

# Table of Contents

[LUIS Documentation](#)

[Overview](#)

[Learn about Language Understanding Intelligent Service](#)

[Quickstarts](#)

[Create a new app](#)

[Call a LUIS endpoint](#)

[Call a LUIS app using Node.js](#)

[Call a LUIS app using client-side JavaScript](#)

[Call a LUIS app using C#](#)

[Call a LUIS app using Java](#)

[Call a LUIS app using Python](#)

[Tutorials](#)

[Integrate LUIS with a bot using Node.js](#)

[Concepts](#)

[Intents](#)

[Entities](#)

[Utterances](#)

[Features](#)

[Support for multiple languages](#)

[How-to guides](#)

[Plan your app](#)

[Build a LUIS app](#)

[Start a new app](#)

[Add intents](#)

[Add utterances](#)

[Add entities](#)

[Improve performance using features](#)

[Train and test the app](#)

[Use active learning](#)

[Publish your app](#)

[Prebuilt models](#)

[Use prebuilt entities](#)

[Use prebuilt domains](#)

[Use the Cortana prebuilt app](#)

[Manage your LUIS app](#)

[Monitor your app using the dashboard](#)

[Manage keys](#)

[Create subscription keys](#)

[Manage versions](#)

[Collaborate on a LUIS app](#)

[Reference](#)

[Authoring APIs](#)

[Endpoint API](#)

[SDKs](#)

[Android](#)

[Node.js](#)

[Python](#)

[Windows](#)

[Resources](#)

[Azure Roadmap](#)

[LUIS FAQ](#)

4 min to read •

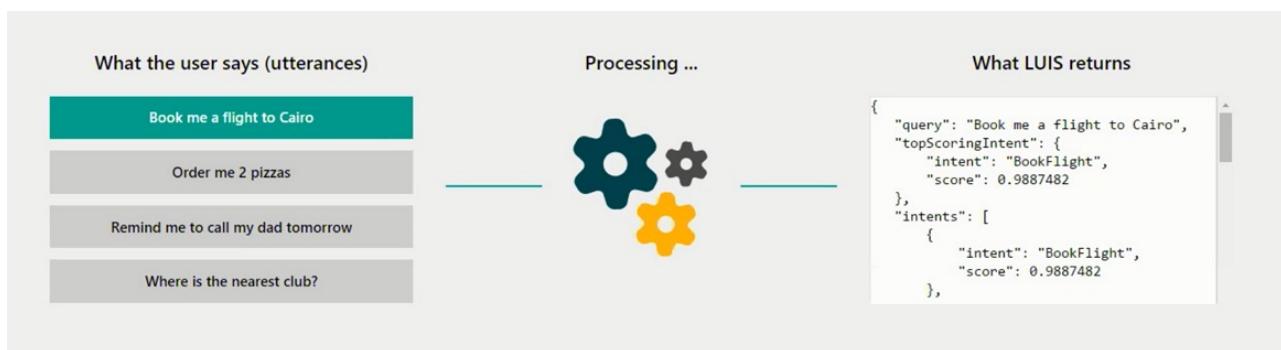
# Learn about Language Understanding Intelligent Service (LUIS)

6/28/2017 • 4 min to read • [Edit Online](#)

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

## What is a LUIS app?

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



## Key concepts

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

## Plan your LUIS app

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

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

## Build and train a LUIS app

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

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

You can also watch a basic [video tutorial](#) on these steps.

## Improve performance using active learning

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

## Configure LUIS programmatically

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

- [LUIS Programmatic API](#).

## Integrate LUIS with a bot

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

### Node.js

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

### C#

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

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

- [LUIS demo bot \(C#\)](#)
- [LUIS demo bot \(Node.js\)](#)

## Integrate LUIS with Speech

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

See [Microsoft Cognitive Services Speech API Overview](#).

# Create your first LUIS app in ten minutes

6/27/2017 • 3 min to read • [Edit Online](#)

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

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

## Before you begin

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

If you don't have an Azure subscription, create a [free account](#) before you begin.

## 1. Create a new app

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

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

The screenshot shows a modal dialog titled "Create a new app". It contains four input fields: "Name (REQUIRED)" with "Application name ..." placeholder, "Culture (REQUIRED)" with "English" selected from a dropdown, "Description (OPTIONAL)" with "Application description ..." placeholder, and "Key to use (OPTIONAL)" with "Choose endpoint key ..." placeholder. A "Create" button is at the bottom left.

Name (REQUIRED)  
Application name ...

Culture (REQUIRED)  
English

Description (OPTIONAL)  
Application description ...

Key to use (OPTIONAL)  
Choose endpoint key ...

Create

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.

The screenshot shows the Microsoft Cognitive Services dashboard for the 'TravelAgent' app. The top navigation bar includes links for Language Understanding, My apps, My keys, Docs, Pricing, Support (which is selected), and About. The user 'Carol Hanna' is signed in. The main area is titled 'Dashboard' and displays facts & statistics about the app's data and endpoint hits. A tip box says: 'You have no intents yet. Intents are the building blocks of your app; they link user requests with the actions that should be taken by your app. Get started by creating your first intent.' It has 'Create an intent' and 'Next tip' buttons. Below this are sections for App status (Last train: Not trained yet, Last published: Not published yet), Intent Count (1 / 80), Entity Count (0 / 10), Prebuilt Entity Count (0 / 5), and Labeled Utterances Count (0). At the bottom are two charts: 'Endpoint Hits Per Period PER DAY (LAST WEEK)' and 'Total Endpoint Hits SINCE APP CREATION'.

## 2. Add intents

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

For more detail on how to add intents, see [Add intents](#).

## 3. Add utterances

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

## 4. Add entities

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

For more information about entities, see [Add entities](#).

## 5. Label entities in utterances

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

## 6. Add prebuilt entities

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

## 7. Train your app

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

## 8. Test your app

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

## 9. Publish your app

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

## 10. Use your app

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

## Next steps

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

# Call a LUIS app using Node.js

10/13/2017 • 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/60340f5f-99c1-4043-8ab9-c810ff16252d?subscription-key=<YOUR_API_KEY>&verbose=false&q=turn%20on%20the%20left%20light
```

. Copy the URL and substitute your subscription key for the value of the `subscription-key` field.

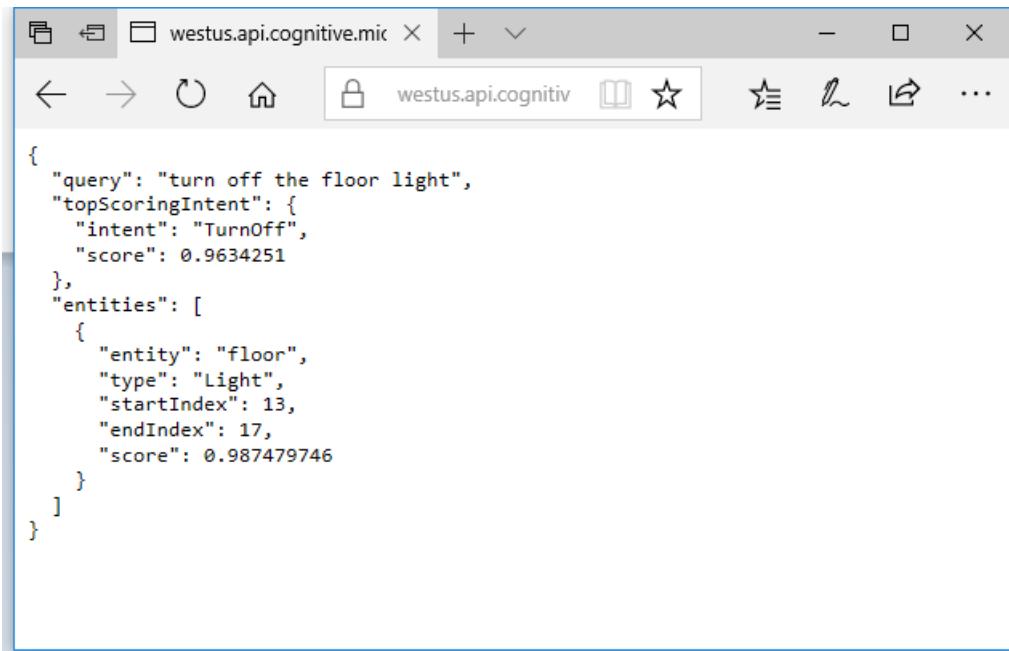
2. Paste the URL into a browser window and press Enter. The browser displays a JSON result that indicates that LUIS detects the `TurnOn` intent and the `Light` entity with the value `left`.



The screenshot shows a Microsoft Edge browser window with the address bar containing "westus.api.cognitive.microsoft.com/luis/v2.0/apps/60340f5f-99c1-4043-8ab9-c810ff16252d?subscription-key=<YOUR\_API\_KEY>&verbose=false&q=turn%20on%20the%20left%20light". The main content area displays the following JSON response:

```
{
  "query": "turn on the left light",
  "topScoringIntent": {
    "intent": "TurnOn",
    "score": 0.9687566
  },
  "entities": [
    {
      "entity": "left",
      "type": "Light",
      "startIndex": 12,
      "endIndex": 15,
      "score": 0.957022846
    }
  ]
}
```

3. Change the value of the `q=` parameter in the URL to `turn off the floor light`, and press enter. The result now indicates that the LUIS detected the `Turnoff` intent and the `Light` entity with value `floor`.



A screenshot of a Microsoft Edge browser window. The address bar shows the URL "westus.api.cognitiv". The main content area displays a JSON object representing the LUIS result for the query "turn off the floor light". The JSON structure includes a top-scoring intent ("TurnOff") and an entity ("floor") of type "Light".

```
{  
  "query": "turn off the floor light",  
  "topScoringIntent": {  
    "intent": "TurnOff",  
    "score": 0.9634251  
  },  
  "entities": [  
    {  
      "entity": "floor",  
      "type": "Light",  
      "startIndex": 13,  
      "endIndex": 17,  
      "score": 0.987479746  
    }  
  ]  
}
```

## 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 60340f5f-99c1-4043-8ab9-c810ff16252d, 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 JavaScript

10/12/2017 • 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/60340f5f-99c1-4043-8ab9-c810ff16252d?subscription-key=<YOUR_API_KEY>&verbose=false&q=turn%20on%20the%20left%20light
```

. Copy the URL and paste in your subscription key for the value of the `subscription-key` field.

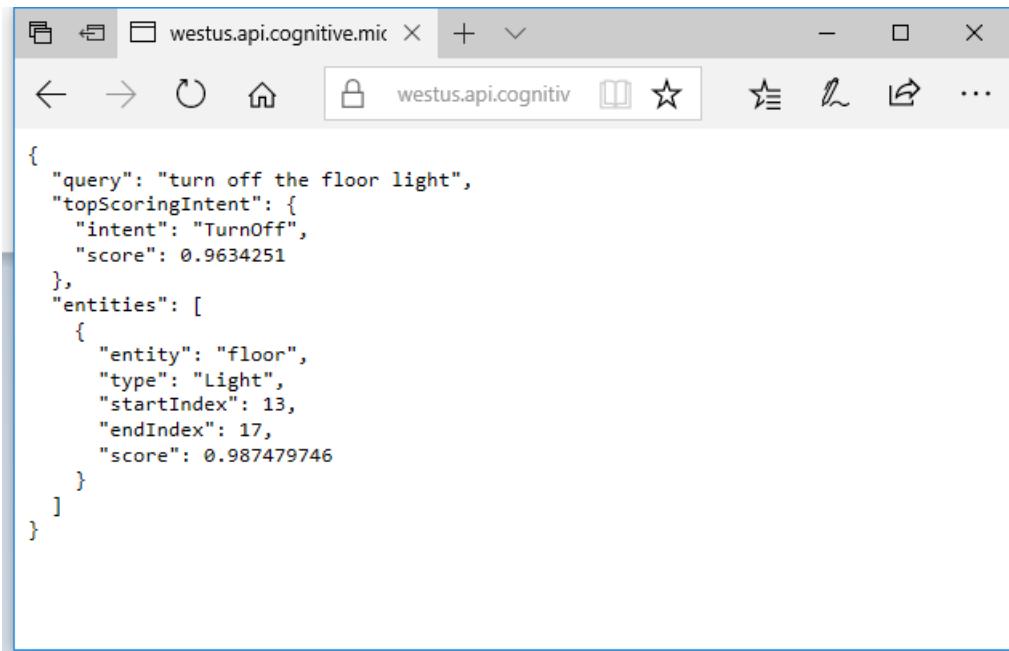
2. Paste the URL into a browser window and press Enter. The browser displays a JSON result that indicates that LUIS detects the `TurnOn` intent and the `Light` entity with the value `left`.



The screenshot shows a Microsoft Edge browser window with the address bar containing "westus.api.cognitive.microsoft.com/luis/v2.0/apps/60340f5f-99c1-4043-8ab9-c810ff16252d?subscription-key=<YOUR\_API\_KEY>&verbose=false&q=turn%20on%20the%20left%20light". The main content area displays the following JSON response:

```
{
  "query": "turn on the left light",
  "topScoringIntent": {
    "intent": "TurnOn",
    "score": 0.9687566
  },
  "entities": [
    {
      "entity": "left",
      "type": "Light",
      "startIndex": 12,
      "endIndex": 15,
      "score": 0.957022846
    }
  ]
}
```

3. Change the value of the `q=` parameter in the URL to `turn off the floor light`, and press enter. The result now indicates that the LUIS detected the `Turnoff` intent and the `Light` entity with value `floor`.



A screenshot of a Microsoft Edge browser window. The address bar shows the URL "westus.api.cognitiv...". The main content area displays a JSON object representing a LUIS prediction result. The JSON structure includes a "query" field ("turn off the floor light"), a "topScoringIntent" object with an "intent" of "TurnOff" and a score of 0.9634251, and an "entities" array containing one entity object. This entity has an "entity" of "floor", a "type" of "Light", a "startIndex" of 13, an "endIndex" of 17, and a "score" of 0.987479746.

```
{  
  "query": "turn off the floor light",  
  "topScoringIntent": {  
    "intent": "TurnOff",  
    "score": 0.9634251  
  },  
  "entities": [  
    {  
      "entity": "floor",  
      "type": "Light",  
      "startIndex": 13,  
      "endIndex": 17,  
      "score": 0.987479746  
    }  
  ]  
}
```

## 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/60340f5f-99c1-4043-8ab9-c810ff16252d?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 C#

10/12/2017 • 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/60340f5f-99c1-4043-8ab9-c810ff16252d?subscription-key=<YOUR_API_KEY>&verbose=false&q=turn%20on%20the%20left%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 `TurnOn` intent and the `Light` entity with the value `left`.



The screenshot shows a Microsoft Edge browser window with the address bar containing "westus.api.cognitive.microsoft.com/luis/v2.0/apps/60340f5f-99c1-4043-8ab9-c810ff16252d?subscription-key=<YOUR\_API\_KEY>&verbose=false&q=turn%20on%20the%20left%20light". The main content area displays the following JSON response:

```
{  
  "query": "turn on the left light",  
  "topScoringIntent": {  
    "intent": "TurnOn",  
    "score": 0.9687566  
  },  
  "entities": [  
    {  
      "entity": "left",  
      "type": "Light",  
      "startIndex": 12,  
      "endIndex": 15,  
      "score": 0.957022846  
    }  
  ]  
}
```

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



A screenshot of a Microsoft Edge browser window. The address bar shows the URL "westus.api.cognitiv...". The main content area displays a JSON object representing a LUIS prediction result. The JSON is as follows:

```
{  
  "query": "turn off the floor light",  
  "topScoringIntent": {  
    "intent": "TurnOff",  
    "score": 0.9634251  
  },  
  "entities": [  
    {  
      "entity": "floor",  
      "type": "Light",  
      "startIndex": 13,  
      "endIndex": 17,  
      "score": 0.987479746  
    }  
  ]  
}
```

## 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 = "60340f5f-99c1-4043-8ab9-c810ff16252d";
            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";

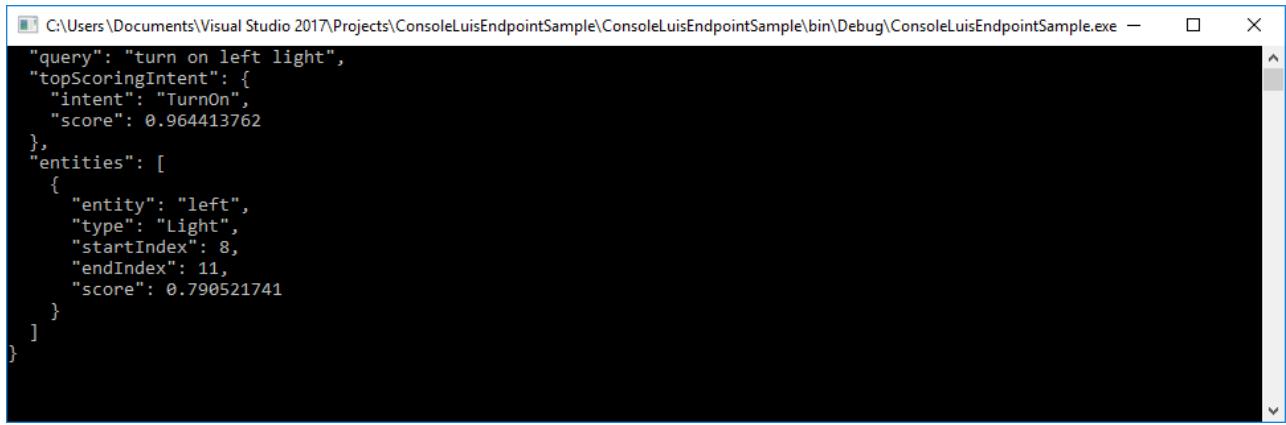
            var uri = "https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/" + luisAppId + "?" +
queryString;
            var response = await client.GetAsync(uri);

            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.



```
C:\Users\Documents\Visual Studio 2017\Projects\ConsoleLuisEndpointSample\ConsoleLuisEndpointSample\bin\Debug\ConsoleLuisEndpointSample.exe

{
  "query": "turn on left light",
  "topScoringIntent": {
    "intent": "TurnOn",
    "score": 0.964413762
  },
  "entities": [
    {
      "entity": "left",
      "type": "Light",
      "startIndex": 8,
      "endIndex": 11,
      "score": 0.790521741
    }
  ]
}
```

## 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

10/12/2017 • 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/60340f5f-99c1-4043-8ab9-c810ff16252d?subscription-key=<YOUR_API_KEY>&verbose=false&q=turn%20on%20the%20left%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 `TurnOn` intent and the `Light` entity with the value `left`.



The screenshot shows a Microsoft Edge browser window with the address bar containing "westus.api.cognitive.microsoft.com/luis/v2.0/apps/60340f5f-99c1-4043-8ab9-c810ff16252d?subscription-key=<YOUR\_API\_KEY>&verbose=false&q=turn%20on%20the%20left%20light". The main content area displays the following JSON response:

```
{  
  "query": "turn on the left light",  
  "topScoringIntent": {  
    "intent": "TurnOn",  
    "score": 0.9687566  
  },  
  "entities": [  
    {  
      "entity": "left",  
      "type": "Light",  
      "startIndex": 12,  
      "endIndex": 15,  
      "score": 0.957022846  
    }  
  ]  
}
```

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



A screenshot of a Microsoft Edge browser window. The address bar shows the URL "westus.api.cognitiv". The main content area displays a JSON object representing the results of a LUIS API call. The JSON structure is as follows:

```
{  
  "query": "turn off the floor light",  
  "topScoringIntent": {  
    "intent": "TurnOff",  
    "score": 0.9634251  
  },  
  "entities": [  
    {  
      "entity": "floor",  
      "type": "Light",  
      "startIndex": 13,  
      "endIndex": 17,  
      "score": 0.987479746  
    }  
  ]  
}
```

## 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 = "60340f5f-99c1-4043-8ab9-c810ff16252d";
            // 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.

```
Problems @ Javadoc Declaration Console 
<terminated> LuisGetRequest [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java
{
  "query": "turn on the left light",
  "topScoringIntent": {
    "intent": "TurnOn",
    "score": 0.9687566
  },
  "entities": [
    {
      "entity": "left",
      "type": "Light",
      "startIndex": 12,
      "endIndex": 15,
      "score": 0.957022846
    }
  ]
}
```

## 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

10/12/2017 • 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/60340f5f-99c1-4043-8ab9-c810ff16252d?subscription-key=<YOUR_API_KEY>&verbose=false&q=turn%20on%20the%20left%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 `TurnOn` intent and the `Light` entity with the value `left`.



The screenshot shows a Microsoft Edge browser window with the address bar containing "westus.api.cognitive.microsoft.com/luis/v2.0/apps/60340f5f-99c1-4043-8ab9-c810ff16252d?subscription-key=<YOUR\_API\_KEY>&verbose=false&q=turn%20on%20the%20left%20light". The main content area displays the following JSON response:

```
{
  "query": "turn on the left light",
  "topScoringIntent": {
    "intent": "TurnOn",
    "score": 0.9687566
  },
  "entities": [
    {
      "entity": "left",
      "type": "Light",
      "startIndex": 12,
      "endIndex": 15,
      "score": 0.957022846
    }
  ]
}
```

3. Change the value of the `q=` parameter in the URL to `turn off the floor light`, and press enter. The result now indicates that the LUIS detected the `Turnoff` intent and the `Light` entity with value `floor`.

```
{  
    "query": "turn off the floor light",  
    "topScoringIntent": {  
        "intent": "TurnOff",  
        "score": 0.9634251  
    },  
    "entities": [  
        {  
            "entity": "floor",  
            "type": "Light",  
            "startIndex": 13,  
            "endIndex": 17,  
            "score": 0.987479746  
        }  
    ]  
}
```

## 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/60340f5f-99c1-4043-8ab9-c810ff16252d?%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

appid = '<app id>'

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/' +
        appid,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.

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

6/29/2017 • 3 min to read • [Edit Online](#)

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

## Before you begin

Before you start, make sure you have the accounts and tools you need for working with LUIS and the [Bot Framework](#).

### LUIS prerequisites

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

### Bot Framework prerequisites

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

## 1. Download the sample bot

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

`SearchHotels`, `ShowHotelsReviews`, and `Help`.

## 2. Import the LUIS model to create the LUIS app

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

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

## 3. Train and publish the LUIS app

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

## 4. Copy the LUIS endpoint URL

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

## 5. Paste the LUIS endpoint into the bot code

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

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

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

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

## 6. Register the bot

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

## 7. Start the bot

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

## 8. Start the emulator

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

## 9. Talk to the bot

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

## Next steps

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

# Intents in LUIS

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 <code>Seattle</code> ?
CheckWeather	{ "type": "date_range", "entity": "this weekend" }	Show me the forecast for <code>this weekend</code>

## 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 <a href="#">Prebuilt entities</a> 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: <ul style="list-style-type: none"><li>• \$Location, including \$FromLocation and \$ToLocation as child entities that represent origin and destination locations.</li><li>• \$TravelClass, including \$First, \$Business, and \$Economy as child entities that represent the travel class.</li></ul>

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.

# Localization support in LUIS apps

6/27/2017 • 1 min to read • [Edit Online](#)

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

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

Luis understands utterances in the following languages. Support for prebuilt entities varies. See [Prebuilt entities in LUIS](#) for details.

Locale	Language	Prebuilt Entity Support
en-US	American English	<input checked="" type="checkbox"/> <input type="checkbox"/>
fr-CA	Canadian French	-
fr-FR	French (France)	<input checked="" type="checkbox"/> <input type="checkbox"/>
it-IT	Italian	<input checked="" type="checkbox"/> <input type="checkbox"/>
nl-NL	Dutch	-
de-DE	German	<input checked="" type="checkbox"/> <input type="checkbox"/>
es-ES	Spanish (Spain)	<input checked="" type="checkbox"/> <input type="checkbox"/>
es-MX	Spanish (Mexico)	-
pt-BR	Portuguese (Brazil)	<input checked="" type="checkbox"/> <input type="checkbox"/>
ja-JP	Japanese	<input checked="" type="checkbox"/> <input type="checkbox"/>
ko-KR	Korean	-
zh-CN	Chinese	<input checked="" type="checkbox"/> <input type="checkbox"/>

## 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 exchangeable with each other.

## Tokenization

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

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

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

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

# Plan your LUIS app

6/27/2017 • 2 min to read • [Edit Online](#)

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

## Identify your domain

A LUIS app is usually centered around a domain-specific topic. For example, you may have a 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.

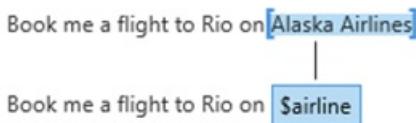
## Identify your entities

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.

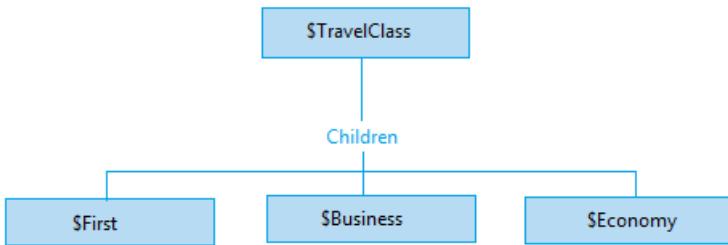
### Simple entity

A simple entity describes a single concept.



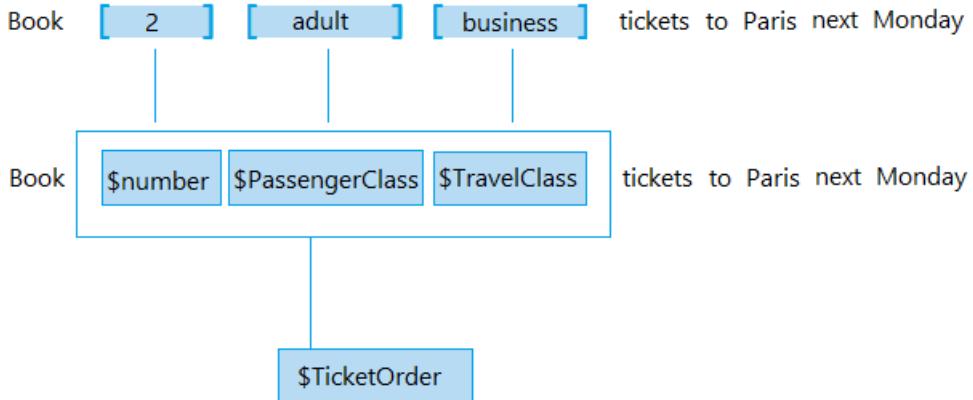
### Hierarchical entity

A hierarchical entity represents a category and its members.



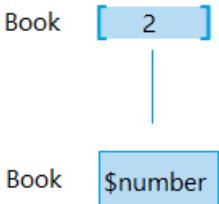
## Composite entity

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



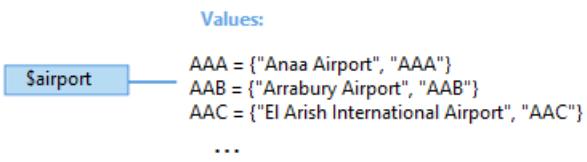
## Prebuilt entity

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".

The screenshot shows a modal dialog titled "Create a new app" with an "X" button in the top right corner. It contains four input fields: "Name (REQUIRED)" with "Application name ..." placeholder, "Culture (REQUIRED)" with "English" selected from a dropdown, "Description (OPTIONAL)" with "Application description ..." placeholder, and "Key to use (OPTIONAL)" with "Choose endpoint key ..." placeholder. A "Create" button is at the bottom left. A note below the dialog states: "\* App culture is the language that your app understands and speaks, not the interface language."

Create a new app

Name (REQUIRED)

Application name ...

Culture (REQUIRED)

English

\* App culture is the language that your app understands and speaks, not the interface language.

Description (OPTIONAL)

Application description ...

Key to use (OPTIONAL)

Choose endpoint key ...

Create

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.



## TravelAgent

## Dashboard

Facts & statistics about the app's data and the received endpoint hits at any period of time ... [Learn more](#)

Dashboard

Intents

Entities

Features

Train &amp; Test

Publish App

[← Back to App list](#)

ⓘ You have no intents yet. Intents are the building blocks of your app; they link user requests with the actions that should be taken by your app. Get started by creating your first intent.

[Create an intent](#) [Next tip](#)

App status      Last train: Not trained yet      Last published: Not published yet

Intent Count	Entity Count	Prebuilt Entity Count	Labeled Utterances Count
1 / 80	0 / 10	0 / 5	0

Endpoint Hits Per Period Total Endpoint Hits SINCE APP CREATION

## 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.

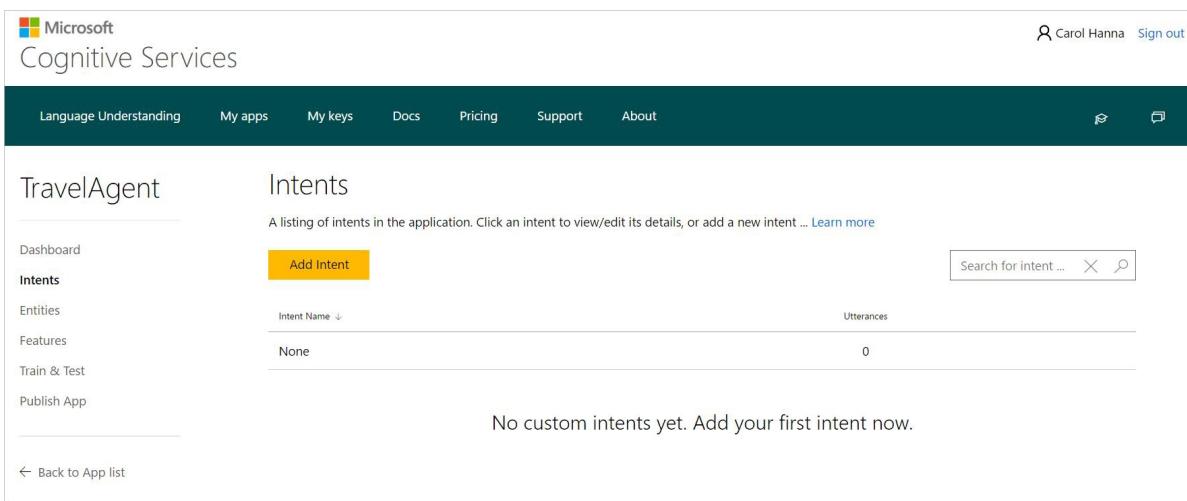
All applications come with the predefined intent, **None**. You should teach it to recognize user statements that are irrelevant to the app. For example, if a user says "Get me a great cookie recipe" in a travel agent app, label that utterance with the **None** intent.

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.

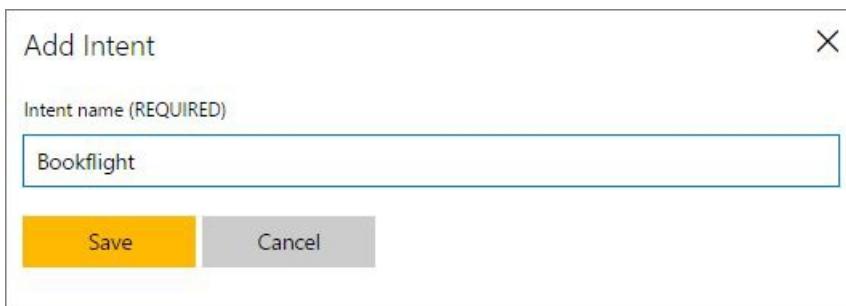
## To add an intent:

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**.



The screenshot shows the Microsoft Cognitive Services portal interface. At the top, there's a navigation bar with links for Language Understanding, My apps, My keys, Docs, Pricing, Support, and About. On the far right, there are user profile and sign-out options. Below the navigation bar, the main header reads "TravelAgent" and "Intents". A sub-header below the main header says "A listing of intents in the application. Click an intent to view/edit its details, or add a new intent ... [Learn more](#)". To the right of this text is a search bar with placeholder text "Search for intent ...". In the center, there's a table-like structure showing one intent entry: "Intent Name" is "None", "Utterances" is "0", and there's a note below stating "No custom intents yet. Add your first intent now.". On the left side, there's a sidebar with links for Dashboard, Intents (which is currently selected and highlighted in blue), Entities, Features, Train & Test, and Publish App. At the bottom left of the main content area, there's a link "← Back to App list".

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](#).

[Language Understanding](#)[My apps](#)[My keys](#)[Docs](#)[Pricing](#)[Support](#)[About](#)

## TravelAgent

### Bookflight

Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... [Learn more](#)

[Dashboard](#)

#### Intents

[Entities](#)[Features](#)[Train & Test](#)[Publish App](#)[← Back to App list](#)[Utterances](#) [Entities in use](#) [Suggested utterances](#)

Type a new utterance &amp; press Enter ...

[Save](#) [Discard](#) [Delete](#) [Reassign Intent](#)

Labels view (Ctrl+E):

Entities

Search in utterances ...

 Utterance text

Predicted Intent

## 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.

[My apps](#) > [TravelAgent](#) > [Intents](#)

### Intents

A listing of intents in the application. Click an intent to view/edit its details, or add a new intent ... [Learn more](#)

[Add Intent](#)

Search for intent ...



Intent Name	Utterances	
Bookflight	0	<a href="#"></a> <a href="#"></a>
GetWeather	0	<a href="#"></a> <a href="#"></a>
None	0	

**To rename an intent:**

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

**Edit Intent**

Intent name (REQUIRED)

[Save](#) [Cancel](#)

**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.

The screenshot shows the Microsoft Cognitive Services Language Understanding interface. At the top, there's a navigation bar with links for Microsoft, Cognitive Services, Language Understanding, My apps, My keys, Docs, Pricing, Support, and About. On the far right, there are search and sign-out options. Below the navigation bar, the main area has a sidebar on the left with links for Dashboard, Intents (which is selected), Entities, Features, Train & Test, and Publish App. The main content area is titled 'TravelAgent' and 'Bookflight'. It says, 'Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... [Learn more](#)'. Below this, there are tabs for Utterances, Entities in use, and Suggested utterances. A text input field says 'Type a new utterance & press Enter ...'. Below it, there are buttons for Save, Discard, Delete, and Reassign Intent. To the right, there's a 'Labels view (Ctrl+E)' dropdown set to 'Entities' and a search bar for 'Search in utterances ...'. At the bottom, there's a section for 'Utterance text' with a checkbox and a 'Predicted Intent' button. At the very bottom of the content area, there's a link to 'Back to App list'.

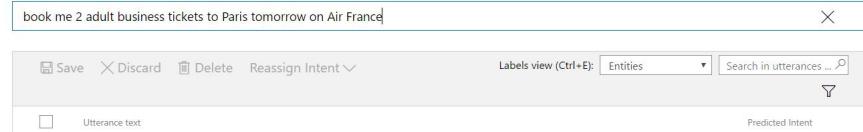
3. Type `book me 2 adult business tickets to Paris tomorrow on Air France` as a new utterance in the text box, and then press Enter. Note that LUIS converts all utterances to lower case.

## TravelAgent

### Bookflight

Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... [Learn more](#)

Utterances Entities in use Suggested utterances



[← Back to App list](#)

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.

## Bookflight

Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... [Learn more](#)

[Utterances](#) [Entities in use](#) [Suggested utterances](#)

X

[Save](#) [Discard](#) [Delete](#) [Reassign Intent](#)

 Labels view (Ctrl+E):   🔍

Utterance text	Predicted Intent	
<input checked="" type="checkbox"/> book me [\$number] adult business tickets to paris [\$datetime] on air france	<b>N.A</b> Bookflight	1

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**.

## Bookflight

Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... [Learn more](#)

[Utterances](#) [Entities in use](#) [Suggested utterances](#)

X

[Save](#) [Discard](#) [Delete](#) [Reassign Intent](#)

 Labels view (Ctrl+E):   🔍

Utterance text	Predicted Intent	
<input checked="" type="checkbox"/> book me [\$number] adult business tickets to paris tomorrow on [air france]	<b>N.A</b> Bookflight	1

🔍

Airline	Airline	+ 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](#).

The screenshot shows the 'BookFlight' intent editor. The utterance is: "book me [number] adult business tickets to [paris] [\$datetime] on [\$Airline]". A dropdown menu is open over the word "[paris]", showing options: "Search/Add entity ...", "Back", "Location::FromLocation", and "Location::ToLocation". The "Location::ToLocation" option is highlighted.

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](#).

The screenshot shows the 'BookFlight' intent editor. The utterance is: "book me [2 adult business] tickets to [Location::ToLocation] [\$datetime] on [\$Airline]". A dropdown menu is open over the phrase "[2 adult business]", showing options: "Search/Add entity ...", "Airline", "TicketsOrder (Composite)", "Location (Hierarchical)", and "TravelClass (Hierarchical)". The "TicketsOrder (Composite)" option is highlighted.

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.

The screenshot shows the Utterances list interface. At the top, there are buttons for Save, Discard, Delete, and Reassign Intent. To the right of these are filters for 'Labels view (Ctrl+E): Entities' and a search bar containing 'reserve'. Below the filters is a small magnifying glass icon. The main list area contains three utterances:

Utterance text	Predicted Intent
reserve a ticket to [\${Location::ToLocation}]	0.99 Bookflight
reserve {2 adult economy} tickets to london	N.A. Bookflight

A green box highlights the first utterance in the list. At the bottom left of the list area, there is a small green box with the number '1'.

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):

- Click the filter button , at the top right corner of the utterances list, to display all filters.
- 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.

## Bookflight

Here you are in full control of this intent: you can manage its utterances, used entities and suggested utterances ... [Learn more](#)

Utterances (6) Entities in use (2) Suggested utterances

The screenshot shows the 'Bookflight' intent management interface. At the top, there's a search bar with placeholder text 'Type a new utterance & press Enter ...'. Below it is a toolbar with 'Save', 'Discard', 'Delete', and 'Reassign Intent' buttons. To the right is a 'Labels view (Ctrl+E)' dropdown set to 'Entities', a search bar 'Search in utterances ...', and a filter icon.

Underneath, there are two tabs: 'Filters' (selected) and 'Changed'. A checkbox 'Utterance text' is checked. The main area lists two utterances:

- book me {[\$number] adult business} tickets to paris [\$datetime] on air france** (Predicted Intent: Airline, Category: BookFlight, Location: Location, Score: 0.9)
- reserve a ticket to [\$Location::ToLocation]** (Predicted Intent: TicketsOrder, TravelClass: TravelClass, datetime: datetime, number: number, Score: 0.9)

A green button labeled '1' is at the bottom center.

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.

The screenshot shows the 'Bookflight' intent management interface with the 'Entities' label view selected in the dropdown. The utterance list shows the same two entries as before, but the entity labels are now displayed in square brackets:

- book me {2 adult business} tickets to [\$Location::ToLocation] [\$datetime] on [\$Airline]** (Predicted Intent: N.A., BookFlight)

A green button labeled '1' is at the bottom center.

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

The screenshot shows the 'Bookflight' intent management interface with the 'Tokens' label view selected in the dropdown. The utterance list shows the same two entries as before, but the entity labels are now displayed in square brackets:

- book me {2 adult business} tickets to [paris] [tomorrow] on [air france]** (Predicted Intent: N.A., BookFlight)

A green button labeled '1' is at the bottom center.

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

Save Discard Delete Reassign Intent

Labels view (Ctrl+E): Composite entities ▾  
Tokens  
Entities  
**Composite entities**

Search in utterances ...

Utterance text

book me { \$TicketsOrder } tickets to paris tomorrow on air france

N.A  
BookFlight

1

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.

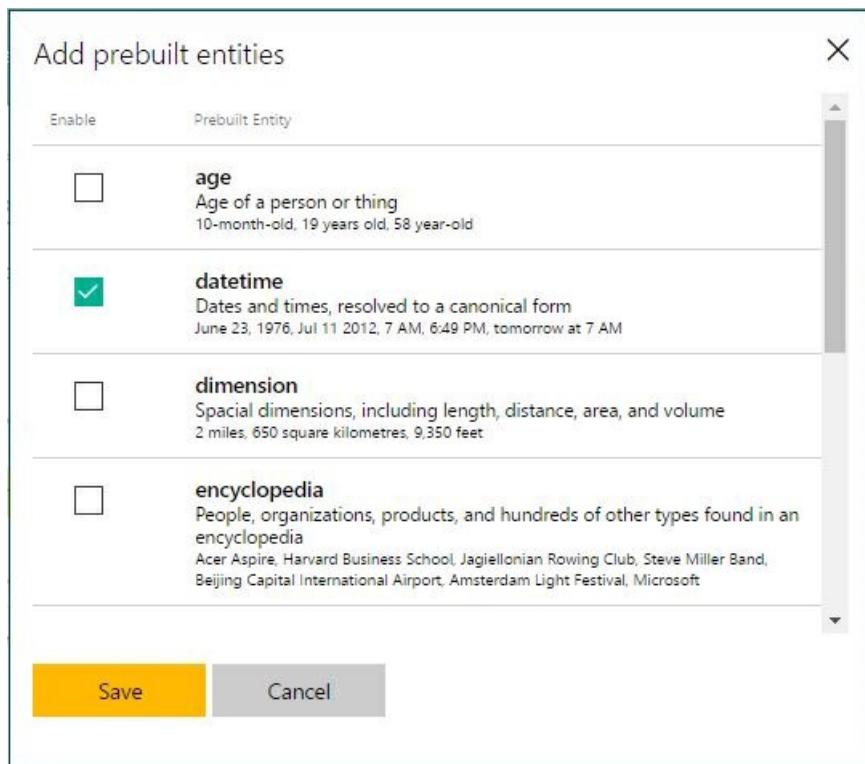
For a full list of prebuilt entities and their use, see [Prebuilt Entities List](#).

### 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**.

The screenshot shows the Microsoft Cognitive Services portal interface. At the top, there's a navigation bar with the Microsoft logo, 'Cognitive Services', and user info ('Thanaa Al Shamy' and 'Sign out'). Below the navigation is a dark green header with links for 'Language Understanding', 'My Apps', 'My Keys', 'Docs', 'Pricing', 'Support', and 'About'. On the far right of the header are two small icons. The main content area has a left sidebar with links for 'Dashboard', 'Intents', 'Entities' (which is currently selected and highlighted in blue), 'Dialogs', 'Features', 'Train & Test', and 'Publish'. The main content area displays the 'TravelAgent' app details. It shows the URL 'My apps > TravelAgent > Entities'. Below this is a section titled 'Entities' with the sub-instruction 'Manage a list of entities in your application and track and control their instances within utterances ... [Learn more](#)'. There are three buttons: 'Add custom entity' (yellow), 'Add prebuilt entity' (yellow), and 'Entities list' (blue). Further down, it says 'Here's a list of all entities in your app. Add and refine entities for a more precise capture of key data ... [Learn more](#)'. At the bottom of the content area, there are input fields for 'Entity Name' and 'Entity Type', and a message 'No entities yet. Add your first new entity now.'

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 <i>simple entity</i> is a generic entity that describes a single concept.
Hierarchical	A <i>hierarchical entity</i> 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 <i>composite entity</i> 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 <i>list entity</i> 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 Entity

Entity name (REQUIRED)

Entity type (REQUIRED)

Save

Cancel

## 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.

Add Entity

Entity name (REQUIRED)

Entity type (REQUIRED)

Child # 1



Child # 2



+ Add child

Save

Cancel

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.

Add Entity

Entity name (REQUIRED)

Entity type (REQUIRED)

Child # 1

Child # 2

Child # 3

+ Add child

**Save**   **Cancel**

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**

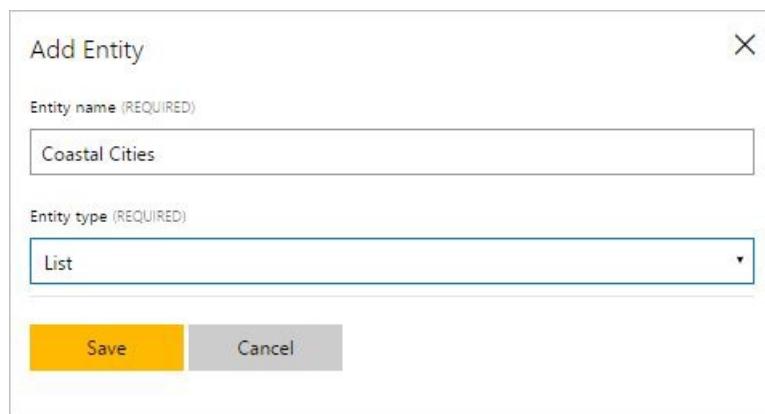
**type.**

Add Entity

Entity name (REQUIRED)  
Coastal Cities

Entity type (REQUIRED)  
List

Save Cancel



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

Coastal Cities

Import Lists Search List Values 0 / 20000

Show Related Values ▾

Canonical Form	Synonyms	+ Add
Enter canonical form	Type & press Enter	+ Add



#### To add list entity values individually:

- 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.
- When you finish typing all synonyms, click **Add**.

Coastal Cities

Import Lists Search List Values 0 / 20000

Show Related Values ▾

Canonical Form	Synonyms	+ Add
Barcelona	Capital City of Catalonia ✕ BCN ✕ Barna ✕ Second Spanish City ✕ Type & press Enter	+ Add



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

#### To import a list of values as a JSON file:

- On the "Coastal Cities" list entity page, click **Import Lists**.
- 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.

Import New Entries

Sub-lists JSON file (REQUIRED)

Choose File Coastal-Cities-List.json

Learn about the supported list syntax ↗

Lists allowed syntax

- The file should be of JSON format.
- The file should contain an array of lists that follow the following JSON format:

```
[  
  {  
    "canonicalForm": "Egypt",  
    "list": [  
      "Cairo",  
      "Alexandria"  
    ]  
  },  
  {  
    "canonicalForm": "USA",  
    "list": [  
      "California",  
      "Texas"  
    ]  
  },  
  {  
    "  
  }  
]
```

Import

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**.

## Coastal Cities

The screenshot shows a user interface for adding a new entity. At the top right is a search bar with a magnifying glass icon and the text 'Search'. To its right is a button labeled 'List Values 1 / 20000'. Below these are two buttons: 'Add all' (gray) and 'Recommend' (yellow). A 'Show Related Values ^' link is located above the main input area. The main input area has tabs for 'Canonical Form' and 'Synonyms'. It contains a text field with placeholder 'Enter canonical form' and a note 'Type & press Enter'. Below the text field is the word 'Barcelona'. To the right of the text field is a small trash bin icon. At the bottom center is a green square button with the number '1'.

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.

The screenshot shows the same user interface after clicking 'Recommend'. The proposed values are now listed in a horizontal row: 'madrid +', 'valencia +', 'malaga +', 'paris +', 'spain +', 'london +', 'sevilla +', 'amsterdam +', 'milan +', and 'berlin +'. The rest of the interface remains the same, including the 'Add all' and 'Recommend' buttons at the top right, the 'Show Related Values ^' link, the 'Canonical Form' and 'Synonyms' tabs, the 'Enter canonical form' text field, and the green '1' button at the bottom.

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:

- On the **Entities** page, click the entity in the **Entities list**.
- 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.

**Edit Entity**

Entity name (REQUIRED)  
Location

Entity type (REQUIRED)  
Hierarchical

Child # 1  
FromLocation

Child # 2  
ToLocation

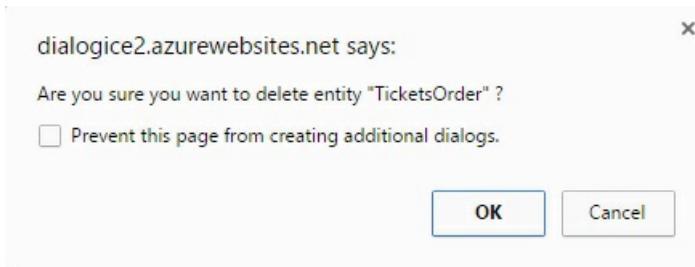
+ Add child

**Save**   **Cancel**

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**.

## Entities

Manage a list of entities in your application and track and control their instances within utterances ... [Learn more](#)

[Entities list](#) [Labeled utterances](#) [Suggested utterances](#)

Labeled utterances for :

Choose Entity ... ▾

Choose Entity ...

<input type="checkbox"/> Save	<input type="checkbox"/> Discard	<input type="checkbox"/> Delete
Intent ▾		
Labels view (Ctrl+E): Entities		
Search in utterances ...		
<input type="button" value="Y"/>		
Predicted Intent		

Utterance text

Airline  
Category  
Location  
TicketsOrder  
TravelClass

## 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 exchangeable 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**.

## TravelAgent

### Features

Use these advanced features to improve performance and avoid mistakes in identifying and interpreting utterances ... [Learn more](#)

Phrase list features (0/10) Pattern features (0/10)

[Add phrase list](#)

Active Phrase list Mode

No phrase lists yet. Add your first phrase list now.

[← Back to App 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.

Add Phrase list

Phrase list name (REQUIRED)  
Cities

Value (REQUIRED)  
Paris, Berlin, Brussels

Phrase list values (REQUIRED)  
London

Related Values  
Add values and click 'Recommend' to get suggestions

Recommend

Is exchangeable?  
 Is active?

Save Cancel

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.

Add Phrase list

Phrase list name (REQUIRED)

Value (REQUIRED)

Phrase list values (REQUIRED)

London Paris Berlin Brussels

Related Values

prague + france + rome + brussels + hilton + montreal +  
vienna + lyon + milan + barcelona +

[Add all](#) [Recommend](#)

Is exchangeable?

Is active?

Save Cancel

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.

## Features

Use these advanced features to improve performance and avoid mistakes in identifying and interpreting utterances ... [Learn more](#)

Phrase list features (1/10) Pattern features (0/10)

Add phrase list

Active	Phrase list	Mode
<input checked="" type="checkbox"/>	Cities: [London,Paris,Berlin,Brussels,montreal,barcelona,lyon,los angeles]	Exchangeable <span>edit</span>

### 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**.

Edit Phrase list

Phrase list name (REQUIRED)

Phrase list values (REQUIRED)

Is exchangeable?

Is active?

**Save** **Cancel**

**To delete a phrase list:**

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

## Pattern features

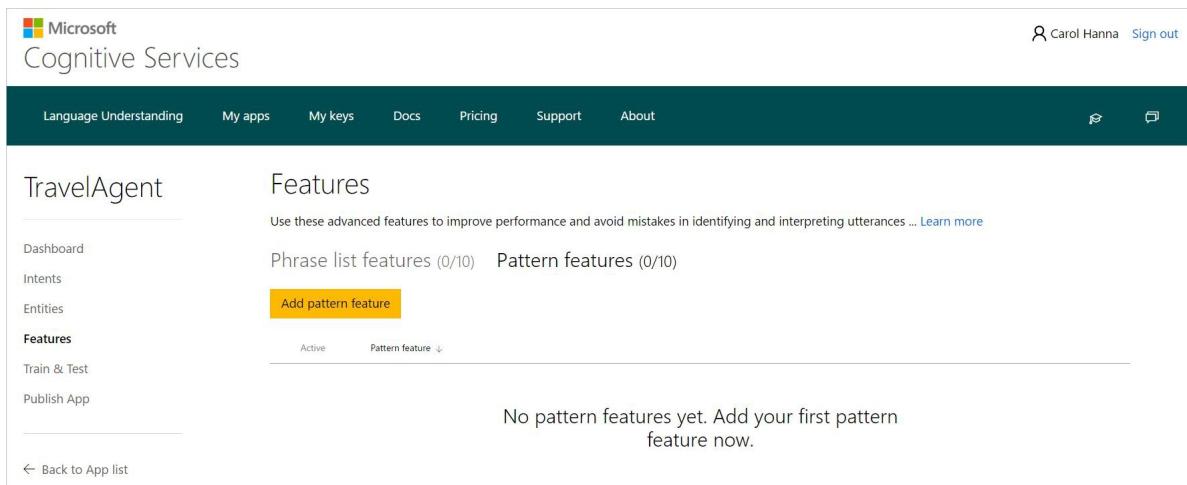
**Pattern** features define a regular expression to help LUIS recognize regular patterns that are frequently used in your application's domain, such as the pattern of flight numbers in a travel app or product codes in a shopping app.

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**

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



The screenshot shows the Microsoft Cognitive Services portal interface. At the top, there's a navigation bar with the Microsoft logo, 'Cognitive Services', and user info ('Carol Hanna Sign out'). Below the navigation bar, the 'TravelAgent' app is selected from a sidebar menu. The main content area is titled 'Features'. It displays a message: 'Use these advanced features to improve performance and avoid mistakes in identifying and interpreting utterances ... [Learn more](#)'. Below this, there are two sections: 'Phrase list features (0/10)' and 'Pattern features (0/10)'. Under 'Pattern features', there's a prominent yellow 'Add pattern feature' button. At the bottom of the 'Features' section, a message says 'No pattern features yet. Add your first pattern feature now.' and a link to 'Back to App list'.

- In the **Add Pattern** dialog box, type "Flight number" in the **Pattern name** text box.
- 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.

Add Pattern X

Pattern name (REQUIRED)

Pattern value (REQUIRED)

Is active?

[Learn about supported regex syntax ^](#)

Regular expression allowed syntax:

- Use '+' to represent 1 or more repetitions, '\*' for 0 or more repetitions and '?' for 0 or 1 repetitions.
- Use '\w' to match a word character, '\d' to match a digit, and '.' to match any character.
- Use '|' to apply a logical OR between characters/groups.
- Use {a,b} to apply a repetition for a character/group.
- Use '^' and '\$' to match beginning and end of utterance.

Save Cancel

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

Add Pattern X

Pattern name (REQUIRED)

Pattern value (REQUIRED)

Is active?

[Learn about supported regex syntax ^](#)

Regular expression allowed syntax:

- Use '+' to represent 1 or more repetitions, '\*' for 0 or more repetitions and '?' for 0 or 1 repetitions.
- Use '\w' to match a word character, '\d' to match a digit, and '.' to match any character.
- Use '|' to apply a logical OR between characters/groups.
- Use {a,b} to apply a repetition for a character/group.
- Use '^' and '\$' to match beginning and end of utterance.

Save Cancel

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**.

The screenshot shows the 'Edit Pattern' dialog box. At the top left is the title 'Edit Pattern' and a close button 'X'. Below the title are two input fields: 'Pattern name (REQUIRED)' with the value 'Flight number' and 'Pattern value (REQUIRED)' with the value '[A-Za-z]{2}[0-9]{3}'. There is a checked checkbox labeled 'Is active?'. Below the checkbox is a link 'Learn about supported regex syntax ▾'. At the bottom are two buttons: a yellow 'Save' button and a grey 'Cancel' button.

#### 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.

### Test your application

Use this tool to test the current and published versions of your application, to check if you are progressing on the right track ... [Learn more](#)

**Train Application** Last train: Aug 25, 2017, 2:29:05 PM | Last publish: Aug 22, 2017, 7:32:16 PM

Interactive Testing   Batch Testing

Enable published model

Type a test utterance & press Enter →

Labels view (Ctrl+E) Entities ▾ | Reset console

#### NOTE

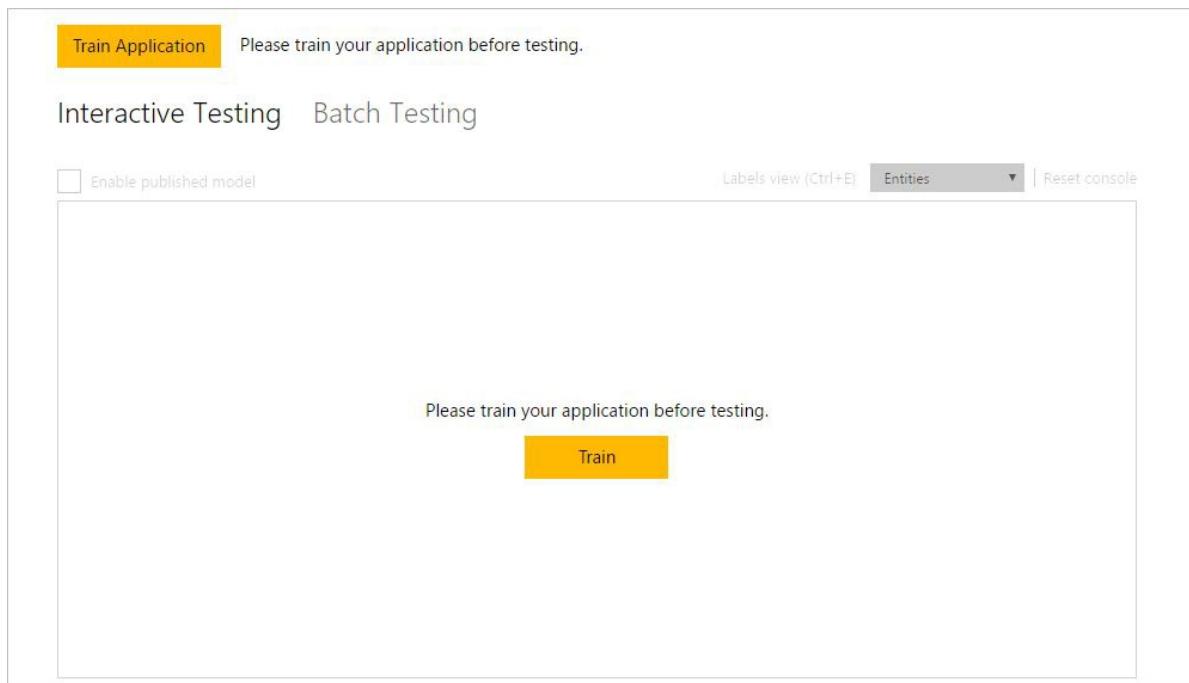
If you have one or more intents in your app that do not contain example utterances, you cannot train your app until you add utterances for all your intents. For more information, see [Add example utterances](#).

## Test your app

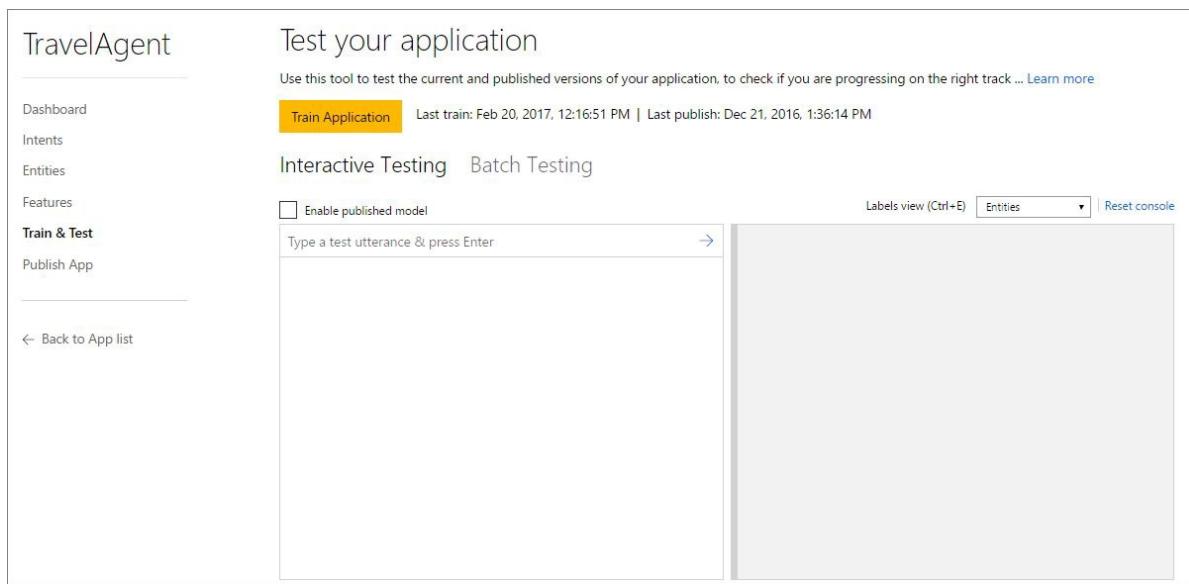
LUIS provides two types of testing: interactive testing and batch testing. You can use either of them by using the corresponding tab on the **Test App** page.

### 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 screenshot shows the LUIS 'Test your application' interface. On the left, a sidebar lists 'Dashboard', 'Intents', 'Entities', 'Features', 'Train & Test' (which is selected), and 'Publish App'. Below the sidebar is a link to 'Back to App list'. The main area has tabs for 'Interactive Testing' and 'Batch Testing'. Under 'Interactive Testing', there is a text input field with placeholder 'Type a test utterance & press Enter' and a large empty result panel. At the top right, there are buttons for 'Labels view (Ctrl+E)', 'Entities' (with a dropdown arrow), and 'Reset console'.

- **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 screenshot shows the 'Interactive Testing' tab with a test utterance 'book me [\$number] flight to [\$Location::ToLocation] [\$datetime]' entered into the text input field. To the right, the 'Current version results' section displays the top scoring intent 'Bookflight (0.99)' and other intents: 'GetWeather (0.06)' and 'None (0.05)'. The interface includes a 'Labels view (Ctrl+E)' button, an 'Entities' dropdown, and a 'Reset console' button at the top right.

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`.

The screenshot shows the LUIS Interactive Testing interface. On the left, there's a text input field with the placeholder "Type a test utterance & press Enter". Below it, the utterance `buy a plane ticket to bangor me` is entered. To the right, there's a "Current version results" section. At the top, it says "Top scoring intent: Weather.GetForecast (0.86)". Below that, under "Other intents", it lists "None (0.09) BookFlight (0.05)".

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.

The screenshot shows the LUIS Interactive Testing interface. On the left, there's a text input field with the placeholder "Type a test utterance & press Enter". Below it, the utterance `book a flight to bangor` is entered. To the right, there's a "Current version results" section. At the top, it says "Top scoring intent: BookFlight (0.71)". Below that, under "Other intents", it lists "None (0.07) Weather.GetForecast (0)".

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.

## BookFlight

Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... [Learn more](#)

Utterances (4) Entities in use (8) Suggested utterances

The screenshot shows the Rasa NLU configuration interface for the 'BookFlight' intent. At the top, there are buttons for Save, Discard, Delete, and Reassign Intent. To the right, there's a 'Labels view (Ctrl+E)' dropdown set to 'Entities' and a search bar labeled 'Search in utterances ...'. Below this, the 'Utterance text' section contains four entries:

- buy a plane ticket to [ bangor me ] (Predicted Intent: 0.83, BookFlight)
- book a flight from [ \$Location::From ] (Predicted Intent: 0.9, BookFlight)
- book me { [ \$number ] [ \$PassengerCategory ] [ \$Location::ToLocation ] [ \$date ] } (Predicted Intent: 0.91, BookFlight)
- book me a flight to [ \$Location::To ] (Predicted Intent: 0.9, BookFlight)

To the right of each utterance, there's a 'Predicted Intent' column with a confidence score and the entity name. Below the utterances, a context menu is open over the third utterance, showing options: 'Search/Add entity ...', 'Wrap in composite entity', 'Airline', 'Location (Hierarchical)', 'PassengerCategory (Hierarchical)', 'TicketsOrder (Composite)', and 'TravelClass (Hierarchical)'. Each option has a right-pointing arrow icon.

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

The screenshot shows the Rasa NLU configuration interface for the 'BookFlight' intent. A context menu is open over the first utterance, 'buy a plane ticket to [ \$Location ]'. The menu includes options: 'Search/Add entity ...', 'Remove label', 'Wrap in composite entity', and two backtracking options ('Back' and 'Location::ToLocation'). The 'Location::ToLocation' option is highlighted with a green background.

10. Click **Save**.

The screenshot shows the Rasa NLU configuration interface for the 'BookFlight' intent. The first utterance, 'buy a plane ticket to [ \$Location::ToLocation ]', is now highlighted with a green background and has a checked checkbox next to it, indicating it is the active utterance.

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

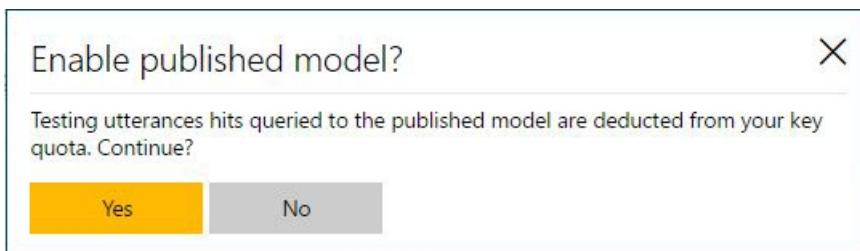
In this step you choose an utterance that's similar to the one you labeled, but not exactly the same. This helps to test your LUIS app's ability to generalize.

## Interactive Testing    Batch Testing

The screenshot shows the LUIS Test App interface. At the top, there are tabs for "Interactive Testing" and "Batch Testing". Below the tabs, there is a checkbox labeled "Enable published model". A text input field says "Type a test utterance & press Enter". To the right of the input field is a blue arrow pointing right. Below the input field is a gray box containing the test utterance: "buy a plane ticket to [\$Location::ToLocation]". To the right of this box, a yellow vertical bar indicates the transition to the results section. The results section is titled "Current version results" and shows the "Top scoring intent" as "BookFlight (0.83)".

#### 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.

The screenshot shows the LUIS Test App interface with the "Enable published model" checkbox checked. The "Interactive Testing" tab is selected. The test utterance "book me a flight to Boston tomorrow" is entered in the input field. The results are split into two sections: "Current version results" and "Published version results". Both sections show the "Top scoring intent" as "BookFlight (0.72)". In the "Published version results" section, there is also a link "Raw JSON view".

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

## Interactive Testing   Batch Testing

The screenshot shows the Interactive Testing interface. On the left, there is a text input field with the placeholder "Type a test utterance & press Enter". Below it, a message box contains the utterance "book me [\$number] flight to [\$Location::ToLocation] [\$datetime]". To the right of the input field is a large orange arrow pointing to the right. On the far right, there is a panel titled "Published version results" which displays a JSON response. The JSON output includes the query, top scoring intent (Bookflight), score (0.999995), and actions. One action is a Bookflight action with parameters for Travel Date (entity: tomorrow, type: builtin.datetime.date). The "Formatted JSON view" link is visible at the top of this panel.

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.

This screenshot shows the Interactive Testing interface with the "Enable published model" checkbox checked. The test utterance "book me [\$number] flight to [\$Location::ToLocation] [\$datetime]" is entered. A tooltip labeled "Entity prediction" appears over the underlined entities. It shows two entries: "Trained: \$number" and "Published: \$builtin.number". To the right, the "Current version results" panel shows the top scoring intent "Bookflight (1)" and other intents "GetWeather (0.12) None (0.05)". The "Published version results" panel on the right shows the same JSON response as the previous screenshot, including the Bookflight action with its parameters.

### 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

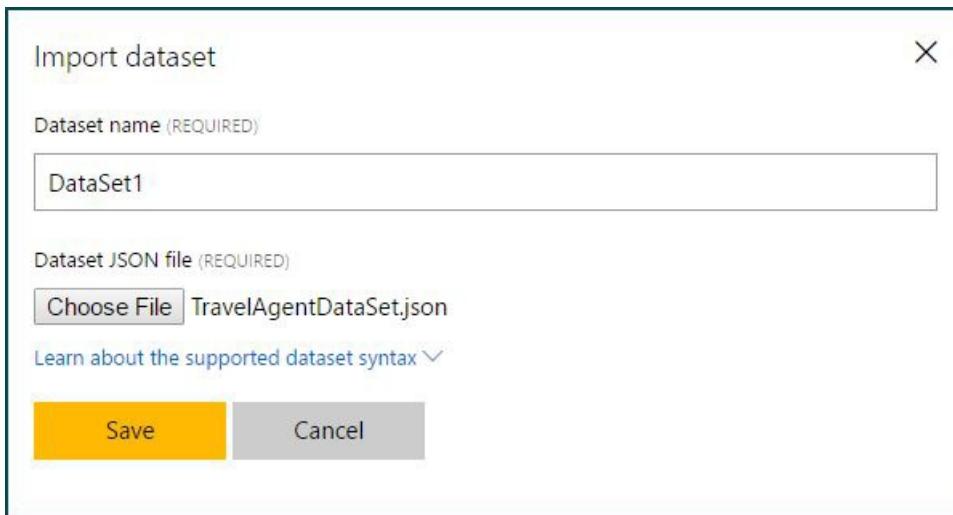
file, known as a *dataset*. The dataset file should be written in JSON format and contains a maximum of 1000 utterances. All you need to do is to import this file to your app and run it to perform the test. Your LUIS app will return the result, enabling you to access detailed analysis of all utterances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

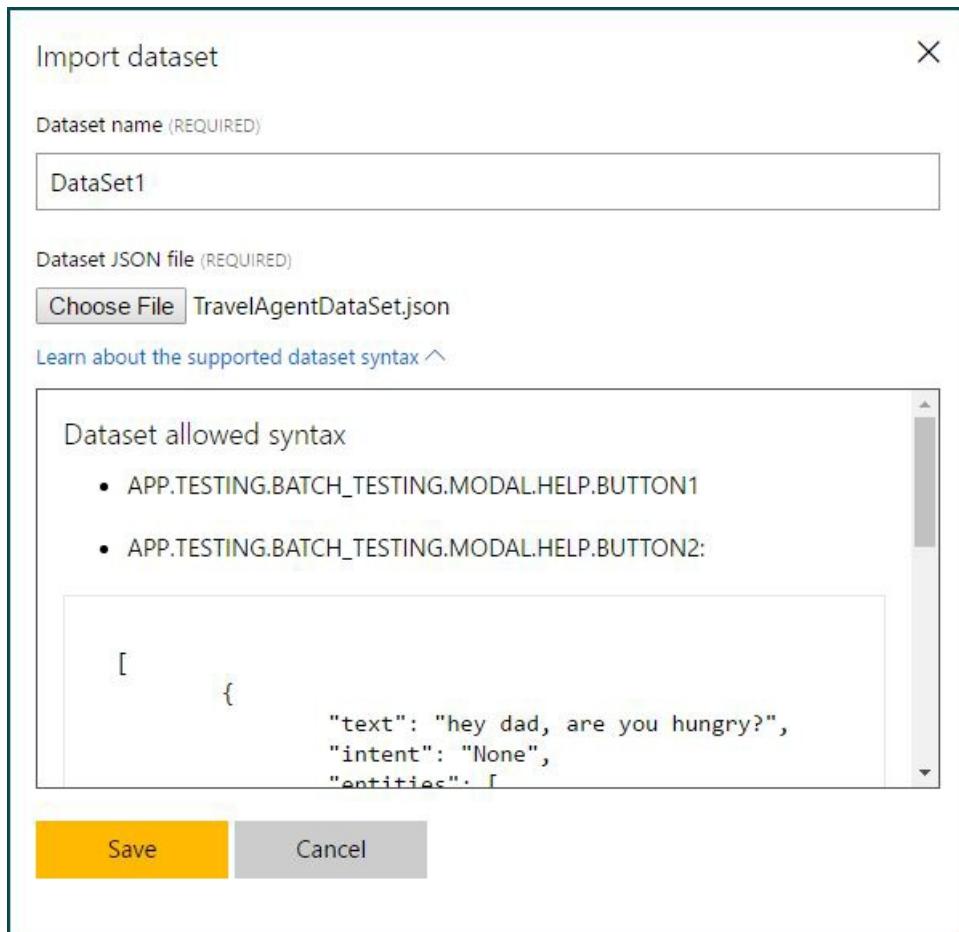
The following procedures will guide you on how to import a dataset file, run a batch test on your current trained app using the imported dataset, and finally to access the test results in a detailed visualized view.

### 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.

Interactive Testing Batch Testing					
<a href="#">Import dataset</a>					
Status	Name	Utterance Count	Last Test Date	Last Test Success	Controls
<a href="#">Test</a>	DataSet1	9	Not tested yet	Not tested yet	<a href="#">Download</a> <a href="#">Edit</a> <a href="#">Delete</a>

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.

Interactive Testing Batch Testing					
<a href="#">Import dataset</a>					
Status	Name	Utterance Count	Last Test Date	Last Test Success	Controls
<a href="#">See results</a>	DataSet1	9	Feb 23, 2017, 3:36:59 PM	33%	<a href="#">Download</a> <a href="#">Edit</a> <a href="#">Delete</a>

In the above screenshot:

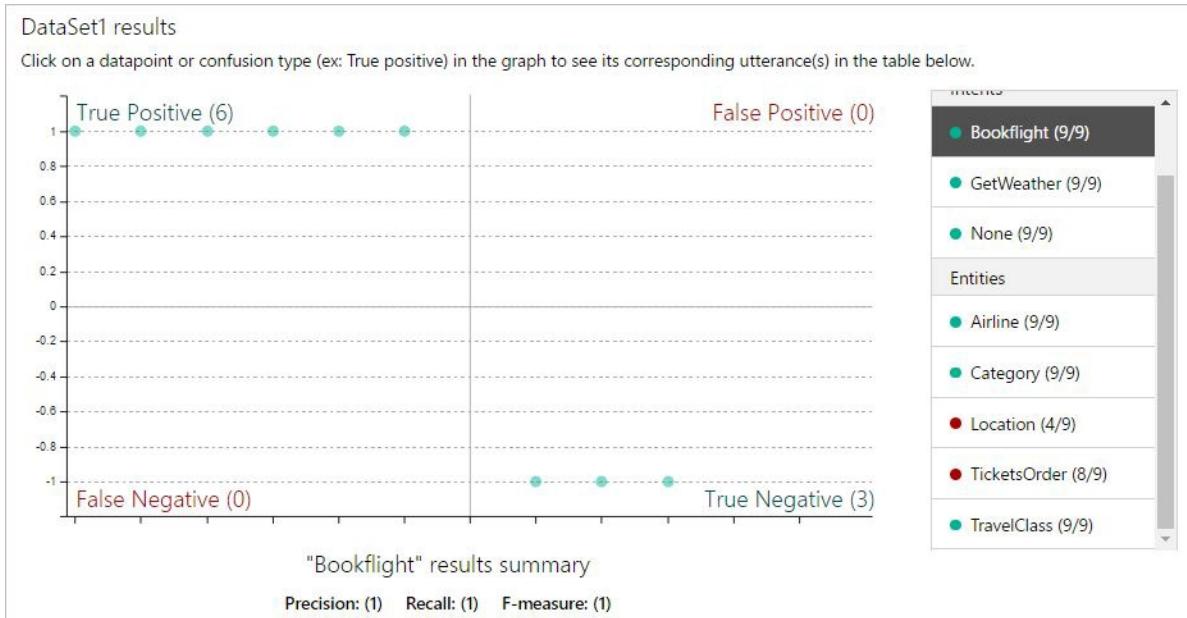
- **Status** of the dataset shows whether or not the dataset result contains errors. In the above example, an

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

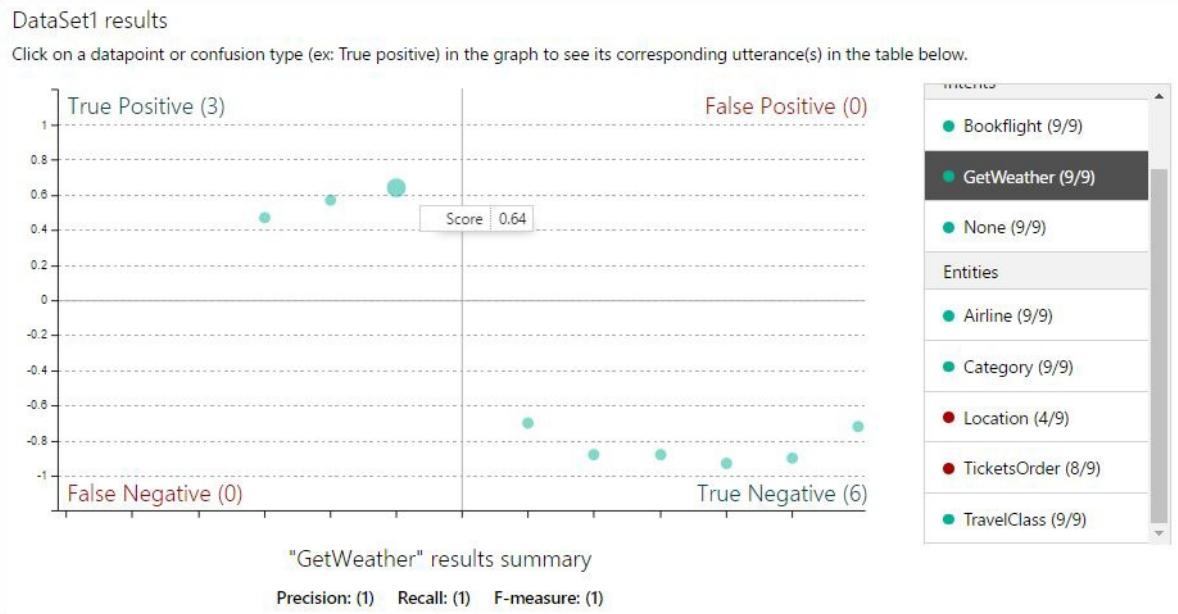
1. Click the **See results** link that appears as a result of running the test (see the above screenshot). A scatter graph (confusion matrix) is displayed, where the data points represent the utterances in the dataset. Green points indicate correct prediction and red ones indicate incorrect prediction.



#### 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.

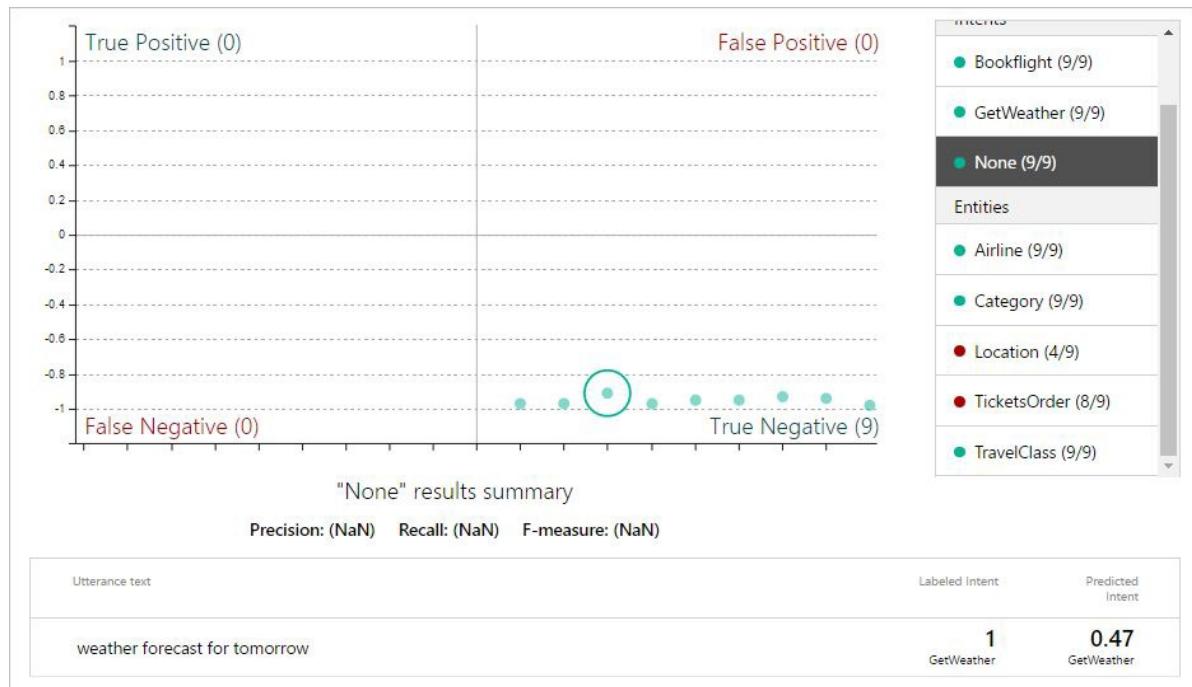
The graph contains 4 sections representing the possible cases of your application's prediction:

- **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.

The screenshot shows the LUIS interface for the "Bookflight" intent. The top navigation bar includes links for Language Understanding, My apps, My keys, Docs, Pricing, Support, and About. Below the navigation is a dark sidebar with links for Dashboard, **Intents** (which is selected), Entities, Features, Train & Test, and Publish App. The main content area is titled "Bookflight" and displays the message "Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... [Learn more](#)". Below this, there are three tabs: Utterances (6), Entities in use (2), and Suggested utterances (which is currently selected). A toolbar at the top of the list includes Save, Discard, Delete, Reassign Intent, Labels view (Ctrl+E), a dropdown for Entities, and a search bar. The list of suggested utterances includes two items:

Utterance text	Suggested Intent
reserve { [ \$ number ] adult economy } tickets to london	1 Bookflight
book me [snumber] flight to [\$Location:ToLocation] [\$datetime]	1 Bookflight

### 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**.

Manage a list of entities in your application and track and control their instances within utterances ... Learn more

Entities list Labeled utterances Suggested utterances

Suggest utterances for : Choose Entity ... ▾

	Utterance text	Suggested Intent
<input type="checkbox"/>	reserve { [ \$ number ] adult economy } tickets to london	1 Bookflight
<input type="checkbox"/>	book me [\$number] flight to [\$Location:ToLocation] [\$datetime]	1 Bookflight
<input type="checkbox"/>	Utterance text	

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.

Manage a list of entities in your application and track and control their instances within utterances ... Learn more

Entities list Labeled utterances Suggested utterances

Suggest utterances for : Location ▾

	Utterance text	Suggested Intent
<input type="checkbox"/>	reserve { [ \$ number ] adult economy } tickets to london	1 Bookflight
<input type="checkbox"/>	book me [\$number] flight to [\$Location:ToLocation] [\$datetime]	1 Bookflight
<input type="checkbox"/>	Utterance text	

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.

## Bookflight

Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... [Learn more](#)

Utterances (4) Entities in use (1) Suggested utterances

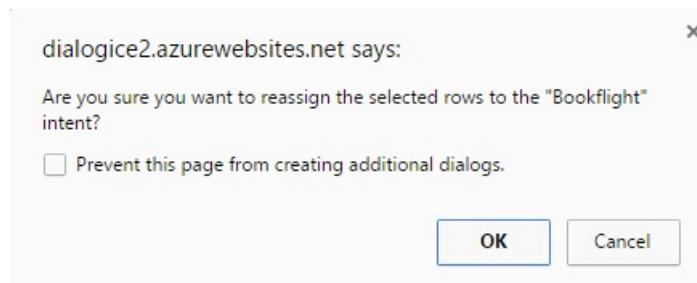
Review the suggested utterances of the current intent and make any required changes to ensure they are correctly labeled ... [Learn more](#)

The screenshot shows the 'Reassign Intent' dialog for the 'Bookflight' intent. The dialog has a search bar at the top. Below it is a table with four rows of utterances:

Utterance text	Suggested Intent	Score	Entity
GetWeather (0.42)	GetWeather	0.42	
book me a flight	Bookflight (0.35)	0.35	[\$number]
i want to travel	None (0.13)	0.13	

A fifth row is highlighted with a green checkmark: "reserve a ticket to [\$Location::ToLocation]". A tooltip "Reassign intent for selected utterance(s)" is shown over this row. At the bottom right of the dialog is a button labeled "1".

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**.

## Entities

Manage a list of entities in your application and track and control their instances within utterances ... [Learn more](#)

Entities list Labeled utterances Suggested utterances

Retrieve suggested utterances that contain a specific entity and make any required changes to improve your model ... [Learn more](#)

Suggest utterances for :

The screenshot shows the 'Entities' list interface. At the top, there are buttons for 'Save', 'Discard', and 'Reassign Intent'. A dropdown menu is open, showing 'Labels view (Ctrl+E): Entities' and a search bar 'Search in utterances ...'. Below this, there is a table with three rows. The first row has an unchecked checkbox and the text 'Utterance text'. The second row has an unchecked checkbox and the text 'book me a flight to [\$Location::ToLocation] on [\$datetime][\$number]'. The third row has a checked checkbox and the text 'i want to travel to [ksa]'. To the right of these rows are 'Suggested Intent' labels: 'Bookflight' (0.67), 'Bookflight' (0.42), and 'GetWeather' (0.42). A dropdown menu is open over the third row, containing options: 'Search/Add entity ...', '< Back', 'Location::FromLocation', and 'Location::ToLocation'. The 'Location::ToLocation' option is highlighted.

Utterance text	Suggested Intent
<input type="checkbox"/> book me a flight to [\$Location::ToLocation] on [\$datetime][\$number]	0.67 Bookflight
<input type="checkbox"/> i want to travel to [ksa]	0.42 Bookflight
<input checked="" type="checkbox"/> reserve a ticket to [\$L	0.42 GetWeather

- 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**.

The screenshot shows the 'Bookflight' intent management interface. At the top, there are buttons for 'Save', 'Discard', and 'Reassign Intent'. A dropdown menu is open, showing 'Labels view (Ctrl+E): Entities' and a search bar 'Search in utterances ...'. Below this, there is a table with four rows. The first row has an unchecked checkbox and the text 'Utterance text'. The second row has an unchecked checkbox and the text 'book me a flight to [\$Location::ToLocation] on may [\$number]'. The third row has an unchecked checkbox and the text 'i want to travel to ksa'. The fourth row has an unchecked checkbox and the text 'reserve a ticket to [\$Location::ToLocation]'. To the right of these rows are 'Suggested Intent' labels: 'Bookflight' (0.99), 'Bookflight' (0.89), and 'Bookflight' (0.97). A dropdown menu is open over the fourth row, containing options: 'Search/Add entity ...', 'Remove label', 'Airline', 'TicketsOrder (Composite)', 'Location (Hierarchical)', 'TravelClass (Hierarchical)', and 'Category (Hierarchical)'. The 'Location (Hierarchical)' option is highlighted.

Utterance text	Suggested Intent
<input type="checkbox"/> book me a flight to [\$Location::ToLocation] on may [\$number]	0.99 Bookflight
<input type="checkbox"/> i want to travel to ksa	0.89 Bookflight
<input type="checkbox"/> reserve a ticket to [\$Location::ToLocation]	0.97 Bookflight

## 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

9/28/2017 • 2 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.

The screenshot shows the 'Publish App' page for the 'TravelAgent' app. On the left, there's a sidebar with navigation links: 'Dashboard', 'Intents', 'Entities', 'Prebuilt domains' (marked as PREVIEW), 'Features', 'Train & Test', and 'Publish App'. The 'Publish App' link is currently selected. The main content area has a title 'Publish App' and a sub-instruction 'Publish your app as a web service or as a chat bot. You can publish a new app or an updated version of a published app ... [Learn more](#)'. Below this is a section titled 'Essentials' with 'Latest publish' and a message 'You haven't published your application yet'. To the right, there's an 'Assigned endpoint key' field with a '(Change)' link and a note about 'StarterKey' (Note: A starter key gives you 1000 endpoint hits per month.). Further down are sections for 'Publish settings' (with a dropdown menu set to 'Production') and 'Endpoint slot' (also set to 'Production'). A note below states 'This slot has no published application.' At the bottom are 'Train' and 'Publish' buttons, with a red message 'Please train your app to publish.'

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

The screenshot shows the LUIS Publish App interface. On the left, there's a sidebar with navigation links: Settings, Dashboard, Intents, Entities, Prebuilt domains (PREVIEW), Features, Train & Test, and Publish App. Under Publish App, there's a link to 'Back to App list'. The main area is titled 'Publish App' and contains the following sections:

- Essentials**: Shows 'Latest publish' as 'Just now' and 'Assigned endpoint key' as 'StarterKey' (Note: A starter key gives you 1000 endpoint hits per month).
- Publish settings**: Includes an 'Endpoint slot' dropdown set to 'Production'.
- Slot info**: Shows 'Published version Id: 0.1' and 'Published date: Sep 26, 2017, 4:01:53 PM'.
- Endpoint url**: Displays the URL <https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/2c7aab1d-ec83-483f-8452-09407b4ff6ae?subscription-key=null&verbose=true&timezoneOffset=0&q=>.
- Checkboxes**: 'Add verbose flag' (checked) and 'Enable bing spell checker'.
- Timezone**: Set to '(GMT) Western Europe Time, London, Lisbo'.
- Buttons**: 'Train' (grayed out) and 'Publish' (yellow).

2. If you want to use a key other than the Starter key, click **Change** next to the **Assigned endpoint key** label. 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 **Endpoint Slot** list.
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. Otherwise, it includes only the top scoring intent.
6. If the app has not been trained, click **Train**.
7. Click **Publish**. The endpoint URL of your published app is displayed.

#### NOTE

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

The screenshot shows the 'Publish settings' section of the LUIS interface. It includes:

- Endpoint slot**: Set to 'Production'.
- Slot info**: Shows 'Published version Id: 0.1' and 'Published date: Sep 26, 2017, 4:01:53 PM'.
- Endpoint url**: Displays the URL <https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/2c7aab1d-ec83-483f-8452-09407b4ff6ae?subscription-key=null&verbose=true&timezoneOffset=0&q=>, which is highlighted with a red border.

**TIP**

The endpoint URL corresponds to the Azure region associated with the endpoint key. For more information, see [Regions and keys](#).

## Test your published endpoint in a browser

If you want to test your published endpoint in a browser using the generated URL, click the URL to open it in your browser, then 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

9/27/2017 • 18 min to read • [Edit Online](#)

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

## **NOTE**

**builtins.datetime** is deprecated. It is replaced by **builtins.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
Phone number	<input checked="" type="checkbox"/>	-	-	-	-	-	-	-	-

## Examples of prebuilt entities

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

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

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

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

## builtin.number resolution

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

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

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

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

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

## Ordinal, percentage and currency resolution

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

### Percentage resolution

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

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

## Ordinal resolution

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

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

## Currency resolution

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

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

## builtin.datetimeV2

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

### NOTE

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

EXAMPLE	PROPERTY DESCRIPTIONS	
<p>The following is an example of a JSON response containing a <code>builtin.datetimeV2</code> entity, of type <code>datetime</code>. For examples of other types of <code>datetimeV2</code> entities, see &lt;a href="#subtypes-of-datetimev2"&gt;Subtypes of <code>datetimeV2</code>.</p>	<p>entity</p>	<p><b>string.</b> Text extracted from the utterance, that represents a date, time, date range or time range.</p>

```

"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.datetimeRange
- builtin.datetimeV2.duration
- builtin.datetimeV2.set

startIndex

**int.** The index in the utterance at which the entity begins.

endIndex

**int.** The index in the utterance at which the entity ends.

resolution

Contains a `values` array that has one, two, or four values.

- The array has one element if the date or time in the utterance is fully specified and unambiguous.
- The array has two elements if the date or date range is ambiguous as to year, or a time or time range is ambiguous as to AM or PM. In the case of an ambiguous date, `values` contains the most recent past and most immediate future instances of the date. See [Ambiguous dates](#) for more examples. In the case of an ambiguous time, `values` contains both the AM and PM times.
- The array has four elements if the utterance contains both a date or date range that is ambiguous as to year, and a time or time range that is ambiguous as to AM or PM. For example, 3:00 April 3rd.

Each element of `values` may contain the following fields:

timex	time, date, or date range expressed in TIMEX format that follows the <a href="#">ISO 8601 standard</a> as well as using the TIMEX3 attributes for annotation using the TimeML language. This annotation is described in the <a href="#">TIMEX guidelines</a> .
type	The subtype, which can be one of the following: datetime, date, time, daterange, datetimera nge, duration, set.
value	<b>Optional.</b> A datetime object in the Format yyyy:MM:d d (date), HH:mm:ss (time) yyyy:MM:d d HH:mm:ss (datetime). If <code>type</code> is <code>duration</code> , the value is the number of seconds (duration) Only used if <code>type</code> is <code>datetime</code> Or <code>date</code> , <code>time</code> , or <code>duration</code> .

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

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 is an utterance that includes the month and date, but not the year. If today's date precedes the date in the utterance in the current year, the most immediate past instance of that date is in the previous year. Otherwise the most immediate past date is in the current year.

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

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

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

```

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

```

### Date range resolution examples

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

```

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

```

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

```

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

```

## Subtypes of datetimeV2

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

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

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
builtin.datetimeV2.date	tomorrow	{ "type": "builtin.datetimeV2.date", "entity": "tomorrow", "resolution": { "values": [ { "timex": "2017-06-21", "type": "date", "value": "2017-06-21" } ] } }
builtin.datetimeV2.date	january 10 2009	{ "type": "builtin.datetimeV2.date", "entity": "january 10 2009", "resolution": { "values": [ { "timex": "2009-01-10", "type": "date", "value": "2009-01-10" } ] } }
builtin.datetimeV2.date	monday	{ "entity": "monday", "type": "builtin.datetimeV2.date", "resolution": { "values": [ { "timex": "XXXX-WXX-1", "type": "date", "value": "2017-06-19" }, { "timex": "XXXX-WXX-1", "type": "date", "value": "2017-06-26" } ] } }

PRE-BUILT ENTITY	EXAMPLE UTTERANCE	JSON
builtin.datetimeV2.daterange	next week	{ "entity": "next week", "type": "builtin.datetime.dateV2.daterange", "resolution": { "values": [ { "timex": "2017-W27", "type": "daterange", "start": "2017-06-26", "end": "2017-07-03"}] } }
builtin.datetimeV2.date	next monday	{ "entity": "next monday", "type": "builtin.datetimeV2.date", "resolution": { "values": [ { "timex": "2017-06-26", "type": "date", "value": "2017-06-26" } ] } }
builtin.datetimeV2.time	3 : 00	{ "type": "builtin.datetimeV2.time", "entity": "3 : 00", "resolution": { "values": [ { "timex": "T03:00", "type": "time", "value": "03:00:00" }, { "timex": "T15:00", "type": "time", "value": "15:00:00" } ] } }
builtin.datetimeV2.time	4 pm	{ "type": "builtin.datetimeV2.time", "entity": "4 pm", "resolution": { "values": [ { "timex": "T16", "type": "time", "value": "16:00:00" } ] } }
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, listed below. In addition, encyclopedia entities often map to multiple types. For example, the query `Ronald Reagan` yields:

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

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.people.person	builtin.encyclopedia.people.person	bryan adams
builtin.encyclopedia.people.person	builtin.encyclopedia.film.producer	walt disney
builtin.encyclopedia.people.person	builtin.encyclopedia.film.cinematographer	adam greenberg
builtin.encyclopedia.people.person	builtin.encyclopedia.royalty.monarch	elizabeth ii
builtin.encyclopedia.people.person	builtin.encyclopedia.film.director	steven spielberg
builtin.encyclopedia.people.person	builtin.encyclopedia.film.writer	alfred hitchcock
builtin.encyclopedia.people.person	builtin.encyclopedia.film.actor	robert de niro
builtin.encyclopedia.people.person	builtin.encyclopedia.martial_arts.martial_artist	bruce lee

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.people.person	builtin.encyclopedia.architecture.architect	james gallier
builtin.encyclopedia.people.person	builtin.encyclopedia.geography.mountainer	jean couzy
builtin.encyclopedia.people.person	builtin.encyclopedia.celebrities.celebrity	angelina jolie
builtin.encyclopedia.people.person	builtin.encyclopedia.music.musician	bob dylan
builtin.encyclopedia.people.person	builtin.encyclopedia.soccer.player	diego maradona
builtin.encyclopedia.people.person	builtin.encyclopedia.baseball.player	babe ruth
builtin.encyclopedia.people.person	builtin.encyclopedia.basketball.player	heiko schaffartzik
builtin.encyclopedia.people.person	builtin.encyclopedia.olympics.athlete	andre agassi
builtin.encyclopedia.people.person	builtin.encyclopedia.basketball.coach	bob huggins
builtin.encyclopedia.people.person	builtin.encyclopedia.amERICAN_football.coach	james franklin
builtin.encyclopedia.people.person	builtin.encyclopedia.cricket.coach	andy flower
builtin.encyclopedia.people.person	builtin.encyclopedia.ice_hockey.coach	david quinn
builtin.encyclopedia.people.person	builtin.encyclopedia.ice_hockey.player	vincent lecavalier
builtin.encyclopedia.people.person	builtin.encyclopedia.government.politician	harold nicolson
builtin.encyclopedia.people.person	builtin.encyclopedia.government.us_president	barack obama
builtin.encyclopedia.people.person	builtin.encyclopedia.government.us_vice_president	dick cheney
builtin.encyclopedia.organization.organization	builtin.encyclopedia.organization.organization	united nations
builtin.encyclopedia.organization.organization	builtin.encyclopedia.sports.league	american league
builtin.encyclopedia.organization.organization	builtin.encyclopedia.ice_hockey.conference	western hockey league
builtin.encyclopedia.organization.organization	builtin.encyclopedia.baseball.division	american league east
builtin.encyclopedia.organization.organization	builtin.encyclopedia.baseball.league	major league baseball

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.organization.organization	builtin.encyclopedia.basketball.conference	national basketball league
builtin.encyclopedia.organization.organization	builtin.encyclopedia.basketball.division	pacific division
builtin.encyclopedia.organization.organization	builtin.encyclopedia.soccer.league	premier league
builtin.encyclopedia.organization.organization	builtin.encyclopedia.american_football.division	afc north
builtin.encyclopedia.organization.organization	builtin.encyclopedia.broadcast.broadcast	nebraska educational telecommunications
builtin.encyclopedia.organization.organization	builtin.encyclopedia.broadcast.tv_station	abc
builtin.encyclopedia.organization.organization	builtin.encyclopedia.broadcast.tv_channel	cnbc world
builtin.encyclopedia.organization.organization	builtin.encyclopedia.broadcast.radio_station	bbc radio 1
builtin.encyclopedia.organization.organization	builtin.encyclopedia.business.operation	bank of china
builtin.encyclopedia.organization.organization	builtin.encyclopedia.music.record_label	pixar
builtin.encyclopedia.organization.organization	builtin.encyclopedia.aviation.airline	air france
builtin.encyclopedia.organization.organization	builtin.encyclopedia.automotive.company	general motors
builtin.encyclopedia.organization.organization	builtin.encyclopedia.music.musical_instrument_company	gibson guitar corporation
builtin.encyclopedia.organization.organization	builtin.encyclopedia.tv.network	cartoon network
builtin.encyclopedia.organization.organization	builtin.encyclopedia.education.educational_institution	cornwall hill college
builtin.encyclopedia.organization.organization	builtin.encyclopedia.education.school	boston arts academy
builtin.encyclopedia.organization.organization	builtin.encyclopedia.education.university	johns hopkins university
builtin.encyclopedia.organization.organization	builtin.encyclopedia.sports.team	united states national handball team

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.organization.organization	builtin.encyclopedia.basketball.team	chicago bulls
builtin.encyclopedia.organization.organization	builtin.encyclopedia.sports.professional_sports_team	boston celtics
builtin.encyclopedia.organization.organization	builtin.encyclopedia.cricket.team	mumbai indians
builtin.encyclopedia.organization.organization	builtin.encyclopedia.baseball.team	houston astros
builtin.encyclopedia.organization.organization	builtin.encyclopedia.american_football.team	green bay packers
builtin.encyclopedia.organization.organization	builtin.encyclopedia.ice_hockey.team	hamilton bulldogs
builtin.encyclopedia.organization.organization	builtin.encyclopedia.soccer.team	fc bayern munich
builtin.encyclopedia.organization.organization	builtin.encyclopedia.government.political_party	pertubuhan kebangsaan melayu singapura
builtin.encyclopedia.time.event	builtin.encyclopedia.time.event	1740 batavia massacre
builtin.encyclopedia.time.event	builtin.encyclopedia.sports.championship_event	super bowl xxxix
builtin.encyclopedia.time.event	builtin.encyclopedia.award.competition	eurovision song contest 2003
builtin.encyclopedia.tv.series_episode	builtin.encyclopedia.tv.series_episode	the magnificent seven
builtin.encyclopedia.tv.series_episode	builtin.encyclopedia.tv.multipart_tv_episode	the deadly assassin
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.commerce.consumer_product	nokia lumia 620
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.music.album	dance pool
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.automotive.model	pontiac fiero
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.computer.computer	toshiba satellite
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.computer.web_browser	internet explorer
builtin.encyclopedia.commerce.brand	builtin.encyclopedia.commerce.brand	diet coke

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.commerce.brand	builtin.encyclopedia.automotive.make	chrysler
builtin.encyclopedia.music.artist	builtin.encyclopedia.music.artist	michael jackson
builtin.encyclopedia.music.artist	builtin.encyclopedia.music.group	the yardbirds
builtin.encyclopedia.music.music_video	builtin.encyclopedia.music.music_video	the beatles anthology
builtin.encyclopedia.theater.play	builtin.encyclopedia.theater.play	camelot
builtin.encyclopedia.sports.fight_song	builtin.encyclopedia.sports.fight_song	the cougar song
builtin.encyclopedia.film.series	builtin.encyclopedia.film.series	the twilight saga
builtin.encyclopedia.tv.program	builtin.encyclopedia.tv.program	late night with david letterman
builtin.encyclopedia.radio.radio_program	builtin.encyclopedia.radio.radio_program	grand ole opry
builtin.encyclopedia.film.film	builtin.encyclopedia.film.film	alice in wonderland
builtin.encyclopedia.cricket.tournament	builtin.encyclopedia.cricket.tournament	cricket world cup
builtin.encyclopedia.government.government	builtin.encyclopedia.government.government	european commission
builtin.encyclopedia.sports.team_owner	builtin.encyclopedia.sports.team_owner	bob castellini
builtin.encyclopedia.music.genre	builtin.encyclopedia.music.genre	eastern europe
builtin.encyclopedia.ice_hockey.division	builtin.encyclopedia.ice_hockey.division	hockeyallsvenskan
builtin.encyclopedia.architecture.style	builtin.encyclopedia.architecture.style	spanish colonial revival architecture
builtin.encyclopedia.broadcast.producer	builtin.encyclopedia.broadcast.producer	columbia tristar television
builtin.encyclopedia.book.author	builtin.encyclopedia.book.author	adam maxwell
builtin.encyclopedia.religion.founding_figure	builtin.encyclopedia.religion.founding_figure	gautama buddha
builtin.encyclopedia.martial_arts.martial_art	builtin.encyclopedia.martial_arts.martial_art	american kenpo
builtin.encyclopedia.sports.school	builtin.encyclopedia.sports.school	yale university
builtin.encyclopedia.business.product_line	builtin.encyclopedia.business.product_line	canon powershot
builtin.encyclopedia.internet.website	builtin.encyclopedia.internet.website	bing

PRE-BUILT ENTITY	PRE-BUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.time.holiday	builtin.encyclopedia.time.holiday	easter
builtin.encyclopedia.food.candy_bar	builtin.encyclopedia.food.candy_bar	cadbury dairy milk
builtin.encyclopedia.finance.stock_exchange	builtin.encyclopedia.finance.stock_exchange	tokyo stock exchange
builtin.encyclopedia.film.festival	builtin.encyclopedia.film.festival	berlin international film festival

# Use prebuilt domains in LUIS apps

10/9/2017 • 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.

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

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.

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

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.

## Prebuilt domains

Prebuilt domains are off-the-shelf collections of intents and entities that you can directly add and use in your application ... [Learn more](#)



Calendar

The Calendar domain provides intents and entities related to calendar ... [Learn more](#)

Added



Camera

The Camera domain provides intents and entities related to using a ... [Learn more](#)

Not added



Communication

The Communication domain provides intents and entities related to ... [Learn more](#)

Not added



Entertainment

The Entertainment domain provides intents and entities related to ... [Learn more](#)

Not added



Events

The Events domain provides intents and entities related to booking ... [Learn more](#)

Not added

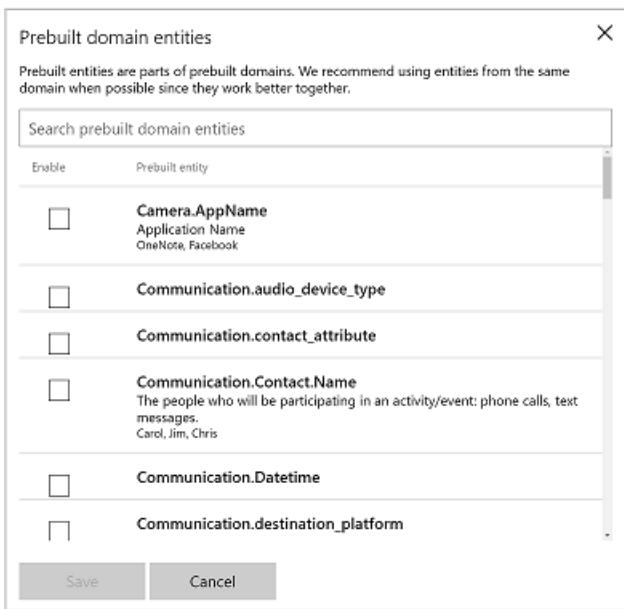


Fitness

The Fitness domain provides intents and entities related to tracking ... [Learn more](#)

Not added

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



## Designing LUIS apps from prebuilt domains

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

### Customizing an entire prebuilt domain

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

#### TIP

The intents and entities in a prebuilt domain work best together. It's better to combine intents and entities from the same domain when possible.

- A best practice is to use related intents from the same domain. For example, if you are customizing the `MakeReservation` intent in the **Places** domain, then select the `Cancel` intent from the **Places** domain instead of the `Cancel` intent in the **Events** or **Utilities** domains.

### Changing the behavior of a prebuilt domain intent

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

#### TIP

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

# Cortana Prebuilt App

9/6/2017 • 13 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:

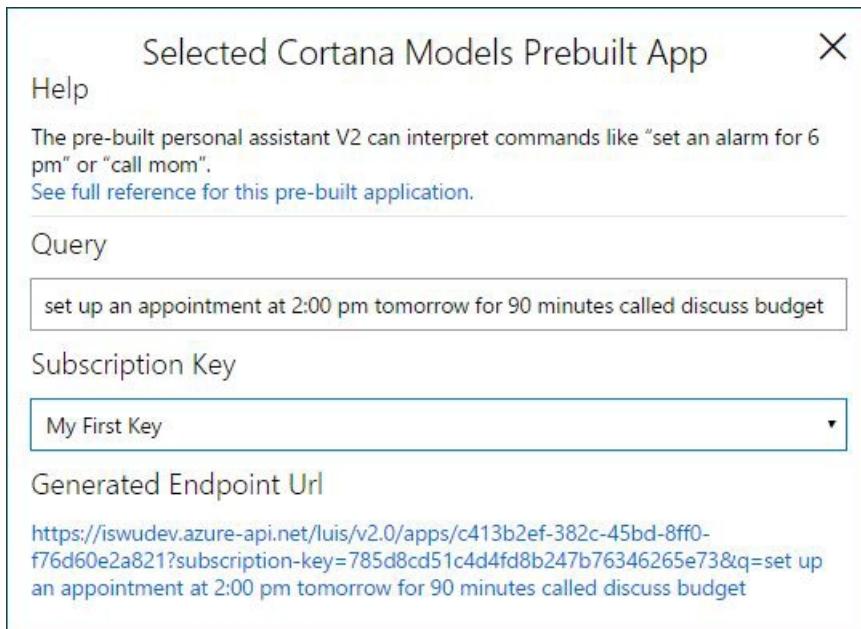
- They provide packages of prebuilt and pretrained intents and entities that are designed to work well with each other. You can integrate a prebuilt domain directly into your app. For example, if you're building a fitness tracker, you can add the **Fitness** domain and have an entire set of intents and 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.

## Use the Cortana Prebuilt App

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



2. In **Query**, type the utterance you want to interpret. For example, type "set up an appointment at 2:00 pm

tomorrow for 90 minutes called discuss budget"

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

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

## Cortana prebuilt intents

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

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

INTENT	EXAMPLE UTTERANCES
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?
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.assignNickname	edit nickname for nickolas add a nickname for john doe
builtin.intent.communication.call_voice_mail	voice mail call voicemail

INTENT	EXAMPLE UTTERANCES
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
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

INTENT	EXAMPLE UTTERANCES
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
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

INTENT	EXAMPLE UTTERANCES
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?
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.changeReminder	change a reminder move up my picture day reminder 30 minutes
builtin.intent.reminder.createSingleReminder	remind me to wake up at 7 am remind me to go trick or treating with luca at 4:40pm
builtin.intent.reminder.deleteReminder	delete this reminder delete my picture reminder
builtin.intent.reminder.findReminder	do I have any reminders set up for today do I have a reminder set for luca's party

INTENT	EXAMPLE UTTERANCES
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
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 <span style="border: 1px solid black; padding: 2px;">off</span> my wake-up alarm is my wake-up alarm <span style="border: 1px solid black; padding: 2px;">on</span>
builtin.alarm.duration	snooze for <span style="border: 1px solid black; padding: 2px;">10 minutes</span> snooze alarm for <span style="border: 1px solid black; padding: 2px;">5 minutes</span>

ENTITY	EXAMPLE OF ENTITY IN UTTERANCE
builtin.alarm.start_date	set an alarm for <b>monday</b> at 7 am set an alarm for <b>tomorrow</b> at noon
builtin.alarm.start_time	create an alarm for <b>30 minutes</b> set the alarm to go off <b>in 20 minutes</b>
builtin.alarm.title	is my <b>wake up</b> alarm on can you set an alarm for quarter to 12 Monday to Friday called <b>take antibiotics</b>
builtin.calendar.absolute_location	create an appointment for tomorrow at <b>123 main street</b> the meeting will take place in <b>cincinnati</b> on the 5th of june
builtin.calendar.contact_name	put a marketing meeting on my calendar and be sure that <b>joe</b> is there I want to set up a lunch at il fornaio and invite <b>paul</b>
builtin.calendar.destination_calendar	add this to my <b>work</b> schedule put this on my <b>personal</b> calendar
builtin.calendar.duration	set up an appointment for <b>an hour</b> at 6 tonight book a <b>2 hour</b> meeting with joe
builtin.calendar.end_date	create a calendar entry called vacation from tomorrow until <b>next monday</b> block my time as busy until <b>monday, october 5th</b>
builtin.calendar.end_time	the meeting ends at <b>5:30 PM</b> schedule it from 11 to <b>noon</b>
builtin.calendar.implicit_location	cancel the appointment at the dmv change the location of miles' birthday to <b>poppy restaurant</b>
builtin.calendar.move_earlier_time	push the meeting forward <b>an hour</b> move the dentist's appointment up <b>30 minutes</b>
builtin.calendar.move_later_time	move my dentist appointment <b>30 minutes</b> push the meeting out <b>an hour</b>
builtin.calendar.original_start_date	reschedule my appointment at the barber from 'Tuesday' to Wednesday move my meeting with ken from <b>monday</b> to Tuesday
builtin.calendar.original_start_time	reschedule my meeting from <b>2:00</b> to 3 change my dentist appointment from <b>3:30</b> to 4
builtin.calendar.start_date	what time does my party start on <b>flag day</b> schedule lunch for the <b>friday after next</b> at noon
builtin.calendar.start_time	I want to schedule it for <b>this morning</b> I want to schedule it in the <b>morning</b>

ENTITY	EXAMPLE OF ENTITY IN UTTERANCE
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
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.communication.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

ENTITY	EXAMPLE OF ENTITY IN UTTERANCE
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
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

ENTITY	EXAMPLE OF ENTITY IN UTTERANCE
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
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?

ENTITY	EXAMPLE OF ENTITY IN UTTERANCE
builtin.places.open_status	when is the mall <b>closed</b> get me the <b>opening</b> hours of the store
builtin.places.place_name	take me to <b>central park</b> look up the <b>eiffel tower</b>
builtin.places.place_type	<b>atms</b> <b>post office</b>
builtin.places.prefer_route	show directions by the <b>shortest</b> route take the <b>fastest</b> route
builtin.places.price_range	give me places that are <b>moderately affordable</b> I want an <b>expensive</b> one
builtin.places.product	where can I get <b>fresh fish</b> around here where around here sells <b>bare minerals</b>
builtin.places.public_transportation_route	bus schedule for the <b>m2</b> bus bus route <b>3x</b>
builtin.places.rating	show <b>3 star</b> restaurants show results that are <b>3 stars or higher</b>
builtin.places.reservation_number	book a table for <b>seven</b> people make a reservation for <b>two</b> at il forno
builtin.places.results_number	show me the <b>10</b> coffee shops closest to here show me top <b>3</b> aquariums
builtin.places.service_provided	where can I go to <b>whale watch</b> by bus ? I need a mechanic to <b>fix my brakes</b>
builtin.places.time	I want places that are open on Saturday at <b>8 am</b>
builtin.places.transportation_company	train schedules for <b>new jersey transit</b> can I get there on <b>bart</b>
builtin.places.transportation_type	where is a music store i can get to <b>on foot</b> ? give me <b>biking</b> directions to mashiko
builtin.places.travel_time	I want to be able to drive <b>less than 15 minutes</b> I want somewhere I can get to in <b>under 15 minutes</b>
builtin.reminder.absolute_location	remind me to call my dad when i land in <b>chicago</b> when I get back to <b>seattle</b> remind me to get gas
builtin.reminder.contact_name	when <b>bob</b> calls, remind me to tell him the joke create a reminder to mention the school bus when I talk to <b>arthur</b>

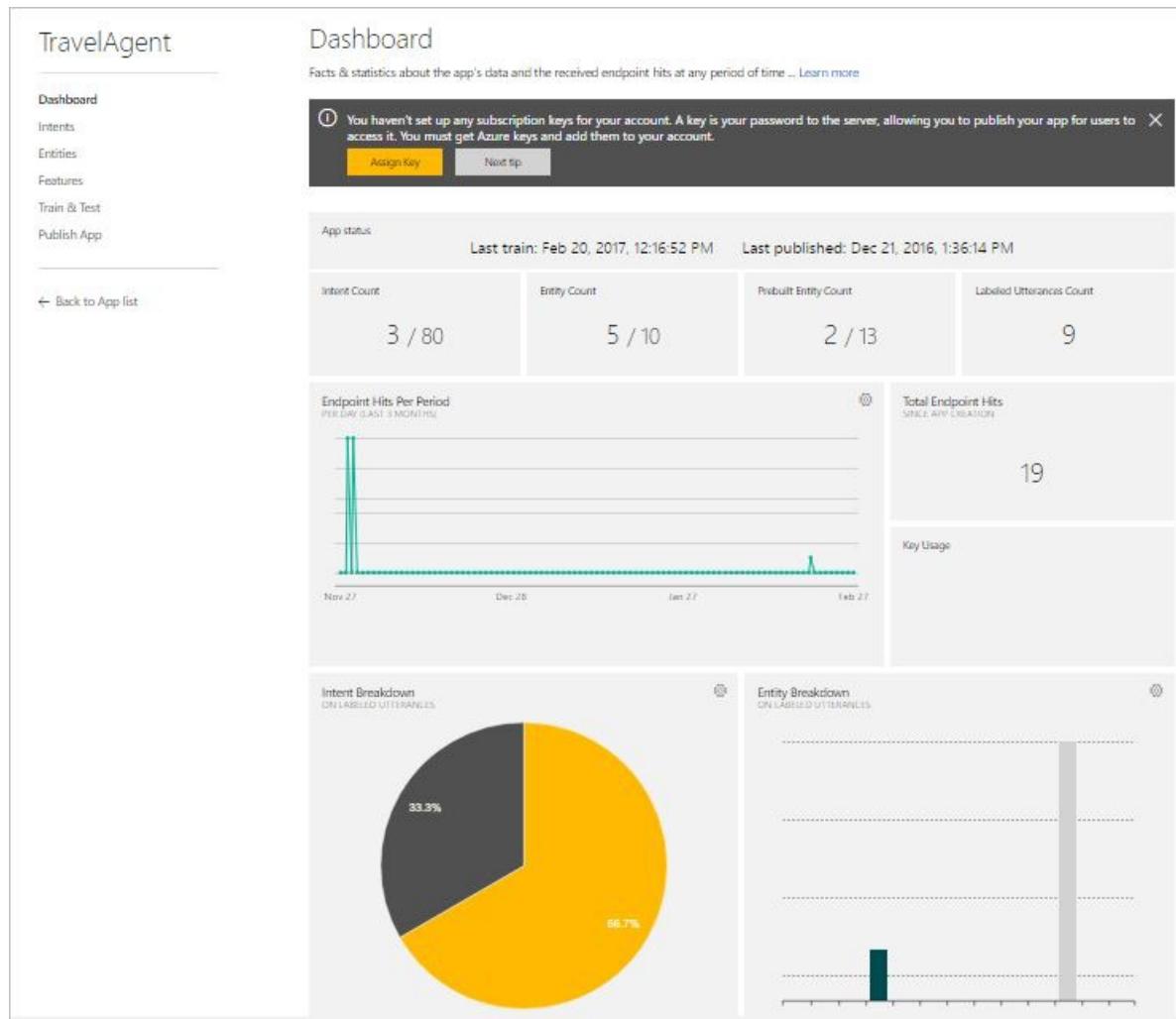
ENTITY	EXAMPLE OF ENTITY IN UTTERANCE
builtin.reminder.leaving_absolute_location	reminder pick up craig when leaving <code>1200 main</code>
builtin.reminder.leaving_implicit_location	remind me to get gas when I leave <code>work</code>
builtin.reminder.original_start_date	change the reminder about the lawn from <code>saturday</code> to Sunday move my reminder about school from <code>monday</code> to Tuesday
builtin.reminder.relationship_name	when my <code>husband</code> calls, remind me to tell him about the pta meeting remind me again when <code>mom</code> calls
builtin.reminder.reminder_text	can you remind me to <code>bring up my small spot of patchy skin</code> when dr smith's calls remind me to <code>pick up dry cleaning</code> at 4:40
builtin.reminder.start_date	remind me the <code>thursday after next</code> at 8 pm remind me <code>next thursday the 18th</code> at 8 pm
builtin.reminder.start_time	create a reminder <code>in 30 minutes</code> create a reminder to water the plants <code>this evening at 7</code>
builtin.weather.absolute_location	will it rain in <code>boston</code> what's the forecast for <code>seattle</code> ?
builtin.weather.date_range	weather in nyc <code>this weekend</code> look up the <code>five day</code> forecast in hollywood florida
builtin.weather.suitable_for	can I go <code>hiking</code> in shorts this weekend? will it be nice enough to <code>walk</code> to the game today?
builtin.weather.temperature_unit	what is the temperature today in <code>kelvin</code> show me the temps in <code>celsius</code>
builtin.weather.time_range	does it look like it will snow <code>tonight</code> ? is it windy right <code>now</code> ?
builtin.weather.weather_condition	show <code>precipitation</code> how thick is the <code>snow</code> at lake tahoe now?

# 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.

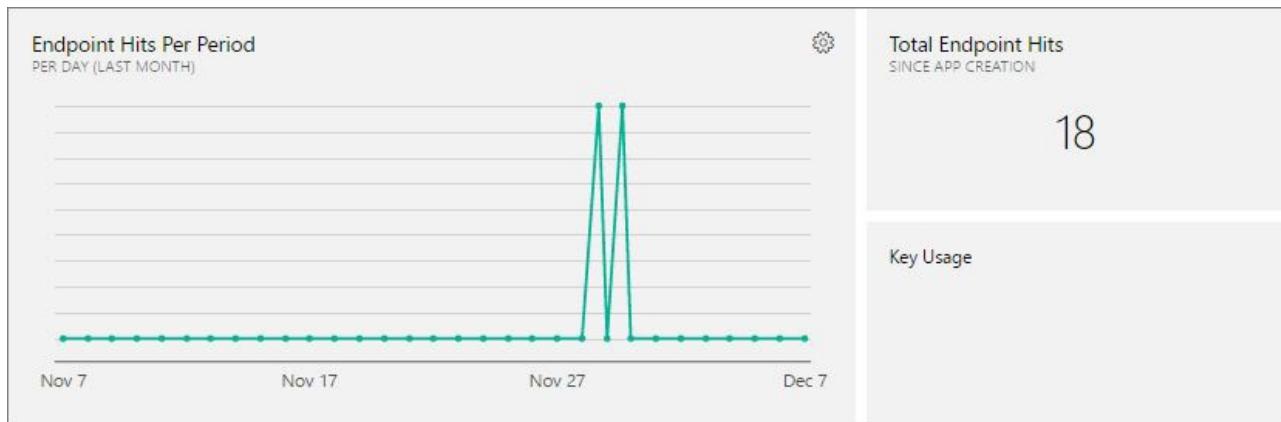
## Model Data Statistics

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

Intent Count	Entity Count	Prebuilt Entity Count	Labeled Utterances Count
3 / 80	5 / 10	2 / 5	9

## 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.

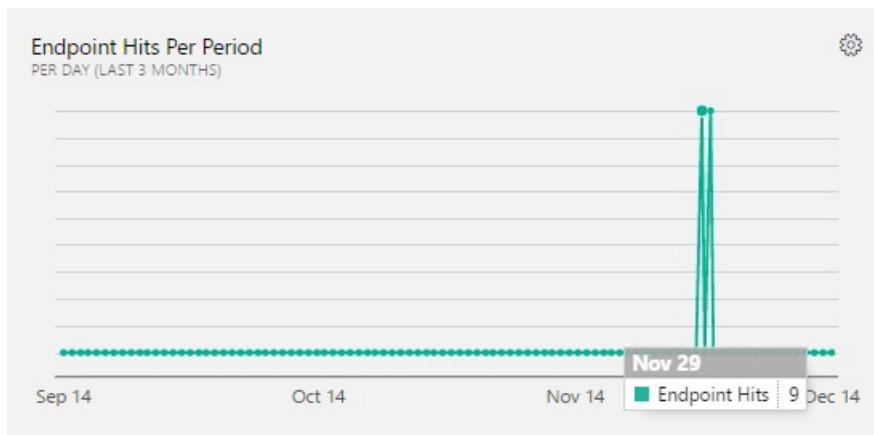


### Total endpoint hits

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.

The screenshot shows a user interface for selecting a time period. At the top right is a back arrow icon. Below it is the title "Endpoint Hits Per Period". Underneath the title is the instruction "Select Period:". A dropdown menu is open, listing several options: "Last Month", "Last Week", "Last Month" (which is highlighted in blue), "Last 3 Months", "Last 6 Months", and "Last Year".

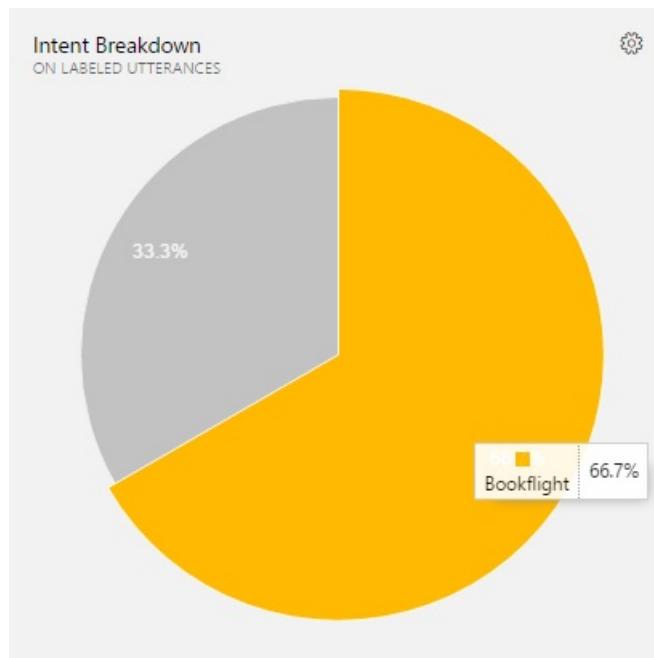
2. Select a period from the list and then click the back arrow to display its hits on the chart.

### 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.

←

## Intent Breakdown

---

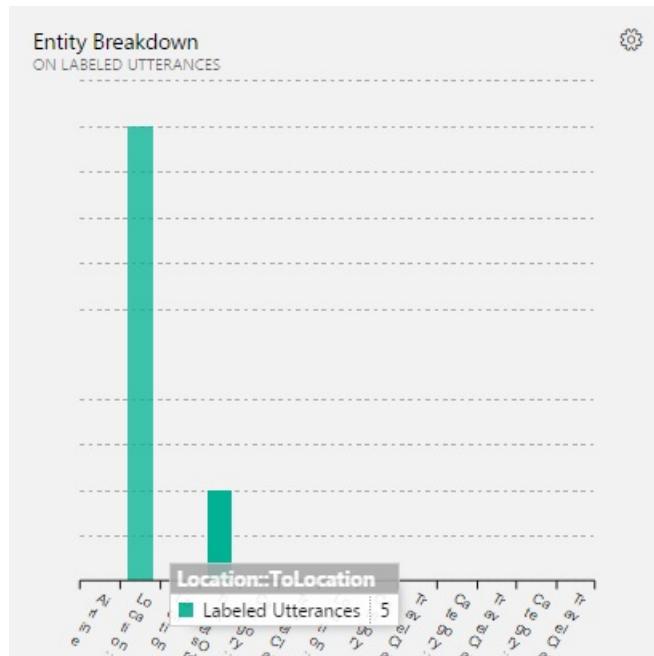
Breakdown based on:

Labeled utterances  
Endpoint hits  
Labeled utterances

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.



## Entity Breakdown

Breakdown based on:

Labeled utterances

Endpoint hits

Labeled utterances

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

# Manage your keys

10/5/2017 • 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 **Change** link to open the **Assign a key to your app** dialog.

The screenshot shows a modal dialog titled "Assign a key to your app". It contains three required fields: "Tenant Id (REQUIRED)", "Subscription Name (REQUIRED)", and "Key (REQUIRED)". Each field has a dropdown menu labeled "Select [field name]". Below the dropdowns are "Save" and "Cancel" buttons. The word "Loading" is displayed above the "Select subscription" dropdown.

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.

Assign a key to your app X

Tenant Id (REQUIRED)

Subscription Name (REQUIRED)

Select subscription ▼

Select subscription

- Internal
- Dev
- Bot
- Test

Save Cancel

5. Select a key associated with the subscription.

Assign a key to your app X

Tenant Id (REQUIRED)

Subscription Name (REQUIRED)

Key (REQUIRED)

Select key ▼

Select key

- TestTravelAgentKey
- TravelAgentKey
- TravelAgentKeyEurope
- TravelAgentKeyAsia

## 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	<a href="https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY">https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</a>
East US 2	<a href="https://eastus2.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY">https://eastus2.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</a>
West Central US	<a href="https://westcentralus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY">https://westcentralus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</a>

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

#### 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	<a href="https://westeurope.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY">https://westeurope.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</a>

## 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

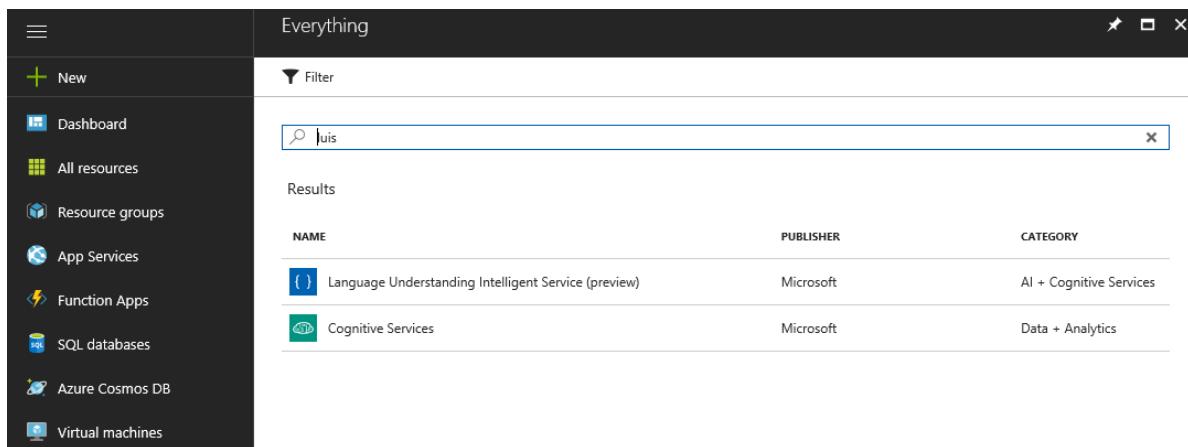
10/9/2017 • 1 min to read • [Edit Online](#)

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.



The screenshot shows the Microsoft Azure portal interface. On the left, there is a dark sidebar with various service icons and links: Dashboard, All resources, Resource groups, App Services, Function Apps, SQL databases, Azure Cosmos DB, and Virtual machines. A green '+' icon is located at the top of this sidebar. The main area has a dark header with the text 'Everything' and a search bar containing the text 'luis'. Below the header, there is a 'Filter' button. The results section is titled 'Results' and contains two items in a table format:

NAME	PUBLISHER	CATEGORY
Language Understanding Intelligent Service (preview)	Microsoft	AI + Cognitive Services
Cognitive Services	Microsoft	Data + Analytics

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

The screenshot shows the Azure portal's 'Create' blade for 'Language Understanding Intelligent Service (LUIS...)'.

**Name:** Enter a name (Luis Internal)

**Subscription:** Luis Internal

**Location:** West US

**Pricing tier:** F0 (5 Calls per second, 10K Calls per month) or S0 (50 Calls per second)

**Consent:**  \* I confirm I have read and understood the notice below.

**Notice:** Microsoft will use data you send to the Cognitive Services to improve Microsoft products and services. Where you send personal data to the Cognitive Services, you are responsible for obtaining sufficient consent from the data subjects. The General Privacy and Security Terms in the Online Services Terms do not apply to the Cognitive Services. Please refer to the Microsoft Cognitive Services section in the [Online Services Terms](#) for details. Microsoft offers policy controls that may be used to [disable new Cognitive Services deployments](#).

- 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.

The screenshot shows the Azure portal interface for managing a Cognitive Services resource named 'dfbLuis1'. The left sidebar lists various service categories like Dashboard, All resources, Resource groups, App Services, Function Apps, SQL databases, Azure Cosmos DB, Virtual machines, and Load balancers. The main content area is titled 'Keys' under 'dfbLuis1 - Keys' and 'Cognitive Services'. It includes sections for Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. Under 'RESOURCE MANAGEMENT', the 'Keys' section is highlighted. It contains fields for 'NAME' (dfbLuis1), 'KEY 1' (<LUIS access key>), and 'KEY 2' (<LUIS access key>). A note at the top right states: 'Notice: It may take up to 10 minutes for the newly (re)generated keys to take effect.'

## Using LUIS access keys in luis.ai

In order to use the access keys step 4, you need to change from the StarterKey to one of the new access keys as part of [publishing your app](#).

Please see [Manage your keys](#) for more information about how to manage your Azure LUIS subscription keys in your [luis.ai account](#).

## Next steps

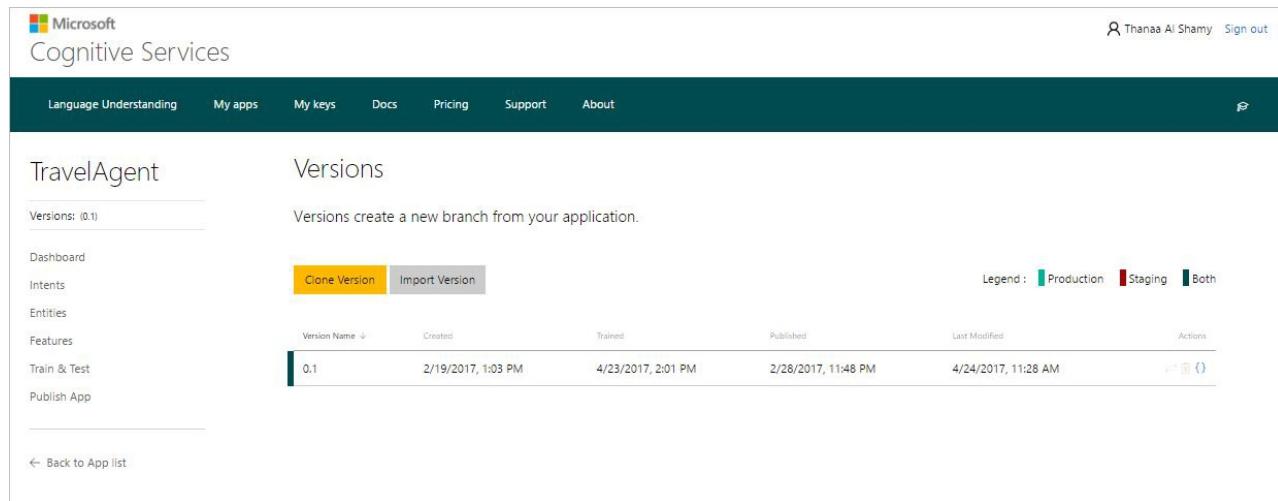
Now that you created your LUIS subscription account, you should understand how to version your application. For more info on how to create and manage versions of your LUIS app, see [Manage Versions](#).

# 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 screenshot shows the Microsoft Cognitive Services portal. In the top navigation bar, there is a Microsoft logo, the text "Cognitive Services", and user information "Thanaa Al Shamy Sign out". Below the navigation bar, there is a dark blue header with links for "Language Understanding", "My apps", "My keys", "Docs", "Pricing", "Support", and "About". On the left side, there is a sidebar for the "TravelAgent" app with sections for "Dashboard", "Intents", "Entities", "Features", "Train & Test", and "Publish App". Below the sidebar, there is a link "← Back to App list". The main content area is titled "Versions" and contains the message "Versions create a new branch from your application.". Below this message, there are two buttons: "Clone Version" (highlighted in yellow) and "Import Version". To the right of these buttons, there is a legend: "Legend : Production (green square), Staging (red square), Both (blue square)". A table follows, showing a single row of version data:

Version Name	Created	Trained	Published	Last Modified	Actions
0.1	2/19/2017, 1:03 PM	4/23/2017, 2:01 PM	2/28/2017, 11:48 PM	4/24/2017, 11:28 AM	

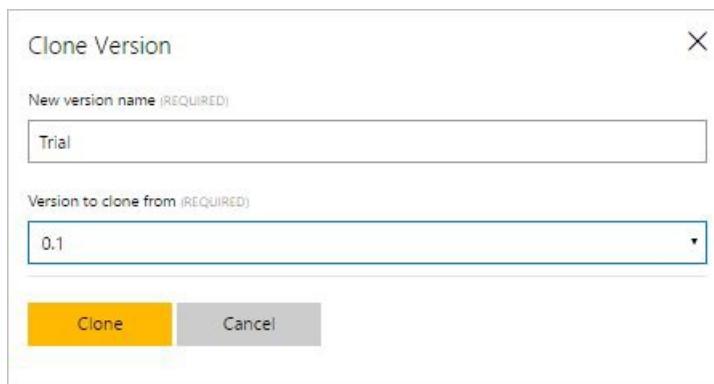
The following procedures will guide you through the actions you need to do while working with versions.

## 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**.



The dialog box has a title "Clone Version" and a close button "X". It contains two input fields: "New version name (REQUIRED)" with the value "Trial" and "Version to clone from (REQUIRED)" with the value "0.1". At the bottom, there are two buttons: "Clone" (highlighted in yellow) and "Cancel".

### 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.

## Versions

Versions create a new branch from your application.

**Clone Version** **Import Version** Legend : ■ Production ■ Staging ■ Both

Version Name	Created	Trained	Published	Last Modified	Actions
0.1	2/19/2017, 1:03 PM	4/23/2017, 2:01 PM	2/28/2017, 11:48 PM	4/24/2017, 11:28 AM	{}
Trial	4/25/2017, 12:34 AM			4/25/2017, 12:34 AM	{}}

### 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.

## Versions

Versions create a new branch from your application.

**Clone Version** **Import Version** Legend : ■ Production ■ Staging ■ Both

Version Name	Created	Trained	Published	Last Modified	Actions
0.1	2/19/2017, 1:03 PM	4/23/2017, 2:01 PM	2/28/2017, 11:48 PM	4/24/2017, 11:28 AM	{}}
0.2	4/26/2017, 12:21 AM			4/26/2017, 12:54 AM	{}}
Trial	4/25/2017, 12:34 AM			4/26/2017, 12:22 AM	{}}

## Import/export a version

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

### To import a version:

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

import.

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



#### To export a version:

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

## Delete a version

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

#### To delete a version:



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

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

6/27/2017 • 1 min to read • [Edit Online](#)

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

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

## Collaborators

Collaborators are people who can also edit this app. ... [Learn more](#)

Original owner: Patti.Owens@outlook.com

Type email here  Add collaborator

Collaborators	Actions
vernakennedy@outlook.com	

- 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.

One of your models has been edited by a collaborator.

My apps My keys Docs Pricing Support

test App Settings

**NOTE**

Collaborators cannot add other collaborators.

# Language Understanding Intelligent Services (LUIS) Frequently Asked Questions

9/28/2017 • 7 min to read • [Edit Online](#)

This article contains answers to some frequently asked questions about 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 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 and are listed in the following table:

TYPE	LIMIT
Prebuilt entities	No limit.
List entities	50 list entities. Each list can contain up to 20000 items
Simple, Hierarchical, and Composite entities	You can define up to 30 entities of these types. 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 the user is asking for, but they don't need to capture every path your code takes. For example, BookFlight and BookHotel may be separate intents in a travel app, but BookInternationalFlight and BookDomesticFlight are too similar, and if your system needs to distinguish them, use entities or other logic rather than intents.

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

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

- Create a top-level LUIS app to determine the category of utterance, and then use the result to send the utterance to the LUIS app for that category.

- Do some preprocessing on the utterance, like matching on regular expressions, to determine which LUIS app or set of apps receives it.

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

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

If reducing the number of intents or dividing your intents into multiple apps won't work for you, provide detailed information about your system and contact support. You can get support by clicking **Support** in [www.luis.ai](http://www.luis.ai), or through Azure technical support if your Azure subscription includes support services.

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

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

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

Luis 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 20000 items.

If you've considered hierarchical, composite, and list entities and still need more than the limit, provide detailed information about your system and contact support. You can get support by clicking **Support** in [www.luis.ai](http://www.luis.ai), or through Azure technical support if your Azure subscription includes support services.

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

The maximum length of a [phrase list](#) is 5000 items. You may 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 on 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 100-200 unlabeled utterances to your app. You can use the suggestion feature to leverage LUIS intelligence in selecting the most informative utterances to label. You can select the intent or entity you want to improve and label the utterances suggested by LUIS. Labeling a few hundred 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 LUIS modeling best practices.

## Is it important to train the None intent?

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

## How can I deal with spelling mistakes in utterances?

You have one of two options:

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

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

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

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

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

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

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

## How do I download a log of user utterances?

By default, utterances from users are logged. To download a log of utterances that users sent to your LUIS app, click the download icon in the entry for your app in **My Apps**. This log is in comma-separated value (CSV) format.

## How can I disable 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, note that this will disable your LUIS app's ability to suggest utterances or improve performance based on user queries. If you set `log=false` due to data privacy concerns be aware that you won't be able to download a record of user utterances from LUIS or use those utterances to improve your app.

## How can I delete data from LUIS?

- If you delete an utterance from your LUIS app, it is removed from the LUIS web service and not available for export.
- If you delete an account, all apps and their utterances are deleted. Data is retained on the servers for 60 days before permanent deletion.

## What is the tenant ID in the Add a key to your app dialog?

In Azure, a tenant represents the client or organization associated with a service. You can find your tenant ID in the Azure portal under Azure Active Directory > Manage > Properties, in the Directory ID field.

The screenshot shows the Azure portal interface. On the left, there's a sidebar with various service icons. The 'Azure Active Directory' icon is highlighted with a red box. The main area has a title bar with 'Save' and 'Discard' buttons. Below that, there are several configuration fields:

- Name:** Contoso
- Country or region:** United States
- Location:** Asia, United States, Europe datacenters
- Notification language:** English
- Global admin can manage Azure Subscriptions:** Yes (button)
- Directory ID:** A placeholder field containing 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx' which is also highlighted with a red box.

A vertical scroll bar is visible on the right side of the main content area.

## Next steps

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