = [{"response":"e7eaf224-8c61-44ed-a6b0-2ab4dc56f1d0"},{"response":"a8a17efd-
f01c-488d-ad44-a31a818cf7d7"},{"response":"bc7c32fc-14a0-4b72-bad4-d345d807f965"},{"response":"727a8d73-cd3b-
4096-bc8d-d7cfba12eb44"}]
called addEntity for entity named Operation.
called addEntity for entity named Device.
called addEntity for entity named Room.
Results of all calls to addEntity= [{"response":"6a7e914f-911d-4c6c-a5bc-377afdce4390"},{"response":"56c35237-
593d-47f6-9d01-2912fa488760"},{"response":"f1dd440c-2ce3-4a20-a817-a57273f169f3"}]
retrying add examples...

Results of add utterances = [{"response":[{"value":{"UtteranceText":"turn on the lights","ExampleId":-
67649},"hasError":false},{"value":{"UtteranceText":"turn the heat on","ExampleId":-69067},"hasError":false},
{"value":{"UtteranceText":"switch on the kitchen fan","ExampleId":-3395901},"hasError":false},{"value":
{"UtteranceText":"turn off bedroom lights","ExampleId":-85402},"hasError":false},{"value":
{"UtteranceText":"turn off air conditioning","ExampleId":-8991572},"hasError":false},{"value":
{"UtteranceText":"kill the lights","ExampleId":-70124},"hasError":false},{"value":{"UtteranceText":"dim the
lights","ExampleId":-174358},"hasError":false},{"value":{"UtteranceText":"hi how are you","ExampleId":-
143722},"hasError":false},{"value":{"UtteranceText":"answer the phone","ExampleId":-69939},"hasError":false},
{"value":{"UtteranceText":"are you there","ExampleId":-149588},"hasError":false},{"value":
{"UtteranceText":"help","ExampleId":-81949},"hasError":false},{"value":{"UtteranceText":"testing the
circuit","ExampleId":-11548708},"hasError":false}]}]
upload done
```

Open the LUIS app in LUIS.ai

Once the script completes, you can log in to luis.ai and see the LUIS app you just created under **My Apps**. You should be able to see the utterances you added under the **TurnOn**, **TurnOff**, and **None** intents.



Next steps

Test and train your app in LUIS.ai

Additional resources

This sample applications uses the following LUIS APIs:

- create app
- add intents
- add entities
- add utterances

Intents in LUIS

9/18/2017 • 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.

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

9/20/2017 • 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, 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. 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.
Simple	A simple entity is a generic entity that describes a single concept.
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.

TYPE	DESCRIPTION
Composite	A composite entity is made up of other entities that form 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.

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

9/18/2017 • 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.
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.

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.

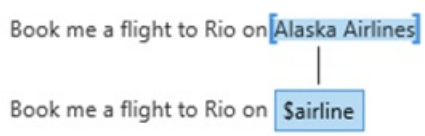
Identify your entities

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.

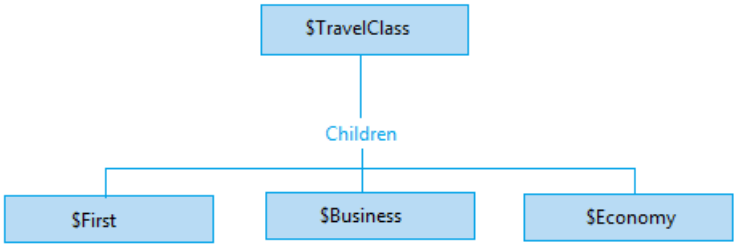
Simple entity

A simple entity describes a single concept.



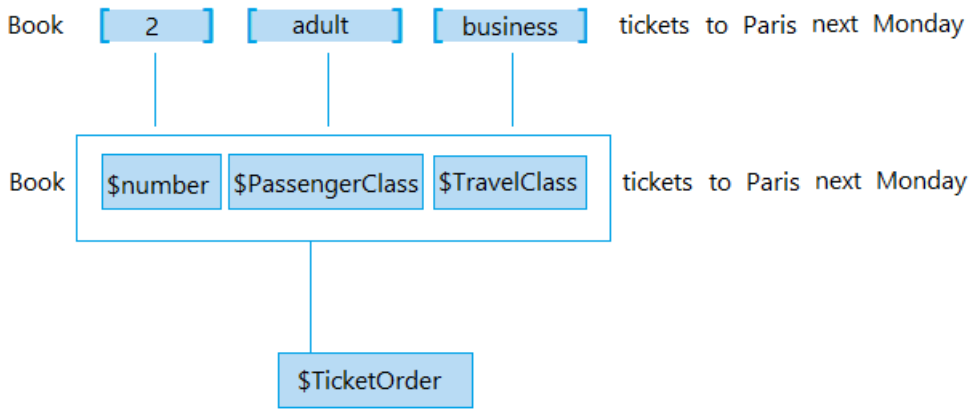
Hierarchical entity

A hierarchical entity represents a category and its members.



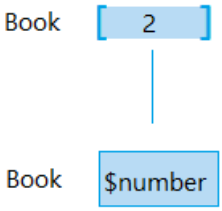
Composite entity

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



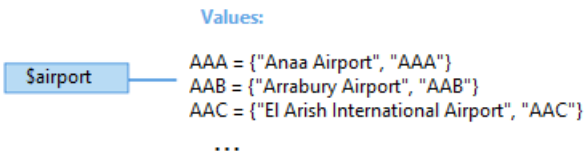
Prebuilt entity

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.

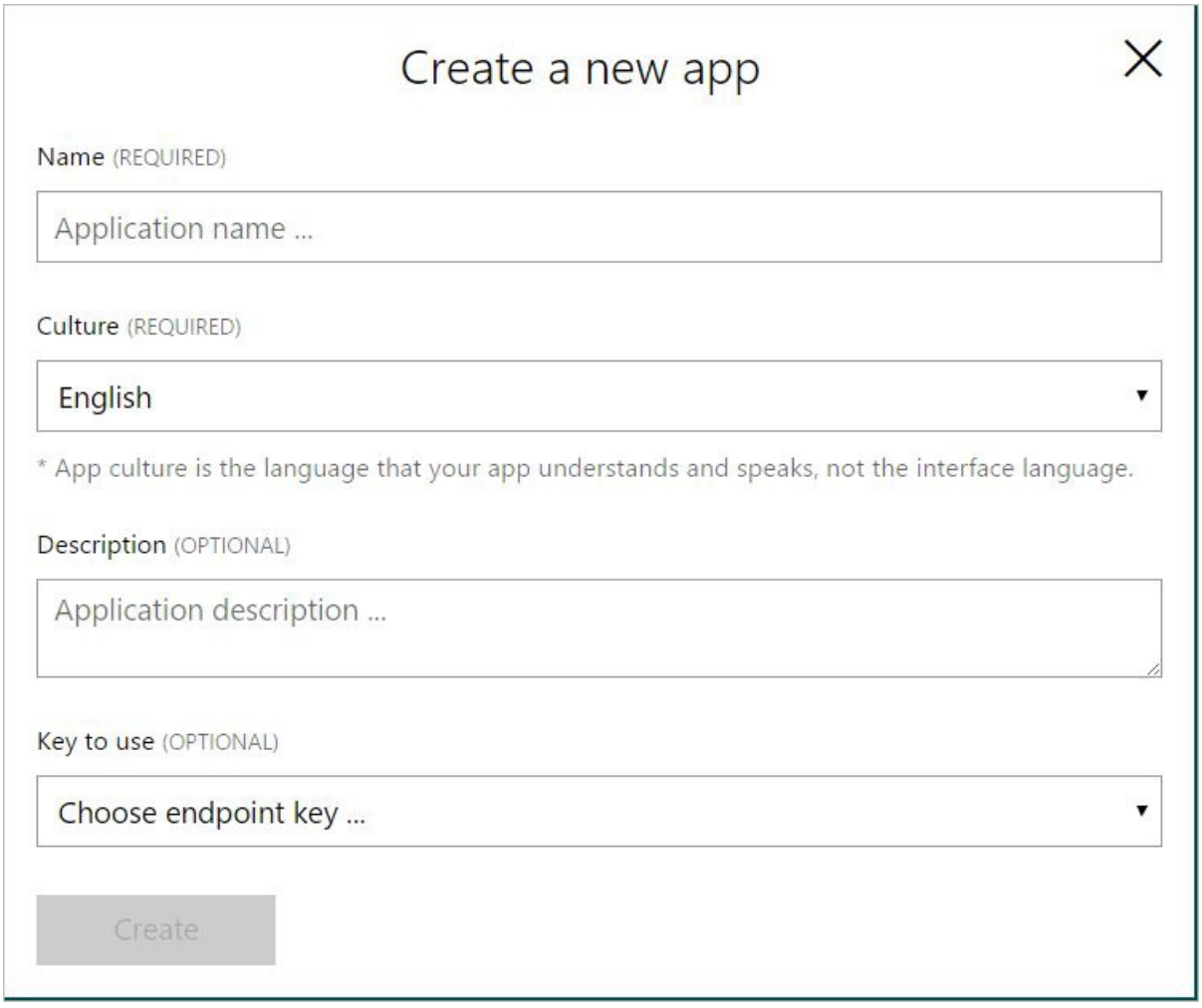
Create a New App

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:

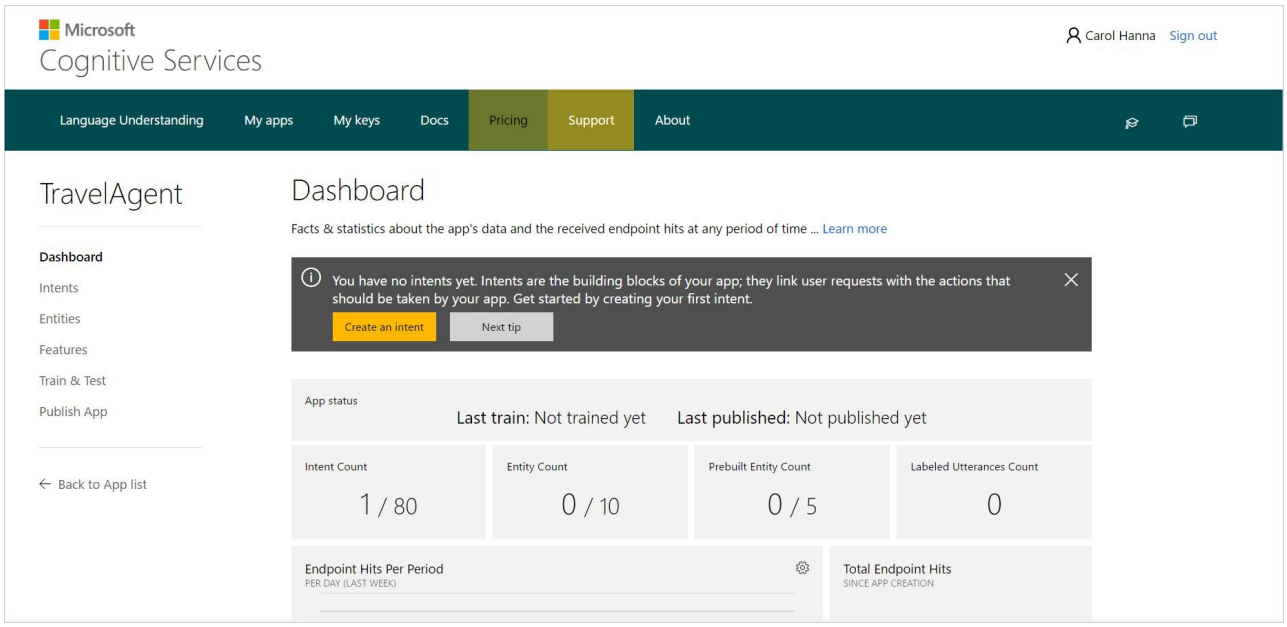
1. On **My Apps** page, click **New App**.
2. In the dialog box, name your application "TravelAgent".



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

9/18/2017 • 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.

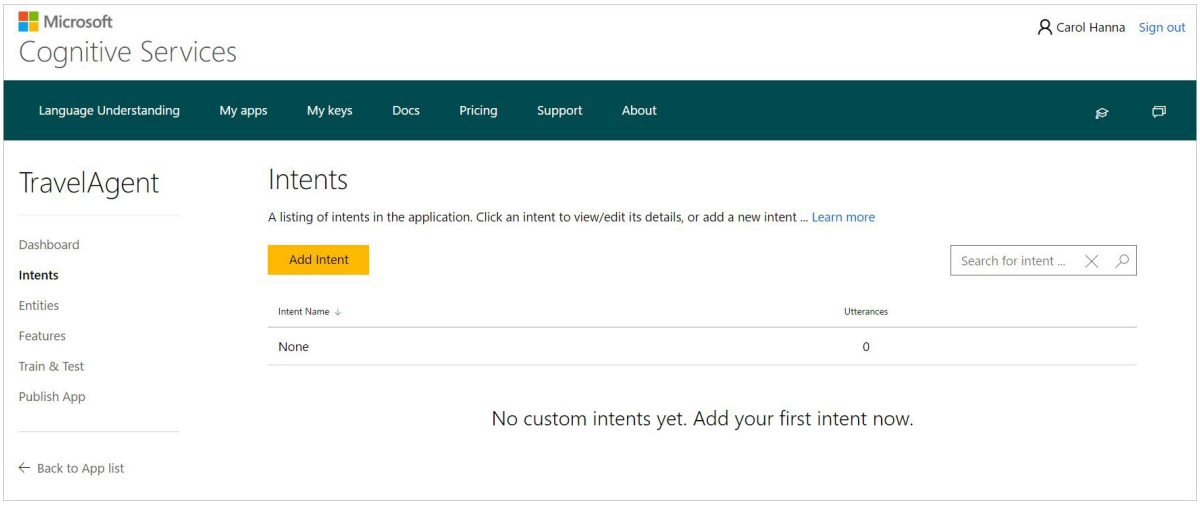
All applications come with the predefined intent, **None**. You should teach it to recognize user statements that are irrelevant to the app. For example, if a user says "Get me a great cookie recipe" in a travel agent app, label that utterance with the **None** intent.

You add and manage your intents from the **Intents** page that is accessed by clicking **Intents** in your application's left panel.

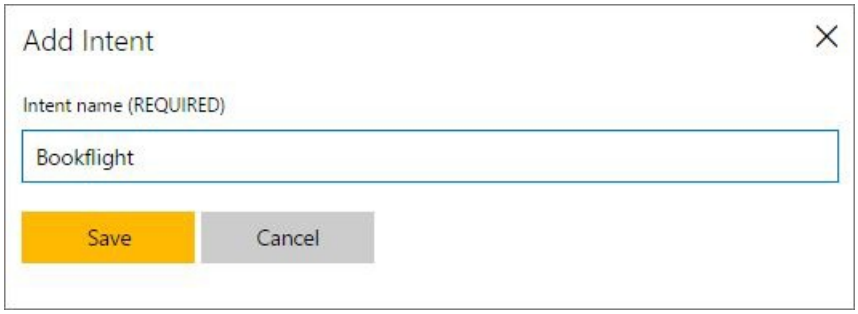
The following procedure demonstrates how to add the "Bookflight" intent in the TravelAgent app.

To add an intent:

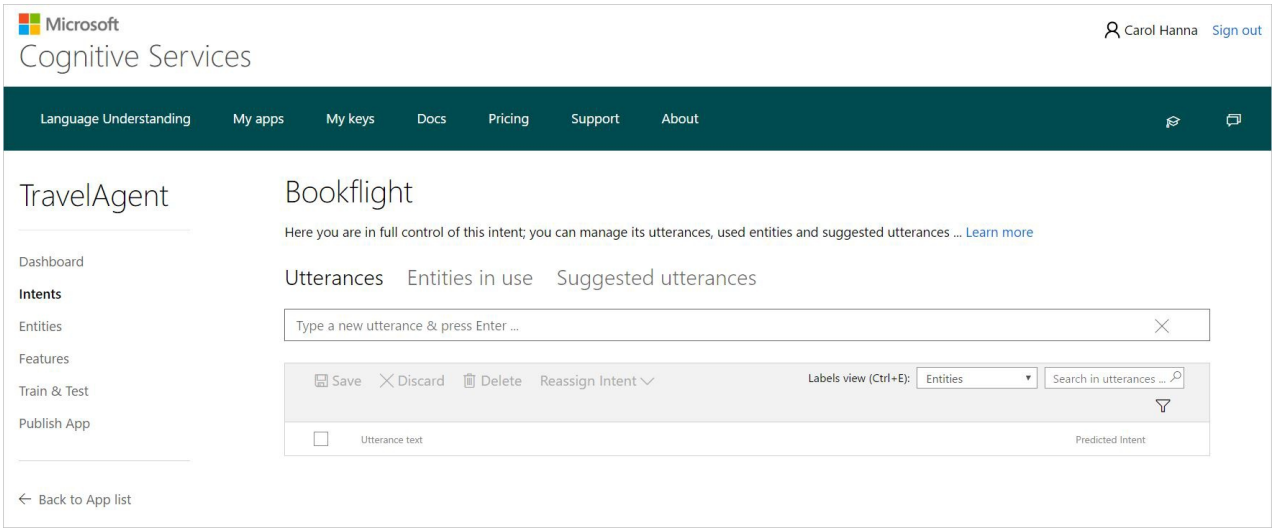
1. Open your app (for example, TravelAgent) by clicking its name on **My Apps** page, and then click **Intents** in the left panel.
2. On the **Intents** page, click **Add intent**.



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

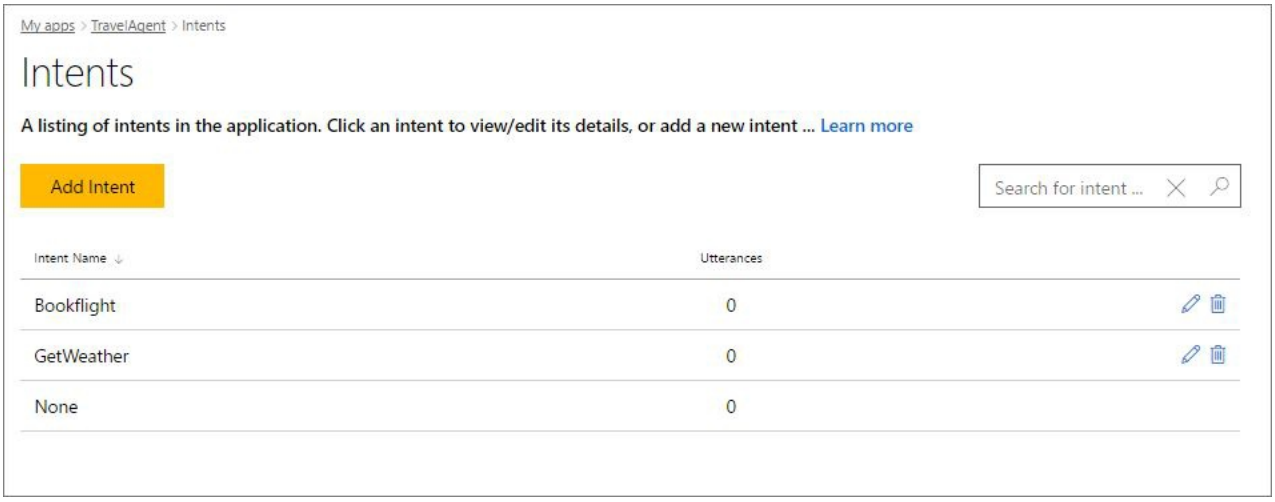


This takes you directly to the intent details page of the newly added intent "Bookflight", like the following screenshot, to add utterances for this intent. For instructions on adding utterances, see Add example utterances.



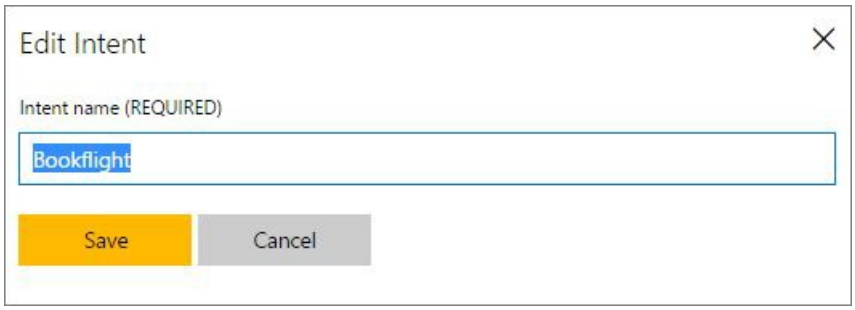
Manage your intents

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.

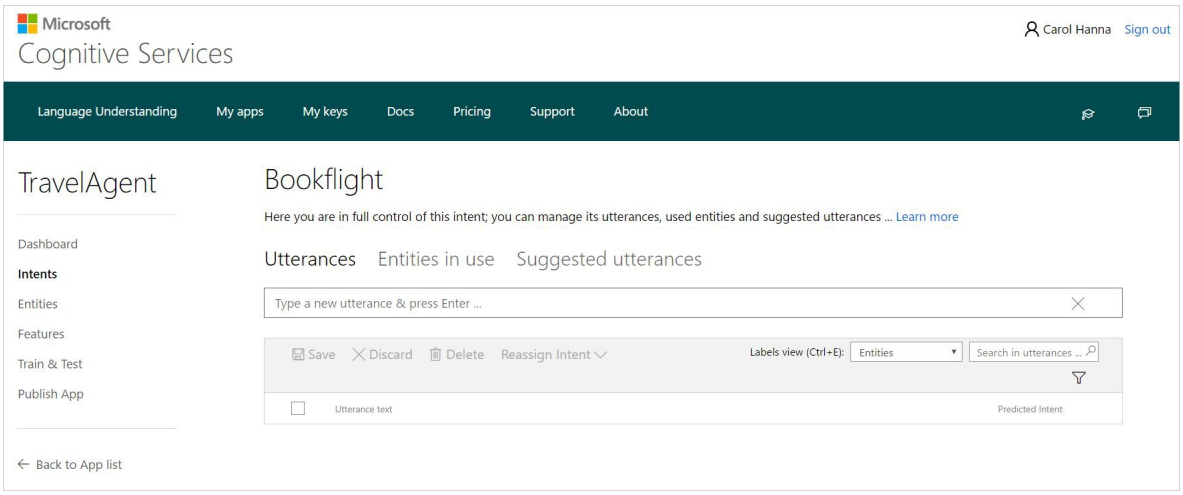
To train LUIS, you need to add example utterances for each intent in your app. LUIS learns from these utterances and your app is able to generalize and understand similar contexts. By constantly adding more utterances and labeling them, you are enhancing your application's language learning experience.

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".

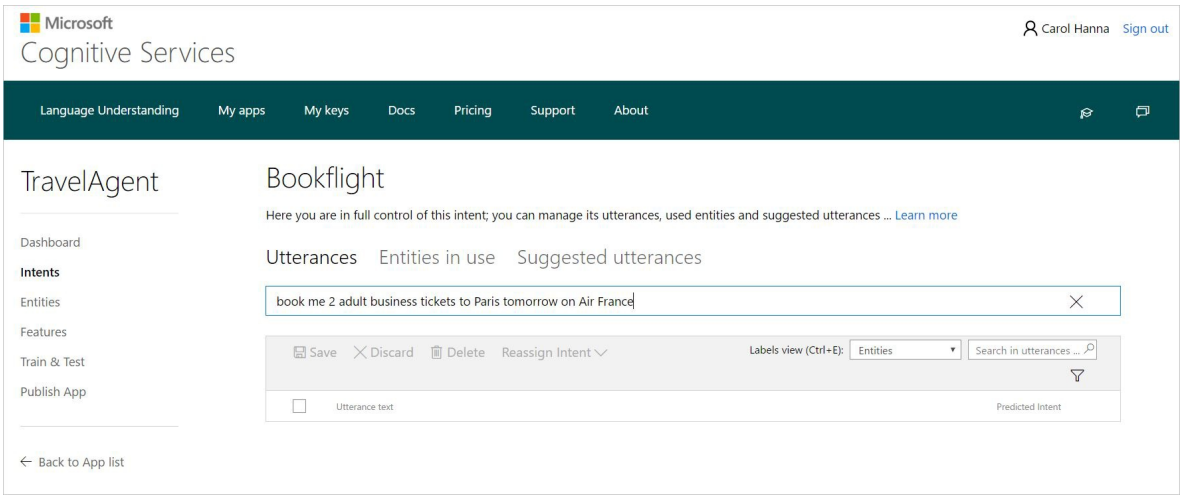
Utterances are added to an intent on the **Utterances** tab of the intent page. The following steps describe how to add example utterances to an intent. In this example we use the "BookFlight" intent in the TravelAgent app.

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.

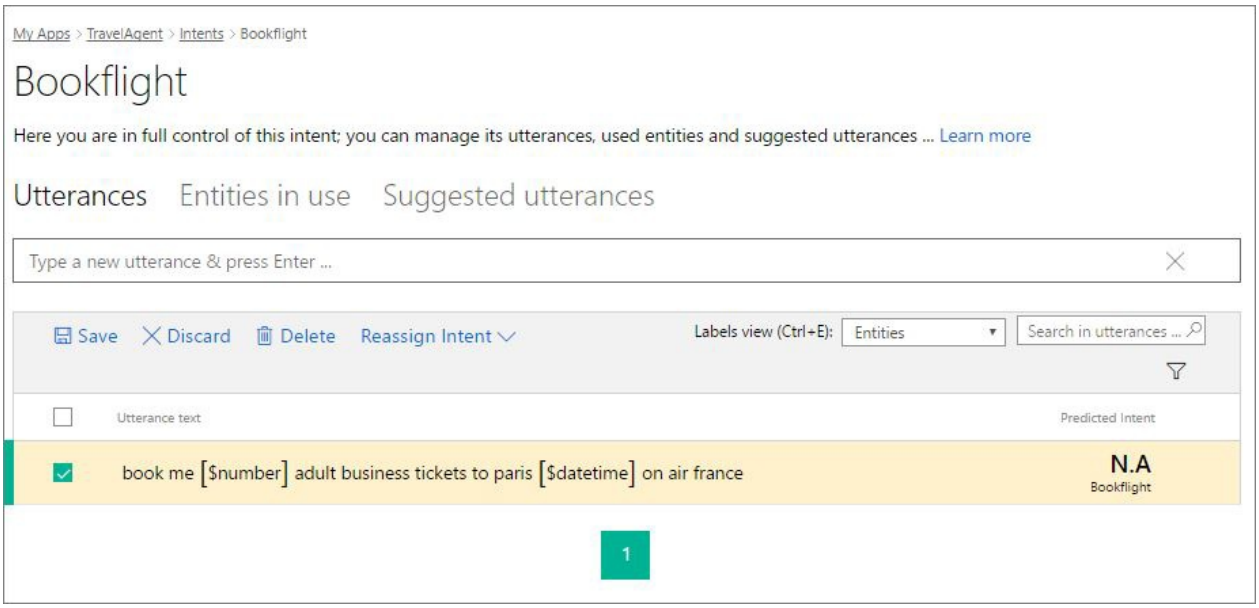
Intent label

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.



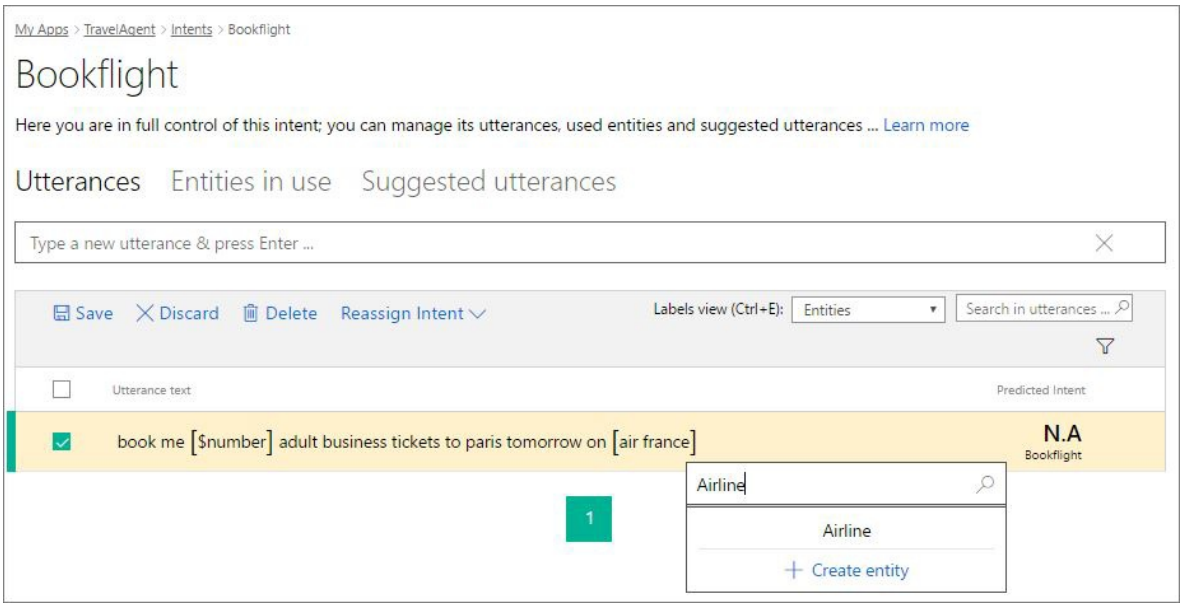
To Learn more about prebuilt entities and how to add them, see Add entities.

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.

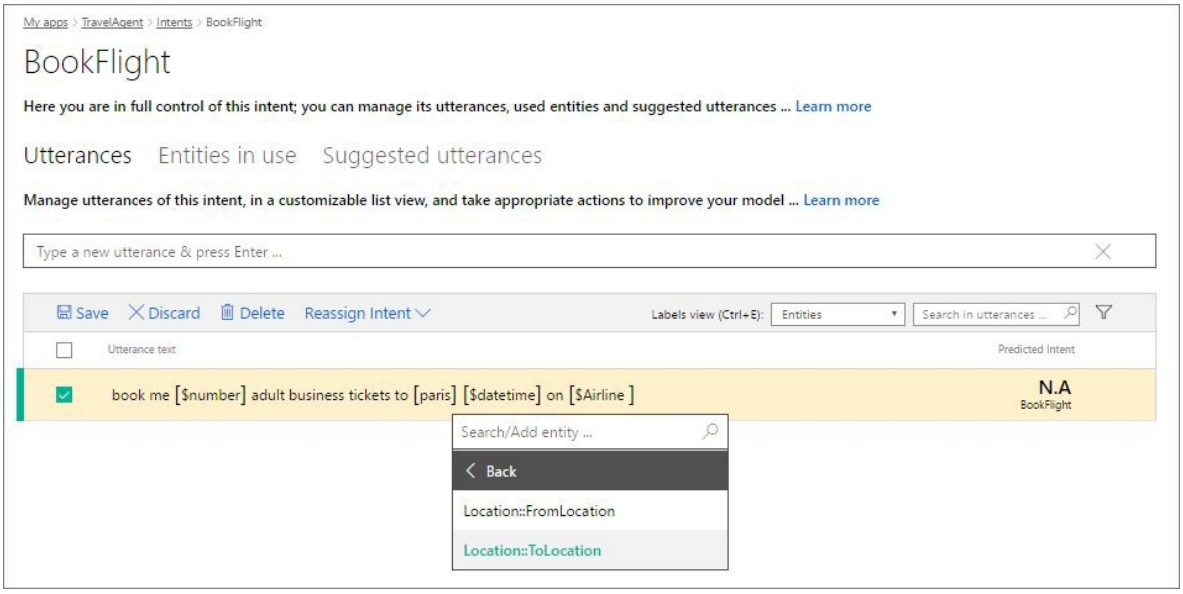
> **NOTE**
>
> When selecting words to label them as entities:
>
> - For a single word, just click it.
> - For a set of two or more words, click at the beginning and then at the end of the set.

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**.



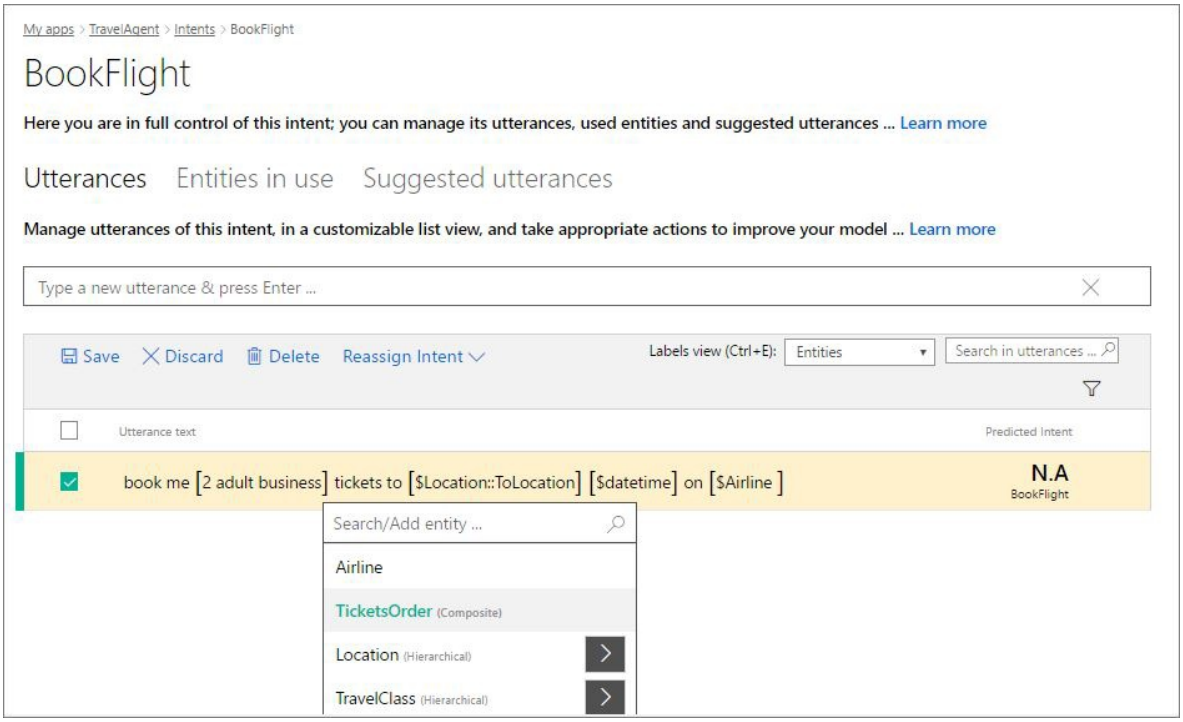
> **NOTE**
>
> This way is used to create a simple entity on the spot (while labeling utterances). Hierarchical and composite entities can only be created from the **Entities** page. For more instructions, see Add entities.

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:

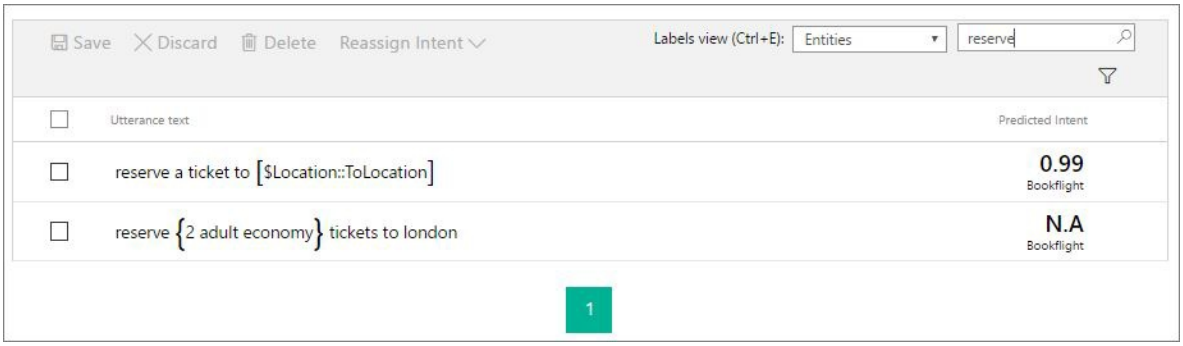
- Click the entity you want to remove its label and click **Remove label** in the entity drop-down box that appears. Then, click **Save** to save this change.

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.

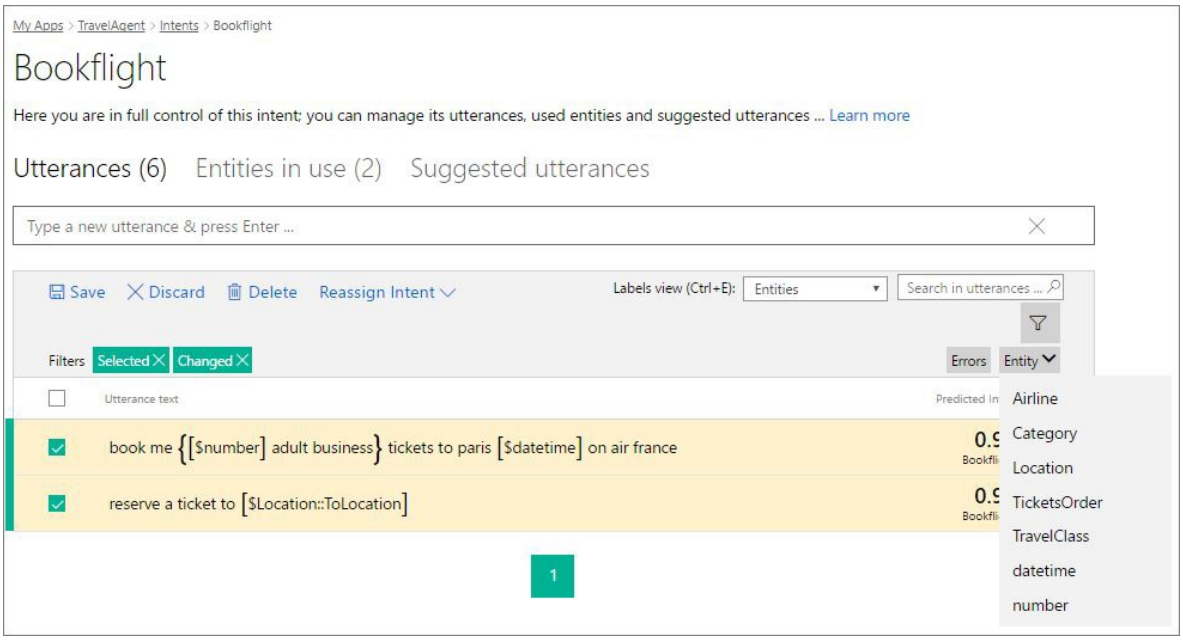
You can apply one or more filters on utterances, as needed. These are the available filters that you can use:

- **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  .

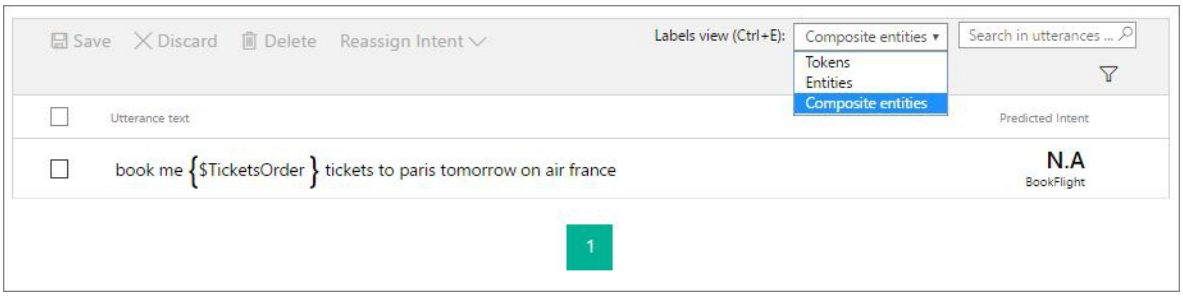
Choose labels view in utterances

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

9/18/2017 • 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. These terms are relevant to the "BookFlight" intent, so you can add them as entities.

You do not need to create entities for every concept in your app, but only for the concepts that the app requires to take action.

You can add, edit, or delete entities in your app through the **Entities list** on the **Entities** page. LUIS offers many types of entities. These include prebuilt entities, custom entities and list entities.

Add prebuilt entities

LUIS provides a set of prebuilt, system-defined entities. These entities cover many common concepts such as date, age, temperature, percentage, dimension, cardinal and ordinal numbers.

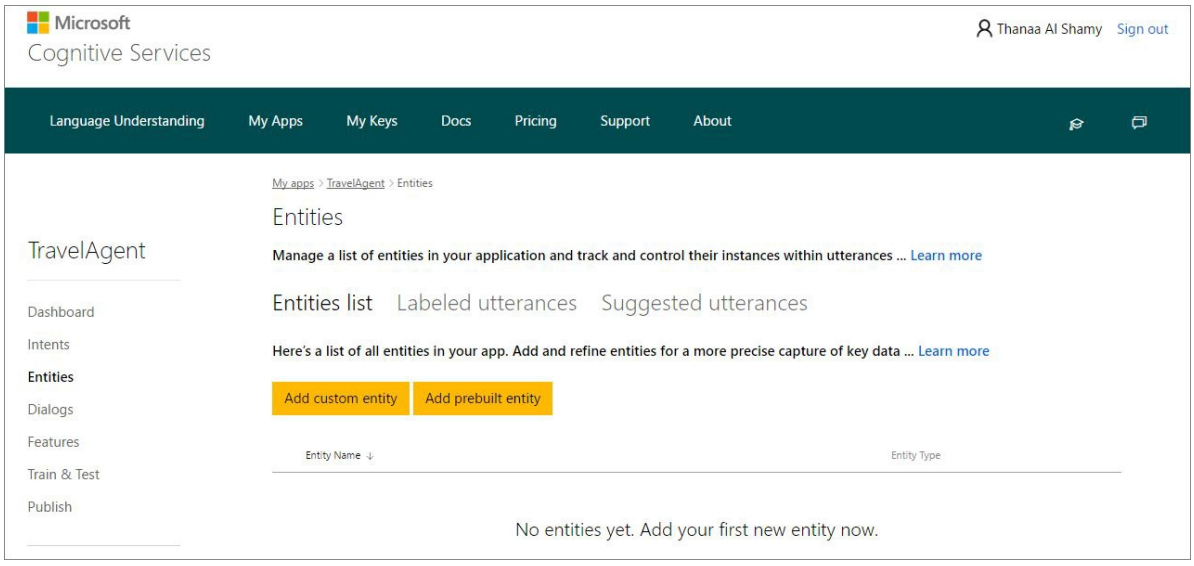
For example, the TravelAgent app may receive a request like "Book me a flight to Boston on May 4." Your app needs to understand date and time words in order to interpret the request properly. Rather than creating entities for such concepts from scratch, you can enable a ready-made prebuilt entity called "datetimeV2."

As another example, a request like "Book me the first flight to Boston" might be sent to a travel agent app. This requires understanding of the ordinal words "first" and "second". You can add a prebuilt entity called "ordinal", for your app to recognize ordinal numbers.

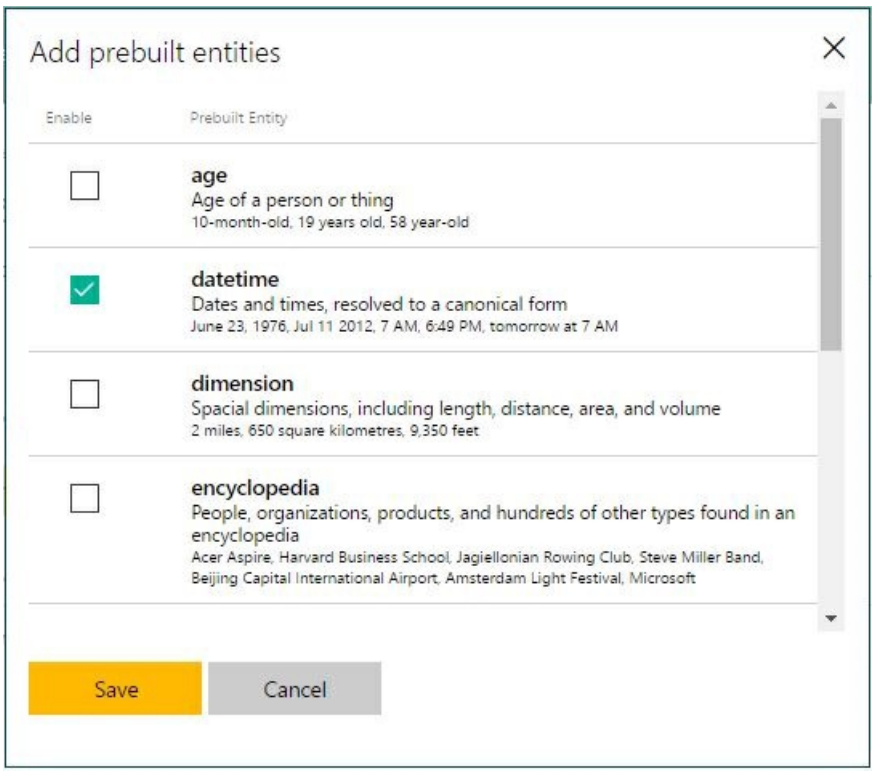
For a full list of prebuilt entities and their use, see Prebuilt Entities List.

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 (for example, "datetimeV2"). Then click **Save**.



Custom entities

Custom entities are the entities you create in your LUIS app. The following are types of custom entities:

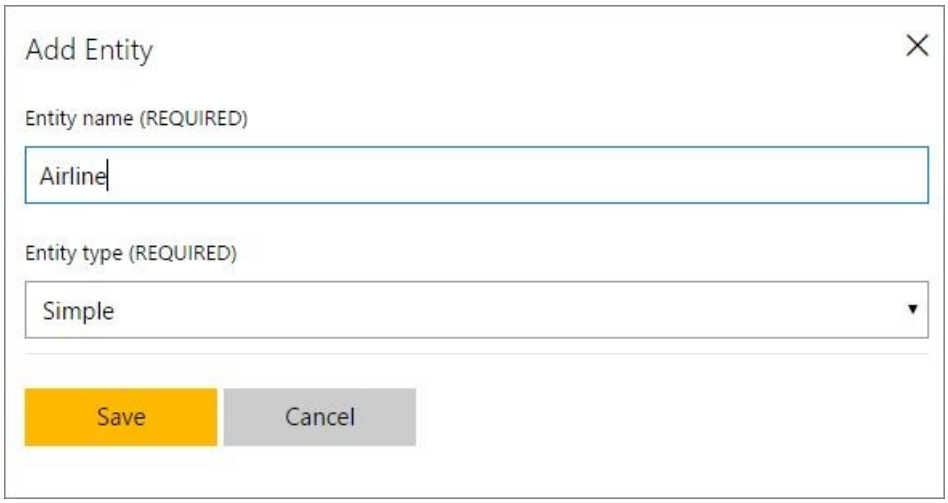
ENTITY TYPE	DESCRIPTION
Simple	A *simple entity* is a generic entity that describes a single concept.
Hierarchical	A *hierarchical entity* defines a category and its members, or a type and its subtypes. For example, a TravelClass hierarchical entity may have FirstClass and EconomyClass as child entities.
Composite	A *composite entity* is made up of other entities that form parts of a whole. For example, a FlightOrder composite entity may have TravelClass, FlightNumber, OriginLocation, DestinationLocation, and NumberOfTickets as child entities.
List	A *list entity* is a fixed list of synonyms or related words.

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

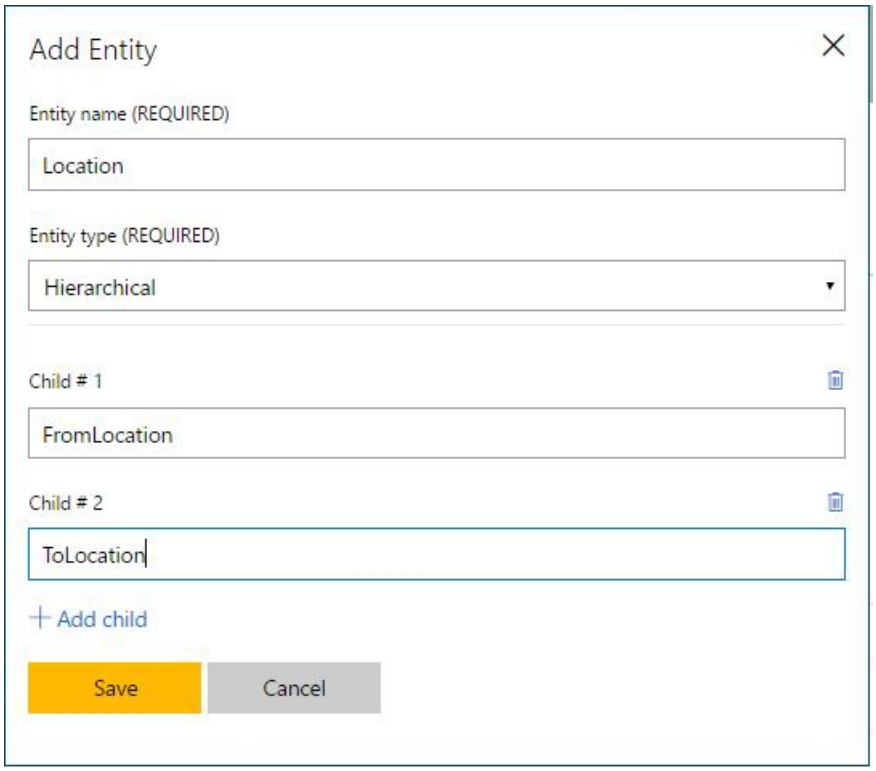
A hierarchical entity defines a relationship between a category and its members, or a type and its subtypes. In these relationships, the hierarchical entity acts as the parent. The child entities are subtypes that share some of the parent entity's characteristics. For example, in the TravelAgent app, you can add three hierarchical entities:

- "Location", including the child entity "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. Make sure to add the child entities at the same time that you create the parent entity. You can add up to 10 child entities 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.

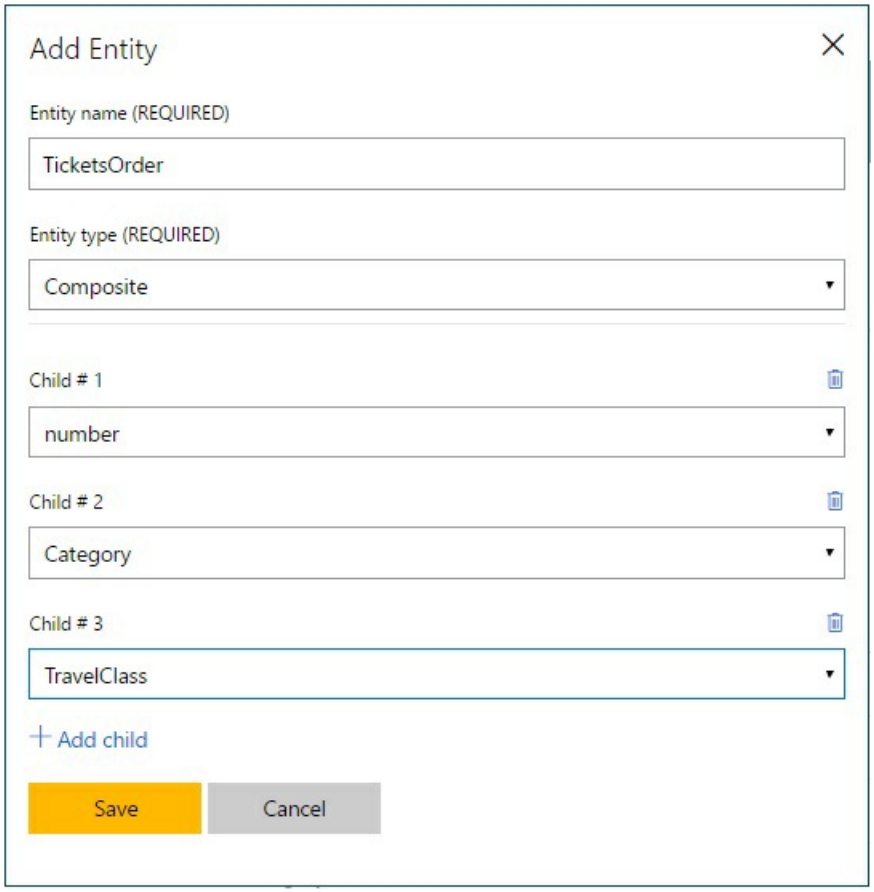
In the TravelAgent app example, a user may say "Book 2 adult business tickets to Paris next Monday." In this example, we can create a composite entity called "TicketsOrder", including three children entities: "number", "category" and "class", which describe the tickets to be booked. Before creating a 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 preceding section Add Prebuilt Entities.
2. Add the hierarchical entity "Category", including the subtypes: "adult", "child" and "infant", and "TravelClass" including "first", "business" and "economy". For more instructions, see the preceding section Add hierarchical entities.

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 identifies 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 is predicted as existing in the utterance, with a prediction score of 100%.

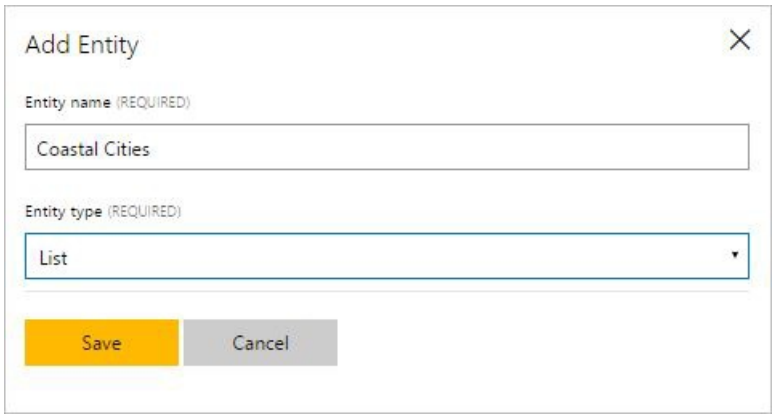
For example, in the TravelAgent app, we can create a list entity named “Coastal Cities” that includes values such as "Barcelona", "Miami", and "Venice" as canonical forms. For each canonical form, you can add synonyms. For example, synonyms of “Barcelona” can be “Capital City of Catalonia”, “BCN”, “Barna”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

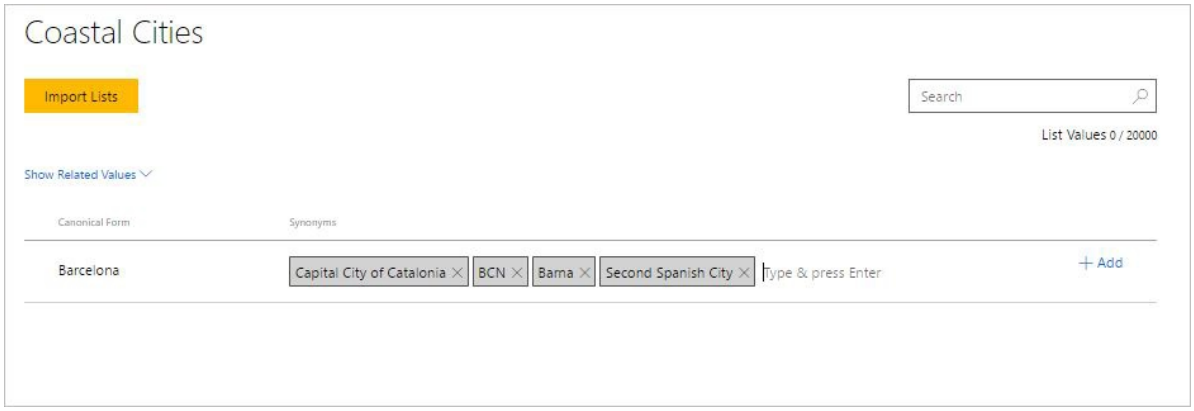
type.



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

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 (for example, Barcelona), and type other forms of it in **Synonyms** (for example, "BCN", "Barna"). 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:

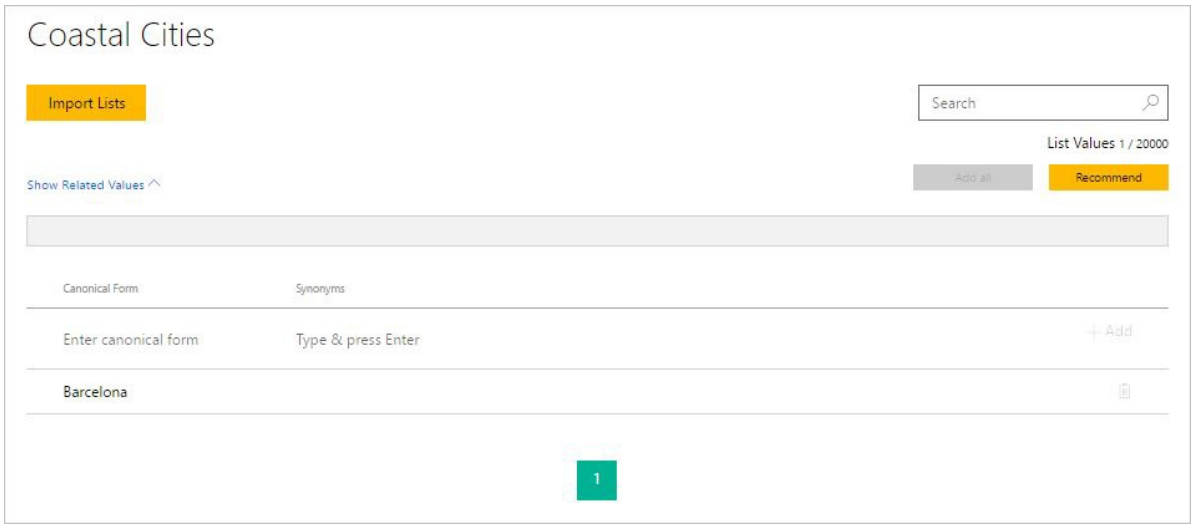
1. On the "Coastal Cities" list entity page, click **Import Lists**.
2. In **Import New Entries** dialog box, click **Choose File** and select the JSON file that includes the list.

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.

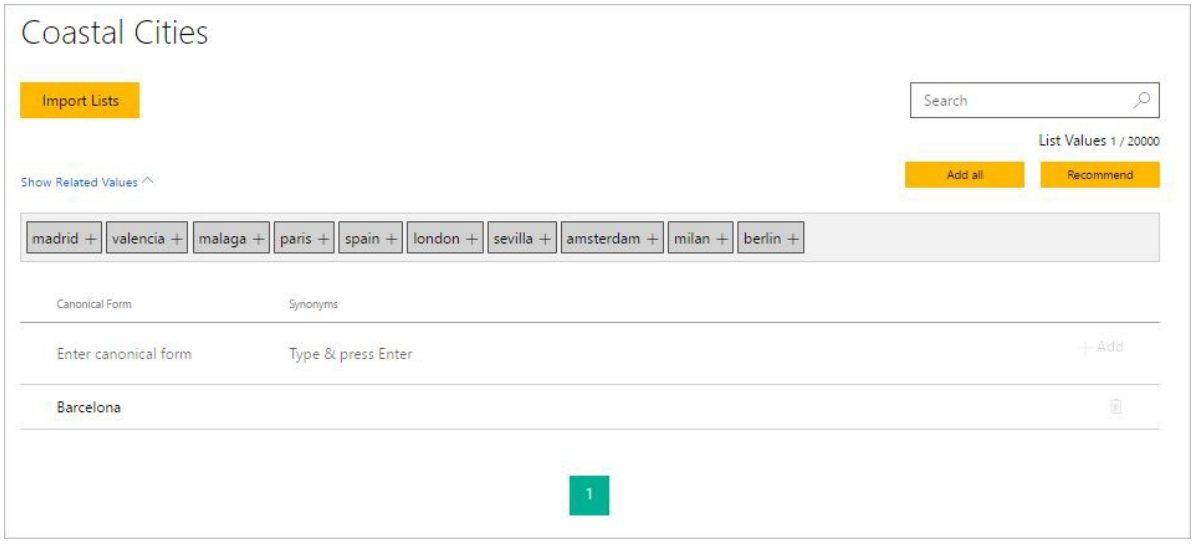
4. Click **Import**.

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 (for example, Barcelona), click **Show Related Values**.



2. Click **Recommend**. You 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.

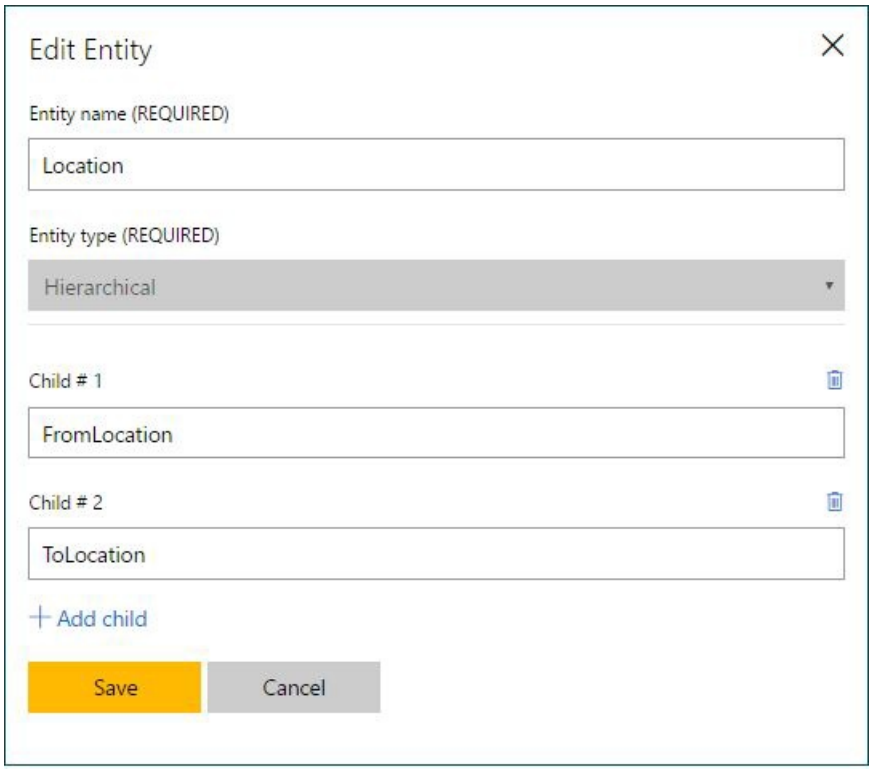
> **NOTE**
>
> - The proposed values are added as canonical forms not synonyms; synonyms should be typed manually.
> - To delete an added canonical form, click the trash bin button corresponding to it.

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:

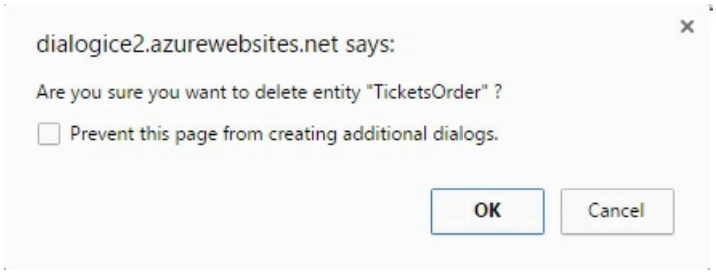
1. On the **Entities** page, click the entity in the **Entities list**.
2. In the **Edit Entity** dialog box, you can edit the entity name and child entity names. For hierarchical or composite entities, you can also add more child entities. The entity type is not editable.



3. Click **Save**.

To delete an entity:

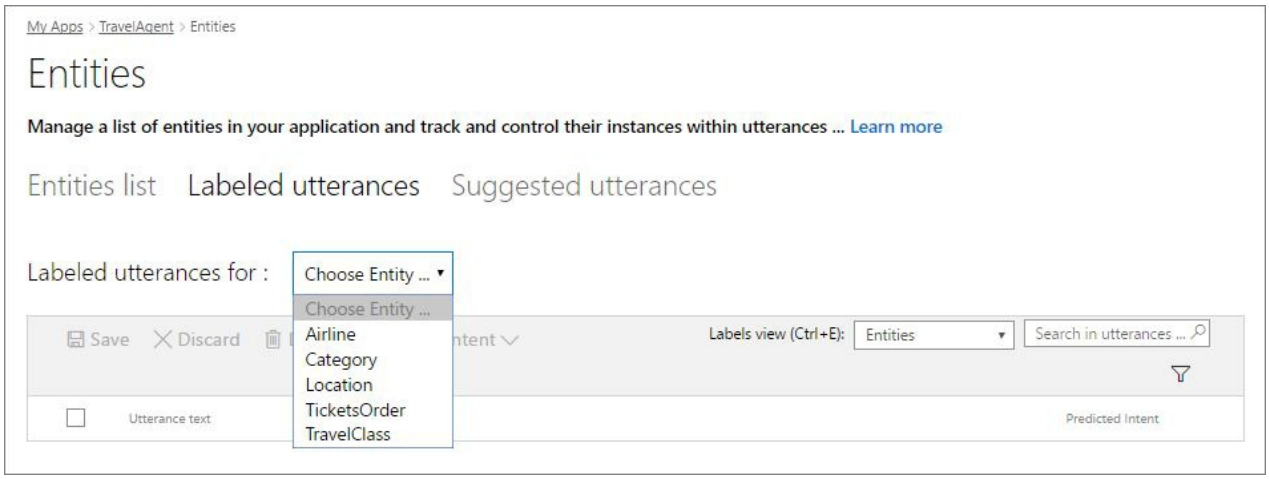
- In the **Entities list**, click the trash bin icon next to the entity you want to delete. Then, click **OK** in the confirmation message to confirm deletion.



> **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

9/18/2017 • 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.

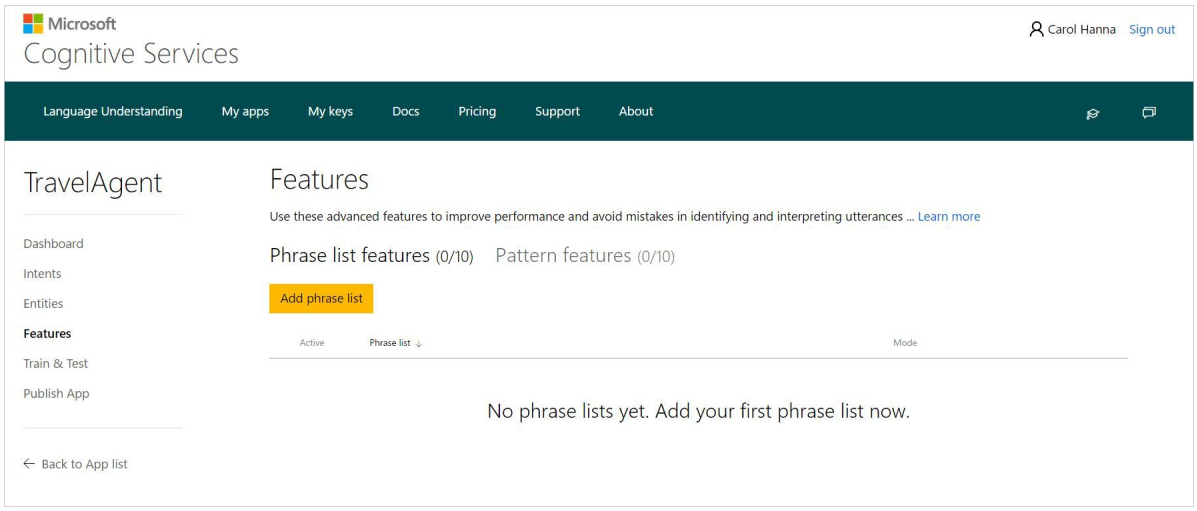
> **NOTE**
> Phrase list features show LUIS some examples of an entity's potential values. The phrase list does not have to include all members of a category, although it may.

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.

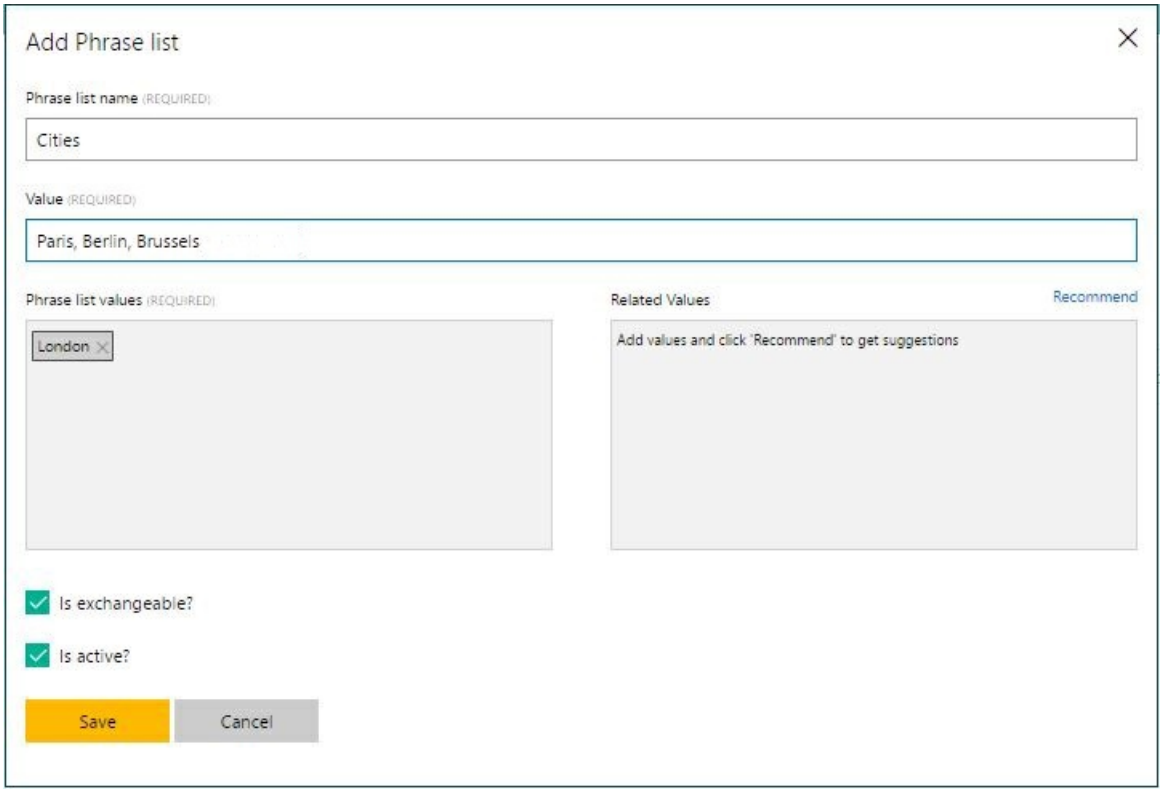
How to add a phrase list

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**.

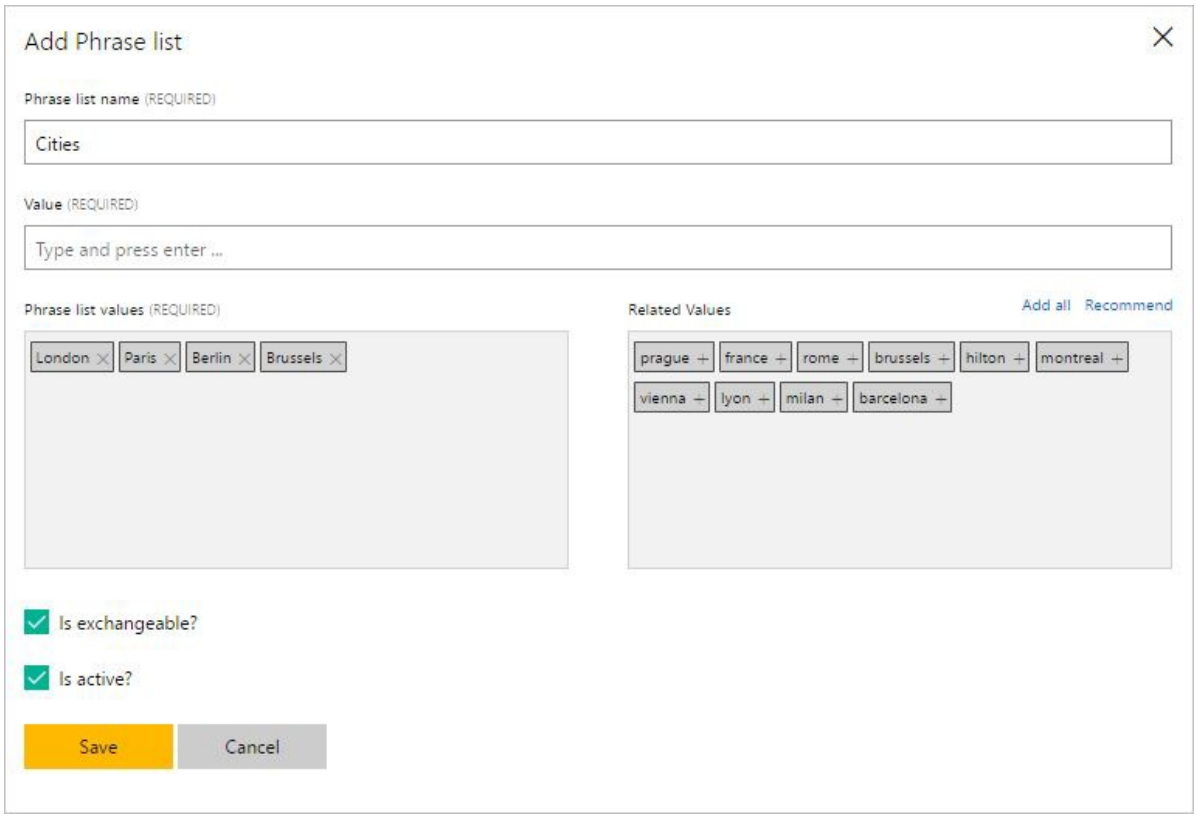


3. In the **Add Phrase List** dialog box, type "Cities" as the name of the phrase list in the **Phrase list name** box.

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.



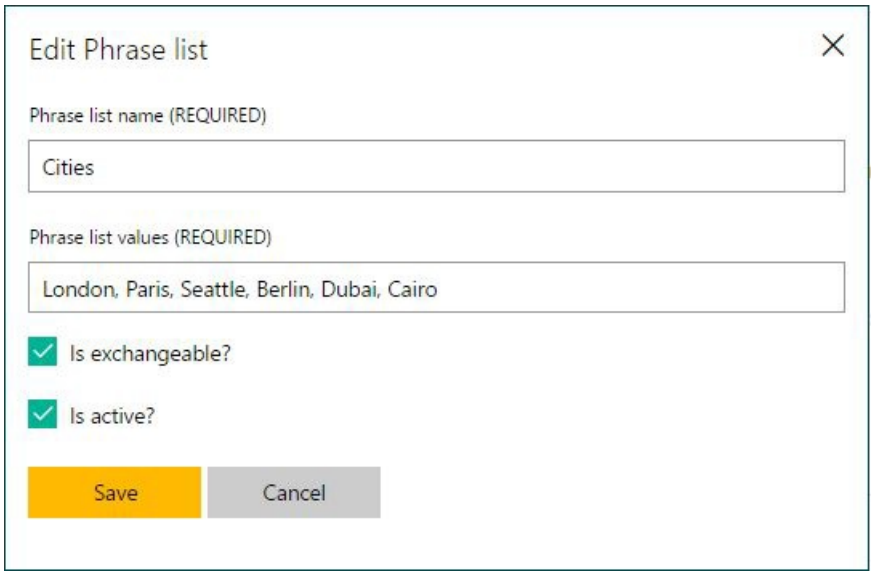
5. If your app culture is English, LUIS can propose some related values to add to your phrase list. Click **Recommend** to get a group of proposed values that are semantically related to the added value(s). You can click any of the proposed values to add it, or click **Add All** to add them all.



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

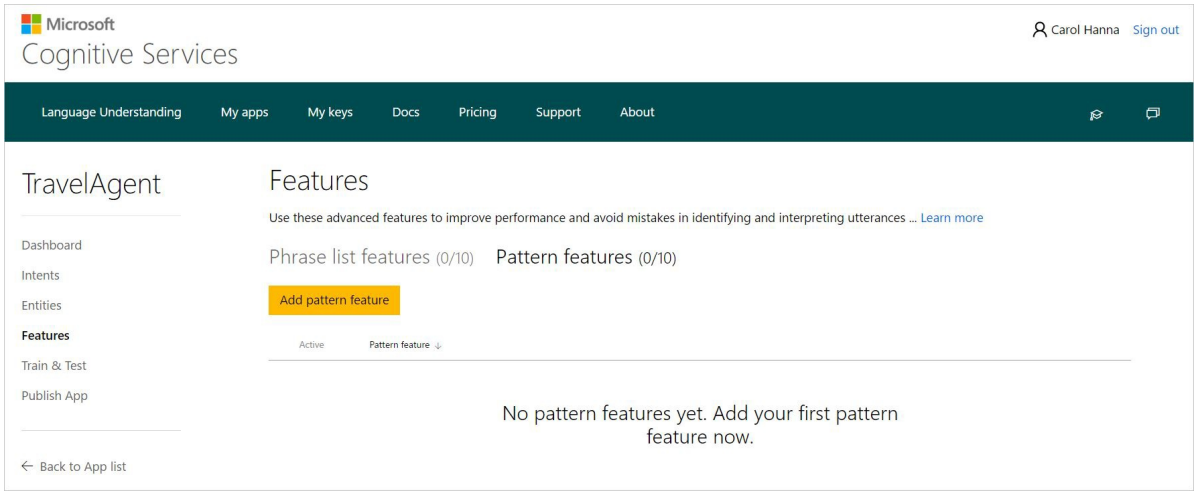
Pattern features define a regular expression to help LUIS recognize regular patterns that are frequently used in your application's domain, such as the pattern of flight numbers in a travel app or product codes in a shopping app.

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.

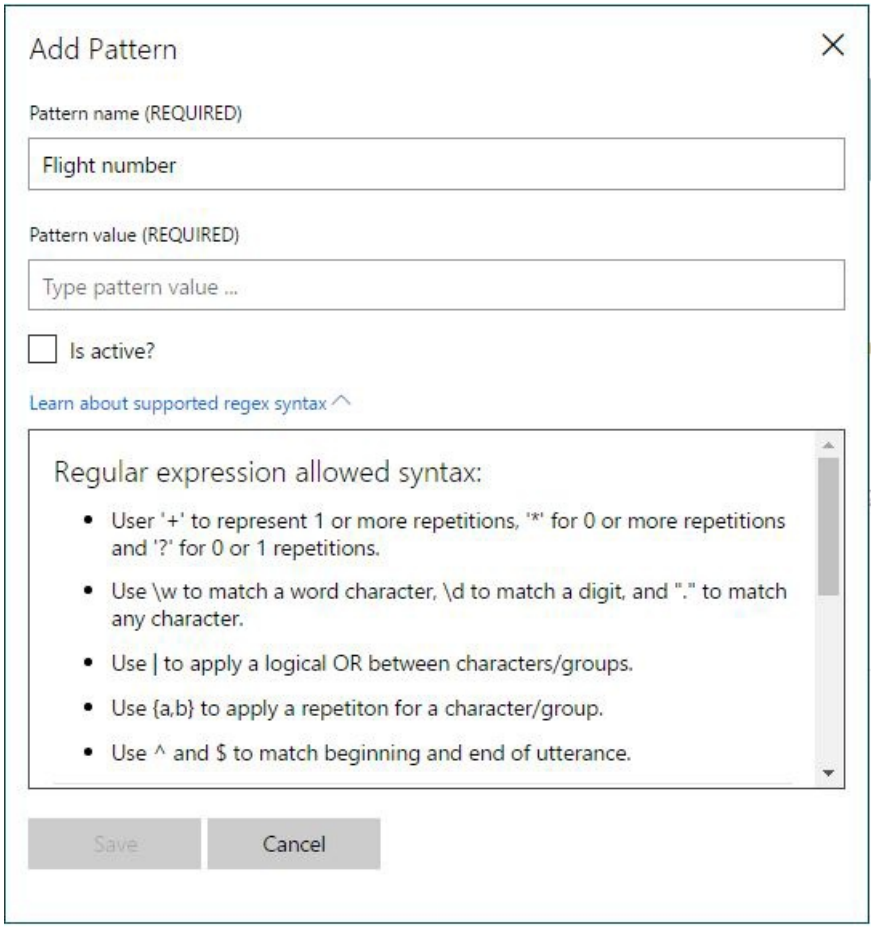
How to add a pattern

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**.

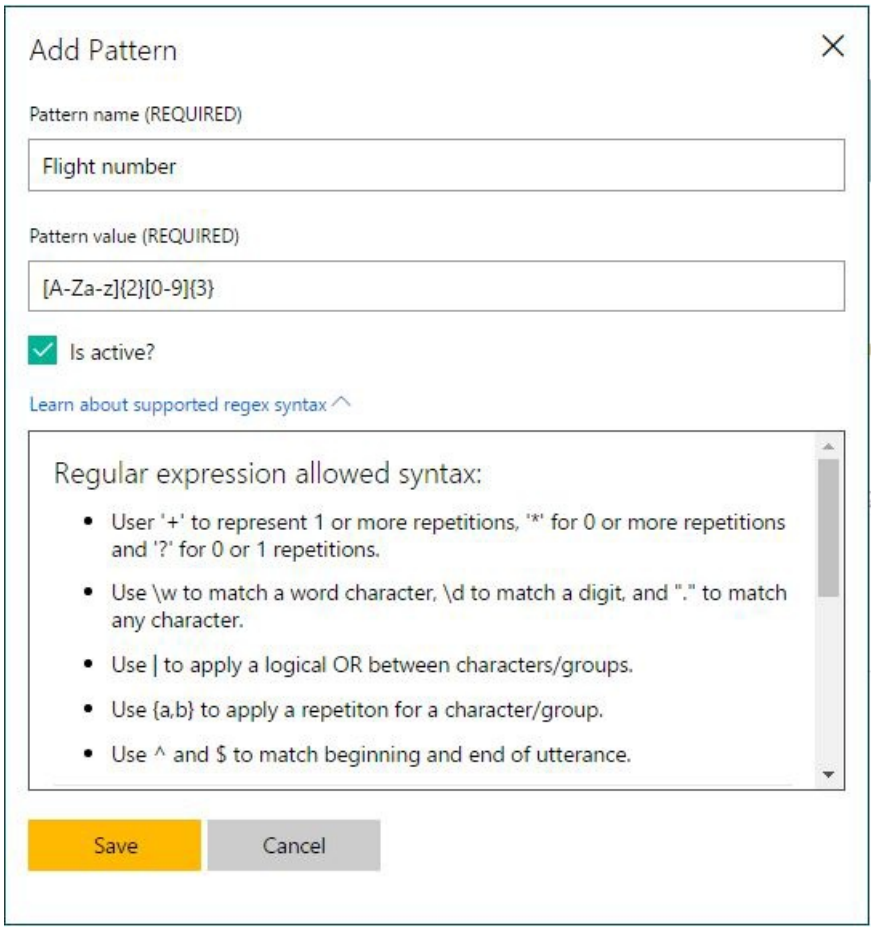


3. In the **Add Pattern** dialog box, type "Flight number" in the **Pattern name** text box.
4. To learn about the supported regex syntax, click **learn about supported regex syntax** to expand the

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.



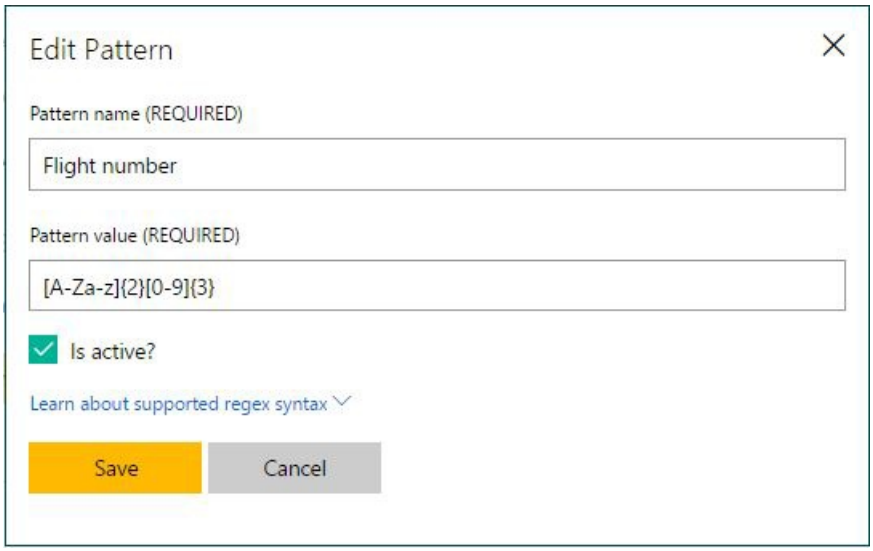
6. Click **Is active** if you want this pattern to be active (i.e. applicable and used) in your app. A feature is active by

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

8/30/2017 • 11 min to read • Edit Online

Training is the process of teaching your LUIS app by example to improve its language understanding. If you make updates by adding, editing, or deleting entities, intents, or utterances to your LUIS app, you need to train your app before testing and publishing. When you train a LUIS app, LUIS generalizes from the examples you have labeled, and learns to recognize the relevant intents and entities in the future, which improves its classification accuracy.

Training and testing is an iterative process. After you train your LUIS app, you test it with sample utterances to see if the intents and entities are recognized correctly. If not, make updates to the LUIS app, train and test again.

Typically, before retraining, you will want to relabel any utterances in which LUIS failed to identify the expected intents and entities. You can find the utterances to relabel using the following procedures:

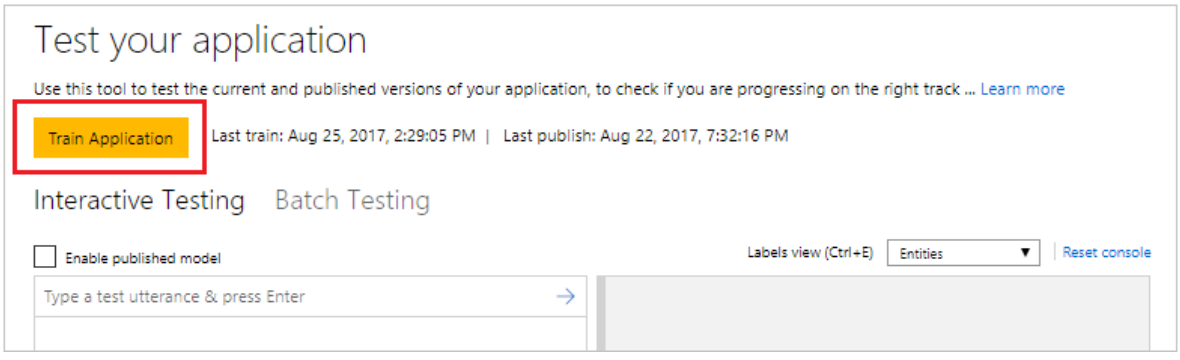
- **Interactive testing**: The interactive testing pane lets you type in an utterance and displays the intents and entities that your LUIS app detects.
- **Suggested utterances**: Relabeling suggested utterances that LUIS identifies for you.
- **Review utterances from users**: LUIS provides a log of utterances from users that have been passed to the LUIS app endpoint. This log includes the intents and entities you can review to see if they've been correctly identified.

In addition to relabeling utterances, you may also try adding new utterances, editing the intent or entity types, and adding features to your LUIS app to improve performance.

Train the current iteration of your app

To start the iterative process of training, you first need to train your LUIS app at least once.

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 LUIS app on the latest updates.



NOTE

If you have one or more intents in your app that do not contain example utterances, you cannot train your app until you add utterances for all your intents. For more information, see Add example utterances.

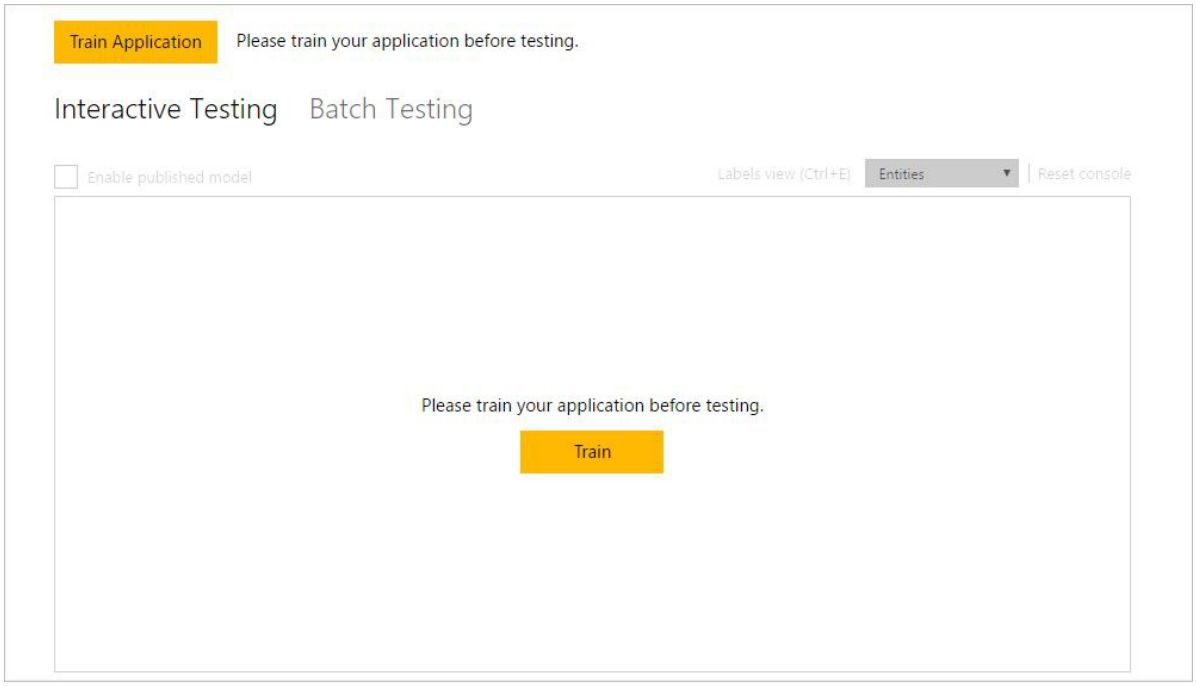
Test your app

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

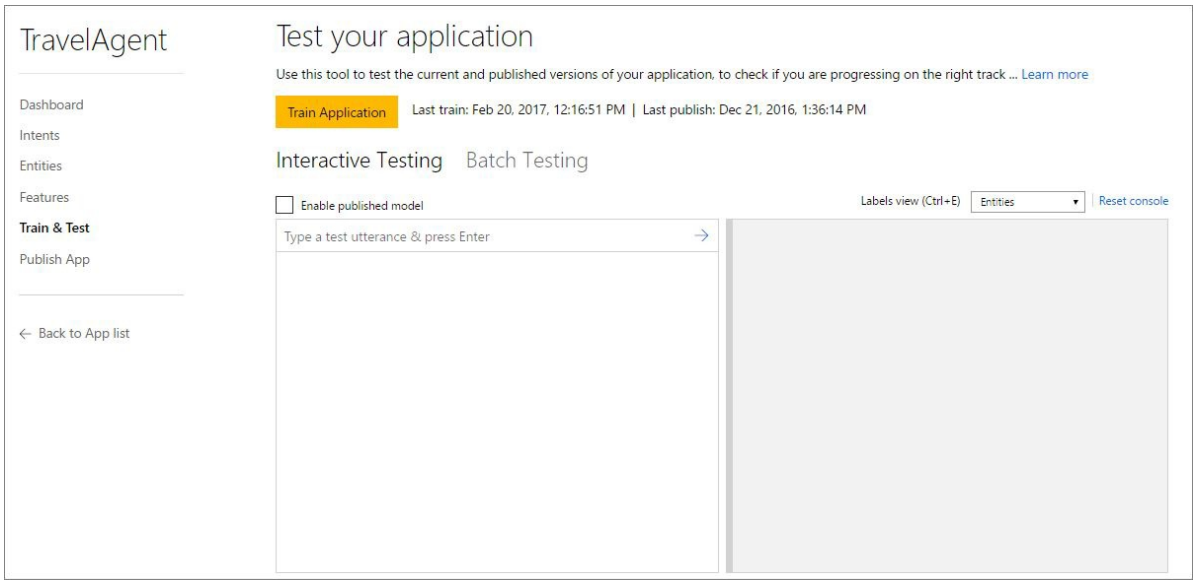
Access the Test App page

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 looks like this screenshot:



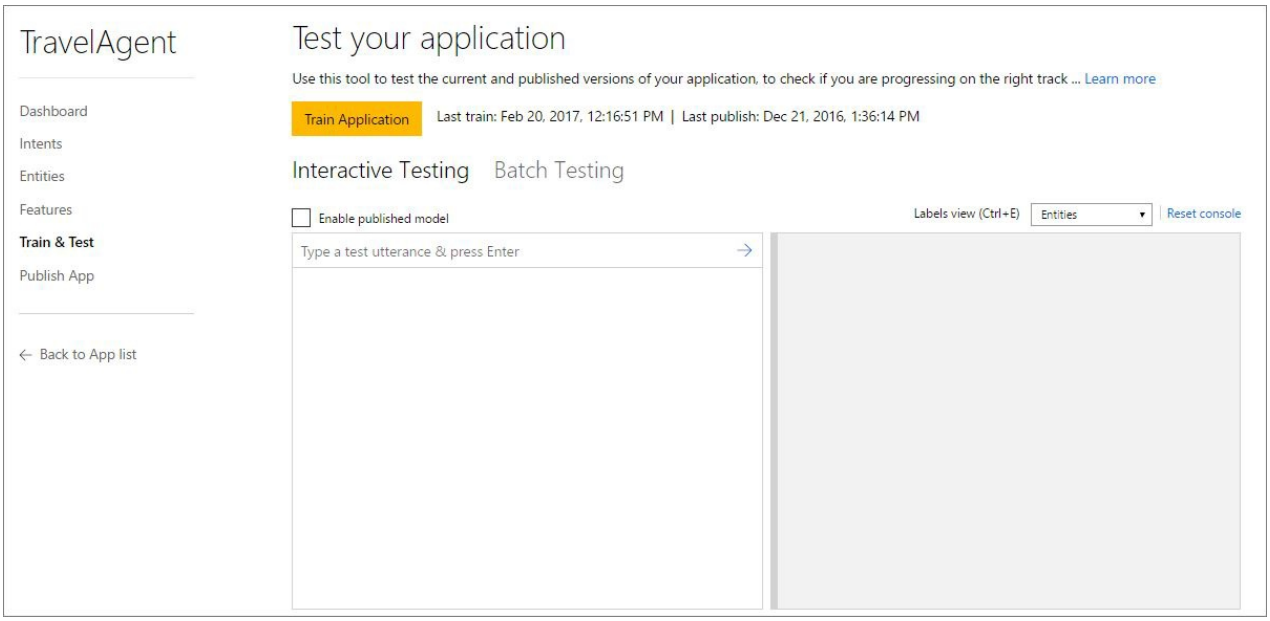
- If your model is trained, your test page looks like this screenshot:



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.

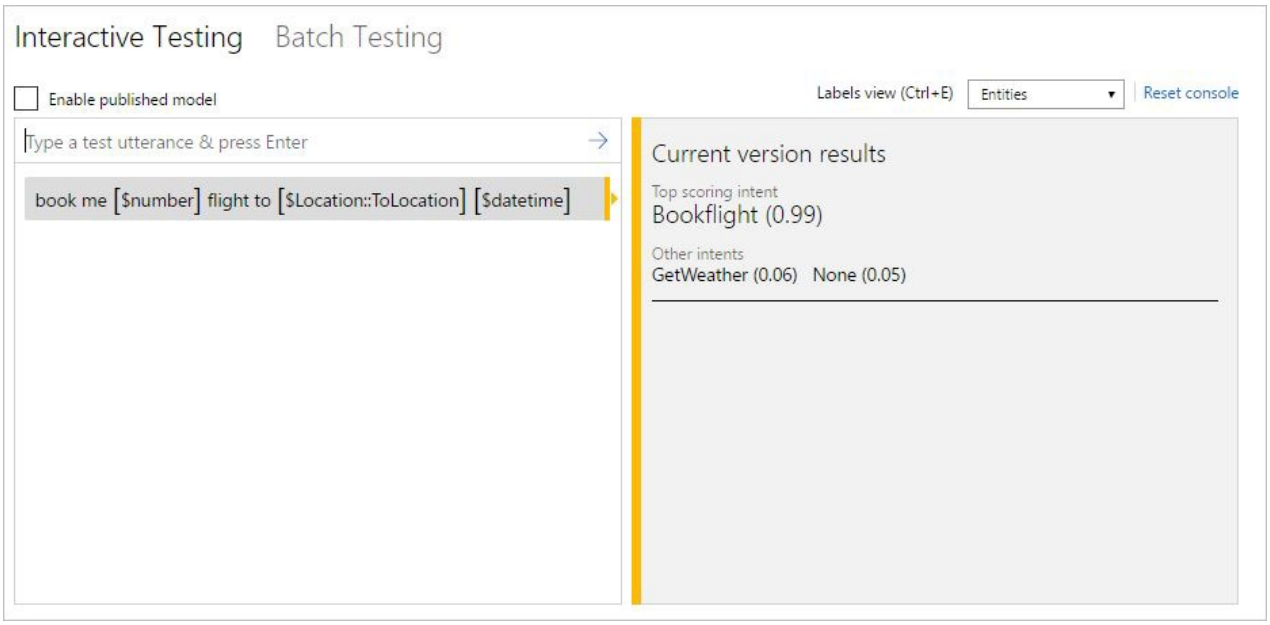
The **Interactive Testing** tab is divided into two sections (as in the screenshot):



- **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.

The following screenshot shows how test results appear in the **Interactive Testing** tab, in which "book me a flight to Boston tomorrow" is entered as a test utterance:



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.

Relabel utterances and retrain

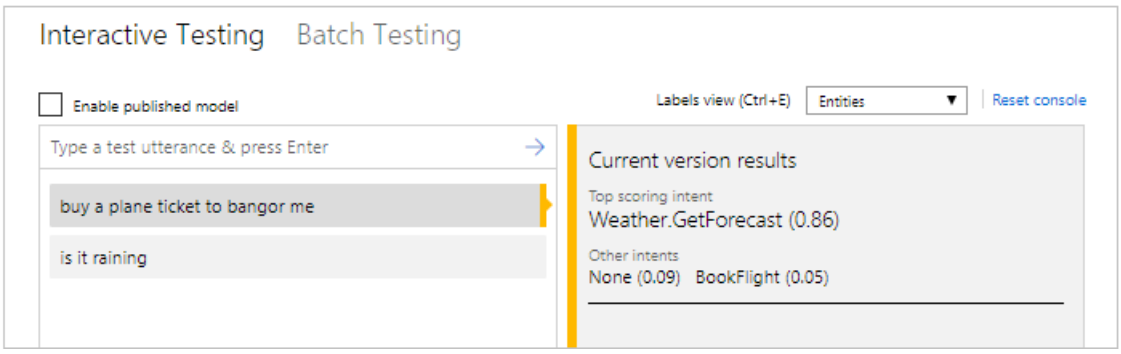
When you perform interactive testing, you may find that LUIS doesn't detect the intent or entities that you expect in some utterances. The following steps walk you through relabeling an utterance and retraining.

Relabel an utterance to retrain intents and entities

1. Import the sample LUIS app Travel Agent - Sample 1. This LUIS app has only a few sample utterances and provides a starting point for training. It has the following intents:

   - BookFlight
   - Weather.GetForecast
   - None

2. On the **Test App** page, in the **Interactive Testing** tab, type in `buy a plane ticket to bangor me` and press Enter. Instead of the `BookFlight` intent, the test results show `Weather.GetForecast`.



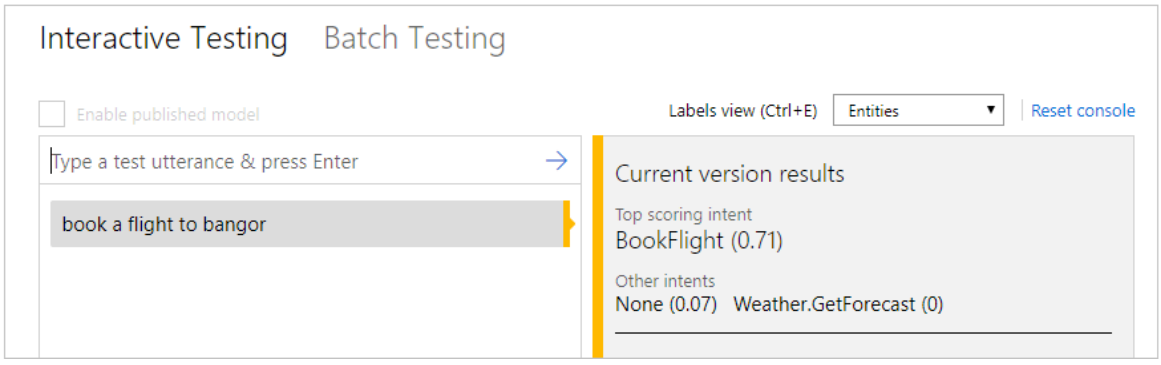
3. To teach LUIS that `buy a plane ticket to bangor me` should be mapped to the `BookFlight` intent instead of `Weather.GetForecast`, go to the **Intents** page, click the **BookFlight** intent, type "buy a plane ticket to bangor me" into the text box, and press Enter. Click **Save**.

4. Go back to the **Train & Test** page and click **Train application**.

5. Type `book a flight to bangor` in the text box and click enter.

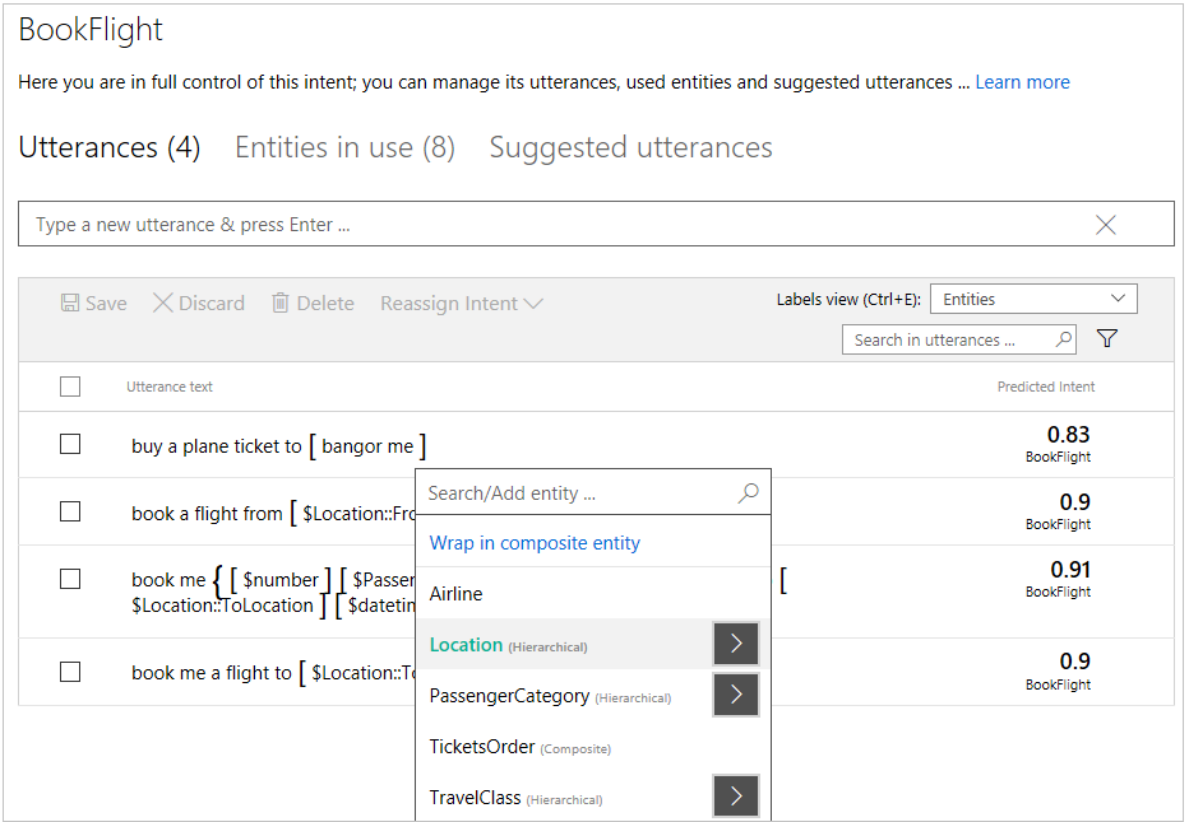
> **NOTE**
> In this step you choose an utterance that's similar to the one you labeled, but not exactly the same. This helps to test your LUIS app's ability to generalize.

6. Now the intent should be correctly detected as `BookFlight`. However, `bangor` isn't detected as a location yet.

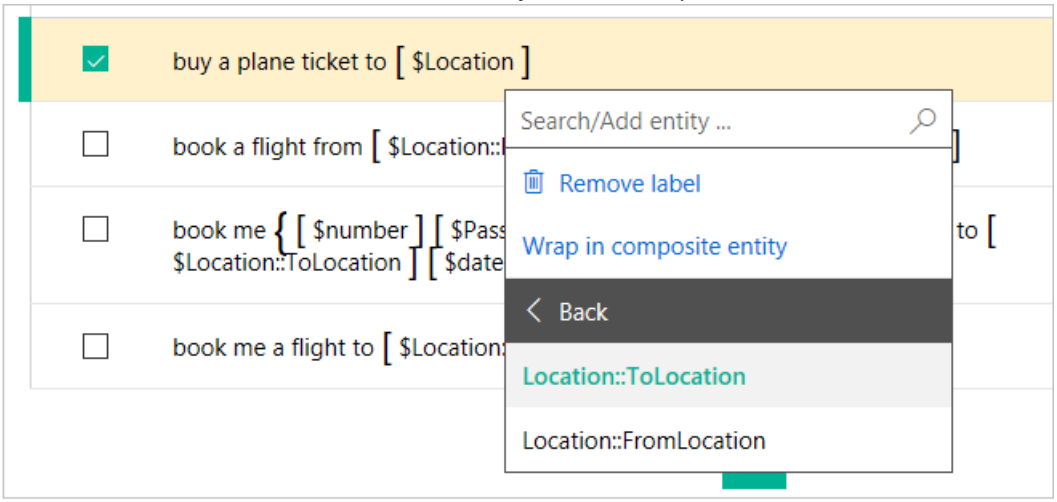


7. Now you need to teach LUIS that `bangor me` in the utterance `buy a plane ticket to bangor me` should be mapped to the `Location` entity. Go to the **Intents** page, click the **BookFlight** intent, and find `buy a plane ticket to bangor me` in the list of utterances.

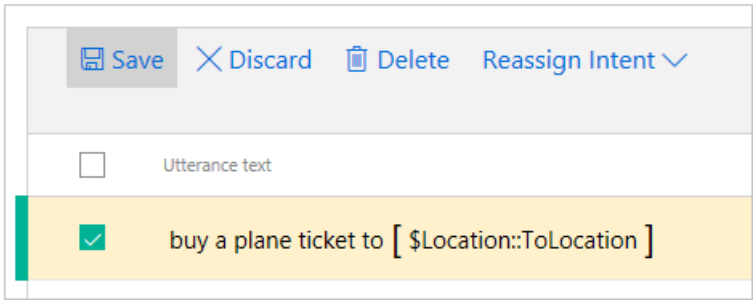
8. Click on the words `bangor me` and choose the **Location** entity from the drop-down list. Click on the arrow to expand the **Location** entity.



9. Select **Location::ToLocation** hierarchical entity from the drop down list.



10. Click **Save**.

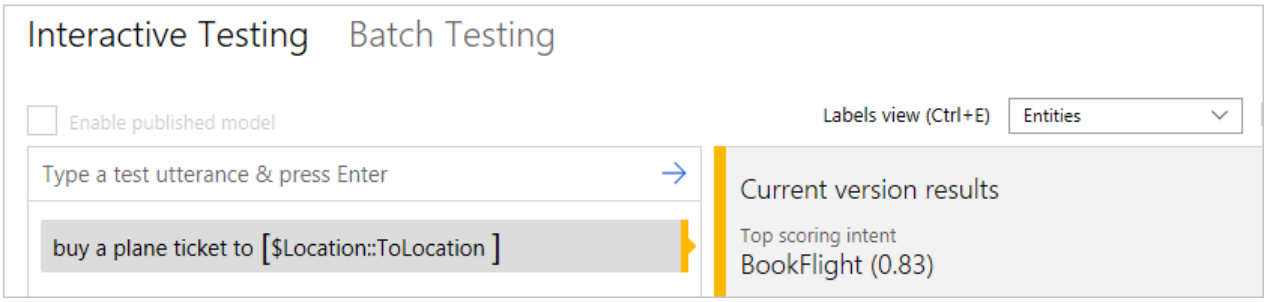


11. Go back to the **Train & Test** page and click **Train application**.

12. Type `buy a plane ticket to paris` in the text box and click enter. Now the location entity is correctly detected.

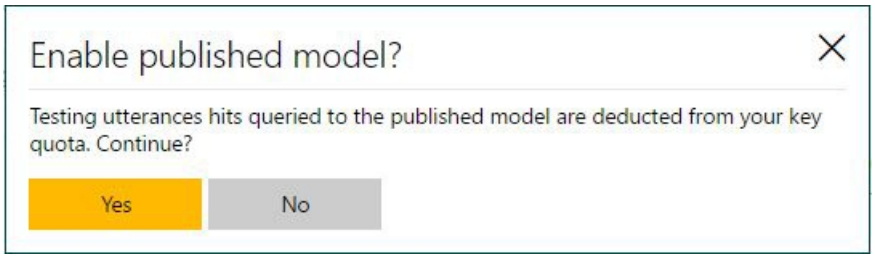
NOTE

In this step you choose an utterance that's similar to the one you labeled, but not exactly the same. This helps to test your LUIS app's ability to generalize.



Perform interactive testing on current and published models

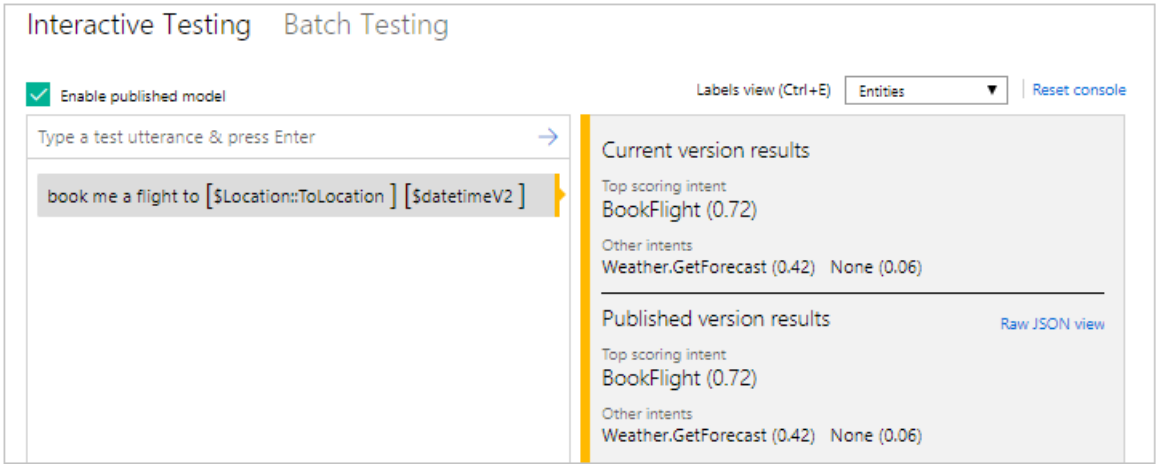
1. On the **Test App** page, **Interactive Testing** tab, click **Enable published model** check box and then click **Yes** in the following confirmation message:



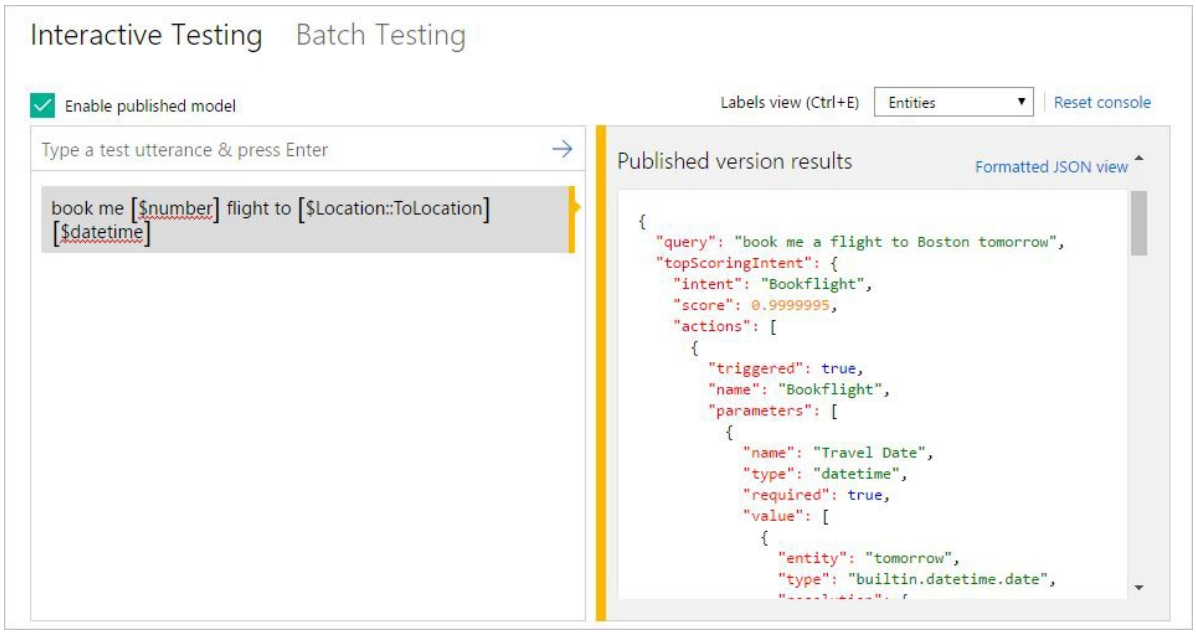
NOTE

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

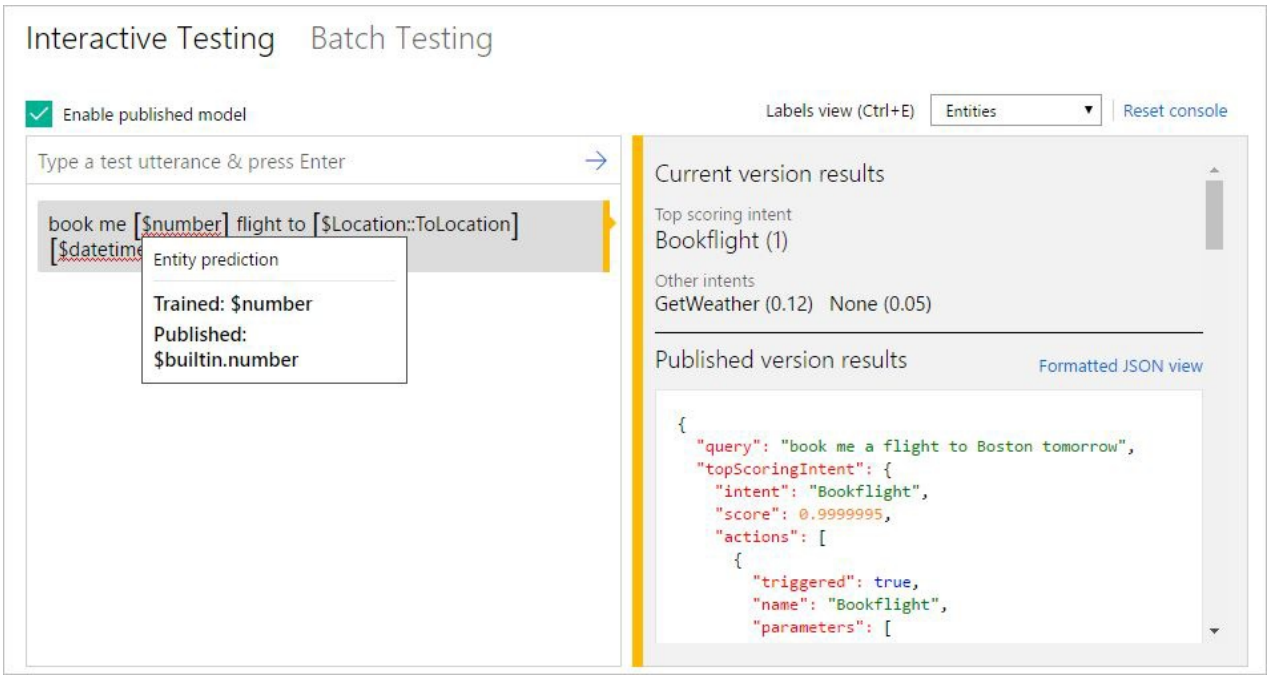
1. Type "book me a flight to Boston tomorrow" as your test utterance and press Enter. The result view on the right side will be split horizontally into two parts (as in the following screenshot) to display results of the test utterance in both the current and published models.



2. To view the test result of your published app in JSON format, click **Raw JSON view**. This looks 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

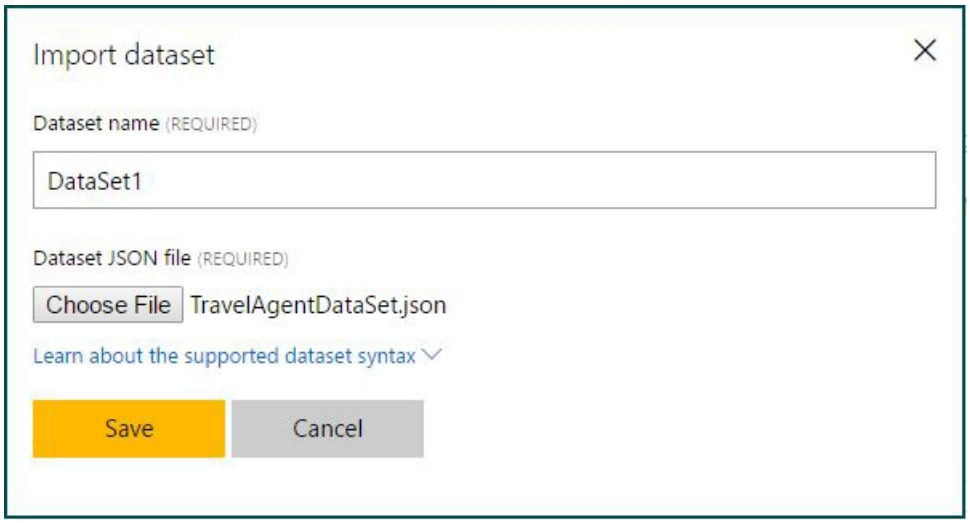
file, known as a *dataset*. The dataset file should be written in JSON format and contains a maximum of 1000 utterances. All you need to do is to import this file to your app and run it to perform the test. Your LUIS app will return the result, enabling you to access detailed analysis of all utterances 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.

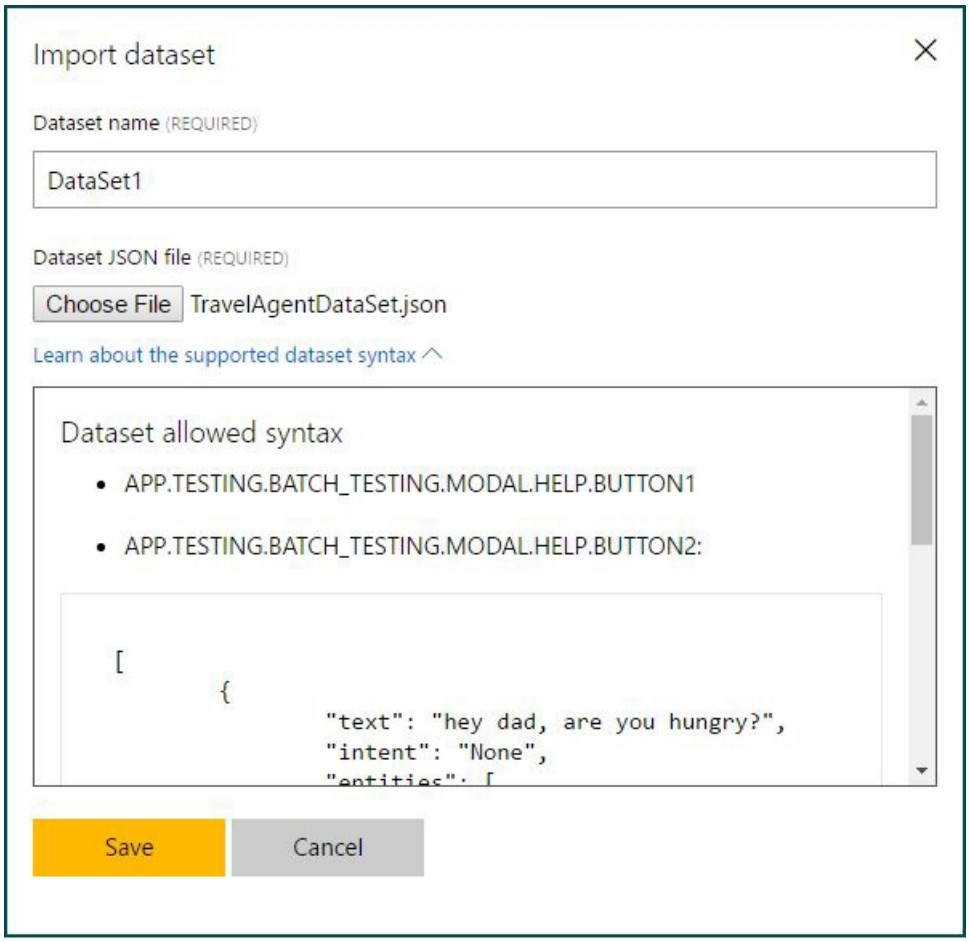
Import a dataset file

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**  .

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:

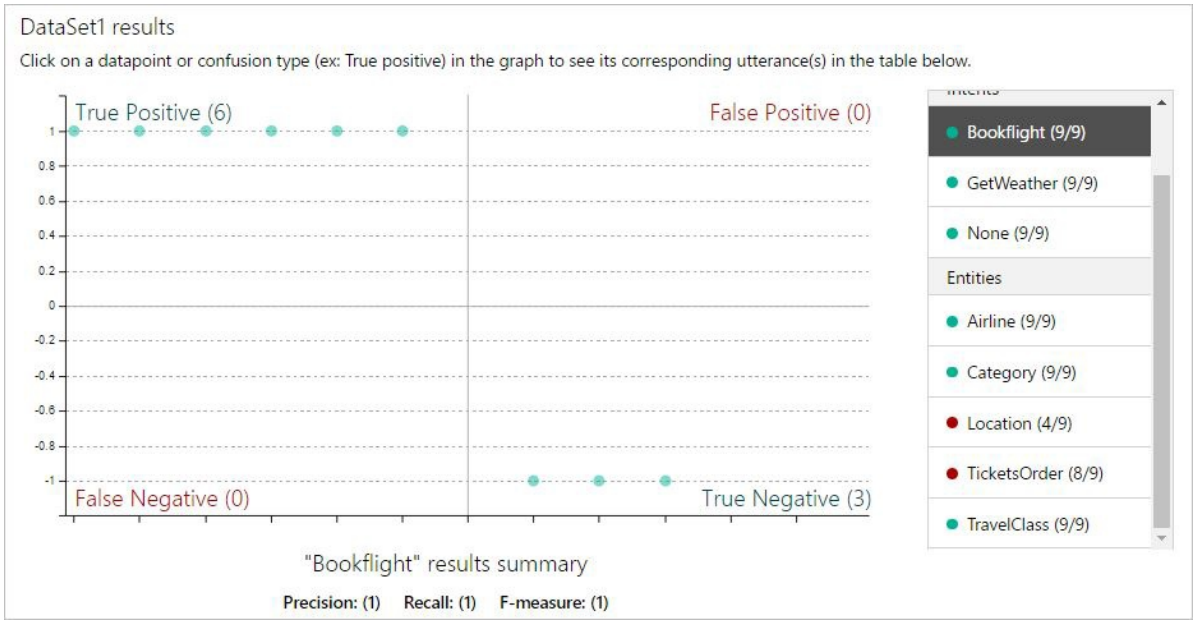
- **Status** of the dataset shows whether or not the dataset result contains errors. In the above example, an

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.

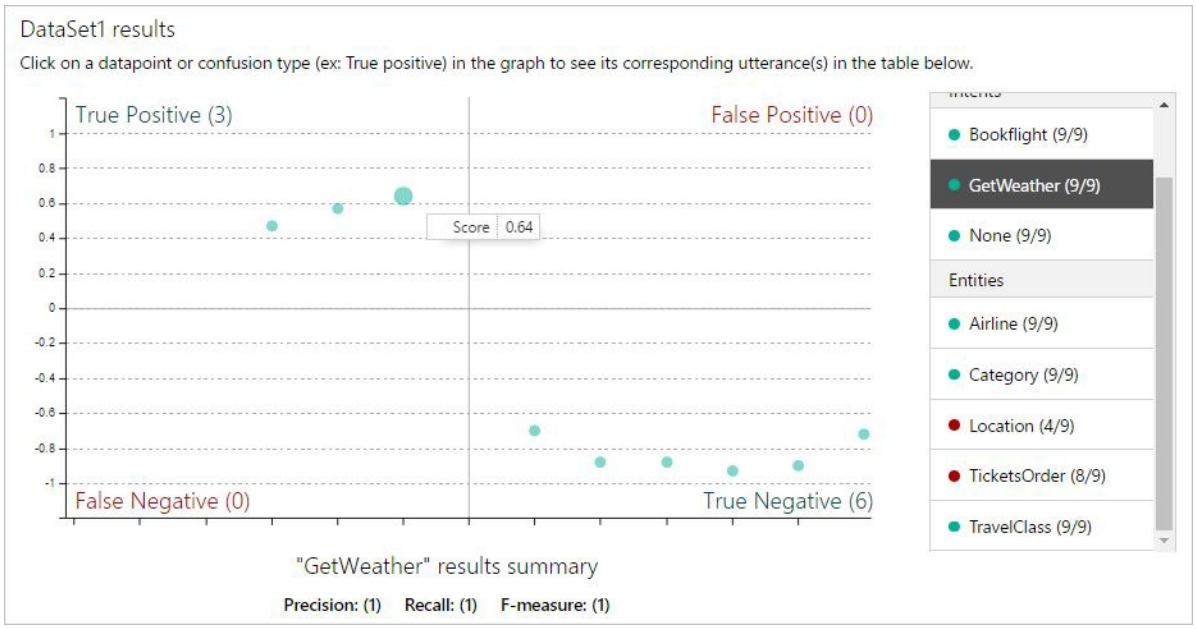
Access test result details in a visualized view

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.

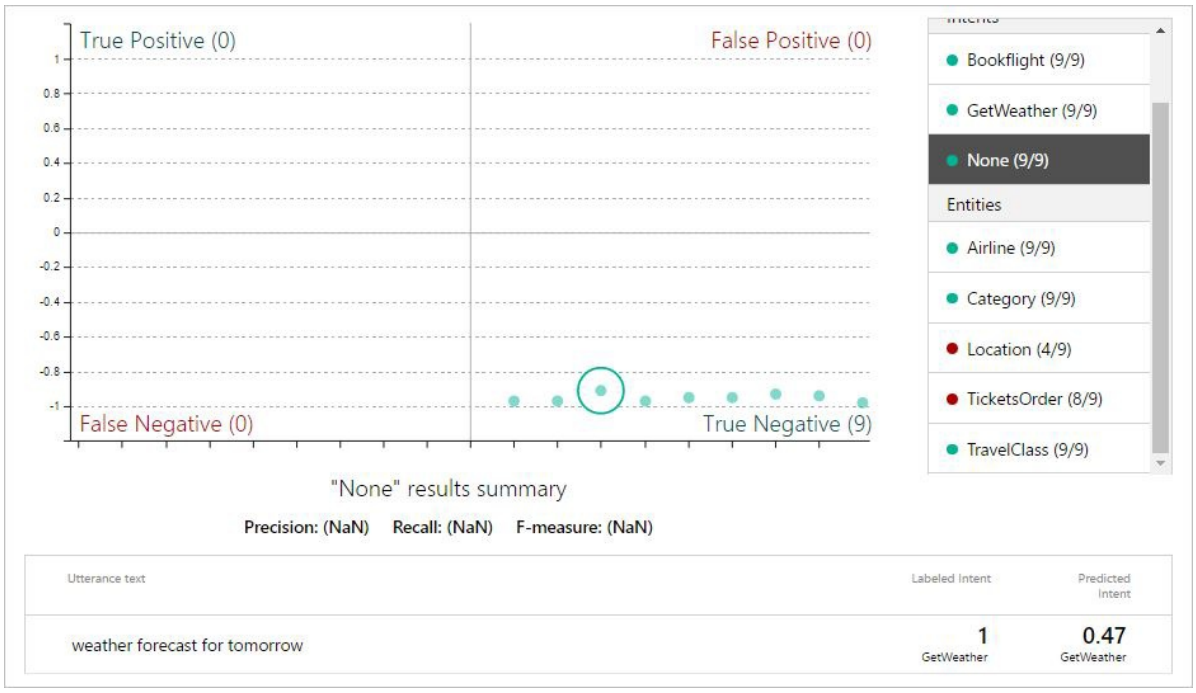
The graph contains 4 sections representing the possible cases of your application's prediction:

- **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.

Next steps

If testing indicates that your LUIS app doesn't recognize the correct intents and entities, you can try to improve your LUIS app's performance by labeling more utterances or adding features.

- Label suggested utterances with LUIS
- Use features to improve your LUIS app's performance

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.

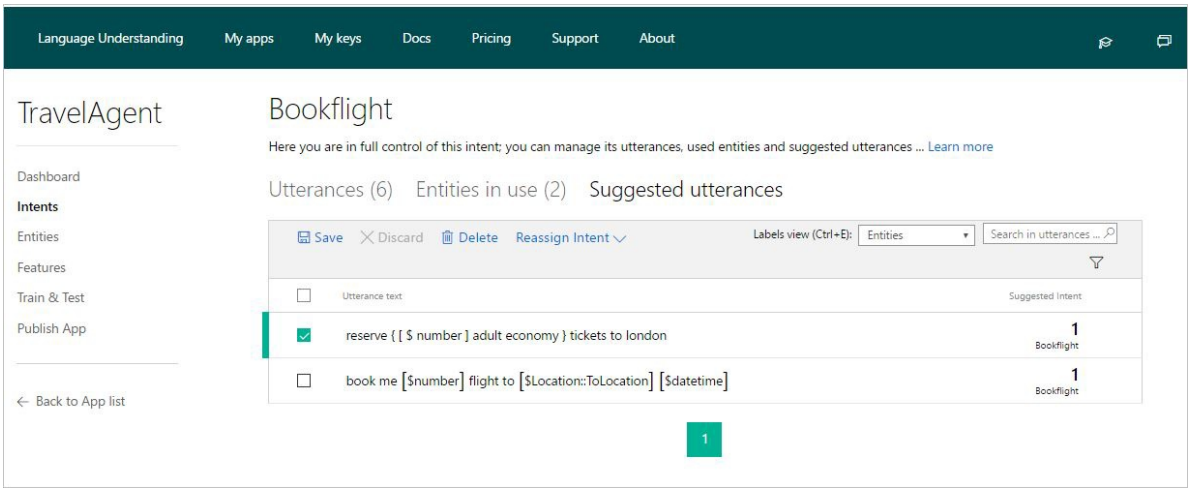
View suggested utterances

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.

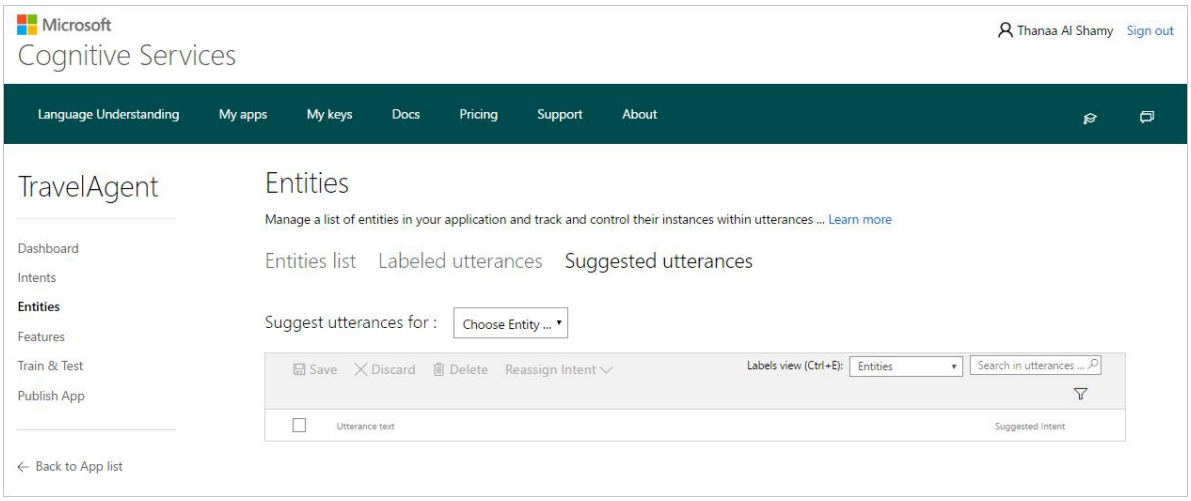
To view suggested utterances per intent:

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.

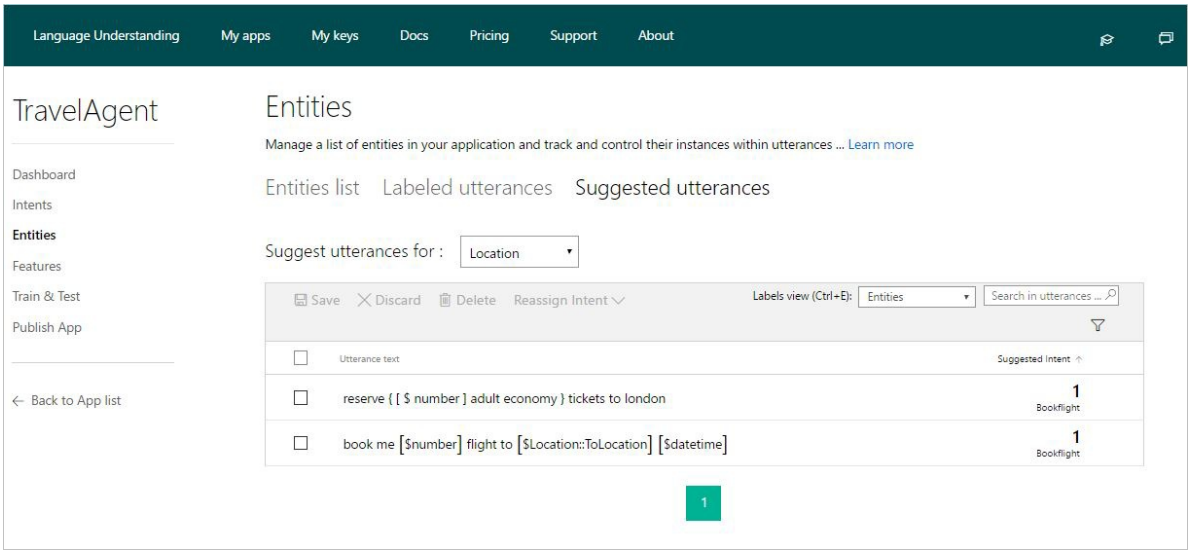


To view suggested utterances per entity:

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.

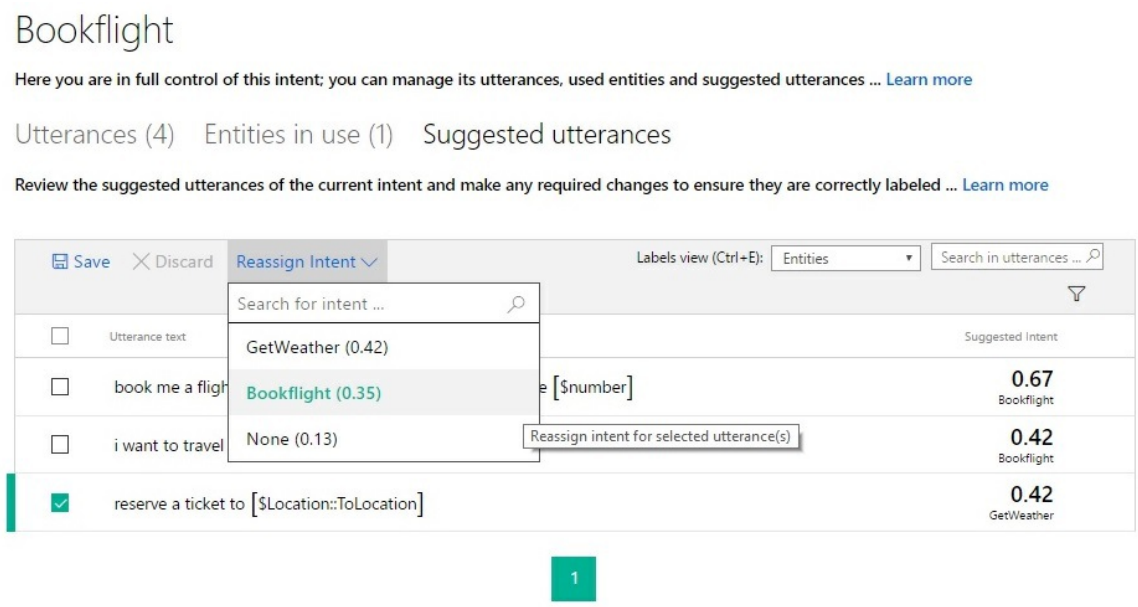
Label suggested utterances

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

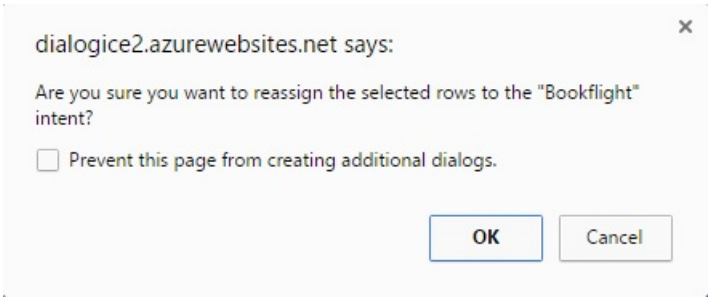
unlabeled entities, correct mis-labeled ones or even remove their labels.

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):

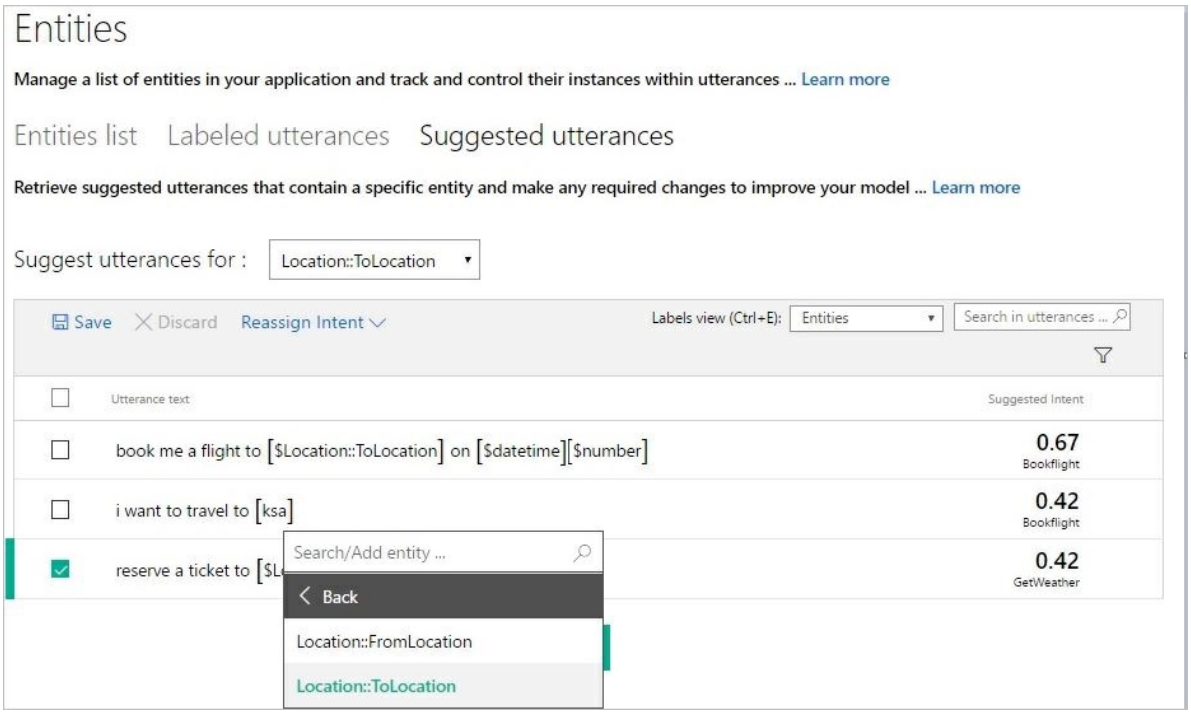
- If the predicted intent is correct and entities are correctly detected and labeled, then just select the utterance and click **Save** without making any changes to accept and save the utterance.
- If the predicted intent is incorrect, you need to change it. For example, in the screenshot below note 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.



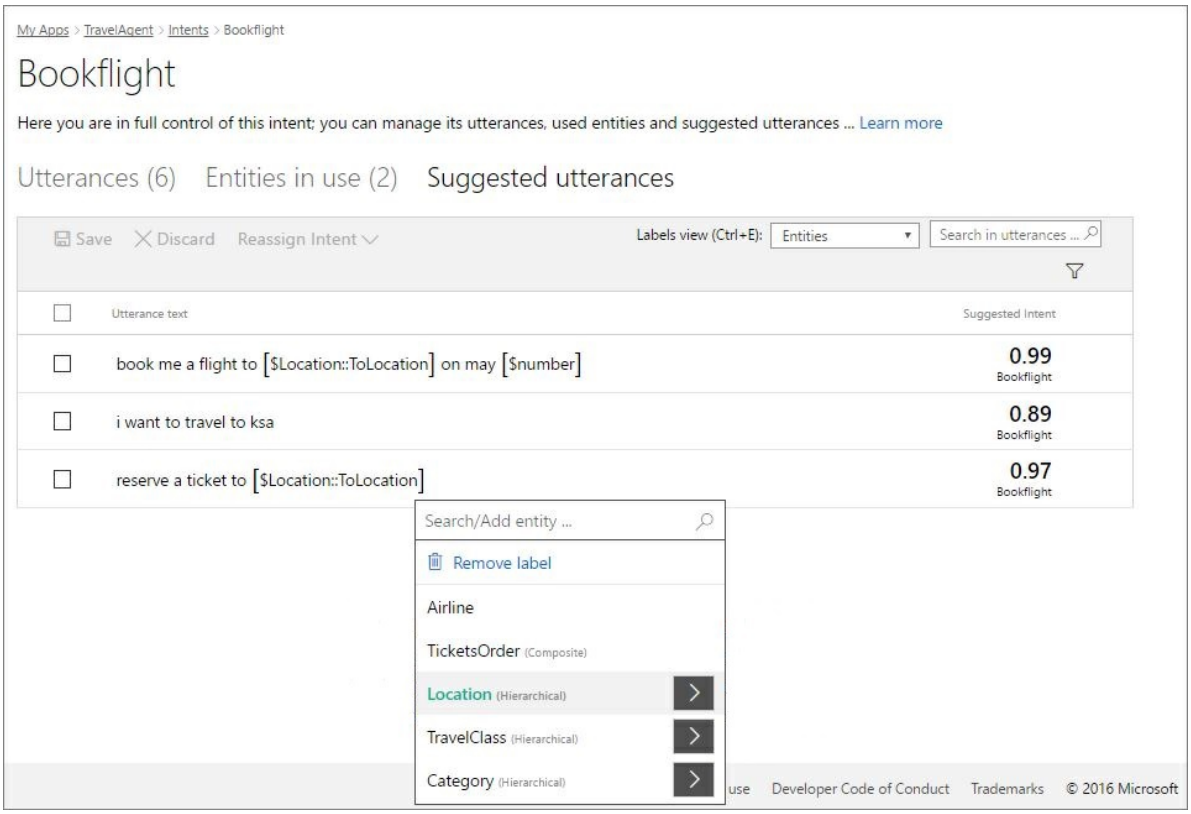
You will get a confirmation message. Click **OK** to confirm this action, and then click **Save** to save your changes.



- 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

11/2/2017 • 3 min to read • Edit Online

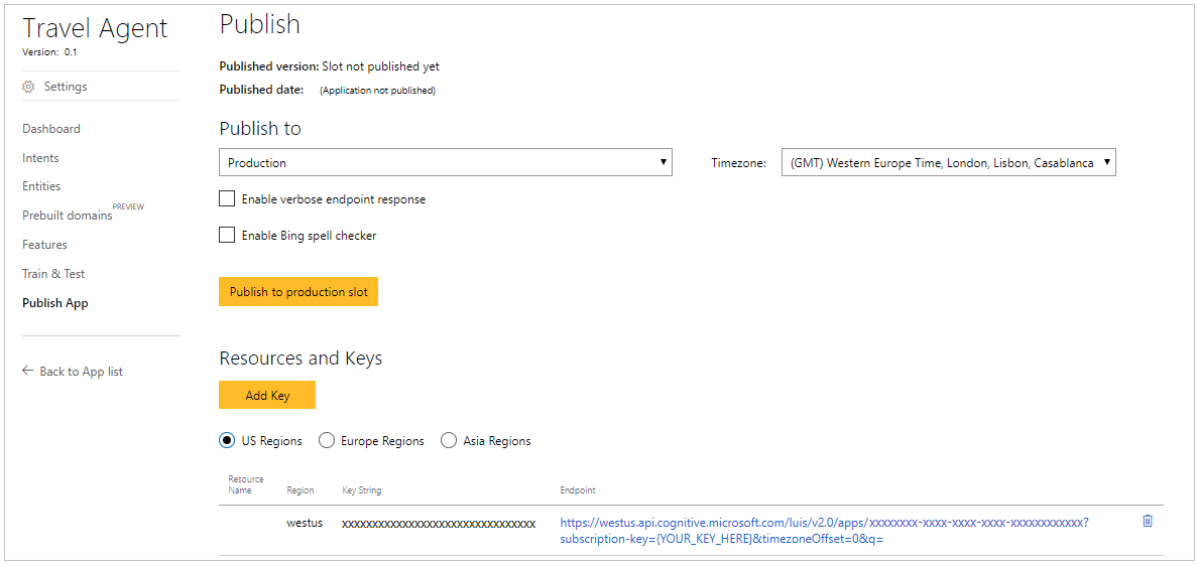
When you finish building and testing your LUIS app, you can publish it as a web service on Azure and get an HTTP endpoint that can be integrated into any client or backend code.

> **TIP**
> You can optionally test your app before publishing it. For instructions, see Train and test your app.

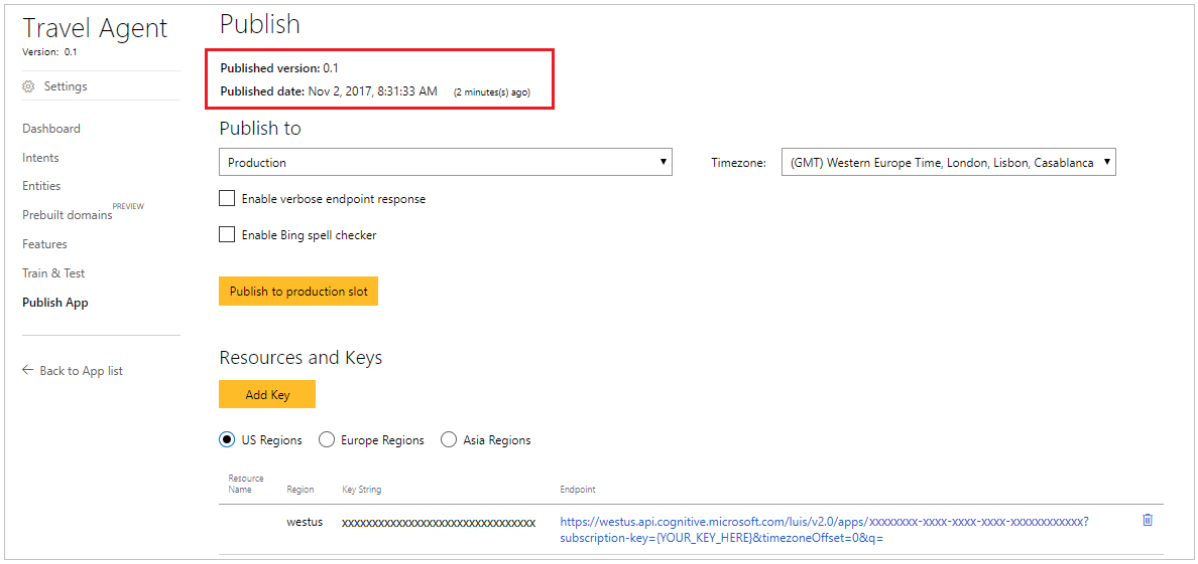
You can either publish your app directly to the **Production Slot** where end users can access and use your model, or you can publish to a **Staging Slot** for working through trials and tests to validate changes before publishing to the production slot.

Publish your app to an HTTP endpoint

1. Open your app by clicking its name on the **My Apps** page, and then click **Publish App** in the left panel. The following screenshot shows the **Publish App** page if you haven't published your app yet.



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



2. The first time you publish, the **Publish App** page shows your starter key. If you want to use a key other than the keys shown, click the **Add Key** button. This action opens a dialog that allows you to select an existing endpoint key to assign to the app. For more information on how to create and add endpoint keys to your LUIS app, see Manage your keys.
3. Choose whether to publish to **Production** or to **Staging** by selecting from the drop-down menu under **Publish to**.
4. If you want to enable Bing Spell Check, click the **Enable Bing Spell Checker** check box.

> **NOTE**
>
> For a limited time, the option to enable Bing Spell Check is included with your LUIS subscription. However, you need to add `spellCheck=true` to the URL when you call the LUIS app endpoint to turn on spell checking. Checking the **Enable Bing Spell Checker** check box appends `spellCheck=true` to the URL that displays in the **Publish app** page when publish is complete.

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 to append a `verbose=true` parameter to the end point URL. Otherwise, it includes only the top scoring intent.
6. If the app has not been trained, click **Train**.
7. Click **Publish to production slot** if you have selected the **Production** option under **Publish to**, or **Publish to staging slot** if you have selected **Staging**. If publish succeeds, you can use the displayed endpoint URL to access your LUIS app.

> **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.

TIP

The endpoint URL corresponds to the Azure region associated with the endpoint key. To see endpoints and keys associated with other regions, use the radio buttons to switch regions. Each row in the **Resources and Keys** table lists Azure resource associated with your account and the endpoint keys associated with that resource, For more information, see Regions and keys.

Test your published endpoint in a browser

You can test your published endpoint in a browser using the generated URL. Copy the URL, then replace the `{YOUR-KEY-HERE}` with one of the keys listed in the **Key String** column for the resource you want to use. To open this URL in your browser, set the URL parameter "&q" to your test query. For example, append `&q=Book me a flight to Boston on May 4` to your URL, and then press Enter. The browser displays the JSON response of your HTTP endpoint.


```
{
  "query": "Book me a flight to Boston on May 4",
  "topScoringIntent": {
    "intent": "BookFlight",
    "score": 0.7119028
  },
  "entities": [
    {
      "entity": "may 4",
      "type": "builtin.datetimeV2.date",
      "startIndex": 30,
      "endIndex": 34,
      "resolution": {
        "values": [
          {
            "timex": "XXXX-05-04",
            "type": "date",
            "value": "2017-05-04"
          },
          {
            "timex": "XXXX-05-04",
            "type": "date",
            "value": "2018-05-04"
          }
        ]
      }
    },
    {
      "entity": "4",
      "type": "builtin.number",
      "startIndex": 34,
      "endIndex": 34,
      "resolution": {
        "value": "4"
      }
    }
  ]
}
```


Next steps

- See Manage keys to add keys to your LUIS app, and learn about how keys map to regions.

- See Train and test your app for instructions on how to test your published app in the test console.

Prebuilt entities

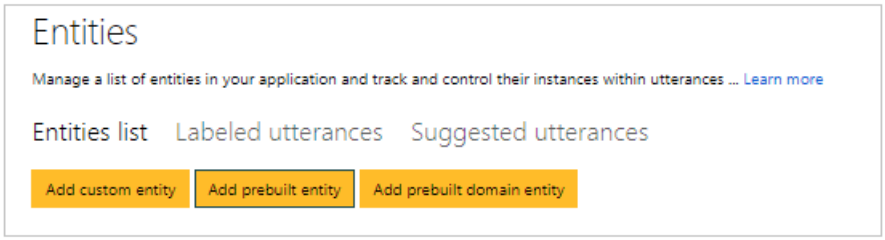
11/6/2017 • 24 min to read • Edit Online

LUIS includes a set of prebuilt entities for recognizing common types of information, like dates, times, numbers, measurements and currency. Prebuilt entity support varies by the culture of your LUIS app. For a full list of the prebuilt entities that LUIS supports, including support by culture, see the prebuilt entity reference.

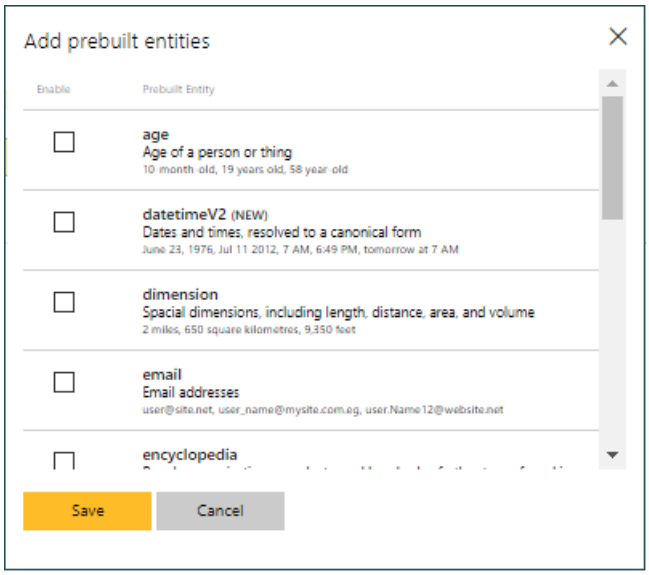
> **NOTE**
>
> **builtin.datetime** is deprecated. It is replaced by **built-in.datetimeV2**, which provides recognition of date and time ranges, as well as improved recognition of ambiguous dates and times.

Add a prebuilt entity

1. Open your 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 (for example, "datetimeV2"). Then click **Save**.

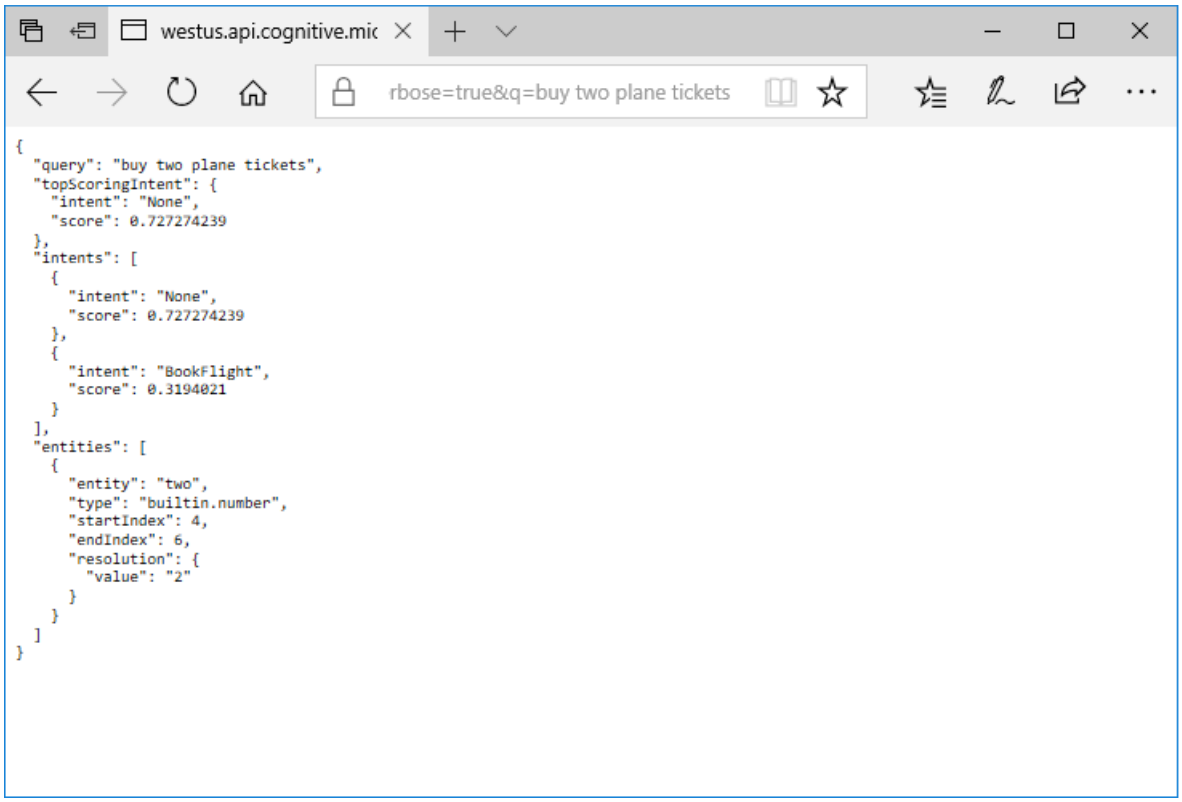


Use a prebuilt number entity

When a prebuilt entity is included in your application, its predictions are included in your published application. The behavior of prebuilt entities is pre-trained and **cannot** be modified. Follow these steps to see how a prebuilt entity works:

1. Add a **number** entity to your app, then Train and publish the app.

2. Click on the endpoint URL in the **Publish App** page to open the LUIS endpoint in a web browser.
3. Append an utterance to the URL that contains an numerical expression. For example, you can type in `buy two plane ticktets`, and see that LUIS identifies `two` as a `builtin.number` entity, and identifies `2` as its value in the `resolution` field. The `resolution` field helps you resolve numbers and dates to a canonical form that's easier for your client application to use.



LUIS can intelligently recognize numbers that aren't in non standard form. Try out different numerical expressions in your utterances and see what LUIS 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 tickets for group travel",
  "topScoringIntent": {
    "intent": "BookFlight",
    "score": 0.905443209
  },
  "entities": [
    {
      "entity": "two dozen",
      "type": "builtin.number",
      "startIndex": 6,
      "endIndex": 14,
      "resolution": {
        "value": "24"
      }
    }
  ]
}
```

Use a prebuilt datetimeV2 entity

The **datetimeV2** prebuilt entity recognizes dates, times, date ranges and time durations. Follow these steps to see how the `datetimeV2` prebuilt entity works:

1. Add a **datetimeV2** entity to your app, then Train and publish the app.
2. Click on the endpoint URL in the **Publish App** page to open the LUIS endpoint in a web browser.
3. Append an utterance to the URL that contains a date range. For example, you can type in `book a flight tomorrow`, and see that LUIS identifies `tomorrow` as a `builtin.datetimeV2.date` entity, and identifies tomorrow's date as its value in the `resolution` field.

The following example shows what the JSON response from LUIS might look like if today's date were October 31st 2017.


```
{
  "query": "book a flight tomorrow",
  "topScoringIntent": {
    "intent": "BookFlight",
    "score": 0.9063408
  },
  "entities": [
    {
      "entity": "tomorrow",
      "type": "builtin.datetimeV2.date",
      "startIndex": 14,
      "endIndex": 21,
      "resolution": {
        "values": [
          {
            "timex": "2017-11-01",
            "type": "date",
            "value": "2017-11-01"
          }
        ]
      }
    }
  ]
}
```


Next Steps

Learn more about prebuilt entities in the prebuilt entity reference.

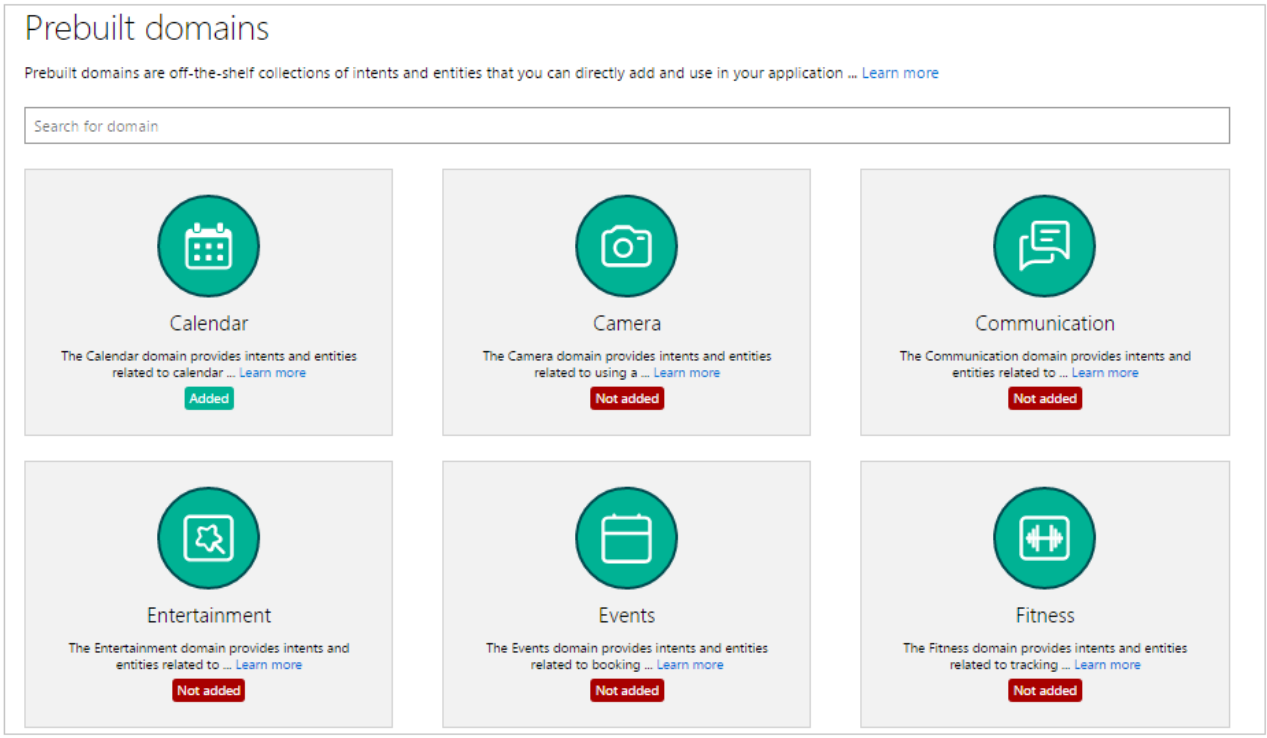
Use prebuilt domains in LUIS apps

11/6/2017 • 3 min to read • Edit Online

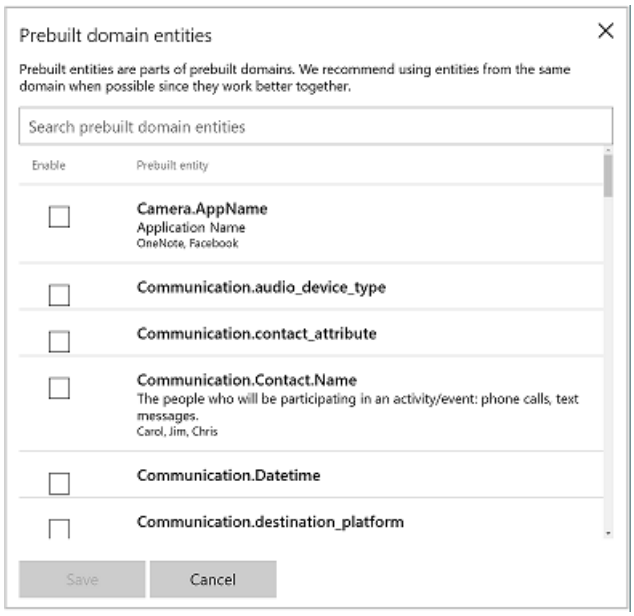
LUIS provides *prebuilt domains*, which are prebuilt sets of intents and entities that work together for domains or common categories of apps. The prebuilt domains have been pre-trained and are ready for use. The intents and entities in a prebuilt domain are fully customizable once you've added them 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.

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.

> **TIP**
> You can find a full listing of the prebuilt domains at https://www.luis.ai/prebuilts.



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. They 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 can to add entities for sports teams. 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.

List of prebuilt domains

LUIS offers 20 prebuilt domains.

PREBUILT DOMAIN	DESCRIPTION
Calendar	The Calendar domain provides intent and entities for adding, deleting, or editing an appointment, checking participants availability, and finding information about a calendar event.
Camera	The Camera domain provides intents and entities for taking pictures, recording videos, and broadcasting video to an application.
Communication	Sending messages and making phone calls.
Entertainment	Handling queries related to music, movies, and TV.
Events	Booking tickets for concerts, festivals, sports games and comedy shows.
Fitness	Handling requests related to tracking fitness activities.
Gaming	Handling requests related to a game party in a multiplayer game.
HomeAutomation	Controlling smart home devices like lights and appliances.
MovieTickets	Booking tickets to movies at a movie theater.
Music	Playing music on a music player.
Note	The Note domain provides intents and entities related to creating, editing, and finding notes.
OnDevice	The OnDevice domain provides intents and entities related to controlling the device.
Places	Handling queries related to places like businesses, institutions, restaurants, public spaces, and addresses.
Reminder	Handling requests related to creating, editing, and finding reminders.
RestaurantReservation	Handling requests to manage restaurant reservations.
Taxi	Handling bookings for a taxi.
Translate	Translating text to a target language.
TV	Controlling TVs.
Utilities	Handling requests that are common in many domains, like "help", "repeat", "start over."

PREBUILT DOMAIN	DESCRIPTION
Weather	Getting weather reports and forecasts.
Web	Navigating to a website.

Cortana Prebuilt App

11/6/2017 • 2 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 LUIS app 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.

Get the endpoint for the Cortana prebuilt app

You can access the Cortana prebuilt app using the following endpoints.

LANGUAGE	ENDPOINT
English	https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/c413b2ef-382c-45bd-8ff0-f76d60e2a821
Chinese	https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/c27c4af7-d44a-436f-a081-841bb834fa29
French	https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/0355ead1-2d08-4955-ab95-e263766e8392
Spanish	https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/cb2675e5-fbea-4f8b-8951-f071e9fc7b38
Italian	https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/30a0fddc-36f4-4488-b022-03de084c1633

NOTE

The endpoint URLs are also available from the apps - Get personal assistant applications API.

Try out the personal assistant app

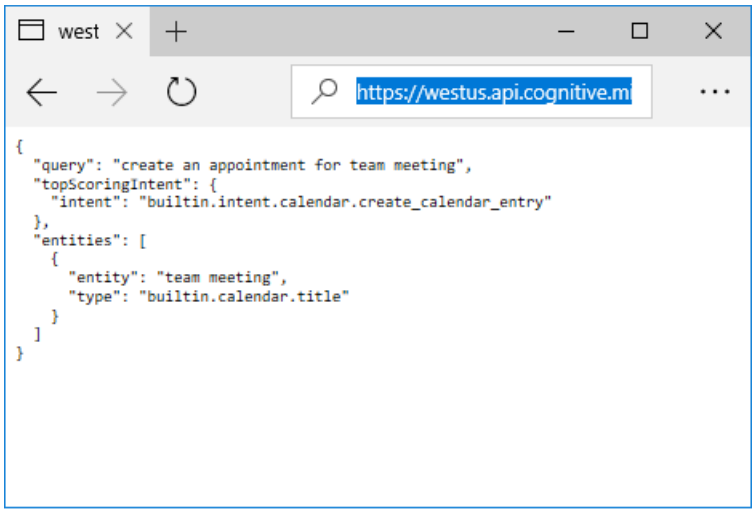
To call the endpoint, you can append your subscription key argument and query string to the endpoint.

For example, if the utterance you want to interpret is "create an appointment for team meeting", then you can append that utterance to the endpoint URL.

```
https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/c413b2ef-382c-45bd-8ff0-f76d60e2a821?subscription-key={YOUR-SUBSCRIPTION-KEY}&q=create an appointment for team meeting
```

You can paste the URL into a web browser, and substitute your subscription key for the `{YOUR-SUBSCRIPTION-KEY}` field.

In the browser you can see that the Cortana prebuilt app identifies `builtin.intent.calendar.create_calendar_entry` as the intent, and `builtin.calendar.title` as the entity type, and for the utterance `create an appointment for team meeting`.



Next steps

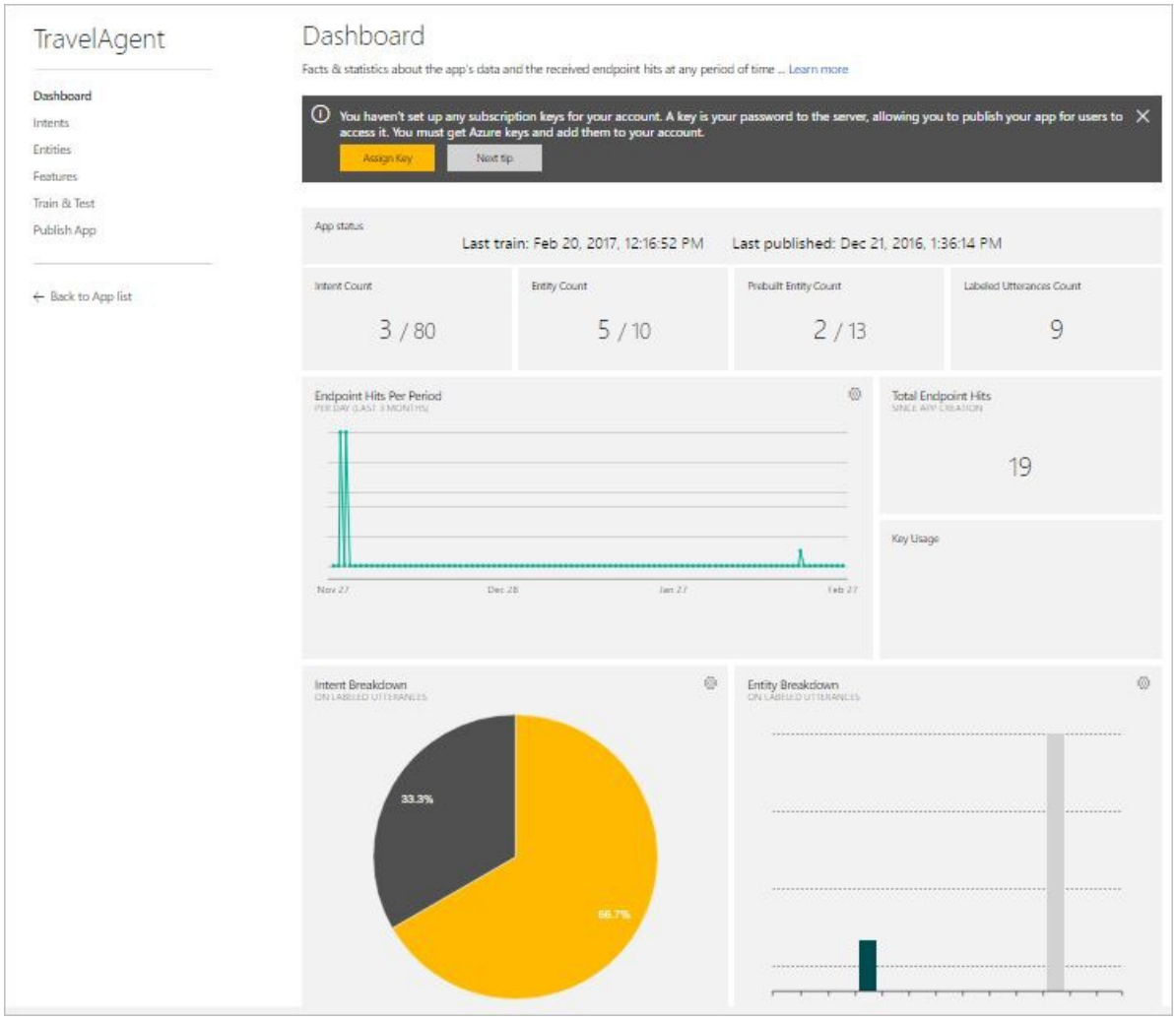
Cortana prebuilt app reference

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

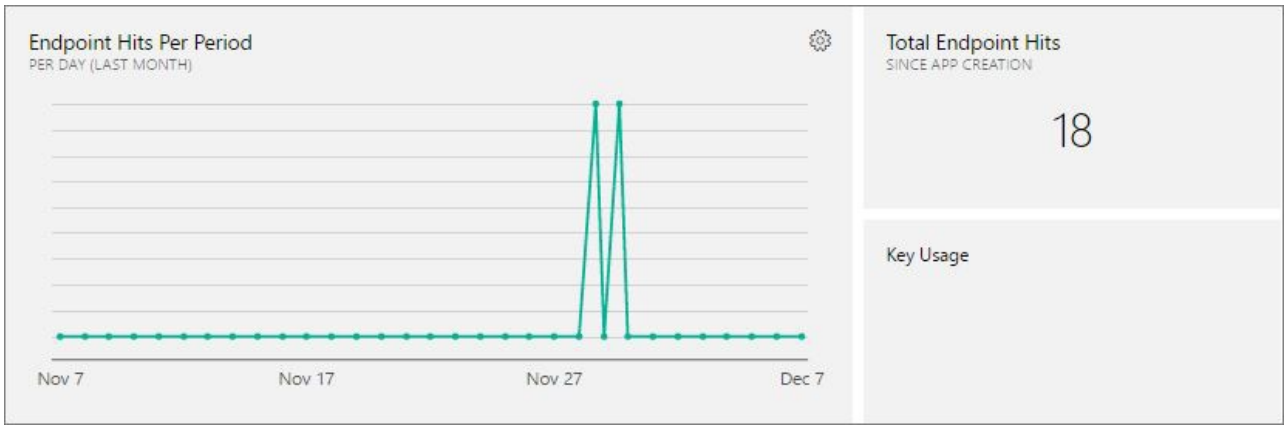
The dashboard displays the application's training and publishing status, including the date and time when the app was last trained and published.

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.

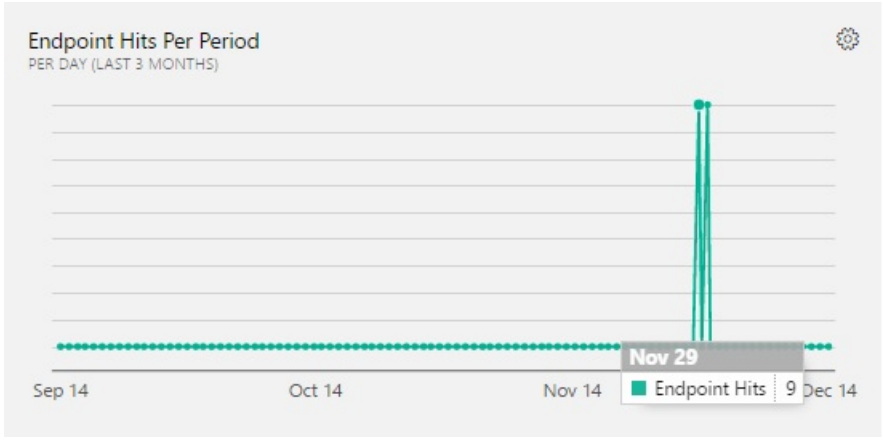


Total endpoint hits

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

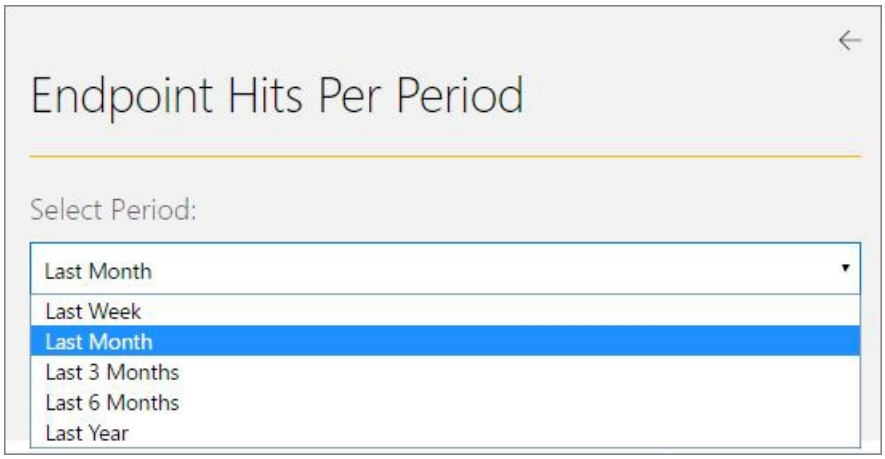
Endpoint hits per period

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.



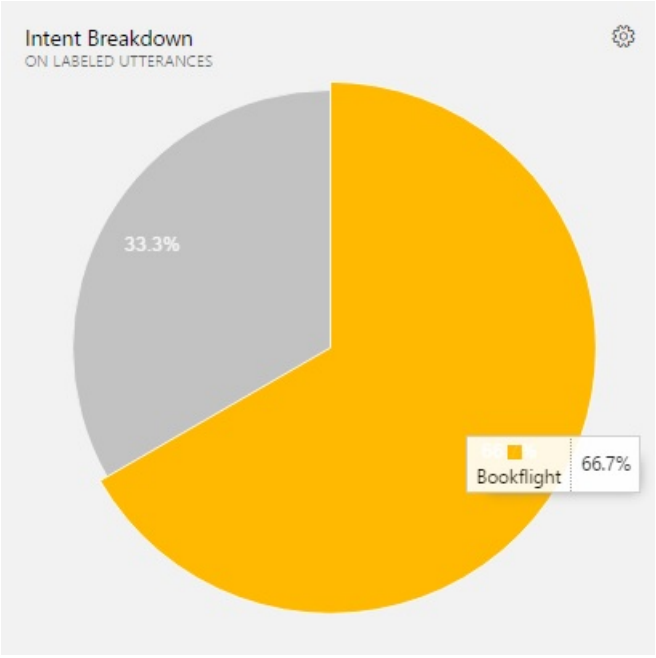
2. Select a period from the list and then click the back arrow to display its hits on the chart.

Key usage

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

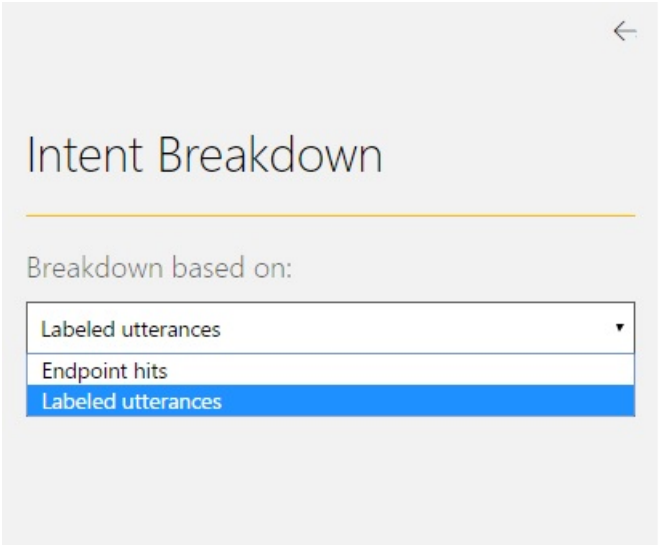
Intent Breakdown

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:

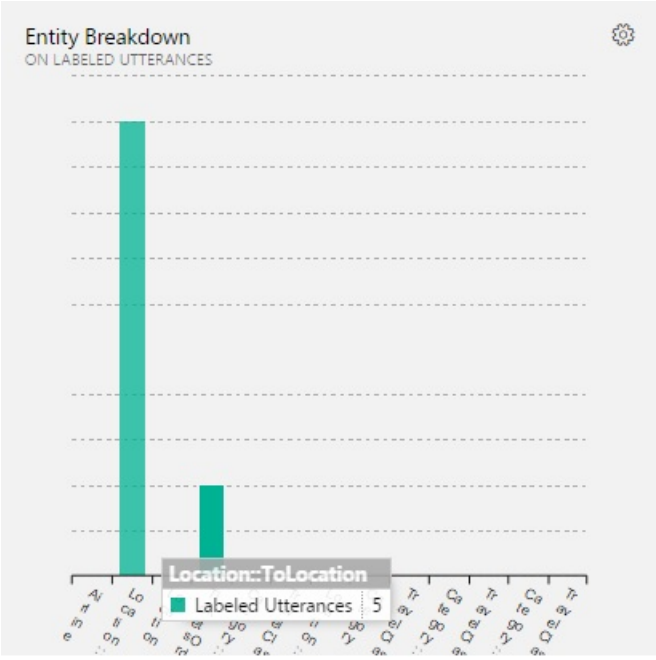
1. Click **Additional Settings** to access the list as in the screenshot below.



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

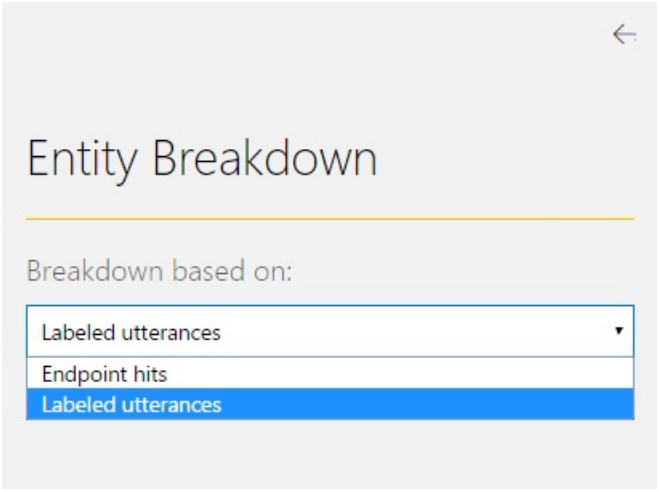
Entity Breakdown

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

11/2/2017 • 5 min to read • Edit Online

A key is your passport to the server that allows you to publish your LUIS app to end users. LUIS has two different types of keys:

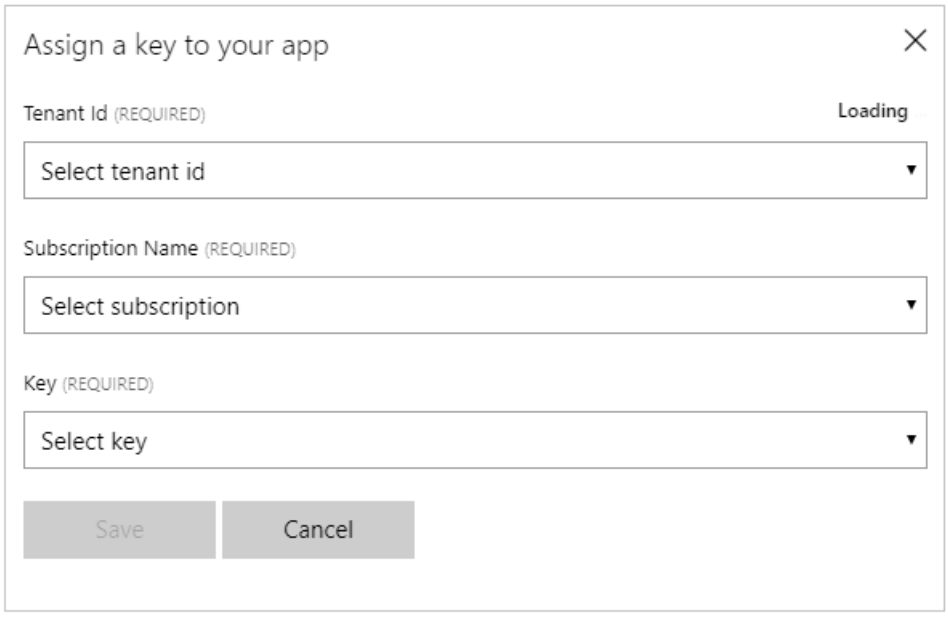
- **Starter Key or Programmatic Key:** A starter key, also known as a programmatic key, is created automatically for LUIS account and it's free. It gives you 1000 endpoint hits per month so you can start using your LUIS app. It also gives you unlimited hits for authoring and editing your application using the LUIS Programmatic APIs. Click here for a complete API Reference.

  > **TIP**
  > To find the Programmatic Key, log in to https://www.luis.ai and click on the account name in the upper-right navigation bar to open **Account Settings**, which displays the Programmatic Key.

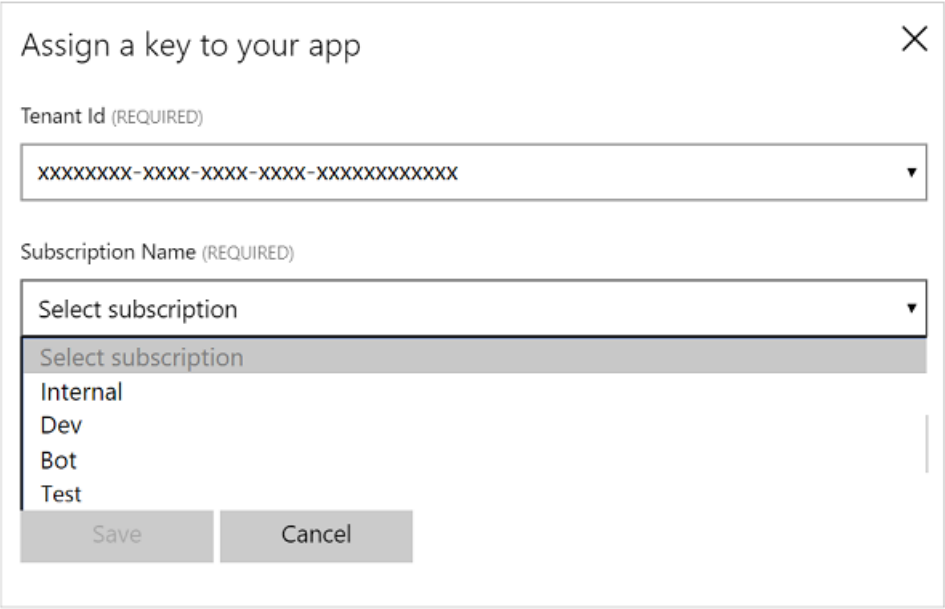
- **Endpoint Key(s):** If you need more than 1000 hits per month you can buy a key from the Microsoft Azure portal. It is essential for publishing your app and accessing your HTTP endpoint. This key allows a quota of endpoint hits based on the usage plan you specified when creating the key. See Cognitive Services Pricing for pricing information.

The process of creating and using endpoint keys involves the following tasks:

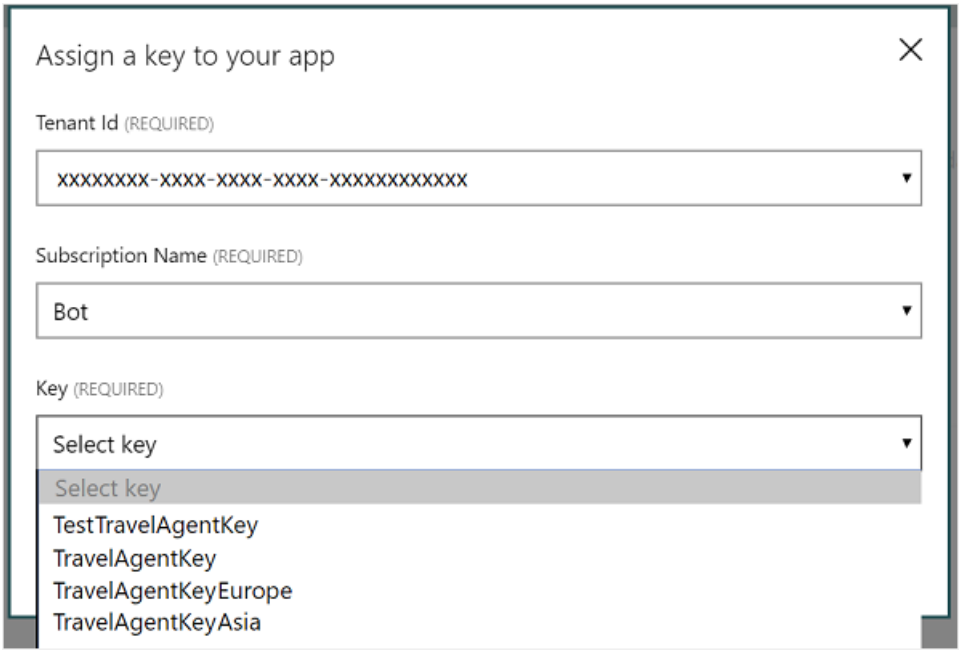
1. Create a key on the Azure portal. For further instructions, see Creating a subscription key using Azure.
2. On the **Publish app** page, click the **Add Key** button to open the **Assign a key to your app** dialog.



3. Select a Tenant ID in the dialog. In Azure, a tenant represents the client or organization associated with a service.
4. Choose the subscription associated with the key you want to add.



5. Select a key associated with the subscription.



Regions and keys

The region to which you publish your LUIS app corresponds to the region or location you specify in the Azure portal when you create a key. When you publish an app, LUIS automatically generates an endpoint URL for the region associated with the 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 2	https://eastus2.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

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

> **IMPORTANT**
>
> If you attempt to publish to https://www.luis.ai using a key in the Europe Azure region, LUIS displays a warning message. Instead, use https://eu.luis.ai.

Publishing to Europe

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

Next steps

Use your key to publish your app in the **Publish app** page. For instructions on publishing, see Publish app.

Creating Subscription Keys on Azure

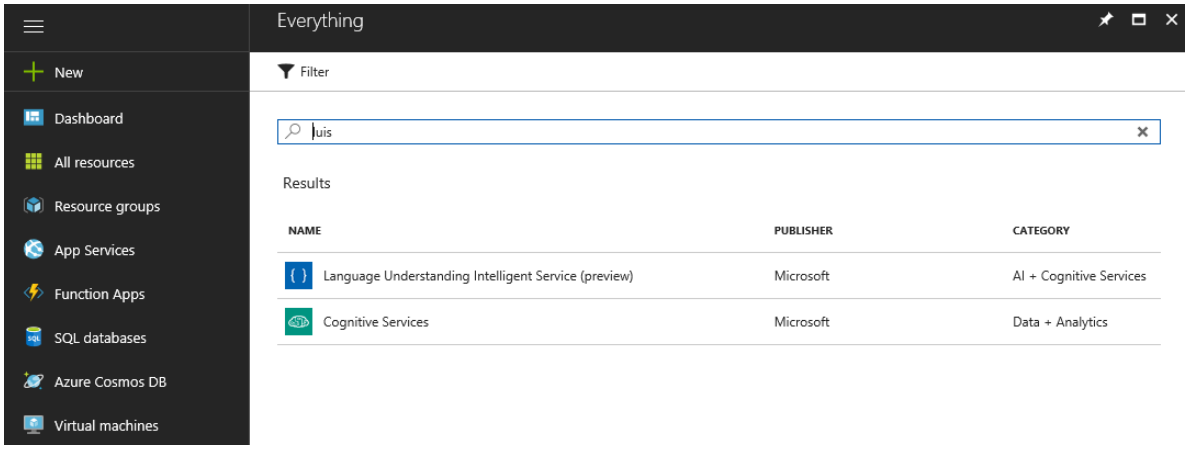
11/3/2017 • 1 min to read • Edit Online

For unlimited traffic to your HTTP endpoint, you must create an Azure subscription for the LUIS service. The Azure subscription creates metered access keys to your endpoint following a payment plan. See Cognitive Services Pricing for pricing information.

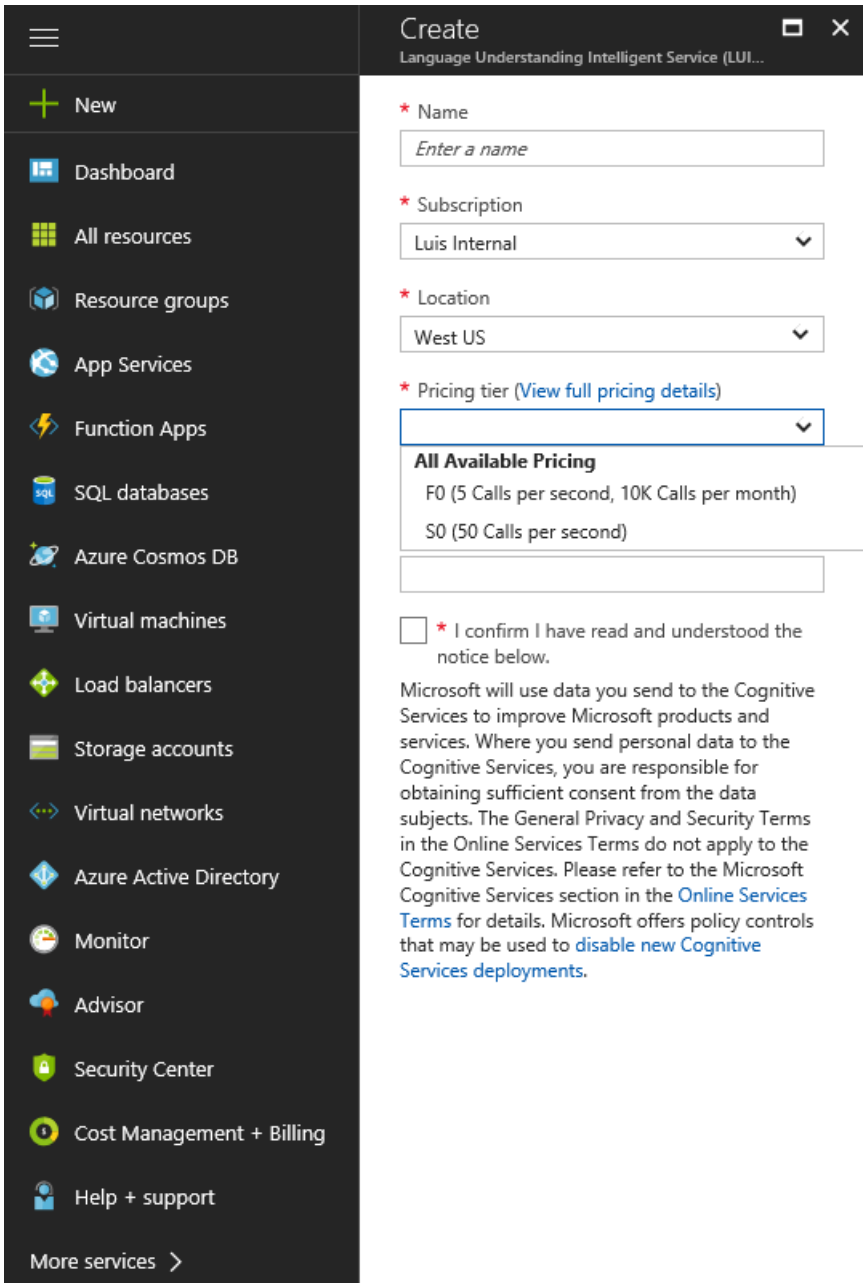
The metered plan allows requests to your LUIS account at a specific rate. If the rate of requests is higher than the allowed rate of your metered account per minute or per month, requests receive an http error of "429: Too Many Requests."

To create your key, follow these steps:

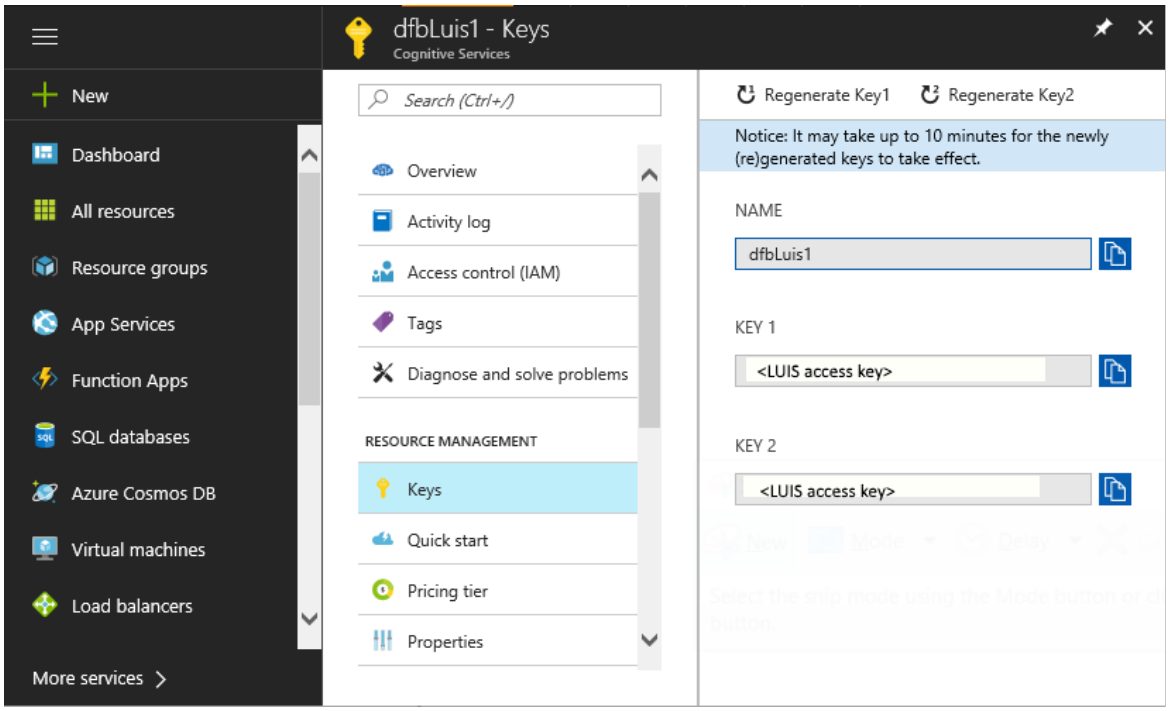
1. Sign in to the **Microsoft Azure portal**
2. Click the green **+** sign in the upper left-hand panel and search for "LUIS" in the marketplace, then click on **Language Understanding Intelligent Service (preview)** and follow the **create experience** to create a LUIS subscription account.



3. Configure the subscription with settings including account name, pricing tiers, etc.



4. Once you have created the LUIS subscription account, you can view the access keys generated in the **Resource Management->Keys** blade. Test your access keys in your **luis.ai account**, or by following the LUIS documentation to create a new endpoint application.



Using LUIS access keys in luis.ai

In order to use the access keys you created in Azure, you need to log in to luis.ai and add the new access keys as part of publishing your app.

Next steps

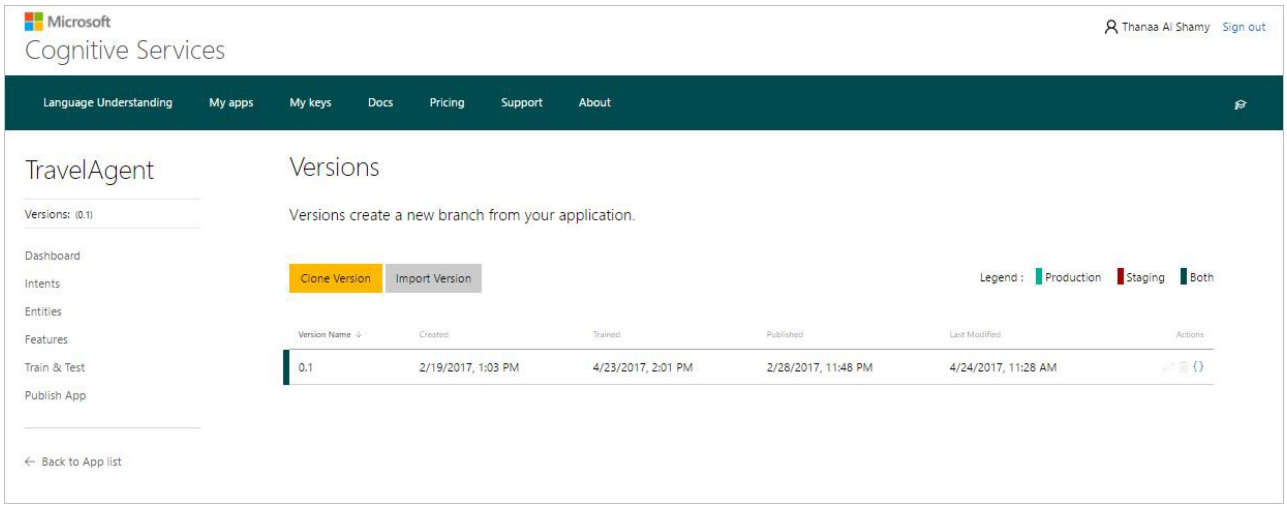
See Manage your keys for more information about how to add and manage your Azure LUIS subscription keys in your **luis.ai account**.

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.



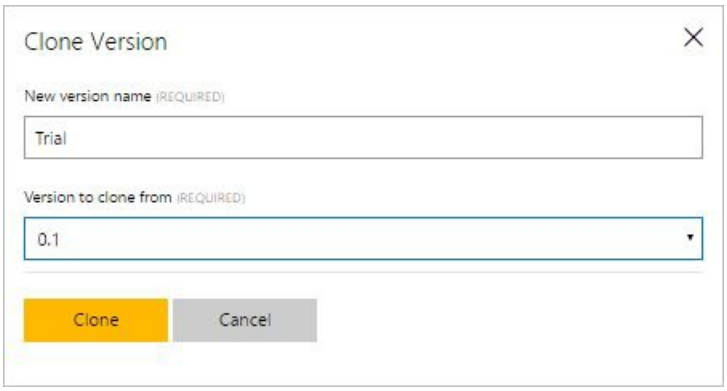
The following procedures will guide you through the actions you need to do while working with versions.

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.

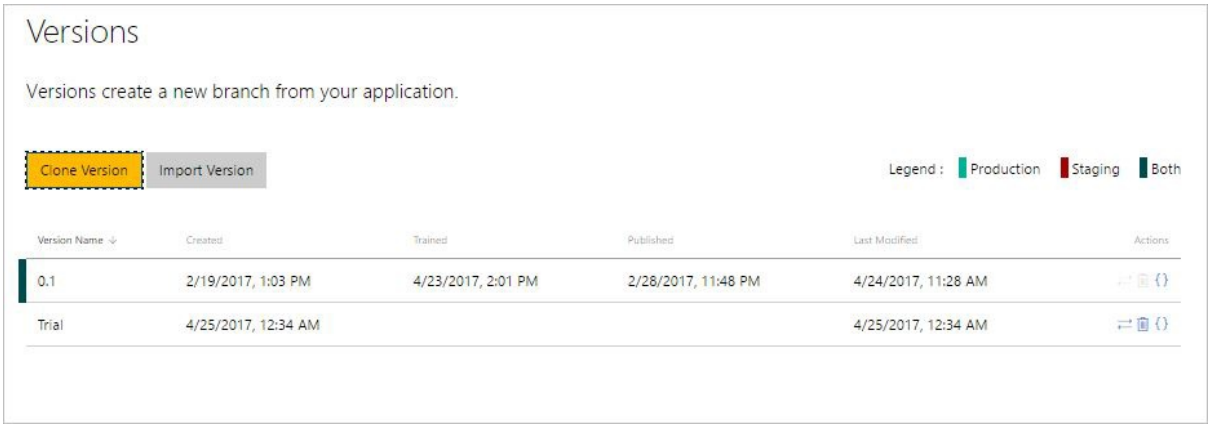
To clone a version:

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.

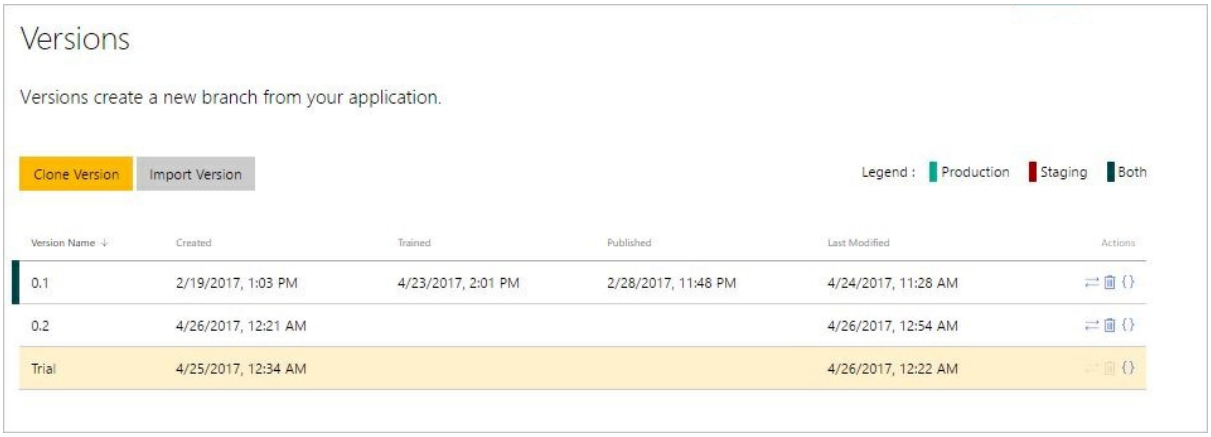
Set a version as active

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

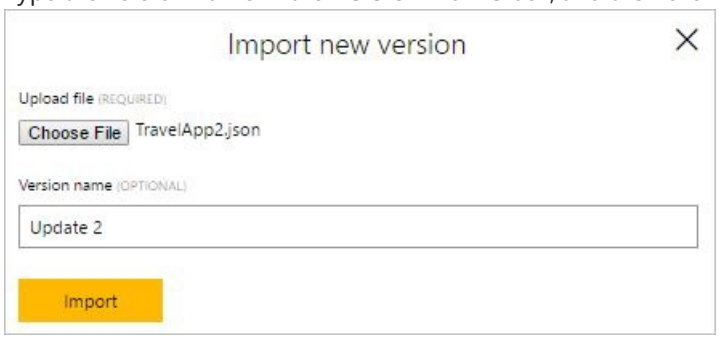
You can import/export a version as a JSON file.

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:

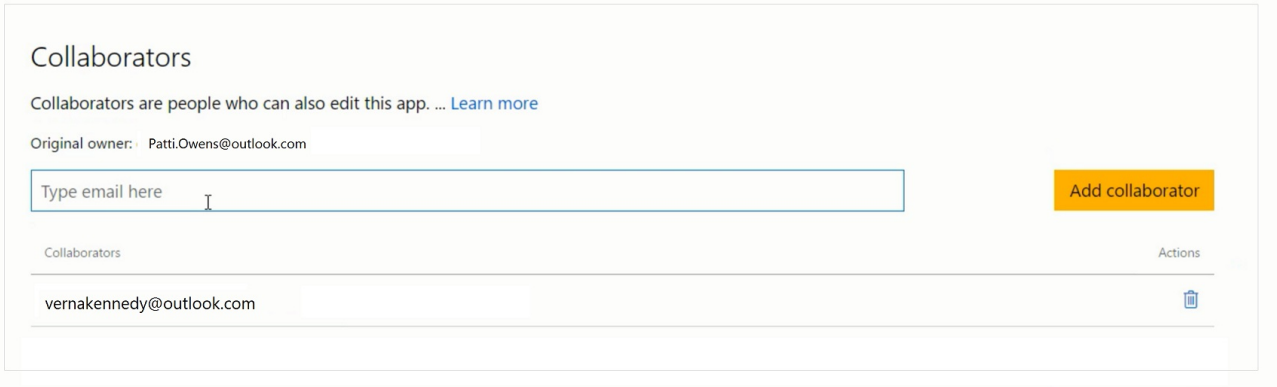
1. Click the **Delete Version** button , corresponding to the version you want to delete.
2. In the confirmation message, click **Yes** to confirm deletion.

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

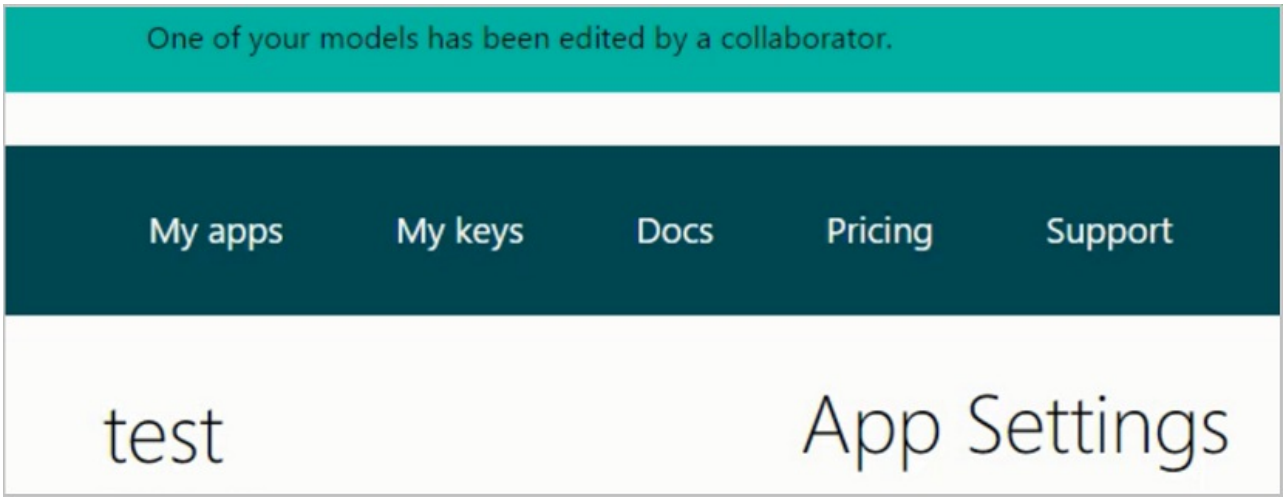
10/18/2017 • 1 min to read • Edit Online

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 notification.



NOTE

Collaborators cannot add other collaborators.

Prebuilt entities reference

11/9/2017 • 19 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 **built-in.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 celsius`	`{ "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 following this table
`builtin.encyclopedia`	See separate table	See separate table following this table

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 <a href=#subtypes-of-datetimev2>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 <a href=#subtypes-of-datetimev2>subtypes of datetimeV2: • builtin.datetimeV2.datetime • builtin.datetimeV2.date • builtin.datetimeV2.time • builtin.datetimeV2.daterange • builtin.datetimeV2.timerange • 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. When there is an ambiguous date, `values` contains the most recent past and most immediate future instances of the date. See Ambiguous dates for more examples. When there is 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, timerange, 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`, `timerange</code>, 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`, `timerange</code>, or `datetimerange`.	

The **builtin.datetimeV2** supports dates between the following ranges.

MIN	MAX
1st January 1900	31st December 2099

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 occurrence 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. 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"
          }
        ]
      }
    }
  ]
```

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 "6pm to 7pm".

```
"entities": [
  {
    "entity": "6pm to 7pm",
    "type": "builtin.datetimeV2.timerange",
    "startIndex": 0,
    "endIndex": 9,
    "resolution": {
      "values": [
        {
          "timex": "(T18,T19,PT1H)",
          "type": "timerange",
          "start": "18:00:00",
          "end": "19:00:00"
        }
      ]
    }
  }
]
```

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.timerange
- 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"}] } }`

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.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"}] } }`
`builtin.datetimeV2.timerange`	`next week`	`{ "entity": "6pm to 7pm", "type": "builtin.datetime.dateV2.timerange", "resolution": { "values": [{ "timex": "(T18,T19,PT1H)", "type": "timerange", "start": "18:00:00", "end": "19:00:00"}] } }`
`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"}] } }`

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
`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" }] } }`

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
`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"} }`
`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"}] } }`

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
`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"} }`
`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"}] } }`

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
`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" }`

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
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 in the following table. 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.cinematograp	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.mart	bruce lee
builtin.encyclopedia.people.person	builtin.encyclopedia.architecture.arch	james gallier

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.people.person	builtin.encyclopedia.geography.mountai	jean couzy
builtin.encyclopedia.people.person	builtin.encyclopedia.celebrities.celeb	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	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.politi	harold nicolson
builtin.encyclopedia.people.person	builtin.encyclopedia.government.us_pre	barack obama
builtin.encyclopedia.people.person	builtin.encyclopedia.government.us_vic	dick cheney
builtin.encyclopedia.organization.orga	builtin.encyclopedia.organization.orga	united nations
builtin.encyclopedia.organization.orga	builtin.encyclopedia.sports.league	american league
builtin.encyclopedia.organization.orga	builtin.encyclopedia.ice_hockey.confer	western hockey league
builtin.encyclopedia.organization.orga	builtin.encyclopedia.baseball.division	american league east
builtin.encyclopedia.organization.orga	builtin.encyclopedia.baseball.league	major league baseball
builtin.encyclopedia.organization.orga	builtin.encyclopedia.basketball.confer	national basketball league
builtin.encyclopedia.organization.orga	builtin.encyclopedia.basketball.divisi	pacific division
builtin.encyclopedia.organization.orga	builtin.encyclopedia.soccer.league	premier league
builtin.encyclopedia.organization.orga	builtin.encyclopedia.american_football	afc north

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.organization.orga	builtin.encyclopedia.broadcast.broadca	nebraska educational telecommunications
builtin.encyclopedia.organization.orga	builtin.encyclopedia.broadcast.tv_stat	abc
builtin.encyclopedia.organization.orga	builtin.encyclopedia.broadcast.tv_chan	cnbc world
builtin.encyclopedia.organization.orga	builtin.encyclopedia.broadcast.radio_s	bbc radio 1
builtin.encyclopedia.organization.orga	builtin.encyclopedia.business.operatio	bank of china
builtin.encyclopedia.organization.orga	builtin.encyclopedia.music.record_labe	pixar
builtin.encyclopedia.organization.orga	builtin.encyclopedia.aviation.airline	air france
builtin.encyclopedia.organization.orga	builtin.encyclopedia.automotive.compan	general motors
builtin.encyclopedia.organization.orga	builtin.encyclopedia.music.musical_ins	gibson guitar corporation
builtin.encyclopedia.organization.orga	builtin.encyclopedia.tv.network	cartoon network
builtin.encyclopedia.organization.orga	builtin.encyclopedia.education.educati	cornwall hill college
builtin.encyclopedia.organization.orga	builtin.encyclopedia.education.school	boston arts academy
builtin.encyclopedia.organization.orga	builtin.encyclopedia.education.univers	johns hopkins university
builtin.encyclopedia.organization.orga	builtin.encyclopedia.sports.team	united states national handball team
builtin.encyclopedia.organization.orga	builtin.encyclopedia.basketball.team	chicago bulls
builtin.encyclopedia.organization.orga	builtin.encyclopedia.sports.profession	boston celtics
builtin.encyclopedia.organization.orga	builtin.encyclopedia.cricket.team	mumbai indians
builtin.encyclopedia.organization.orga	builtin.encyclopedia.baseball.team	houston astros
builtin.encyclopedia.organization.orga	builtin.encyclopedia.american_football	green bay packers
builtin.encyclopedia.organization.orga	builtin.encyclopedia.ice_hockey.team	hamilton bulldogs
builtin.encyclopedia.organization.orga	builtin.encyclopedia.soccer.team	fc bayern munich
``builtin.encyclopedia.organization.orga nization	builtin.encyclopedia.government.politica l_party	pertubuhan kebangsaan melayu singapura```
builtin.encyclopedia.time.event	builtin.encyclopedia.time.event	1740 batavia massacre

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.time.event	builtin.encyclopedia.sports.championsh	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
builtin.encyclopedia.tv.series_episode	builtin.encyclopedia.tv.multipart_tv_e	the deadly assassin
builtin.encyclopedia.commerce.consumer	builtin.encyclopedia.commerce.consumer	nokia lumia 620
builtin.encyclopedia.commerce.consumer	builtin.encyclopedia.music.album	dance pool
builtin.encyclopedia.commerce.consumer	builtin.encyclopedia.automotive.model	pontiac fiero
builtin.encyclopedia.commerce.consumer	builtin.encyclopedia.computer.computer	toshiba satellite
builtin.encyclopedia.commerce.consumer	builtin.encyclopedia.computer.web_brow	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_progr	builtin.encyclopedia.radio.radio_progr	grand ole opry
builtin.encyclopedia.film.film	builtin.encyclopedia.film.film	alice in wonderland
builtin.encyclopedia.cricket.tournamen	builtin.encyclopedia.cricket.tournamen	cricket world cup
builtin.encyclopedia.government.governi	builtin.encyclopedia.government.governi	european commission
builtin.encyclopedia.sports.team_owner	builtin.encyclopedia.sports.team_owner	bob castellini
builtin.encyclopedia.music.genre	builtin.encyclopedia.music.genre	eastern europe

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.ice_hockey.divisi	builtin.encyclopedia.ice_hockey.divisi	hockeyallsvenskan
builtin.encyclopedia.architecture.styl	builtin.encyclopedia.architecture.styl	spanish colonial revival architecture
builtin.encyclopedia.broadcast.produce	builtin.encyclopedia.broadcast.produce	columbia tristar television
builtin.encyclopedia.book.author	builtin.encyclopedia.book.author	adam maxwell
builtin.encyclopedia.religion.founding	builtin.encyclopedia.religion.founding	gautama buddha
builtin.encyclopedia.martial_arts.mart	builtin.encyclopedia.martial_arts.mart	american kenpo
builtin.encyclopedia.sports.school	builtin.encyclopedia.sports.school	yale university
builtin.encyclopedia.business.product_	builtin.encyclopedia.business.product_	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_exc	builtin.encyclopedia.finance.stock_exc	tokyo stock exchange
builtin.encyclopedia.film.festival	builtin.encyclopedia.film.festival	berlin international film festival

Cortana prebuilt app reference

11/6/2017 • 12 min to read • Edit Online

This reference lists the intents and entities that the Cortana prebuilt app recognizes.

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?
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?

INTENT	EXAMPLE UTTERANCES
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
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

INTENT	EXAMPLE UTTERANCES
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 on speakerphone
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
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

INTENT	EXAMPLE UTTERANCES
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
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?

INTENT	EXAMPLE UTTERANCES
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 up my picture day reminder 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
builtin.intent.reminder.snooze	snooze that reminder until tomorrow snooze this reminder
builtin.intent.reminder.turn_off_reminder	turn off reminder dismiss airport pickup 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

INTENT	EXAMPLE UTTERANCES
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
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`

ENTITY	EXAMPLE OF ENTITY IN UTTERANCE
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
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` 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

ENTITY	EXAMPLE OF ENTITY IN UTTERANCE
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`
builtin.comminication.source_platform	call him from `skype` call him from my `personal line`
builtin.mystuff.attachment	with documents `attached` find the email `attachment` bob sent
builtin.mystuff.contact_name	find the spreadsheet lisa sent to `me` where's the document i sent to `susan` last night
builtin.mystuff.data_source	`c:\dev\` my `desktop`
builtin.mystuff.data_type	locate the `document` i worked on last night
builtin.mystuff.end_date	show me the docs i worked on `between yesterday and today` find what doc i was working on `before thursday the 31st`
builtin.mystuff.end_time	find files i saved `before noon` find what doc i was working on `before 4pm`
builtin.mystuff.file_action	open the spreadsheet i `saved` yesterday find the spreadsheet kevin `created`
builtin.mystuff.from_contact_name	find the proposal `jason` sent me open `isaac` 's last email
builtin.mystuff.keyword	show me the `french conjugation` files find the `marketing plan` i drafted yesterday
builtin.mystuff.location	the document i edited at `work` photos i took in `paris`
builtin.mystuff.message_category	look for my `new` emails search for my `high priority` email
builtin.mystuff.message_type	check my `email` show me my `text` messages
builtin.mystuff.source_platform	search my `hotmail` email for email from john find the document i sent from `work`

ENTITY	EXAMPLE OF ENTITY IN UTTERANCE
builtin.mystuff.start_date	find notes from `january` find the email i send rob `after january 1st`
builtin.mystuff.start_time	find that email i sent rob sometime before 2pm but `after noon`? find the worksheet kristin sent to me that I edited `last night`
builtin.mystuff.title	`c:\dev\mystuff.txt` `*.txt`
builtin.mystuff.transform_action	`download` the file john sent me `open` my annotation guidelines doc
builtin.note.note_text	create a grocery list including `pork chops, applesauce and milk` make a note to `buy milk`
builtin.note.title	make a note called `grocery list` make a note called `people to call`
builtin.ondevice.music_artist_name	play everything by `rufus wainwright` play `garth brooks` music
builtin.ondevice.music_genre	show `classic rock` songs play my `classical` music from the baroque period
builtin.ondevice.music_playlist	shuffle all britney spears from `workout` playlist play `breakup` playlist
builtin.ondevice.music_song_name	play `summertime` play `me and bobby mcgee`
builtin.ondevice.setting_type	`quiet hours` `airplane mode`
builtin.places.absolute_location	take me to the intersection of `5th and pike` no, I want directions to `1 microsoft way redmond wa 98052`
builtin.places.atmosphere	look for `interesting` places to go out where can I find a `casual` restaurant
builtin.places.audio_device_type	call the post office on `hands free` call papa john's with `speakerphone`
builtin.places.avoid_route	avoid the `toll road` get me to san francisco avoiding the `construction on 101`
builtin.places.cuisine	`halal` deli near mountain view `kosher` fine dining on the peninsula

ENTITY	EXAMPLE OF ENTITY IN UTTERANCE
builtin.places.date	make a reservation for `next friday the 12th` is mashiko open on `mondays` ?
builtin.places.discount_type	find a `coupon` for macy's find me a `coupon`
builtin.places.distance	is there a good diner `within 5 miles` of here find ones `within 15 miles`
builtin.places.from_absolute_location	directions from `45 elm street` to home get me directions from `san francisco` to palo alto
builtin.places.from_place_name	driving from the `post office` to 56 center street get me directions from `home depot` to lowes
builtin.places.from_place_type	directions to downtown from `work` get me directions from the `drug store` to home
builtin.places.meal_type	nearby places for `dinner` find a good place for a business `lunch`
builtin.places.nearby	show me some cool shops `near` me are there any good Lebanese restaurants `around` here?
builtin.places.open_status	when is the mall `closed` get me the `opening` hours of the store
builtin.places.place_name	take me to `central park` look up the `eiffel tower`
builtin.places.place_type	`atms` `post office`
builtin.places.prefer_route	show directions by the `shortest` route take the `fastest` route
builtin.places.price_range	give me places that are `moderately affordable` I want an `expensive` one
builtin.places.product	where can I get `fresh fish` around here where around here sells `bare minerals`
builtin.places.public_transportation_route	bus schedule for the `m2` bus bus route `3x`
builtin.places.rating	show `3 star` restaurants show results that are `3 stars or higher`
builtin.places.reservation_number	book a table for `seven` people make a reservation for `two` at il fornaio

ENTITY	EXAMPLE OF ENTITY IN UTTERANCE
builtin.places.results_number	show me the `10` coffee shops closest to here show me top `3` aquariums
builtin.places.service_provided	where can I go to `whale watch` by bus ? I need a mechanic to `fix my brakes`
builtin.places.time	I want places that are open on Saturday at `8 am`
builtin.places.transportation_company	train schedules for `new jersey transit` can I get there on `bart`
builtin.places.transportation_type	where is a music store i can get to `on foot` ? give me `biking` directions to mashiko
builtin.places.travel_time	I want to be able to drive `less than 15 minutes` I want somewhere I can get to in `under 15 minutes`
builtin.reminder.absolute_location	remind me to call my dad when i land in `chicago` when I get back to `seattle` remind me to get gas
builtin.reminder.contact_name	when `bob` calls, remind me to tell him the joke create a reminder to mention the school bus when I talk to `arthur`
builtin.reminder.leaving_absolute_location	reminder pick up craig when leaving `1200 main`
builtin.reminder.leaving_implicit_location	remind me to get gas when I leave `work`
builtin.reminder.original_start_date	change the reminder about the lawn from `saturday` to Sunday move my reminder about school from `monday` to Tuesday
builtin.reminder.relationship_name	when my `husband` calls, remind me to tell him about the pta meeting remind me again when `mom` calls
builtin.reminder.reminder_text	can you remind me to `bring up my small spot of patchy skin` when dr smith'calls remind me to `pick up dry cleaning` at 4:40
builtin.reminder.start_date	remind me the `thursday after next` at 8 pm remind me `next thursday the 18th` at 8 pm
builtin.reminder.start_time	create a reminder `in 30 minutes` create a reminder to water the plants `this evening at 7`
builtin.weather.absolute_location	will it rain in `boston` what's the forecast for `seattle` ?

ENTITY	EXAMPLE OF ENTITY IN UTTERANCE
builtin.weather.date_range	weather in nyc `this weekend` look up the `five day` forecast in hollywood florida
builtin.weather.suitable_for	can I go `hiking` in shorts this weekend? will it be nice enough to `walk` to the game today?
builtin.weather.temperature_unit	what is the temperature today in `kelvin` show me the temps in `celsius`
builtin.weather.time_range	does it look like it will snow `tonight`? is it windy right `now`?
builtin.weather.weather_condition	show `precipitation` how thick is the `snow` at lake tahoe now?

Language Understanding Intelligent Services FAQ

11/10/2017 • 8 min to read • Edit Online

This article contains answers to frequently asked questions about Language Understanding Intelligent Services (LUIS).

How do I interpret LUIS scores?

Your system should use the highest scoring intent regardless of its value. For example, a score below 0.5 does not necessarily mean that LUIS has low confidence. Providing more training data can help increase the score of the most-likely intent.

What is the maximum number of intents and entities that a LUIS app can support?

A LUIS app can support up to 80 intents.

Limits on entities depend on the entity type, as shown in the following table:

TYPE	LIMIT
Prebuilt entities	No limit.
List entities	50 list entities. Each list can contain up to 20,000 items.
Simple, hierarchical, and composite entities	You can define up to 30 of these types of entities. A hierarchical entity can consist of up to 10 child entities. A composite entity can consist of up to 20 child entities.

I want to build a LUIS app with more than the maximum number of 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 that the user is asking for, but they don't need to capture every path your code takes. For example, BookFlight and BookHotel might be separate intents in a travel app, but BookInternationalFlight and BookDomesticFlight are too similar. 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, and group related intents. This approach is a good best practice if you're using multiple apps for your system. For example, let's say you're developing an office assistant that has over 80 intents. If 20 intents relate to scheduling meetings, 20 are about reminders, 20 are about getting information about colleagues, and 20 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, such as matching on regular expressions, to determine which LUIS app

or set of apps receives it.

When you're deciding which approach to use with multiple LUIS apps, consider the following trade-offs:

- **Saving suggested utterances for training**: Your LUIS apps get a performance boost when you label the user utterances that the apps receive, 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**: To improve responsiveness, you might ordinarily design a system to reduce the number of REST API calls that happen in series. But if you send the utterance to multiple LUIS apps and pick the intent with the highest score, you can call the apps 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.

If reducing the number of intents or dividing your intents into multiple apps doesn't work for you, contact support. To do so, gather detailed information about your system, go to the Language Understanding Intelligent Service site, and then select **Support**. If your Azure subscription includes support services, contact Azure technical support.

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 might 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 might 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 also provides the list entity type that is not machine-learned but allows your LUIS app to specify a fixed list of values. A list entity can have up to 20,000 items.

If you've considered hierarchical, composite, and list entities and still need more than the limit, contact support. To do so, gather detailed information about your system, go to the Language Understanding Intelligent Service site, and then select **Support**. If your Azure subscription includes support services, contact Azure technical support.

What are the limits on the number and size of phrase lists?

The maximum length of a phrase list is 5,000 items. You can use a maximum of 10 phrase lists per LUIS app.

What is the limit on the length of an utterance?

The maximum length of an utterance is 500 characters.

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

The best way to build your 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 to 200 unlabeled utterances to your app. As you select the most informative utterances to label, you can use the suggestion feature to take advantage of LUIS intelligence. You can select the intent or entity that you want to improve, and then label the utterances that are suggested by LUIS. Labeling a few hundred utterances 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 the best practices for LUIS modeling.

Is it important to train the None intent?

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

How can I deal with spelling mistakes in utterances?

You have one of two options:

- Use a spell checker on your utterances before sending them to the LUIS endpoint. This option might be the easier of the two.
- For greatest diversity, label utterances that have spelling mistakes so that LUIS can learn proper spelling as well as typos. This option requires more labeling effort.

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

The errors indicate that there is some discrepancy between your labels and the predictions from your models. To address the problem, do one or both of the following:

- To help LUIS improve discrimination among intents, add more labels.
- To help LUIS learn faster, add phrase-list features that introduce domain-specific vocabulary.

I have an app in one language and want 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 file 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. Finally, import the app to have a LUIS app in the target language.

How do I download a log of user utterances?

By default, your LUIS app logs utterances from users. To download a log of utterances that users send to your LUIS app, under **My App**, select the download icon in the entry for your app. The log is formatted as a comma-separated value (CSV) file.

How can I disable the logging of utterances?

You can turn off the logging of user utterances by setting `log=false` in the URL when your client application queries LUIS. However, turning off logging disables your LUIS app's ability to suggest utterances or improve performance that's based on user queries. If you set `log=false` because of data-privacy concerns, you won't be able to download a record of user utterances from LUIS or use those utterances to improve your app.

Can I delete data from LUIS?

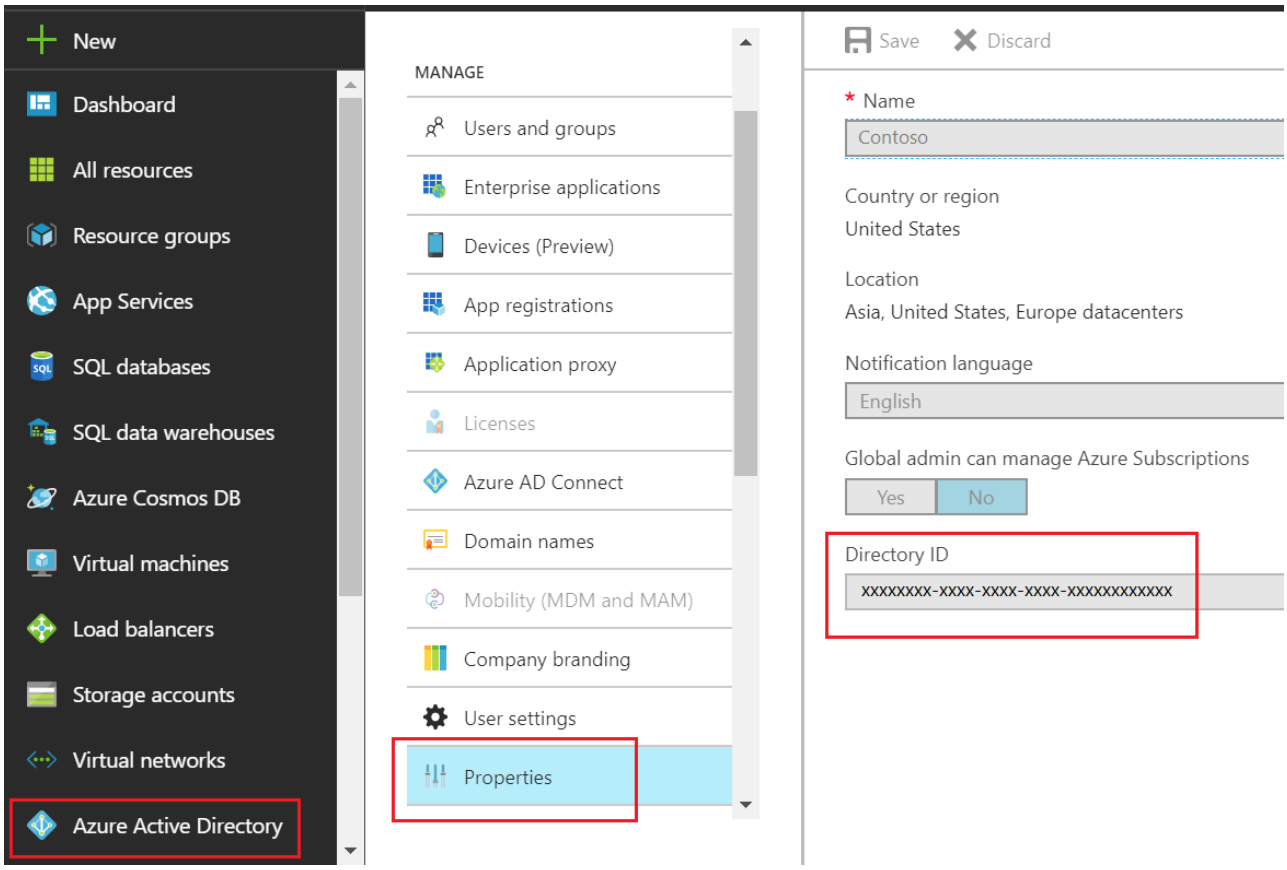
- If you delete an utterance from your LUIS app, it is removed from the LUIS web service and is unavailable for export.
- If you delete an account, all apps and their utterances are deleted. The data is retained on the servers for 60 days before it is deleted permanently.

What are the transaction limits on the Authoring API?

To edit your LUIS app programmatically, you use a programmatic key with the Authoring API. Programmatic authoring allows up to 100,000 calls per month and five transactions per second.

What is the tenant ID in the "Add a key to your app" window?

In Azure, a tenant represents the client or organization that's associated with a service. Find your tenant ID in the Azure portal in the **Directory ID** box by selecting **Azure Active Directory** > **Manage** > **Properties**.



Next steps

To learn more about LUIS, see the following resources:

- Stack Overflow questions tagged with LUIS
- MSDN Language Understanding Intelligent Services (LUIS) Forum