

Contents

[LUIS Documentation](#)

[Overview](#)

[What is Language Understanding?](#)

[What's new](#)

[Quickstarts](#)

[Build prebuilt domain app](#)

[Use LUIS portal](#)

[1. Create an app](#)

[2. Deploy an app](#)

[Use .NET SDK](#)

[Create and manage app](#)

[Query prediction endpoint](#)

[Use Python SDK](#)

[Create and manage app](#)

[Use REST APIs](#)

[Get intent](#)

[C#](#)

[Go](#)

[Java](#)

[Node.js](#)

[Python](#)

[Change model](#)

[C#](#)

[Go](#)

[Java](#)

[Node.js](#)

[Python](#)

[Tutorials](#)

[Build app](#)

Determine user intentions

Identify common user data with prebuilt models

Get sentiment from user utterances

Get well-formatted data with regular expressions

Get exact text matches with lists

Get related data with entity roles

Group and extract related data with composite entities

Get names with simple entities

Improve app

Fix unsure predictions

Batch test datasets

Add common utterance formats

Extract contextually-related patterns

Extract free-form data

Use with bot

With C#

1. Create basic bot
2. Add telemetry to Application Insights

With Node.js

1. Create basic bot
2. Add telemetry to Application Insights

Samples

Code samples

Concepts

Best practices

Using LUIS and QnA Maker

Plan your app

Author App

Authoring cycle

Collaboration

Versioning

Design Model

- [Intents](#)
- [Entities](#)
- [Utterances](#)
- [Roles](#)
- [Prebuilt models](#)
- [Improve app performance](#)
 - [Prediction score](#)
 - [Review endpoint utterances](#)
 - [Phrase lists](#)
 - [Patterns](#)
- [Data management](#)
 - [Altering data](#)
 - [Converting data](#)
 - [Extracting data](#)
 - [Data Storage](#)
- [Test](#)
 - [Interactive testing](#)
 - [Batch testing](#)
- [Enterprise](#)
 - [Design strategies](#)
- [Security](#)
 - [App access](#)
 - [Authoring and endpoint keys](#)
- [How-to guides](#)
 - [Start a new app](#)
 - [Build Model](#)
 - [Add an intent](#)
 - [Add entity to utterance](#)
 - [Add entity without utterance](#)
 - [Use prebuilt models](#)
 - [Use domains](#)
 - [Use intents](#)

- [Use entities](#)
- [With REST APIs](#)
 - [Build app programmatically using Node.js](#)
 - [Use list entity to match exact text using Node.js](#)
- [Integrate](#)
 - [With Bing Spell Check v7](#)
 - [With Speech service](#)
- [Improve prediction accuracy](#)
 - [Using the dashboard](#)
 - [Add Phrase list](#)
 - [Review endpoint utterances](#)
 - [Add Patterns](#)
- [Train](#)
- [Test](#)
 - [Interactive testing](#)
 - [Batch testing](#)
- [Publish](#)
 - [Manage app](#)
 - [Using subscription keys with your LUIS app](#)
 - [Manage versions](#)
 - [Manage authors and collaborators](#)
 - [Manage authoring key](#)
 - [Spread endpoint quota across keys with Traffic Manager](#)
- [Migrate API](#)
 - [Migrate to V2 REST API](#)
 - [Migrate to V3 REST API](#)
- [Use containers](#)
 - [Install and run containers](#)
 - [Configure containers](#)
 - [Use container instances](#)
- [Reference](#)
 - [User privacy](#)

[Regions](#)

[Boundaries](#)

[Application settings](#)

[Prebuilt entity reference](#)

[Entities per culture](#)

[Age entity](#)

[Currency entity](#)

[DatetimeV2 entity](#)

[Dimension entity](#)

[Deprecated entities](#)

[Email entity](#)

[GeographyV2 entity](#)

[KeyPhrase entity](#)

[Number entity](#)

[Ordinal entity](#)

[Ordinal V2 entity](#)

[Percentage entity](#)

[PersonName entity](#)

[Phonenumber entity](#)

[Temperature entity](#)

[URL entity](#)

[Prebuilt domain reference](#)

[Custom entity reference](#)

[Composite entity](#)

[List entity](#)

[Pattern.any entity](#)

[Regular expression entity](#)

[Simple entity](#)

[Azure CLI](#)

[Azure RM PowerShell](#)

[REST APIs](#)

[Authoring V2 REST APIs](#)

[Endpoint V2 REST API](#)

[HTTP codes](#)

[SDKs](#)

[C#](#)

[NuGet - authoring](#)

[NuGet - runtime](#)

[Reference](#)

[Samples](#)

[Go](#)

[SDK - authoring](#)

[SDK - runtime](#)

[Reference](#)

[Java](#)

[Maven - authoring](#)

[Maven - runtime](#)

[Reference](#)

[Samples](#)

[Node.js](#)

[NPM - authoring](#)

[NPM - runtime](#)

[Samples](#)

[Python](#)

[Pip SDK](#)

[Reference](#)

[Samples](#)

[Related](#)

[Azure Bot Service](#)

[Bot Builder 3.x SDK C#](#)

[Bot Builder 4.x SDK C#](#)

[QnA maker](#)

[Bing Speech API](#)

[Bing Spell Check API](#)

[Text Analytics](#)
[Recognizers-Text](#)
[Intelligent Kiosk](#)

[Resources](#)

[Regional availability](#)
[Language and region support](#)
[Azure Roadmap](#)
[Glossary](#)
[Troubleshooting](#)
[Pricing and limits](#)
[Compliance](#)
[Stack Overflow](#)
[UserVoice](#)
[Search](#)
[Blog](#)
[Knowledge Base](#)

What is Language Understanding (LUIS)?

7/26/2019 • 4 minutes to read • [Edit Online](#)

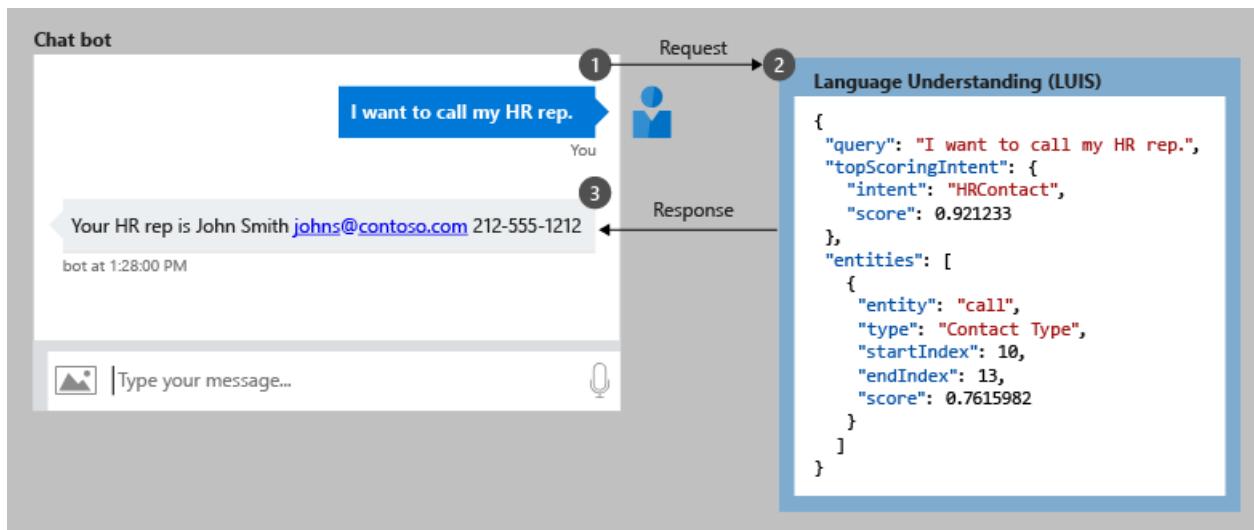
Language Understanding (LUIS) is a cloud-based API service that applies custom machine-learning intelligence to a user's conversational, natural language text to predict overall meaning, and pull out relevant, detailed information.

A client application for LUIS is any conversational application that communicates with a user in natural language to complete a task. Examples of client applications include social media apps, chat bots, and speech-enabled desktop applications.



Use LUIS in a chat bot

Once the LUIS app is published, a client application sends utterances (text) to the LUIS natural language processing endpoint [API](#) and receives the results as JSON responses. A common client application for LUIS is a chat bot.



STEP	ACTION
1	The client application sends the user <i>utterance</i> (text in their own words), "I want to call my HR rep." to the LUIS endpoint as an HTTP request.

STEP	ACTION
2	LUIS applies the learned model to the natural language text to provide intelligent understanding about the user input. LUIS returns a JSON-formatted response, with a top intent, "HRContact". The minimum JSON endpoint response contains the query utterance, and the top scoring intent. It can also extract data such as the Contact Type entity.
3	The client application uses the JSON response to make decisions about how to fulfill the user's requests. These decisions can include some decision tree in the bot framework code and calls to other services.

The LUIS app provides intelligence so the client application can make smart choices. LUIS doesn't provide those choices.

Natural language processing

A LUIS app contains a domain-specific natural language model. You can start the LUIS app with a prebuilt domain model, build your own model, or blend pieces of a prebuilt domain with your own custom information.

- **Prebuilt model** LUIS has many prebuilt domain models including intents, utterances, and prebuilt entities. You can use the prebuilt entities without having to use the intents and utterances of the prebuilt model. [Prebuilt domain models](#) include the entire design for you and are a great way to start using LUIS quickly.
- **Custom Entities** LUIS gives you several ways to identify your own custom intents and entities including machine-learned entities, specific or literal entities, and a combination of machine-learned and literal.

Build the LUIS model

Build the model with the [authoring](#) APIs or with the LUIS portal.

The LUIS model begins with categories of user intentions called **intents**. Each intent needs examples of user **utterances**. Each utterance can provide a variety of data that needs to be extracted with **entities**.

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

Query prediction endpoint

After the model is built and published to the endpoint, the client application sends utterances to the published prediction [endpoint](#) API. The API applies the model to the text for analysis. The API responds with the prediction results in a JSON format.

The minimum JSON endpoint response contains the query utterance, and the top scoring intent. It can also extract data such as the following **Contact Type** entity.

```
{
  "query": "I want to call my HR rep.",
  "topScoringIntent": {
    "intent": "HRContact",
    "score": 0.921233
  },
  "entities": [
    {
      "entity": "call",
      "type": "Contact Type",
      "startIndex": 10,
      "endIndex": 13,
      "score": 0.7615982
    }
  ]
}
```

Improve model prediction

After a LUIS model is published and receives real user utterances, LUIS provides several methods to improve prediction accuracy: [active learning](#) of endpoint utterances, [phrase lists](#) for domain word inclusion, and [patterns](#) to reduce the number of utterances needed.

Development lifecycle

Luis provides tools, versioning, and collaboration with other LUIS authors to integrate into the full development life cycle at the level of the client application and the language model.

Implementing LUIS

LUIS, as a REST API, can be used with any product, service, or framework that makes an HTTP request. The following list contains the top Microsoft products and services used with LUIS.

The top client application for LUIS is:

- [Web app bot](#) quickly creates a LUIS-enabled chat bot to talk with a user via text input. Uses [Bot Framework](#) version [4.x](#) for a complete bot experience.

Tools to quickly and easily use LUIS with a bot:

- [LUIS CLI](#) The NPM package provides authoring and prediction with as either a stand-alone command line tool or as import.
- [LUISGen](#) LUISGen is a tool for generating strongly typed C# and typescript source code from an exported LUIS model.
- [Dispatch](#) allows several LUIS and QnA Maker apps to be used from a parent app using dispatcher model.
- [LUDown](#) LUDown is a command line tool that helps manage language models for your bot.

Other Cognitive Services used with LUIS:

- [QnA Maker](#) allows several types of text to combine into a question and answer knowledge base.
- [Bing Spell Check API](#) provides text correction before prediction.
- [Speech service](#) converts spoken language requests into text.
- [Conversation learner](#) allows you to build bot conversations quicker with LUIS.
- [Project personality chat](#) to handle bot small talk.

Samples using LUIS:

- [Conversational AI](#) GitHub repository.
- [Language Understanding](#) Azure samples

Next steps

Author a new LUIS app with a [prebuilt](#) or [custom](#) domain. [Query the prediction endpoint](#) of a public IoT app.

What's new in Language Understanding

8/9/2019 • 2 minutes to read • [Edit Online](#)

Learn what's new in the service. These items may release notes, videos, blog posts, and other types of information. Bookmark this page to keep up-to-date with the service.

Release notes

July 23, 2019

- Update the [Recognizers-Text](#) to 1.2.3
 - Age, Temperature, Dimension, and Currency recognizers in Italian.
 - Improvement in Holiday recognition in English to correctly calculate Thanksgiving-based dates.
 - Improvements in French DateTime to reduce false positives of non-Date and non-Time entities.
 - Support for calendar/school/fiscal year and acronyms in English DateRange.
 - Improved PhoneNumber recognition in Chinese and Japanese.
 - Improved support for NumberRange in English.
 - Performance improvements.

June 24, 2019

- [OrdinalV2 prebuilt entity](#) to support ordering such as next, previous, and last. English culture only.

May 6, 2019 - //Build Conference

The following features were released at the Build 2019 Conference:

- [Preview of V3 API migration guide](#)
- [Improved analytics dashboard](#)
- [Improved prebuilt domains](#)
- [Dynamic list entities](#)
- [External entities](#)

Blogs

[Bot Framework](#)

Videos

2019 Build videos

[How to use Azure Conversational AI to scale your business for the next generation](#)

Service updates

[Azure update announcements for Cognitive Services](#)

Quickstart: Use prebuilt Home automation app

7/26/2019 • 2 minutes to read • [Edit Online](#)

In this quickstart, create a LUIS app that uses the prebuilt domain `HomeAutomation` for turning lights and appliances on and off. This prebuilt domain provides intents, entities, and example utterances for you. When you're finished, you'll have a LUIS endpoint running in the cloud.

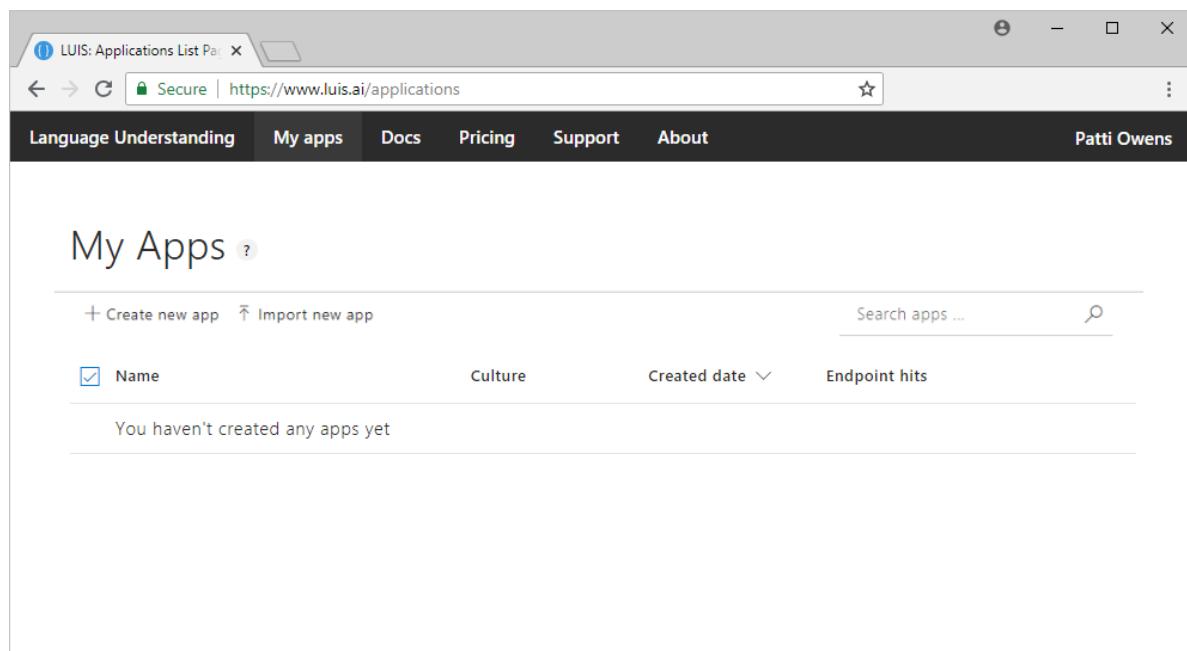
Prerequisites

For this article, you need a free LUIS account, created on the LUIS portal at <https://www.luis.ai>.

Create a new app

You can create and manage your applications on **My Apps**.

1. Sign in to the LUIS portal.
2. Select **Create new app**.



3. In the dialog box, name your application "Home Automation".

Create new app

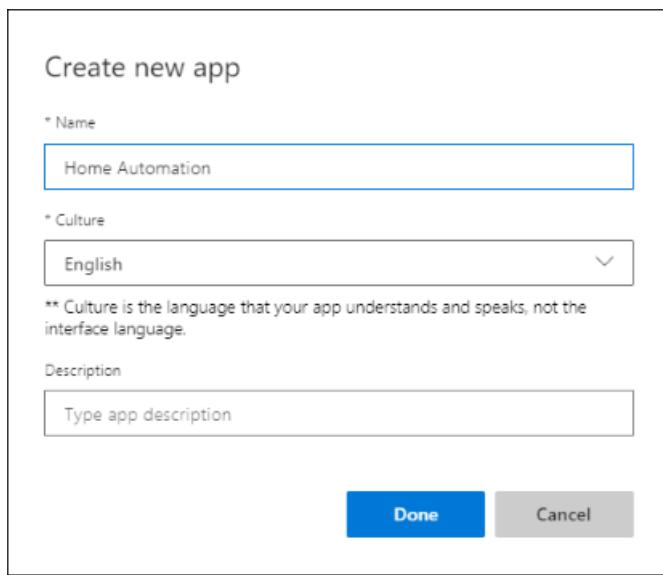
* Name
Home Automation

* Culture
English

** Culture is the language that your app understands and speaks, not the interface language.

Description
Type app description

Done **Cancel**



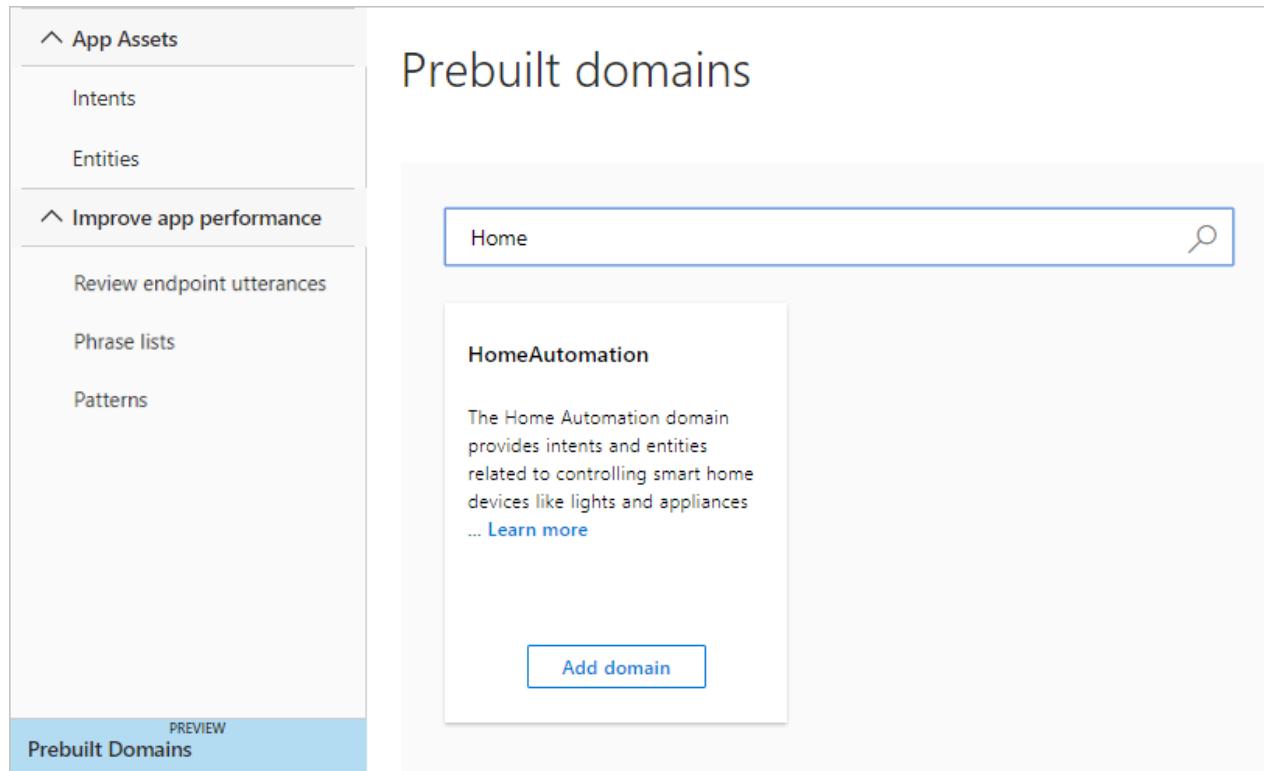
4. Choose your application culture. For this Home Automation app, choose English. Then select **Done**. LUIS creates the Home Automation app.

NOTE

The culture cannot be changed once the application is created.

Add prebuilt domain

Select **Prebuilt domains** in the left-side navigation pane. Then search for "Home". Select **Add domain**.



App Assets

Intents

Entities

Improve app performance

Review endpoint utterances

Phrase lists

Patterns

PREVIEW

Prebuilt Domains

Prebuilt domains

Home

HomeAutomation

The Home Automation domain provides intents and entities related to controlling smart home devices like lights and appliances ... [Learn more](#)

Add domain

When the domain is successfully added, the prebuilt domain box displays a **Remove domain** button.

^ App Assets

- Intents
- Entities

^ Improve app performance

- Review endpoint utterances
- Phrase lists
- Patterns

PREVIEW

Prebuilt Domains

Prebuilt domains

Home

HomeAutomation

The Home Automation domain provides intents and entities related to controlling smart home devices like lights and appliances ... [Learn more](#)

[Remove domain](#)

This screenshot shows the 'Prebuilt Domains' section of the LUIS interface. On the left, there's a navigation pane with sections for 'App Assets' (Intents, Entities), 'Improve app performance' (Review endpoint utterances, Phrase lists, Patterns), and a 'PREVIEW' section for 'Prebuilt Domains'. The 'Prebuilt Domains' section is currently selected. It displays a card for the 'HomeAutomation' domain, which includes a brief description, a 'Learn more' link, and a 'Remove domain' button. A search bar is at the top right.

Intents and entities

Select **Intents** in the left-side navigation pane to review the HomeAutomation domain intents. Each intent has sample utterances.

▼ App Assets

- Intents**
- Entities

▼ Improve app performance

- Review endpoint utterances
- Phrase lists
- Patterns

Intents ?

+ Create new intent + Add prebuilt domain intent Search intents ...

Name	Labeled Utterances
None	0
QueryState	86
SetDevice	45
TurnDown	133
TurnOff	146
TurnOn	141
TurnUp	128

This screenshot shows the 'Intents' section of the LUIS interface. The left sidebar shows 'App Assets' (Intents selected) and 'Improve app performance' (Review endpoint utterances, Phrase lists, Patterns). The main area displays a list of intents for the 'HomeAutomation' domain, each with its name and the count of labeled utterances. There are buttons to 'Create new intent' and 'Add prebuilt domain intent' at the top, and a search bar.

NOTE

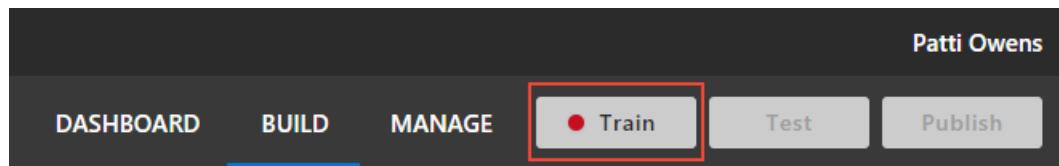
None is an intent provided by all LUIS apps. You use it to handle utterances that don't correspond to functionality your app provides.

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

The screenshot shows the LUIS Intents page for the "TurnOff" app. The left sidebar has sections for App Assets (Intents, Entities), Improve app performance (Review endpoint utterances, Phrase lists, Patterns), and Prebuilt Domains (PREVIEW). The main area shows the intent "TurnOff" with a label count of 1. It includes a toolbar with Edit, Reassign intent, Add as pattern, Delete utterance(s), Search, Filter, and View options. Below the toolbar is a section for Example utterance with a text input field and a Score button. A list of 15 labeled utterances is shown, each with a score from 0.99 to 1.00. At the bottom are navigation buttons for page 1 of 15, a Next button, and a Get Started button.

Train the LUIS app

1. In the top right side of the LUIS website, select the **Train** button.



2. Training is complete when you see the green status bar at the top of the website confirming success.



Test your app

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

The screenshot shows the Interactive Testing pane with a single test utterance: "Turn off the lights".

Check that the top scoring intent corresponds to the intent you expected for each test utterance.

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

The screenshot shows the LUIS Test pane. At the top, there are buttons for 'Start over' and 'Batch testing panel'. Below is a text input field labeled 'Type a test utterance' containing 'turn off the lights'. A dark bar displays the prediction 'HomeAutomation.TurnOff (0.99)'. To the right of the prediction is an 'Inspect' button.

Select **Test** again to collapse the test pane.

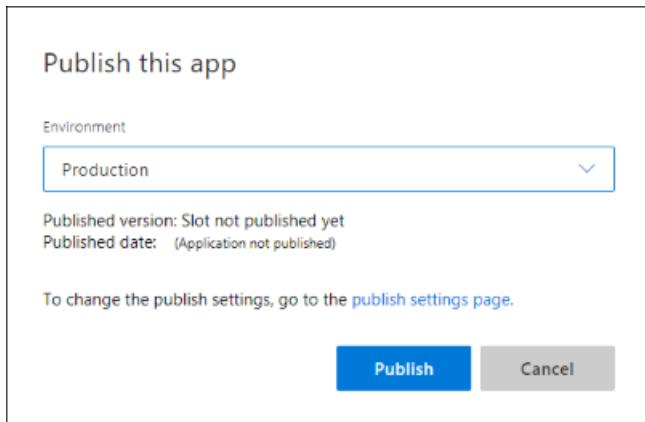
Publish the app to get the endpoint URL

In order to receive a LUIS prediction in a chat bot or other client application, you need to publish the app to the endpoint.

1. Select **Publish** in the top right navigation.



2. Select the **Production** slot and the **Publish** button.



3. Publishing is complete when you see the green status bar at the top of the website confirming success.

Publishing complete. Refer to the list of endpoints to access your endpoint URL

4. Select the **endpoints** link in the green status bar to go to the **Keys and endpoints** page. The endpoint URLs are listed at the bottom.

Query the endpoint with a different utterance

1. In the **Manage** section (top right menu), on the **Keys and endpoints** page (left menu), select the **endpoint** URL at the bottom of the page. This action opens another browser tab with the endpoint URL in the address bar.

The endpoint URL looks like

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?verbose=true&subscription-key=<YOUR_KEY>&<optional-name-value-pairs>&q=<user-utterance-text>
```

2. Go to the end of the URL in the address and enter `turn off the living room light`, and then press Enter. The browser displays the JSON response of your HTTP endpoint.



The screenshot shows a browser window with the URL `westus.api.cognitive.microsoft.com`. The page content displays a JSON object representing a language understanding result:

```
{
  "query": "turn off the living room light",
  "topScoringIntent": {
    "intent": "HomeAutomation.TurnOff",
    "score": 0.984057844
  },
  "entities": [
    {
      "entity": "living room",
      "type": "HomeAutomation.Room",
      "startIndex": 13,
      "endIndex": 23,
      "score": 0.9619945
    }
  ]
}
```

Clean up resources

When no longer needed, delete the LUIS app. To do so, select **My apps** from the top left menu. Select the ellipsis (...) to the right of the app name in the app list, select **Delete**. On the pop-up dialog **Delete app?**, select **Ok**.

Next steps

You can call the endpoint from code:

[Call a LUIS endpoint using code](#)

Quickstart: Create a new app in the LUIS portal

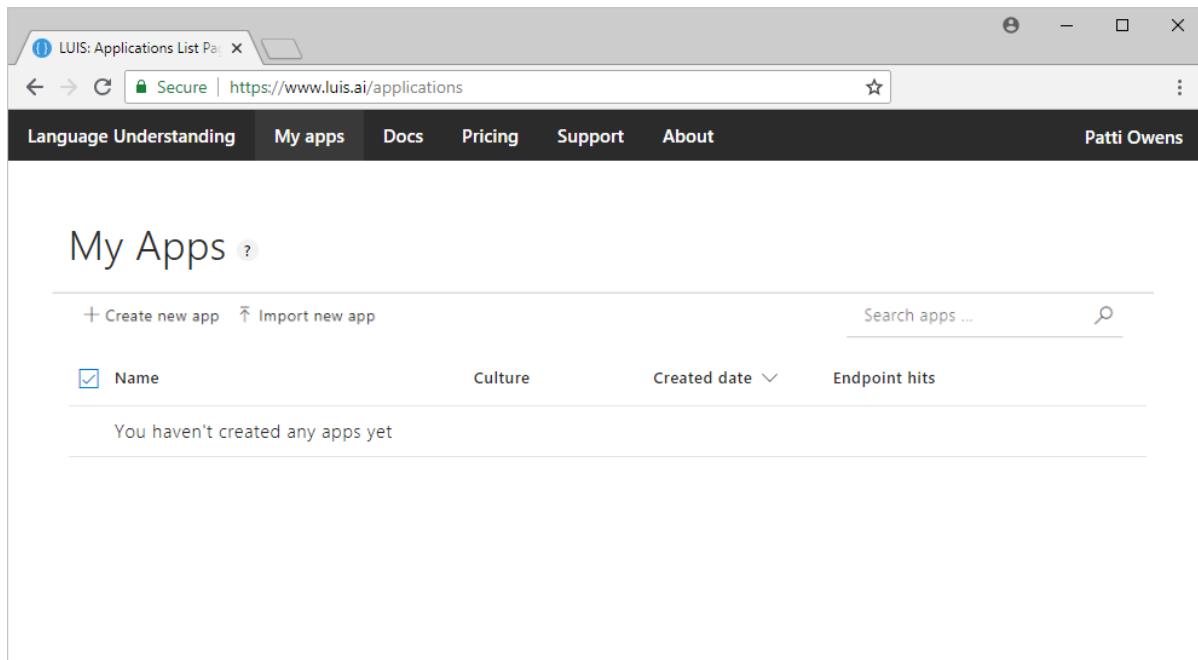
7/26/2019 • 4 minutes to read • [Edit Online](#)

In this quickstart, you build a new app in the [LUIS portal](#). First you create the basic parts of an app, **intents**, and **entities**. Then you test the app by providing a sample user utterance in the interactive test panel to get the predicted intent.

Building an app is free and doesn't require an Azure subscription. When you're ready to deploy your app, see the [quickstart to deploy an app](#). It shows you how to create an Azure Cognitive Service Resource and assign it to the app.

Create an app

1. Open the [LUIS portal](#) in a browser and sign in. If it's your first time signing in, you need to create a free LUIS portal user account.
2. Select **Create new app** from the context toolbar.



3. In the pop-up window, configure the app with the following settings and then select **Done**.

SETTING NAME	VALUE	PURPOSE
Name	myEnglishApp	Unique LUIS app name required
Culture	English	Language of utterances from users, en-us required
Description	App made with LUIS Portal	Description of app optional

Create new app

Name (Required)
Type app name

Culture (Required)
English

** Culture is the language that your app understands and speaks, not the interface language.

Description
Type app description

Done **Cancel**

Create intents

After the LUIS app is created, you need to create intents. Intents are a way to categorize text from users. For example, a human resources app might have two functions. To help people:

1. Find and apply for jobs
2. Find forms to apply for jobs

The app's two different *intentions* align to the following intents:

INTENT	EXAMPLE TEXT FROM USER KNOWN AS AN <i>UTTERANCE</i>
ApplyForJob	I want to apply for the new software engineering position in Cairo.
FindForm	Where is the job transfer form href-123456?

To create intents, complete the following steps:

1. After the app is created, you are on the **Intents** page of the **Build** section. Select **Create new intent**.

The screenshot shows the LUIS Intents page. On the left, there's a sidebar with 'App Assets' expanded, showing 'Intents' selected (highlighted in blue). The main area is titled 'Intents' with a question mark icon. At the top right, there are two buttons: '+ Create new intent' (highlighted with a red box) and '+ Add prebuilt domain intent'. Below these buttons is a search bar with the placeholder 'Search intents ...' and a magnifying glass icon. Underneath the search bar, there's a table with one row. The first column is labeled 'Name' with a dropdown arrow and contains the value 'None'. The second column is labeled 'Labeled Utterances' and contains the value '0'.

2. Enter the intent name, `FindForm`, and then select **Done**.

Create new intent

Intent name (Required)
FindForm

Done **Cancel**

Add an example utterance

You add example utterances after you create intents. Example utterances are text that a user enters in a chat bot or other client application. They map the intention of the user's text to a LUIS intent.

For this example application's `FindForm` intent, example utterances will include the form number. The client application needs the form number to fulfill the user's request, so it's important to include it in the utterance.

The screenshot shows the Microsoft Bot Framework Emulator interface. On the left, there's a sidebar with 'App Assets' expanded, showing 'Intents' selected. The main area is titled 'FindForm' with a blue edit icon. Below it, 'Labelled entities:' is listed. A toolbar at the top includes 'Edit', 'Reassign intent', '+ Add as pattern', 'Delete utterance(s)', 'Search', 'Filter', and 'View options'. Under 'Example utterance', there's a text input field containing 'Looking for hrf-123456', which is highlighted with a red border. Below the input field, it says 'No items created'.

Add the following 15 example utterances to the `FindForm` intent.

#	EXAMPLE UTTERANCES
1	Looking for hrf-123456
2	Where is the human resources form hrf-234591?
3	hrf-345623, where is it
4	Is it possible to send me hrf-345794
5	Do I need hrf-234695 to apply for an internal job?
6	Does my manager need to know I'm applying for a job with hrf-234091
7	Where do I send hrf-234918? Do I get an email response if it was received?
8	hrf-234555
9	When was hrf-234987 updated?
10	Do I use form hrf-876345 to apply for engineering positions
11	Was a new version of hrf-765234 submitted for my open req?
12	Do I use hrf-234234 for international jobs?
13	hrf-234598 spelling mistake
14	will hrf-234567 be edited for new requirements
15	hrf-123456, hrf-123123, hrf-234567

By design, these example utterances vary in the following ways:

- utterance length
- punctuation
- word choice
- verb tense (is, was, will be)
- word order

Create a regular expression entity

To return the form number in the runtime prediction response, the form must be marked as an entity. Because the form number text is highly structured, you can mark it using a regular expression entity. Create the entity with the following steps:

1. Select **Entities** from the menu on the left.
2. Select **Create new entity** on the **Entities** page.
3. Enter the name `Human Resources Form Number`, select the **Regex** entity type, and enter the regular expression, `hrf-[0-9]{6}`. This entry matches the literal characters, `hrf-`, and allows for exactly 6 digits.

What type of entity do you want to create?

Entity name (Required)
Human Resources Form Number

Entity type (Required)
Regex

Regex (Required)
hrf-[0-9]{6}

A **regex entity** is an entity that matches based on the regular expression defined.
Regex entities are not machine learned entities.

Done **Cancel**

4. Select **Done**.

Add example utterances to the None intent

The **None** intent is the fallback intent and shouldn't be left empty. This intent should contain one utterance for every 10 example utterances that you've added for the other intents of the app.

The **None** intent's example utterances should be outside of your client application domain.

1. Select **Intents** from the left menu, and then select **None** from the intents list.
2. Add the following example utterances to the intent:

NONE INTENT EXAMPLE UTTERANCES	
Barking dogs are annoying	
Order a pizza for me	
Penguins in the ocean	

For this human resources app, these example utterances are outside the domain. If your human resources domain include animals, food, or the ocean, then you should use different example utterances for the **None** intent.

intent.

Train the app

In the menu in the upper right, select **Train** to apply the intent and entity model changes to the current version of the app.

Look at the regular expression entity in the example utterances

1. Verify the entity is found in the **FindForm** intent by selecting **Intents** from the left menu. Then select **FindForm** intent.

The entity is marked where it appears in the example utterances. If you want to see the original text instead of the entity name, toggle **Entities View** from the toolbar.

Utterance	Score
Human Resources Form Number, Human Resources Form Number	1.00
Human Resources Form Number	
will Human Resources Form Number be edited for new requirements	0.99
Human Resources Form Number spelling mistake	0.98
do i use Human Resources Form Number for international jobs ?	0.99
was a new version of Human Resources Form Number submitted for my open req ?	0.99
do i use form Human Resources Form Number to apply for engineering positions	0.99
when was Human Resources Form Number updated ?	0.99
Human Resources Form Number	0.97
where do i send Human Resources Form Number	
? do i get an email response it was received ?	1.00
does my manager need to know i ' m applying for a job with Human Resources Form Number	1.00

Test your new app with the interactive test pane

Use the interactive **Test** pane in the LUIS portal to validate that the entity is extracted from new utterances the app hasn't seen yet.

1. Select **Test** from the upper-right menu.
2. Add a new utterance and then press Enter:

```
Is there a form named href-234098
```

The screenshot shows the LUIS Inspect interface. In the 'Utterance' section, the input 'is there a form named hrf-234098' is displayed. Below it, the 'Top scoring intent' is listed as 'FindForm (0.977)' with an 'Edit' link. A red box highlights this intent. In the 'Entities' section, 'Human Resources Form Number --> hrf - 234098' is listed. Another red box highlights this entity extraction. The 'Top matched pattern' section shows 'No matched patterns'. The 'Sentiment' section has a note to 'Enable sentiment analysis to get sentiment score'.

The top predicted intent is correctly **FindForm** with over 90% confidence (0.977). The **Human Resources Form Number** entity is extracted with a value of hrf-234098.

Clean up resources

When you're done with this quickstart and aren't moving on to the next quickstart, select **My apps** from the top navigation menu. Then select the app's left check box from the list and select **Delete** from the context toolbar above the list.

My Apps			
Rename		Export	Import container endpoint logs
Export endpoint logs		Delete	Search apps
Name	Culture	Created date	Endpoint hits
<input checked="" type="checkbox"/> myEnglishApp (V 0.1)	en-us	2/28/19	0
<input type="checkbox"/> HumanResources (V batchtest)	en-us	2/26/19	2
<input type="checkbox"/> DND-IoT-Lights-FULL (V 0.1)	en-us	1/25/18	0

Next steps

2. Deploy an app

Quickstart: Deploy an app in the LUIS portal

7/26/2019 • 3 minutes to read • [Edit Online](#)

When your LUIS app is ready to return utterance predictions to a client application (for example, a chat bot), you need to deploy the app to the prediction endpoint.

In this quickstart, you learn to deploy an application. You create a prediction endpoint resource, assign the resource to the app, train the app, and publish the app.

Prerequisites

- Get an [Azure subscription](#).
- Complete the [previous portal quickstart](#) or [download and import the app](#).

Create the endpoint resource

You create the prediction endpoint resource in the Azure portal. This resource should only be used for endpoint prediction queries. Do not use this resource for authoring changes to the app.

1. Sign in to the [Azure portal](#).
2. Select the green + sign in the upper-left panel. Search for `Cognitive Services` in the marketplace and select it.
3. Configure the subscription with the following settings:

SETTING	VALUE	PURPOSE
Name	<code>my-cognitive-service-resource</code>	The name of the Azure resource. You need this name when you assign the resource to the app in the LUIS portal.
Subscription	Your subscription	Select one of the subscriptions associated with your account.
Location	West US	The Azure region for this resource.
Pricing tier	S0	The default pricing tier for this resource.
Resource group	<code>my-cognitive-service-resource-group</code>	Create a new resource group for all your cognitive service resources. When you're done with the resources, you can delete the resource group to clean up your subscription.

Create

All Cognitive Services

* Name
my-cognitive-service-resource ✓

* Subscription
documentationteam

* Location
West US

Location specifies the region only for included regional services. This does not specify a region for included non-regional services. [See service status page](#) for more details.

* Pricing tier ([View full pricing details](#))
S0

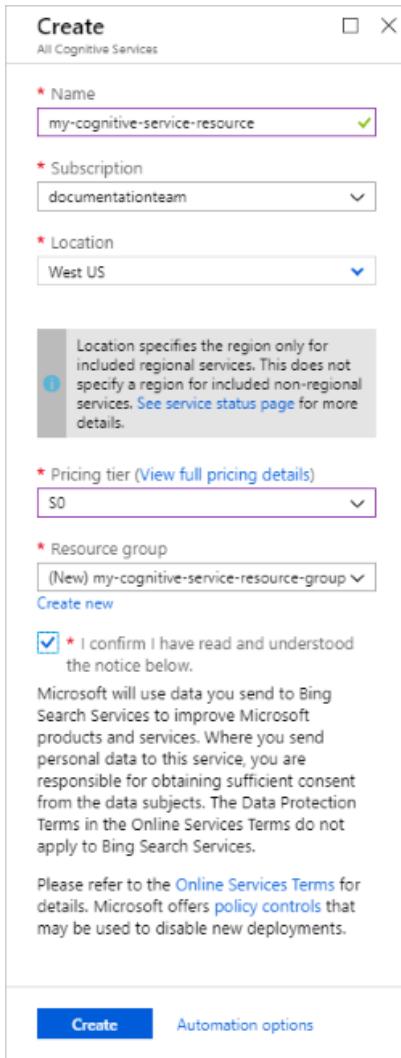
* Resource group
(New) my-cognitive-service-resource-group ✓
[Create new](#)

* I confirm I have read and understood the notice below.

Microsoft will use data you send to Bing Search Services to improve Microsoft products and services. Where you send personal data to this service, you are responsible for obtaining sufficient consent from the data subjects. The Data Protection Terms in the Online Services Terms do not apply to Bing Search Services.

Please refer to the [Online Services Terms](#) for details. Microsoft offers [policy controls](#) that may be used to disable new deployments.

Create [Automation options](#)



4. Select **Create** to create the Azure resource.

In the next section, you learn how to connect this new resource to a LUIS app in the LUIS portal.

Assign the resource key to the LUIS app in the LUIS portal

Every time you create a new resource for LUIS, you need to assign the resource to the LUIS app. After it's assigned, you won't need to do this step again unless you create a new resource. You might create a new resource to expand the regions of your app or to support a higher number of prediction queries.

1. Sign in to the [LUIS portal](#) and choose the **myEnglishApp** app from the apps list.
2. Select **Manage** in the upper-right menu, and then select **Keys and endpoints**.
3. To add the LUIS, select **Assign Resource +**.

4. Select your tenant, subscription, and resource name. Select **Assign resource**.

5. Find the new row in the table and copy the endpoint URL. It's correctly constructed to make an **HTTP GET** request to the LUIS API endpoint for a prediction.

Train and publish the app

Train the app when you're ready to test it. Publish the app when you want the currently trained version to be available to client applications from the query prediction endpoint runtime.

1. If the app is untrained, select **Train** from the menu in the upper right.
2. Select **Publish** from the top menu. Accept the default environment settings, and select **Publish**.
3. When the green success notification bar appears at the top of the browser window, select **Refer to the list of endpoints**.

✓ Publishing complete. **Refer to the list of endpoints** to access your endpoint URL
4. On the **Keys and Endpoint settings** page, find the list of assigned resources and corresponding endpoint URLs at the bottom.
5. Select the endpoint URL associated with your new resource name. This action opens a web browser with a

correctly constructed URL to make a `GET` request to the prediction endpoint runtime.

6. The `q=` at the end of the URL is short for **query** and is where the user's utterance is appended to the GET request. After the `q=`, enter the same user utterance used at the end of the previous quickstart:

```
Is there a form named hrf-234098
```

The browser shows the response, which is the same JSON your client application will receive:

```
{
  "query": "Is there a form named hrf-234098",
  "topScoringIntent": {
    "intent": "FindForm",
    "score": 0.9768753
  },
  "intents": [
    {
      "intent": "FindForm",
      "score": 0.9768753
    },
    {
      "intent": "None",
      "score": 0.0216071066
    }
  ],
  "entities": [
    {
      "entity": "hrf-234098",
      "type": "Human Resources Form Number",
      "startIndex": 22,
      "endIndex": 31
    }
  ]
}
```

This response gives you more information than the default test pane in the previous tutorial. To see this same level of information in the test pane, you must publish the app. After the app is published, select **Compare with published** in the test pane. Use **Show JSON view** in the published test pane to see the same JSON as the previous step. In this way, you can compare the current app you're working on with an app that is published to the endpoint.

The screenshot shows three panels: Test, Inspect, and Additional Settings.

Test Panel: Shows the input "is there a form named hrf-234098" and the result "FindForm (0.977)" with an "Inspect" button.

Inspect Panel: Shows the utterance "is there a form named hrf-234098".

Additional Settings Panel: Shows the published status as "production" (version 0.1).

Test Panel	Inspect Panel	Additional Settings
Type a test utterance	Utterance is there a form named hrf-234098	Published: production version 0.1
is there a form named hrf-234098	Top scoring intent FindForm (0.977) Edit	Utterance is there a form named hrf-234098
FindForm (0.977)	Entities Human Resources Form Number --> hrf - 234098	Top scoring intent FindForm (0.977)
Inspect	Top matched pattern No matched patterns	Entities Human Resources Form Number --> hrf-234098
	Sentiment Enable sentiment analysis to get sentiment score	Top matched pattern No matched patterns
	Sentiment Enable sentiment analysis to get sentiment score	Sentiment Enable sentiment analysis to get sentiment score

Clean up resources

When you're done with this quickstart, select **My apps** from the top navigation menu. Select the app's check box from the list, and then select **Delete** from the context toolbar above the list.

The screenshot shows the "My Apps" page with three listed applications:

Name	Culture	Created date	Endpoint hits
<input checked="" type="checkbox"/> myEnglishApp (V 0.1)	en-us	2/28/19	0
<input type="checkbox"/> HumanResources (V batchtest)	en-us	2/26/19	2
<input type="checkbox"/> DND-IoT-Lights-FULL (V 0.1)	en-us	1/25/18	0

A red box highlights the "Delete" button in the toolbar above the table.

Next steps

[Identify common intents and entities](#)

Quickstart: Language Understanding (LUIS) authoring client library for .NET

8/12/2019 • 9 minutes to read • [Edit Online](#)

Get started with the Language Understanding (LUIS) authoring client library for .NET. Follow these steps to install the package and try out the example code for basic tasks. Language Understanding (LUIS) enables you to apply custom machine-learning intelligence to a user's conversational, natural language text to predict overall meaning, and pull out relevant, detailed information.

Use the Language Understanding (LUIS) authoring client library for .NET to:

- Create an app
- Add intents, entities, and example utterances
- Add features such as a phrase list
- Train and publish app

[Reference documentation](#) | [Library source code](#) | [Authoring Package \(NuGet\)](#) | [C# Samples](#)

Prerequisites

- Language Understanding (LUIS) portal account - [Create one for free](#)
- The current version of [.NET Core](#).

Setting up

Get your Language Understanding (LUIS) authoring key

Get your [authoring key](#), and [create an environment variable](#) for the key, named `COGNITIVESERVICE_AUTHORIZING_KEY`.

Create a new C# application

Create a new .NET Core application in your preferred editor or IDE.

1. In a console window (such as cmd, PowerShell, or Bash), use the `dotnet new` command to create a new console app with the name `language-understanding-quickstart`. This command creates a simple "Hello World" C# project with a single source file: `Program.cs`.

```
dotnet new console -n language-understanding-quickstart
```

2. Change your directory to the newly created app folder.

3. You can build the application with:

```
dotnet build
```

The build output should contain no warnings or errors.

```
...  
Build succeeded.  
0 Warning(s)  
0 Error(s)  
...
```

Install the SDK

Within the application directory, install the Language Understanding (LUIS) authoring client library for .NET with the following command:

```
dotnet add package Microsoft.Azure.CognitiveServices.Language.LUIS.Authoring --version 3.0.0
```

If you're using the Visual Studio IDE, the client library is available as a downloadable NuGet package.

Object model

The Language Understanding (LUIS) authoring client is a [LUISAuthoringClient](#) object that authenticates to Azure, which contains your authoring key.

Once the client is created, use this client to access functionality including:

- Apps - [create, delete, publish](#)
- Example utterances - [add by batch, delete by ID](#)
- Features - manage [phrase lists](#)
- Model - manage [intents](#) and entities
- Pattern - manage [patterns](#)
- Train - [train](#) the app and poll for [training status](#)
- [Versions](#) - manage with clone, export, and delete

Code examples

These code snippets show you how to do the following with the Language Understanding (LUIS) authoring client library for .NET:

- [Create an app](#)
- [Add entities](#)
- [Add intents](#)
- [Add example utterances](#)
- [Train the app](#)
- [Publish the app](#)

Add the dependencies

From the project directory, open the **Program.cs** file in your preferred editor or IDE. Replace the existing `using` code with the following `using` directives:

```
using Microsoft.Azure.CognitiveServices.Language.LUIS.Authoring;
using Microsoft.Azure.CognitiveServices.Language.LUIS.Authoring.Models;

using System;
using System.Collections.Generic;
using System.Linq;
using System.Security;
using System.Threading.Tasks;
```

Authenticate the client

1. Create variables to manage your authoring key pulled from an environment variable named `COGNITIVESERVICES_AUTHORIZING_KEY`. If you created the environment variable after the application is launched, the editor, IDE, or shell running it will need to be closed and reloaded to access the variable. The methods will be created later.
2. Create variables to hold your authoring region and endpoint. The region of your authoring key depends on where you are authoring. The [three authoring regions](#) are:
 - Australia - `australiaeast`
 - Europe - `westeurope`
 - U.S. and other regions - `westus` (Default)

```
private const string key_var = "COGNITIVESERVICE_AUTHORIZING_KEY";
private static readonly string authoring_key = Environment.GetEnvironmentVariable(key_var);

// Note you must use the same region as you used to get your authoring key.
private const string region_var = "COGNITIVESERVICE_REGION";
private static readonly string region = Environment.GetEnvironmentVariable(region_var);
private static readonly string endpoint = "https://" + region + ".api.cognitive.microsoft.com";
```

3. Create an [ApiKeyServiceClientCredentials](#) object with your key, and use it with your endpoint to create an [LUISAuthoringClient](#) object.

```
// Generate the credentials and create the client.
var credentials = new
Microsoft.Azure.CognitiveServices.Language.LUIS.Authoring.ApiKeyServiceClientCredentials(authoring_key);
var client = new LUISAuthoringClient(credentials, new System.Net.Http.DelegatingHandler[] { })
{
    Endpoint = "https://" + region + ".api.cognitive.microsoft.com"
};
```

Create a LUIS app

1. Create a LUIS app to contain the natural language processing (NLP) model holding intents, entities, and example utterances.
2. Create a [ApplicationCreateObject](#). The name and language culture are required properties.
3. Call the [Apps.AddAsync](#) method. The response is the app ID.

```

// Return the application ID and version.
async static Task<ApplicationInfo> CreateApplication(LUISAuthoringClient client)
{
    string app_name = String.Format("Contoso {0}", DateTime.Now);
    string app_description = "Flight booking app built with LUIS .NET SDK.";
    string app_version = "0.1";
    string app_culture = "en-us";

    var app_info = new ApplicationCreateObject()
    {
        Name = app_name,
        InitialVersionId = app_version,
        Description = app_description,
        Culture = app_culture
    };
    var app_id = await client.Apps.AddAsync(app_info);
    Console.WriteLine("Created new LUIS application {0}\n with ID {1}.", app_info.Name, app_id);
    return new ApplicationInfo() { ID = app_id, Version = app_version };
}

```

Create intent for the app

The primary object in a LUIS app's model is the intent. The intent aligns with a grouping of user utterance *intentions*. A user may ask a question, or make a statement looking for a particular *intended* response from a bot (or other client application). Examples of intentions are booking a flight, asking about weather in a destination city, and asking about contact information for customer service.

Create a [ModelCreateObject](#) with the name of the unique intent then pass the app ID, version ID, and the [ModelCreateObject](#) to the [Model.AddIntentAsync](#) method. The response is the intent ID.

```

async static Task AddIntents(LUISAuthoringClient client, ApplicationInfo app_info)
{
    await client.Model.AddIntentAsync(app_info.ID, app_info.Version, new ModelCreateObject()
    {
        Name = "FindFlights"
    });
    Console.WriteLine("Created intent FindFlights");
}

```

Create entities for the app

While entities are not required, they are found in most apps. The entity extracts information from the user utterance, necessary to fulfill the user's intention. There are several types of [prebuilt](#) and custom entities, each with their own data transformation object (DTO) models. Common prebuilt entities to add to your app include [number](#), [datetimeV2](#), [geographyV2](#), [ordinal](#).

This **AddEntities** method created a [Location](#) simple entity with two roles, a [Class](#) simple entity, a [Flight](#) composite entity and adds several prebuilt entities.

It is important to know that entities are not marked with an intent. They can and usually do apply to many intents. Only example user utterances are marked for a specific, single intent.

Creation methods for entities are part of the [Model](#) class. Each entity type has its own data transformation object (DTO) model, usually containing the word [model](#) in the [Models](#) namespace.

```

// Create entity objects
async static Task AddEntities(LUISAuthoringClient client, ApplicationInfo app_info)
{
    // Add simple entity
    var simpleEntityIdLocation = await client.Model.AddEntityAsync(app_info.ID, app_info.Version, new
ModelCreateObject()
{
    Name = "Location"
});

    // Add 'Origin' role to simple entity
    await client.Model.CreateEntityRoleAsync(app_info.ID, app_info.Version, simpleEntityIdLocation, new
EntityRoleCreateObject()
{
    Name = "Origin"
});

    // Add 'Destination' role to simple entity
    await client.Model.CreateEntityRoleAsync(app_info.ID, app_info.Version, simpleEntityIdLocation, new
EntityRoleCreateObject()
{
    Name = "Destination"
});

    // Add simple entity
    var simpleEntityIdClass = await client.Model.AddEntityAsync(app_info.ID, app_info.Version, new
ModelCreateObject()
{
    Name = "Class"
});

    // Add prebuilt number and datetime
    await client.Model.AddPrebuiltAsync(app_info.ID, app_info.Version, new List<string>
{
    "number",
    "datetimeV2",
    "geographyV2",
    "ordinal"
});

    // Composite entity
    await client.Model.AddCompositeEntityAsync(app_info.ID, app_info.Version, new CompositeEntityModel()
{
    Name = "Flight",
    Children = new List<string>() { "Location", "Class", "number", "datetimeV2", "geographyV2", "ordinal" }
});
    Console.WriteLine("Created entities Location, Class, number, datetimeV2, geographyV2, ordinal.");
}

```

Add example utterance to intent

In order to determine an utterance's intention and extract entities, the app needs examples of utterances. The examples need to target a specific, single intent and should mark all custom entities. Prebuilt entities do not need to be marked.

Add example utterances by creating a list of [ExampleLabelObject](#) objects, one object for each example utterance. Each example should mark all entities with a dictionary of name/value pairs of entity name and entity value. The entity value should be exactly as it appears in the text of the example utterance.

Call [Examples.BatchAsync](#) with the app ID, version ID, and the list of examples. The call responds with a list of results. You need to check each example's result to make sure it was successfully added to the model.

```

async static Task AddUtterances(LUISAuthoringClient client, ApplicationInfo app_info)
{
    var utterances = new List<ExampleLabelObject>()
    {
        CreateUtterance ("FindFlights", "find flights in economy to Madrid on July 1st", new Dictionary<string,
string>() { {"Flight", "economy to Madrid"}, { "Location", "Madrid" }, { "Class", "economy" } }),
        CreateUtterance ("FindFlights", "find flights from seattle to London in first class", new
Dictionary<string, string>() { { "Flight", "London in first class" }, { "Location", "London" }, { "Class",
"first" } }),
        CreateUtterance ("FindFlights", "find flights to London in first class", new Dictionary<string, string>
() { { "Flight", "London in first class" }, { "Location", "London" }, { "Class", "first" } }),

        //Role not supported in SDK yet
        //CreateUtterance ("FindFlights", "find flights to Paris in first class", new Dictionary<string,
string>() { { "Flight", "London in first class" }, { "Location::Destination", "Paris" }, { "Class", "first" }
})
    };
    var resultsList = await client.Examples.BatchAsync(app_info.ID, app_info.Version, utterances);

    foreach (var x in resultsList)
    {
        var result = (!x.HasError.GetValueOrDefault()) ? "succeeded": "failed";
        Console.WriteLine("{0} {1}", x.Value.ExampleId, result);
    }
}
// Create utterance with marked text for entities
static ExampleLabelObject CreateUtterance(string intent, string utterance, Dictionary<string, string> labels)
{
    var entity_labels = labels.Select(kv => CreateLabel(utterance, kv.Key, kv.Value)).ToList();
    return new ExampleLabelObject()
    {
        IntentName = intent,
        Text = utterance,
        EntityLabels = entity_labels
    };
}
// Mark beginning and ending of entity text in utterance
static EntityLabelObject CreateLabel(string utterance, string key, string value)
{
    var start_index = utterance.IndexOf(value, StringComparison.InvariantCultureIgnoreCase);
    return new EntityLabelObject()
    {
        EntityName = key,
        StartCharIndex = start_index,
        EndCharIndex = start_index + value.Length
    };
}

```

The **CreateUtterance** and **CreateLabel** methods are utility methods to help you create objects.

Train the app

Once the model is created, the LUIS app needs to be trained for this version of the model. A trained model can be used in a [container](#), or [published](#) to the staging or product slots.

The [Train.TrainVersionAsync](#) method needs the app ID and the version ID.

A very small model, such as this quickstart shows, will train very quickly. For production-level applications, training the app should include a polling call to the [GetStatusAsync](#) method to determine when or if the training succeeded. The response is a list of [ModelTrainingInfo](#) objects with a separate status for each object. All objects must be successful for the training to be considered complete.

```
async static Task Train_App(LUISAuthoringClient client, ApplicationInfo app)
{
    var response = await client.Train.TrainVersionAsync(app.ID, app.Version);
    Console.WriteLine("Training status: " + response.Status);
}
```

Publish a Language Understanding app

Publish the LUIS app using the [PublishAsync](#) method. This publishes the current trained version to the specified slot at the endpoint. Your client application uses this endpoint to send user utterances for prediction of intent and entity extraction.

```
// Publish app, display endpoint URL for the published application.
async static Task Publish_App(LUISAuthoringClient client, ApplicationInfo app)
{
    ApplicationPublishObject obj = new ApplicationPublishObject
    {
        VersionId = app.Version,
        IsStaging = true
    };
    var info = await client.Apps.PublishAsync(app.ID, obj);
    Console.WriteLine("Endpoint URL: " + info.EndpointUrl);
}
```

Run the application

Run the application with the dotnet `run` command from your application directory.

```
dotnet run
```

Clean up resources

If you want to clean up, you can delete the LUIS app. Deleting the app is done with the [Apps.DeleteAsync](#) method. You can also delete the app from the [LUIS portal](#).

Next steps

[Use the .Net SDK to query the prediction endpoint](#)

- [What is the Language Understanding \(LUIS\) API?](#)
- [What's new?](#)
- [Intents, entities, and example utterances, and prebuilt entities](#)
- The source code for this sample can be found on [GitHub](#).

Quickstart: Query prediction endpoint with C# .NET SDK

7/26/2019 • 3 minutes to read • [Edit Online](#)

Use the .NET SDK, found on [NuGet](#), to send a user utterance to Language Understanding (LUIS) and receive a prediction of the user's intention.

This quickstart sends a user utterance, such as `turn on the bedroom light`, to a public Language Understanding application, then receives the prediction and displays the top-scoring intent `HomeAutomation.TurnOn` and the entity `HomeAutomation.Room` found within the utterance.

Prerequisites

- [Visual Studio Community 2017 edition](#)
- C# programming language (included with VS Community 2017)
- Public app ID: df67dcdb-c37d-46af-88e1-8b97951ca1c2

NOTE

The complete solution is available from the [cognitive-services-language-understanding](#) GitHub repository.

Looking for more documentation?

- [SDK reference documentation](#)

Get Cognitive Services or Language Understanding key

In order to use the public app for home automation, you need a valid key for endpoint predictions. You can use either a Cognitive Services key (created below with the Azure CLI), which is valid for many cognitive services, or a `Language Understanding` key.

Use the following [Azure CLI command to create a Cognitive Service key](#):

```
az cognitiveservices account create \
-n my-cog-serv-resource \
-g my-cog-serv-resource-group \
--kind CognitiveServices \
--sku S0 \
-l WestEurope \
--yes
```

Create .NET Core project

Create a .NET Core console project in Visual Studio Community 2017.

1. Open Visual Studio Community 2017.
2. Create a new project, from the **Visual C#** section, choose **Console App (.NET Core)**.
3. Enter the project name `QueryPrediction`, leave the remaining default values, and select **OK**. This creates a simple project with the primary code file named **Program.cs**.

Add SDK with NuGet

1. In the **Solution Explorer**, select the Project in the tree view named **QueryPrediction**, then right-click. From the menu, select **Manage NuGet Packages....**
2. Select **Browse** then enter `Microsoft.Azure.CognitiveServices.Language.LUIS.Runtime`. When the package information displays, select **Install** to install the package into the project.
3. Add the following *using* statements to the top of **Program.cs**. Do not remove the existing *using* statement for `System`.

```
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Azure.CognitiveServices.Language.LUIS.Runtime;
using Microsoft.Azure.CognitiveServices.Language.LUIS.Runtime.Models;
```

Create a new method for the prediction

Create a new method, `GetPrediction` to send the query to the query prediction endpoint. The method will create and configure all necessary objects then return a `Task` with the `LuisResult` prediction results.

```
static async Task<LuisResult> GetPrediction() {
}
```

Create credentials object

Add the following code to the `GetPrediction` method to create the client credentials with your Cognitive Service key.

Replace `<REPLACE-WITH-YOUR-KEY>` with your Cognitive Service key's region. The key is in the [Azure portal](#) on the Keys page for that resource.

```
// Use Language Understanding or Cognitive Services key
// to create authentication credentials
var endpointPredictionkey = "<REPLACE-WITH-YOUR-KEY>";
var credentials = new ApiKeyServiceClientCredentials(endpointPredictionkey);
```

Create Language Understanding client

In the `GetPrediction` method, after the preceding code, add the following code to use the new credentials, creating a `LUISRuntimeClient` client object.

Replace `<REPLACE-WITH-YOUR-KEY-REGION>` with your key's region, such as `westus`. The key region is in the [Azure portal](#) on the Overview page for that resource.

```
// Create Luis client and set endpoint
// region of endpoint must match key's region, for example `westus`
var luisClient = new LUISRuntimeClient(credentials, new System.Net.Http.DelegatingHandler[] { });
luisClient.Endpoint = "https://<REPLACE-WITH-YOUR-KEY-REGION>.api.cognitive.microsoft.com";
```

Set query parameters

In the `GetPrediction` method, after the preceding code, add the following code to set the query parameters.

```
// public Language Understanding Home Automation app
var appId = "df67dcdb-c37d-46af-88e1-8b97951ca1c2";

// query specific to home automation app
var query = "turn on the bedroom light";

// common settings for remaining parameters
Double? timezoneOffset = null;
var verbose = true;
var staging = false;
var spellCheck = false;
String bingSpellCheckKey = null;
var log = false;
```

Query prediction endpoint

In the `GetPrediction` method, after the preceding code, add the following code to set the query parameters:

```
// Create prediction client
var prediction = new Prediction(luisClient);

// get prediction
return await prediction.ResolveAsync(appId, query, timezoneOffset, verbose, staging, spellCheck,
bingSpellCheckKey, log, CancellationToken.None);
```

Display prediction results

Change the **Main** method to call the new `GetPrediction` method and return the prediction results:

```
static void Main(string[] args)
{
    var luisResult = GetPrediction().Result;

    // Display query
    Console.WriteLine("Query:'{0}'", luisResult.Query);

    // Display most common properties of query result
    Console.WriteLine("Top intent is '{0}' with score {1}",
luisResult.TopScoringIntent.Intent,luisResult.TopScoringIntent.Score);

    // Display all entities detected in query utterance
    foreach (var entity in luisResult.Entities)
    {
        Console.WriteLine("{0}:'{1}' begins at position {2} and ends at position {3}", entity.Type,
entity.Entity, entity.StartIndex, entity.EndIndex);
    }

    Console.Write("done");
}
```

Run the project

Build the project in Studio and run the project to see the output of the query:

```
Query:'turn on the bedroom light'
Top intent is 'HomeAutomation.TurnOn' with score 0.809439957
HomeAutomation.Room:'bedroom' begins at position 12 and ends at position 18
```

Next steps

Learn more about the [.NET SDK](#) and the [.NET reference documentation](#).

[Tutorial: Build LUIS app to determine user intentions](#)

Quickstart: Language Understanding (LUIS) authoring client library for Python

8/9/2019 • 6 minutes to read • [Edit Online](#)

Get started with the Language Understanding (LUIS) authoring client library for python. Follow these steps to install the package and try out the example code for basic tasks. Language Understanding (LUIS) enables you to apply custom machine-learning intelligence to a user's conversational, natural language text to predict overall meaning, and pull out relevant, detailed information.

Use the Language Understanding (LUIS) authoring client library for Python to:

- Create an app.
- Add intents, entities, and example utterances.
- Add features, such as a phrase list.
- Train and publish an app.

[Reference documentation](#) | [Library source code](#) | [Authoring Package \(Pypi\)](#) | [Samples](#)

Prerequisites

- Language Understanding (LUIS) portal account: [Create one for free](#).
- [Python 3.x](#)

Setting up

Get your Language Understanding (LUIS) authoring key

Get your [authoring key](#), and [create an environment variable](#) for the key, named `LUIS_AUTHORIZING_KEY` and an environment variable for the region of the key, `LUIS_REGION`.

Install the Python library for LUIS

Within the application directory, install the Language Understanding (LUIS) authoring client library for python with the following command:

```
pip install azure-cognitiveservices-language-luis
```

Object model

The Language Understanding (LUIS) authoring client is a [LUISAuthoringClient](#) object that authenticates to Azure, which contains your authoring key.

Once the client is created, use this client to access functionality including:

- Apps - [create](#), [delete](#), [publish](#)
- Example utterances - [add by batch](#), [delete by ID](#)
- Features - manage [phrase lists](#)
- Model - manage [intents](#) and entities
- Pattern - manage [patterns](#)
- Train - [train](#) the app and poll for [training status](#)

- [Versions](#) - manage with clone, export, and delete

Code examples

These code snippets show you how to do the following with the Language Understanding (LUIS) authoring client library for python:

- [Create an app](#)
- [Add entities](#)
- [Add intents](#)
- [Add example utterances](#)
- [Train the app](#)
- [Publish the app](#)

Create a new python application

Create a new Python application in your preferred editor or IDE. Then import the following libraries.

```
from azure.cognitiveservices.language.luis.authoring import LUISAuthoringClient
from msrest.authentication import CognitiveServicesCredentials

import datetime, json, os, time
```

Create variables for your resource's Azure endpoint and key. If you created the environment variable after you launched the application, you will need to close and reopen the editor, IDE, or shell running it to access the variable.

```
key_var_name = 'LUIS_AUTHORING_KEY'
if not key_var_name in os.environ:
    raise Exception('Please set/export the environment variable: {}'.format(key_var_name))
authoring_key = os.environ[key_var_name]

region_var_name = 'LUIS_REGION'
if not region_var_name in os.environ:
    raise Exception('Please set/export the environment variable: {}'.format(region_var_name))
region = os.environ[region_var_name]
endpoint = "https://{}.api.cognitive.microsoft.com".format(region)
```

Authenticate the client

Create an [CognitiveServicesCredentials](#) object with your key, and use it with your endpoint to create an [LUISAuthoringClient](#) object.

```
# Instantiate a LUIS client
client = LUISAuthoringClient(endpoint, CognitiveServicesCredentials(authoring_key))
```

Create a LUIS app

1. Create a LUIS app to contain the natural language processing (NLP) model holding intents, entities, and example utterances.
2. Create a [AppsOperation](#) object's [add](#) method to create the app. The name and language culture are required properties.

```

def create_app():
    # Create a new LUIS app
    app_name      = "Contoso {}".format(datetime.datetime.now())
    app_desc      = "Flight booking app built with LUIS Python SDK."
    app_version   = "0.1"
    app_locale    = "en-us"

    app_id = client.apps.add(dict(name=app_name,
                                   initial_version_id=app_version,
                                   description=app_desc,
                                   culture=app_locale))

    print("Created LUIS app {} with ID {}".format(app_name, app_id))
    return app_id, app_version

```

Create intent for the app

The primary object in a LUIS app's model is the intent. The intent aligns with a grouping of user utterance *intentions*. A user may ask a question, or make a statement looking for a particular *intended* response from a bot (or other client application). Examples of intentions are booking a flight, asking about weather in a destination city, and asking about contact information for customer service.

Use the [model.add_intent](#) method with the name of the unique intent then pass the app ID, version ID, and new intent name.

```

def add_intents(app_id, app_version):
    intentId = client.model.add_intent(app_id, app_version, "FindFlights")

    print("Intent FindFlights {} added.".format(intentId))

```

Create entities for the app

While entities are not required, they are found in most apps. The entity extracts information from the user utterance, necessary to fulfill the user's intention. There are several types of [prebuilt](#) and custom entities, each with their own data transformation object (DTO) models. Common prebuilt entities to add to your app include [number](#), [datetimeV2](#), [geographyV2](#), [ordinal](#).

This **add_entities** method created a `Location` simple entity with two roles, a `Class` simple entity, a `Flight` composite entity and adds several prebuilt entities.

It is important to know that entities are not marked with an intent. They can and usually do apply to many intents. Only example user utterances are marked for a specific, single intent.

Creation methods for entities are part of the [ModelOperations](#) class. Each entity type has its own data transformation object (DTO) model.

```

def add_entities(app_id, app_version):

    locationEntityId = client.model.add_entity(app_id, app_version, "Location")
    print("locationEntityId {} added.".format(locationEntityId))

    originRoleId = client.model.create_entity_role(app_id, app_version, locationEntityId, "Origin")
    print("originRoleId {} added.".format(originRoleId))

    destinationRoleId = client.model.create_entity_role(app_id, app_version, locationEntityId, "Destination")
    print("destinationRoleId {} added.".format(destinationRoleId))

    classEntityId = client.model.add_entity(app_id, app_version, name="Class")
    print("classEntityId {} added.".format(classEntityId))

    client.model.add_prewritten(app_id, app_version, prewritten_extractor_names=["number", "datetimeV2",
    "geographyV2", "ordinal"])

    compositeEntityId = client.model.add_composite_entity(app_id, app_version, name="Flight",
        children=["Location", "Class", "number", "datetimeV2", "geographyV2",
    "ordinal"])
    print("compositeEntityId {} added.".format(compositeEntityId))

```

Add example utterance to intent

In order to determine an utterance's intention and extract entities, the app needs examples of utterances. The examples need to target a specific, single intent and should mark all custom entities. Prebuilt entities do not need to be marked.

Add example utterances by creating a list of [ExampleLabelObject](#) objects, one object for each example utterance. Each example should mark all entities with a dictionary of name/value pairs of entity name and entity value. The entity value should be exactly as it appears in the text of the example utterance.

Call [examples.batch](#) with the app ID, version ID, and the list of examples. The call responds with a list of results. You need to check each example's result to make sure it was successfully added to the model.

```

def add_utterances(app_id, app_version):
    # Now define the utterances
    utterances = [create_utterance("FindFlights", "find flights in economy to Madrid",
        ("Flight", "economy to Madrid"),
        ("Location", "Madrid"),
        ("Class", "economy")),

        create_utterance("FindFlights", "find flights to London in first class",
        ("Flight", "London in first class"),
        ("Location", "London"),
        ("Class", "first")),

        create_utterance("FindFlights", "find flights from seattle to London in first class",
        ("Flight", "flights from seattle to London in first class"),
        ("Location", "London"),
        ("Location", "Seattle"),
        ("Class", "first"))]

    # Add the utterances in batch. You may add any number of example utterances
    # for any number of intents in one call.
    client.examples.batch(app_id, app_version, utterances)
    print("{} example utterance(s) added.".format(len(utterances)))

```

Train the app

Once the model is created, the LUIS app needs to be trained for this version of the model. A trained model can be used in a [container](#), or [published](#) to the staging or product slots.

The [train.train_version](#) method needs the app ID and the version ID.

A very small model, such as this quickstart shows, will train very quickly. For production-level applications, training the app should include a polling call to the [get_status](#) method to determine when or if the training succeeded. The response is a list of [ModelTrainingInfo](#) objects with a separate status for each object. All objects must be successful for the training to be considered complete.

```
def train_app(app_id, app_version):
    response = client.train.train_version(app_id, app_version)
    waiting = True
    while waiting:
        info = client.train.get_status(app_id, app_version)

        # get_status returns a list of training statuses, one for each model. Loop through them and make sure
        all are done.
        waiting = any(map(lambda x: 'Queued' == x.details.status or 'InProgress' == x.details.status, info))
        if waiting:
            print ("Waiting 10 seconds for training to complete...")
            time.sleep(10)
```

Publish a Language Understanding app

Publish the LUIS app using the [app.publish](#) method. This publishes the current trained version to the specified slot at the endpoint. Your client application uses this endpoint to send user utterances for prediction of intent and entity extraction.

```
def publish_app(app_id, app_version):
    response = client.apps.publish(app_id, app_version, is_staging=True)
    print("Application published. Endpoint URL: " + response.endpoint_url)
```

Run the application

Run the application with the `python` command on your quickstart file.

```
python quickstart-file.py
```

Clean up resources

If you want to clean up and remove a Cognitive Services subscription, you can delete the resource or resource group. Deleting the resource group also deletes any other resources associated with it.

- [Portal](#)
- [Azure CLI](#)

Next steps

[Build a LUIS app to determine user intentions](#)

- [What is the Language Understanding \(LUIS\) API?](#)
- [What's new?](#)
- [Intents, entities, and example utterances, and prebuilt entities](#)

- The source code for this sample can be found on [GitHub](#).

Quickstart: Get intent using C#

7/26/2019 • 4 minutes to read • [Edit Online](#)

In this quickstart, use an available public LUIS app to determine a user's intention from conversational text. Send the user's intention as text to the public app's HTTP prediction endpoint. At the endpoint, LUIS applies the public app's model to analyze the natural language text for meaning, determining overall intent and extracting data relevant to the app's subject domain.

This quickstart uses the endpoint REST API. For more information, see the [endpoint API documentation](#).

For this article, you need a free [LUIS](#) account.

Prerequisites

- [Visual Studio Community 2017 edition](#)
- C# programming language (included with VS Community 2017)
- Public app ID: df67dcdb-c37d-46af-88e1-8b97951ca1c2

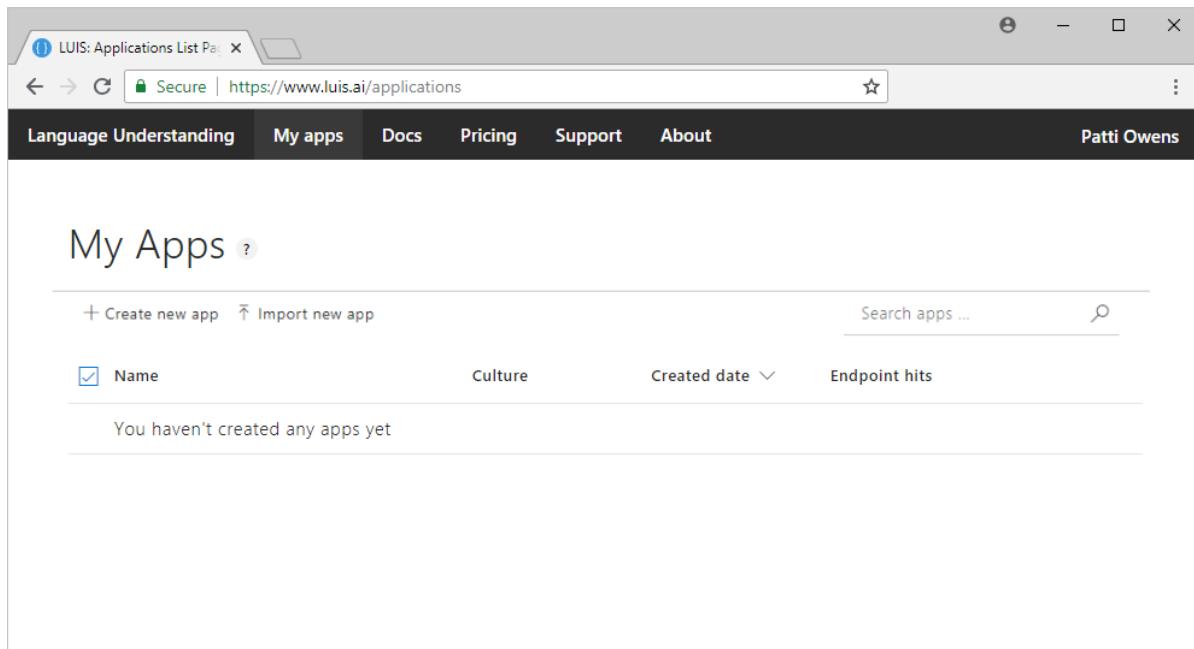
NOTE

The complete solution is available from the [cognitive-services-language-understanding](#) GitHub repository.

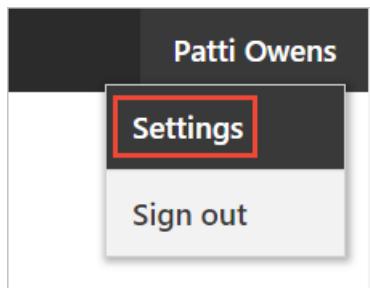
Get LUIS key

Access to the prediction endpoint is provided with an endpoint key. For the purposes of this quickstart, use the free starter key associated with your LUIS account.

1. Sign in using your LUIS account.



2. Select your name in the top right menu, then select **Settings**.



3. Copy the value of the **Authoring key**. You will use it later in the quickstart.

Language Understanding My apps Docs Pricing Support About Patti Owens

User settings

Authoring Key

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx (Reset) ?

Other settings

Country (Required)

United States

Company

Microsoft

The authoring key allows free unlimited requests to the authoring API and up to 1000 queries to the prediction endpoint API per month for all your LUIS apps.

Get intent with browser

To understand what a LUIS prediction endpoint returns, view a prediction result in a web browser. In order to query a public app, you need your own key and the app ID. The public IoT app ID, `df67dcdb-c37d-46af-88e1-8b97951ca1c2`, is provided as part of the URL in step one.

The format of the URL for a **GET** endpoint request is:

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?subscription-key=<YOUR-KEY>&q=<user-utterance>
```

1. The endpoint of the public IoT app is in this format:

```
https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951ca1c2?  
subscription-key=<YOUR_KEY>&q=turn on the bedroom light
```

Copy the URL and substitute your key for the value of `<YOUR_KEY>`.

2. Paste the URL into a browser window and press Enter. The browser displays a JSON result that indicates that LUIS detects the `HomeAutomation.TurnOn` intent as the top intent and the `HomeAutomation.Room` entity with the value `bedroom`.

```
{  
  "query": "turn on the bedroom light",  
  "topScoringIntent": {  
    "intent": "HomeAutomation.TurnOn",  
    "score": 0.809439957  
  },  
  "entities": [  
    {  
      "entity": "bedroom",  
      "type": "HomeAutomation.Room",  
      "startIndex": 12,  
      "endIndex": 18,  
      "score": 0.8065475  
    }  
  ]  
}
```

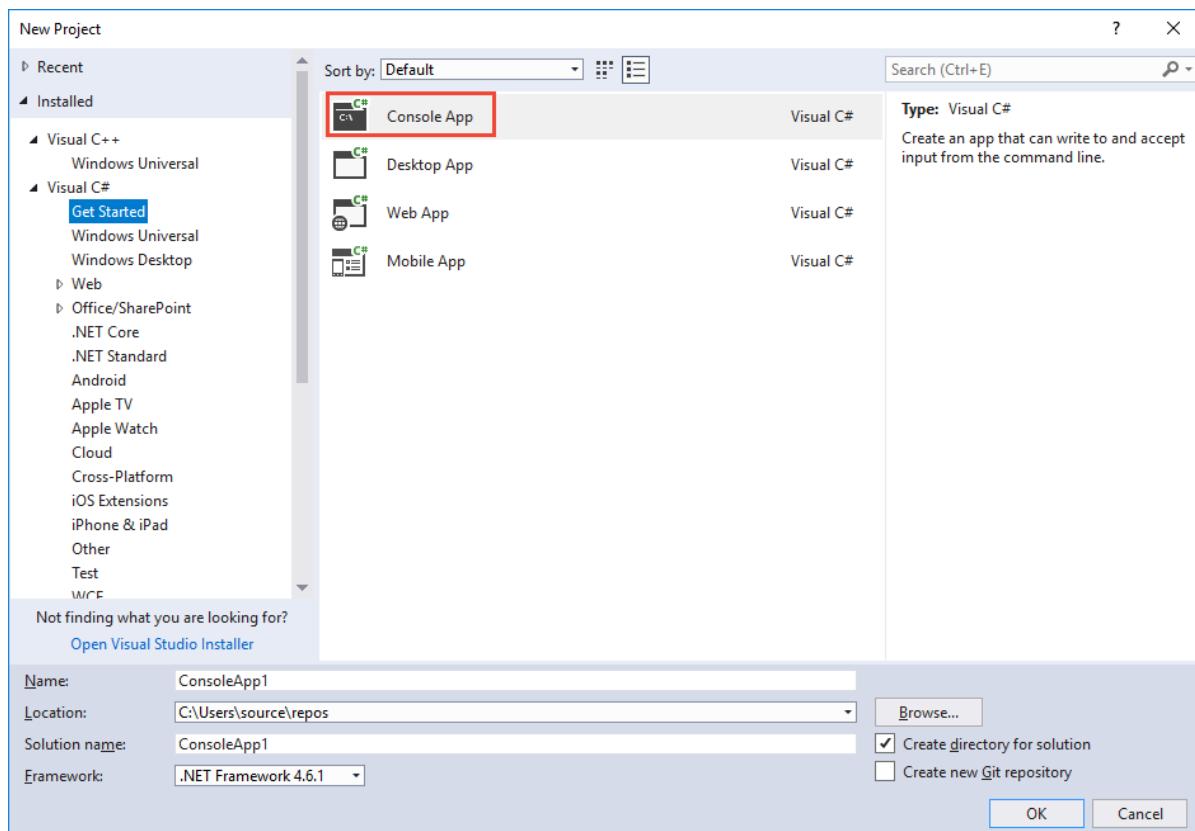
3. Change the value of the `q=` parameter in the URL to `turn off the living room light`, and press Enter. The result now indicates that LUIS detected the `HomeAutomation.TurnOff` intent as the top intent and the `HomeAutomation.Room` entity with value `living room`.

```
{  
  "query": "turn off the living room light",  
  "topScoringIntent": {  
    "intent": "HomeAutomation.TurnOff",  
    "score": 0.984057844  
  },  
  "entities": [  
    {  
      "entity": "living room",  
      "type": "HomeAutomation.Room",  
      "startIndex": 13,  
      "endIndex": 23,  
      "score": 0.9619945  
    }  
  ]  
}
```

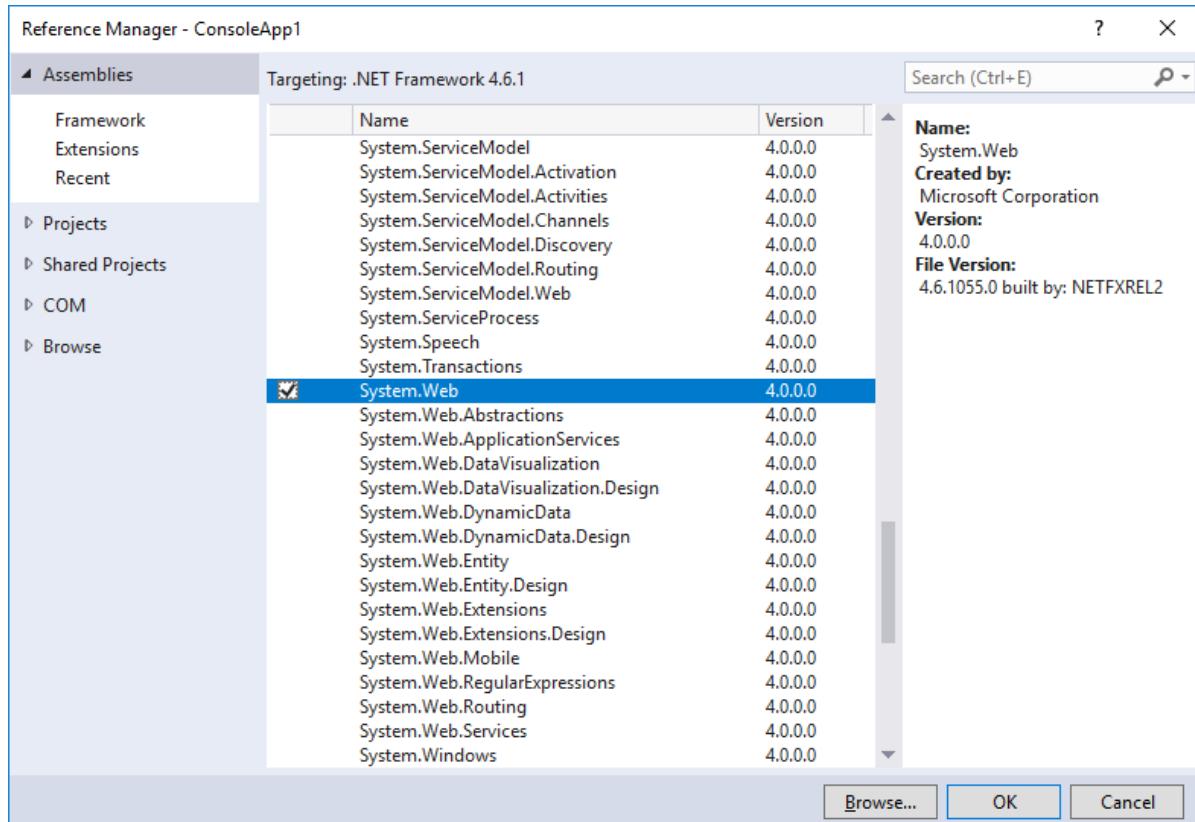
Get intent programmatically

Use C# to query the prediction endpoint GET [API](#) to get the same results as you saw in the browser window in the previous section.

1. Create a new console application in Visual Studio.



2. In the Visual Studio project, in the Solutions Explorer, select **Add reference**, then select **System.Web** from the Assemblies tab.



3. Overwrite Program.cs with the following code:

```

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

/*
You can use the authoring key instead of the endpoint key.
The authoring key allows 1000 endpoint queries a month.

*/

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

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

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

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

            // 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";

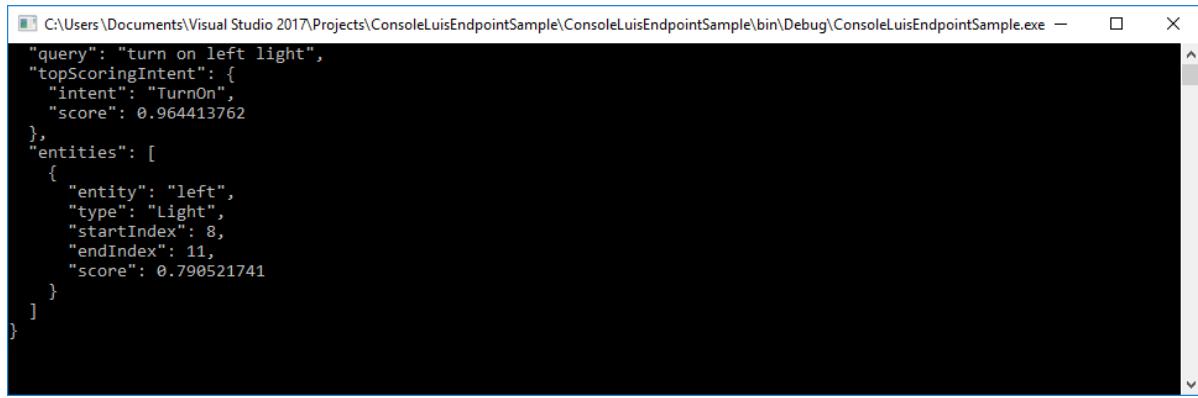
            var endpointUri = "https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/" + luisAppId
+ "?" + queryString;
            var response = await client.GetAsync(endpointUri);

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

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

```

4. Replace the value of `YOUR_KEY` with your LUIS key.
5. Build and 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
    }
  ]
}
```

LUIS keys

This quickstart uses the authoring key for convenience. The key is primarily for authoring the model but does allow a small number (1000) of endpoint requests. When you are ready for more endpoint requests in a test, stage or production environment, create a **Language Understanding** resource in the Azure portal and assign it to the LUIS app in the LUIS portal.

Clean up resources

When you are finished with this quickstart, close the Visual Studio project and remove the project directory from the file system.

Next steps

[Add utterances and train with C#](#)

Quickstart: Get intent using Go

7/26/2019 • 4 minutes to read • [Edit Online](#)

In this quickstart, pass utterances to a LUIS endpoint and get intent and entities back.

In this quickstart, use an available public LUIS app to determine a user's intention from conversational text. Send the user's intention as text to the public app's HTTP prediction endpoint. At the endpoint, LUIS applies the public app's model to analyze the natural language text for meaning, determining overall intent and extracting data relevant to the app's subject domain.

This quickstart uses the endpoint REST API. For more information, see the [endpoint API documentation](#).

For this article, you need a free [LUIS](#) account.

Prerequisites

- [Go](#) programming language
- [Visual Studio Code](#)
- Public app ID: df67dcdb-c37d-46af-88e1-8b97951ca1c2

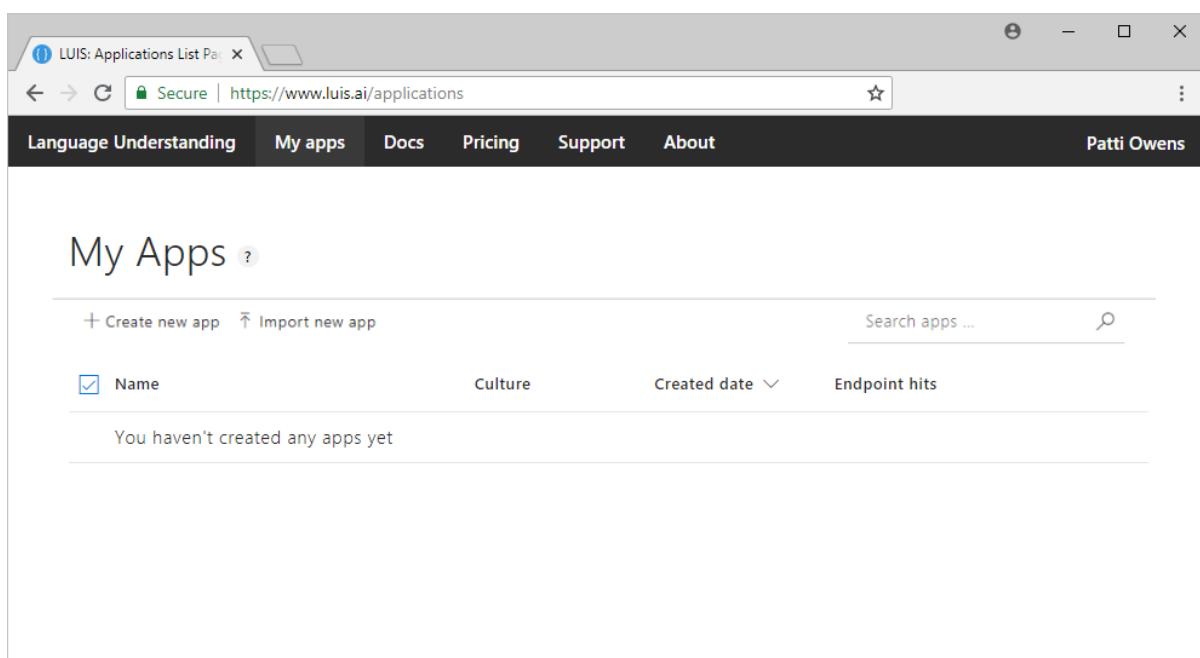
NOTE

The complete solution is available from the [cognitive-services-language-understanding](#) GitHub repository.

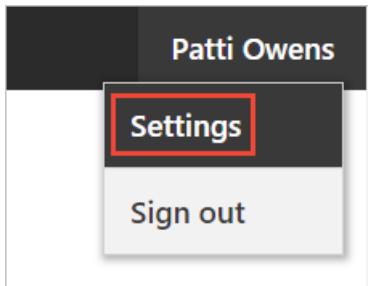
Get LUIS key

Access to the prediction endpoint is provided with an endpoint key. For the purposes of this quickstart, use the free starter key associated with your LUIS account.

1. Sign in using your LUIS account.



2. Select your name in the top right menu, then select **Settings**.



3. Copy the value of the **Authoring key**. You will use it later in the quickstart.

Language Understanding My apps Docs Pricing Support About Patti Owens

User settings

Authoring Key

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx (Reset) ?

Other settings

Country (Required)
United States

Company
Microsoft

The authoring key allows free unlimited requests to the authoring API and up to 1000 queries to the prediction endpoint API per month for all your LUIS apps.

Get intent with browser

To understand what a LUIS prediction endpoint returns, view a prediction result in a web browser. In order to query a public app, you need your own key and the app ID. The public IoT app ID, `df67dcdb-c37d-46af-88e1-8b97951ca1c2`, is provided as part of the URL in step one.

The format of the URL for a **GET** endpoint request is:

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?subscription-key=<YOUR-KEY>&q=<user-utterance>
```

1. The endpoint of the public IoT app is in this format:

```
https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951ca1c2?  
subscription-key=<YOUR_KEY>&q=turn on the bedroom light
```

Copy the URL and substitute your key for the value of `<YOUR_KEY>`.

2. Paste the URL into a browser window and press Enter. The browser displays a JSON result that indicates that LUIS detects the `HomeAutomation.TurnOn` intent as the top intent and the `HomeAutomation.Room` entity with the value `bedroom`.

```
{  
  "query": "turn on the bedroom light",  
  "topScoringIntent": {  
    "intent": "HomeAutomation.TurnOn",  
    "score": 0.809439957  
  },  
  "entities": [  
    {  
      "entity": "bedroom",  
      "type": "HomeAutomation.Room",  
      "startIndex": 12,  
      "endIndex": 18,  
      "score": 0.8065475  
    }  
  ]  
}
```

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

```
{  
  "query": "turn off the living room light",  
  "topScoringIntent": {  
    "intent": "HomeAutomation.TurnOff",  
    "score": 0.984057844  
  },  
  "entities": [  
    {  
      "entity": "living room",  
      "type": "HomeAutomation.Room",  
      "startIndex": 13,  
      "endIndex": 23,  
      "score": 0.9619945  
    }  
  ]  
}
```

Get intent programmatically

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

1. Create a new file named `endpoint.go`. Add the following code:

```

package main

/* Do dependencies */
import (
    "fmt"
    "flag"
    "net/http"
    "net/url"
    "io/ioutil"
    "log"
)

/*
Analyze text

appID = public app ID = be402ffc-57f4-4e1f-9c1d-f0d9fa520aa4
endpointKey = Azure Language Understanding key, or Authoring key if it still has quota
region = endpoint region
utterance = text to analyze

*/
func endpointPrediction(appID string, endpointKey string, region string, utterance string) {

    var endpointUrl = fmt.Sprintf("https://%s.api.cognitive.microsoft.com/luis/v2.0/apps/%s?
subscription-key=%s&verbose=false&q=%s", region, appID, endpointKey, url.QueryEscape(utterance))

    response, err := http.Get(endpointUrl)

    // 401 - check value of 'subscription-key' - do not use authoring key!
    if err!=nil {
        // handle error
        fmt.Println("error from Get")
        log.Fatal(err)
    }

    response2, err2 := ioutil.ReadAll(response.Body)

    if err2!=nil {
        // handle error
        fmt.Println("error from ReadAll")
        log.Fatal(err2)
    }

    fmt.Println("response")
    fmt.Println(string(response2))
}

func main() {

    var appID = flag.String("appID", "df67dcdb-c37d-46af-88e1-8b97951ca1c2", "LUIS appID")
    var endpointKey = flag.String("endpointKey", "", "LUIS endpoint key")
    var region = flag.String("region", "", "LUIS app publish region")
    var utterance = flag.String("utterance", "turn on the bedroom light", "utterance to predict")

    flag.Parse()

    fmt.Println("appID has value", *appID)
    fmt.Println("endpointKey has value", *endpointKey)
    fmt.Println("region has value", *region)
    fmt.Println("utterance has value", *utterance)

    endpointPrediction(*appID, *endpointKey, *region, *utterance)

}

```

- With a command prompt in the same directory as where you created the file, enter `go build endpoint.go` to

compile the Go file. The command prompt does not return any information for a successful build.

3. Run the Go application from the command line by entering the following text in the command prompt:

```
go run endpoint.go -appId df67dcdb-c37d-46af-88e1-8b97951ca1c2 -endpointKey <add-your-key> -region westus
```

Replace `<add-your-key>` with the value of your key.

The command prompt response is:

```
appId has value df67dcdb-c37d-46af-88e1-8b97951ca1c2
endpointKey has valuexxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
region has value westus
utterance has value turn on the bedroom light
response
{
    "query": "turn on the bedroom light",
    "topScoringIntent": {
        "intent": "HomeAutomation.TurnOn",
        "score": 0.809439957
    },
    "entities": [
        {
            "entity": "bedroom",
            "type": "HomeAutomation.Room",
            "startIndex": 12,
            "endIndex": 18,
            "score": 0.8065475
        }
    ]
}
```

LUIS keys

This quickstart uses the authoring key for convenience. The key is primarily for authoring the model but does allow a small number (1000) of endpoint requests. When you are ready for more endpoint requests in a test, stage or production environment, create a **Language Understanding** resource in the Azure portal and assign it to the LUIS app in the LUIS portal.

Clean up resources

Close the Go file and remove it from the file system.

Next steps

[Add utterances](#)

Quickstart: Get intent using Java

7/26/2019 • 4 minutes to read • [Edit Online](#)

In this quickstart, pass utterances to a LUIS endpoint and get intent and entities back.

In this quickstart, use an available public LUIS app to determine a user's intention from conversational text. Send the user's intention as text to the public app's HTTP prediction endpoint. At the endpoint, LUIS applies the public app's model to analyze the natural language text for meaning, determining overall intent and extracting data relevant to the app's subject domain.

This quickstart uses the endpoint REST API. For more information, see the [endpoint API documentation](#).

For this article, you need a free [LUIS](#) account.

Prerequisites

- [JDK SE](#) (Java Development Kit, Standard Edition)
- [Visual Studio Code](#) or your favorite IDE
- Public app ID: df67dcdb-c37d-46af-88e1-8b97951ca1c2

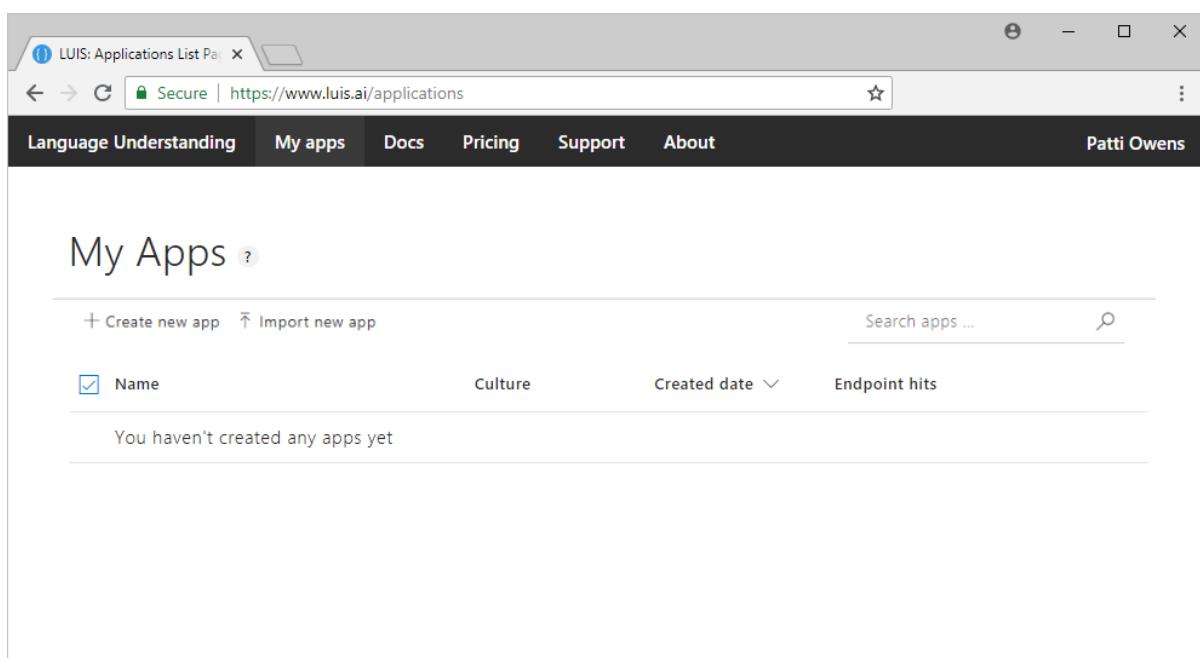
NOTE

The complete solution is available from the [cognitive-services-language-understanding](#) GitHub repository.

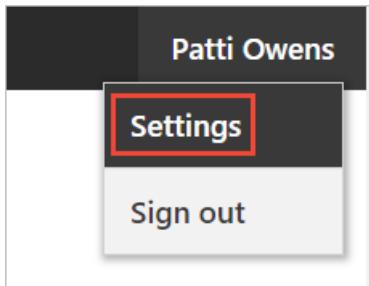
Get LUIS key

Access to the prediction endpoint is provided with an endpoint key. For the purposes of this quickstart, use the free starter key associated with your LUIS account.

1. Sign in using your LUIS account.



2. Select your name in the top right menu, then select **Settings**.



3. Copy the value of the **Authoring key**. You will use it later in the quickstart.

The authoring key allows free unlimited requests to the authoring API and up to 1000 queries to the prediction endpoint API per month for all your LUIS apps.

Get intent with browser

To understand what a LUIS prediction endpoint returns, view a prediction result in a web browser. In order to query a public app, you need your own key and the app ID. The public IoT app ID, `df67dcdb-c37d-46af-88e1-8b97951ca1c2`, is provided as part of the URL in step one.

The format of the URL for a **GET** endpoint request is:

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?subscription-key=<YOUR-KEY>&q=<user-utterance>
```

1. The endpoint of the public IoT app is in this format:

```
https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951ca1c2?  
subscription-key=<YOUR_KEY>&q=turn on the bedroom light
```

Copy the URL and substitute your key for the value of `<YOUR_KEY>`.

2. Paste the URL into a browser window and press Enter. The browser displays a JSON result that indicates that LUIS detects the `HomeAutomation.TurnOn` intent as the top intent and the `HomeAutomation.Room` entity with the value `bedroom`.

```
{
  "query": "turn on the bedroom light",
  "topScoringIntent": {
    "intent": "HomeAutomation.TurnOn",
    "score": 0.809439957
  },
  "entities": [
    {
      "entity": "bedroom",
      "type": "HomeAutomation.Room",
      "startIndex": 12,
      "endIndex": 18,
      "score": 0.8065475
    }
  ]
}
```

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

```
{
  "query": "turn off the living room light",
  "topScoringIntent": {
    "intent": "HomeAutomation.TurnOff",
    "score": 0.984057844
  },
  "entities": [
    {
      "entity": "living room",
      "type": "HomeAutomation.Room",
      "startIndex": 13,
      "endIndex": 23,
      "score": 0.9619945
    }
  ]
}
```

Get intent programmatically

You can use Java to access the same results you saw in the browser window in the previous step. Be sure to add the Apache libraries to your project.

1. Copy the following code to create a class in a file named `LuisGetRequest.java`:

```
// This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/downloads.cgi)
// You need to add the following Apache HTTP client libraries to your project:
// httpclient-4.5.3.jar
// httpcore-4.4.11.jar
// commons-logging-4.0.6.jar

import java.net.URI;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

public class LuisGetRequest {
```

```

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

    try
    {

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

        // Add your endpoint key
        // You can use the authoring key instead of the endpoint key.
        // The authoring key allows 1000 endpoint queries a month.
        String EndpointKey = "YOUR-KEY";

        // Begin endpoint URL string building
        URIBuilder endpointURLbuilder = new
        URIBuilder("https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/" + AppId + "?");

        // query text
        endpointURLbuilder.setParameter("q", "turn on the left light");

        // create URL from string
        URI endpointURL = endpointURLbuilder.build();

        // create HTTP object from URL
        HttpGet request = new HttpGet(endpointURL);

        // set key to access LUIS endpoint
        request.setHeader("Ocp-Apim-Subscription-Key", EndpointKey);

        // access LUIS endpoint - analyze text
        HttpResponse response = httpClient.execute(request);

        // get response
        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 `YOUR-KEY` variable with your LUIS key.
3. Replace with your file path and compile the java program from a command line:
`javac -cp .;<FILE_PATH>* LuisGetRequest.java`.
4. Replace with your file path and run the application from a command line:
`java -cp .;<FILE_PATH>* LuisGetRequest.java`. It displays the same JSON that you saw earlier in the browser window.



```
<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": 0.968756
  },
  "entities": [
    {
      "entity": "left",
      "type": "Light",
      "startIndex": 12,
      "endIndex": 15,
      "score": 0.957022846
    }
  ]
}
```

LUIS keys

This quickstart uses the authoring key for convenience. The key is primarily for authoring the model but does allow a small number (1000) of endpoint requests. When you are ready for more endpoint requests in a test, stage or production environment, create a **Language Understanding** resource in the Azure portal and assign it to the LUIS app in the LUIS portal.

Clean up resources

Delete the Java file/project folder.

Next steps

[Add utterances](#)

Quickstart: Get intent using Node.js

7/26/2019 • 3 minutes to read • [Edit Online](#)

In this quickstart, use an available public LUIS app to determine a user's intention from conversational text. Send the user's intention as text to the public app's HTTP prediction endpoint. At the endpoint, LUIS applies the public app's model to analyze the natural language text for meaning, determining overall intent and extracting data relevant to the app's subject domain.

This quickstart uses the endpoint REST API. For more information, see the [endpoint API documentation](#).

For this article, you need a free [LUIS](#) account.

Prerequisites

- [Node.js](#) programming language
- [Visual Studio Code](#)
- Public app ID: df67dcdb-c37d-46af-88e1-8b97951ca1c2

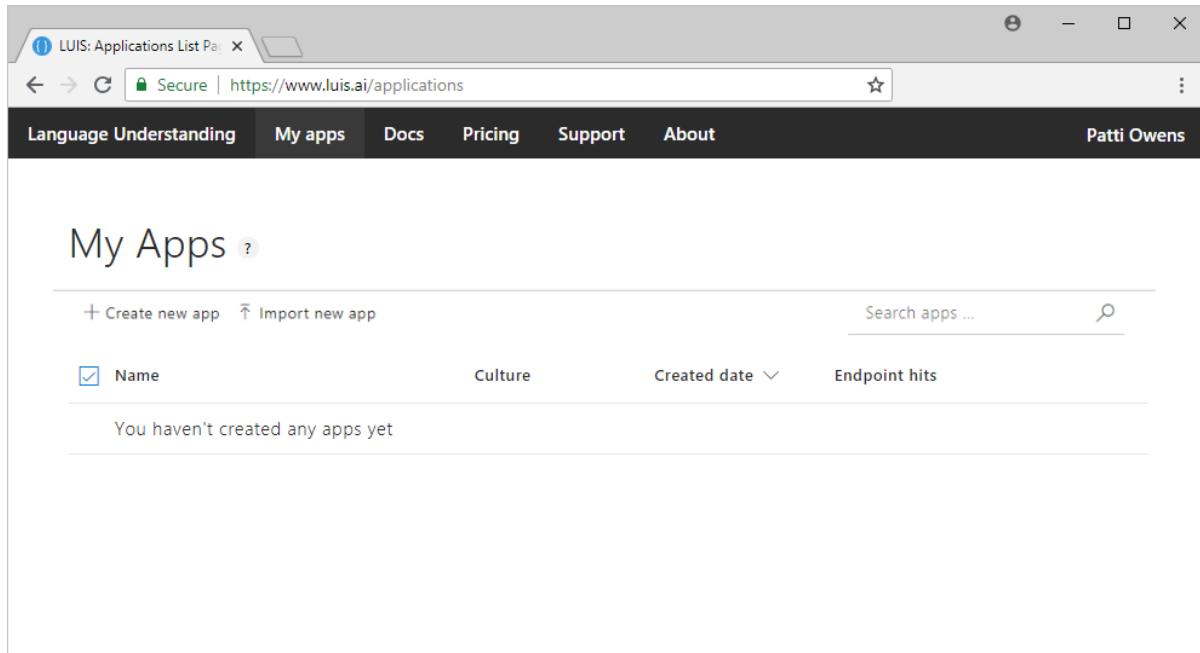
NOTE

The complete Node.js solution is available from the [Azure-Samples GitHub repository](#).

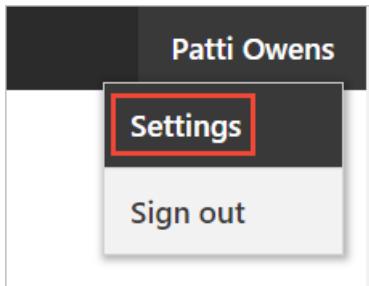
Get LUIS key

Access to the prediction endpoint is provided with an endpoint key. For the purposes of this quickstart, use the free starter key associated with your LUIS account.

1. Sign in using your LUIS account.



2. Select your name in the top right menu, then select **Settings**.



3. Copy the value of the **Authoring key**. You will use it later in the quickstart.

Language Understanding My apps Docs Pricing Support About Patti Owens

User settings

Authoring Key

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx (Reset) ?

Other settings

Country (Required)
United States

Company
Microsoft

The authoring key allows free unlimited requests to the authoring API and up to 1000 queries to the prediction endpoint API per month for all your LUIS apps.

Get intent with browser

To understand what a LUIS prediction endpoint returns, view a prediction result in a web browser. In order to query a public app, you need your own key and the app ID. The public IoT app ID, `df67dcdb-c37d-46af-88e1-8b97951ca1c2`, is provided as part of the URL in step one.

The format of the URL for a **GET** endpoint request is:

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?subscription-key=<YOUR-KEY>&q=<user-utterance>
```

1. The endpoint of the public IoT app is in this format:

```
https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951ca1c2?  
subscription-key=<YOUR_KEY>&q=turn on the bedroom light
```

Copy the URL and substitute your key for the value of `<YOUR_KEY>`.

2. Paste the URL into a browser window and press Enter. The browser displays a JSON result that indicates that LUIS detects the `HomeAutomation.TurnOn` intent as the top intent and the `HomeAutomation.Room` entity with the value `bedroom`.

```
{  
  "query": "turn on the bedroom light",  
  "topScoringIntent": {  
    "intent": "HomeAutomation.TurnOn",  
    "score": 0.809439957  
  },  
  "entities": [  
    {  
      "entity": "bedroom",  
      "type": "HomeAutomation.Room",  
      "startIndex": 12,  
      "endIndex": 18,  
      "score": 0.8065475  
    }  
  ]  
}
```

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

```
{  
  "query": "turn off the living room light",  
  "topScoringIntent": {  
    "intent": "HomeAutomation.TurnOff",  
    "score": 0.984057844  
  },  
  "entities": [  
    {  
      "entity": "living room",  
      "type": "HomeAutomation.Room",  
      "startIndex": 13,  
      "endIndex": 23,  
      "score": 0.9619945  
    }  
  ]  
}
```

Get intent programmatically

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');

// Analyze text
//
// utterance = user's text
//
function getLuisIntent(utterance) {

    // endpoint URL
    var endpoint =
        "https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/";

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

    // Read LUIS key from environment file ".env"
    // You can use the authoring key instead of the endpoint key.
    // The authoring key allows 1000 endpoint queries a month.
    var endpointKey = process.env.LUIS_ENDPOINT_KEY;

    // Create query string
    var queryParams = {
        "verbose": true,
        "q": utterance,
        "subscription-key": endpointKey
    }

    // append query string to endpoint URL
    var luisRequest =
        endpoint + luisAppId +
        '?' + querystring.stringify(queryParams);

    // HTTP Request
    request(luisRequest,
        function (err,
            response, body) {

            // HTTP Response
            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(JSON.stringify(data.intents));
            }
        });
}

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

```

2. Create `.env` file with the following text or set these variables in the system environment:

```
LUIS_APP_ID=df67dcdb-c37d-46af-88e1-8b97951ca1c2  
LUIS_ENDPOINT_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

3. Set the `LUIS_ENDPOINT_KEY` environment variable to your key.
4. Install dependencies by running the following command at the command-line: `npm install`.
5. Run the code with `npm start`. It displays the same values that you saw earlier in the browser window.

LUIS keys

This quickstart uses the authoring key for convenience. The key is primarily for authoring the model but does allow a small number (1000) of endpoint requests. When you are ready for more endpoint requests in a test, stage or production environment, create a **Language Understanding** resource in the Azure portal and assign it to the LUIS app in the LUIS portal.

Clean up resources

Delete the Node.js file.

Next steps

[Add utterances](#)

Quickstart: Get intent using Python

7/26/2019 • 4 minutes to read • [Edit Online](#)

In this quickstart, pass utterances to a LUIS endpoint and get intent and entities back.

In this quickstart, use an available public LUIS app to determine a user's intention from conversational text. Send the user's intention as text to the public app's HTTP prediction endpoint. At the endpoint, LUIS applies the public app's model to analyze the natural language text for meaning, determining overall intent and extracting data relevant to the app's subject domain.

This quickstart uses the endpoint REST API. For more information, see the [endpoint API documentation](#).

For this article, you need a free [LUIS](#) account.

Prerequisites

- [Python 3.6](#) or later.
- [Visual Studio Code](#)

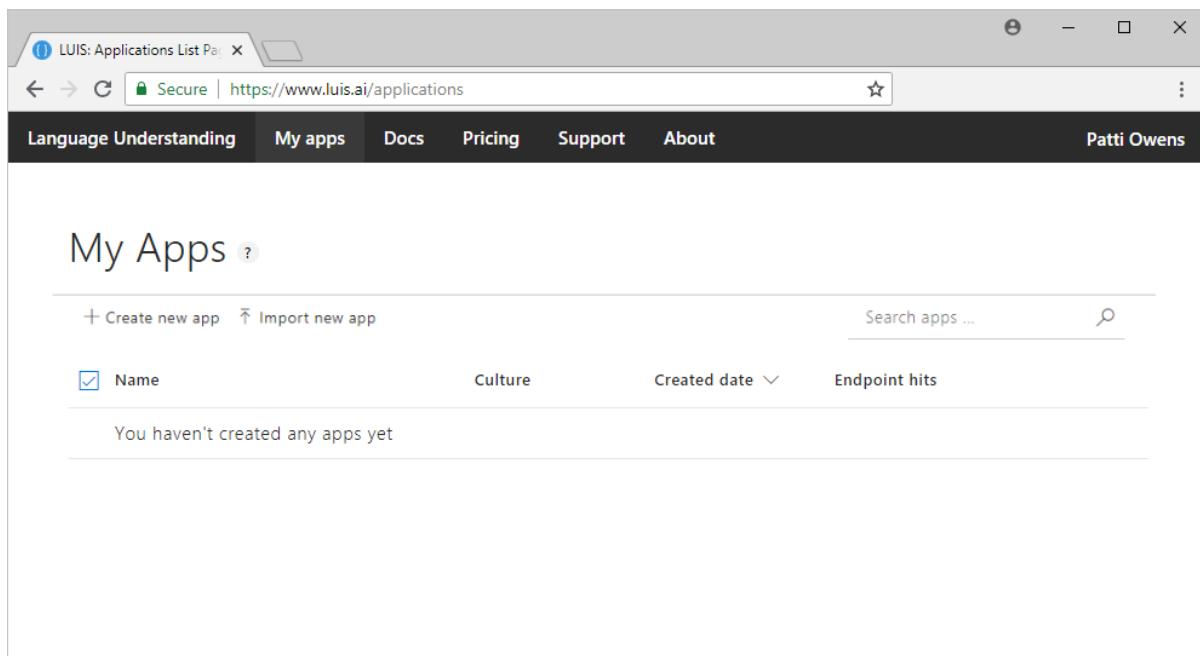
NOTE

The complete solution is available from the [cognitive-services-language-understanding](#) GitHub repository.

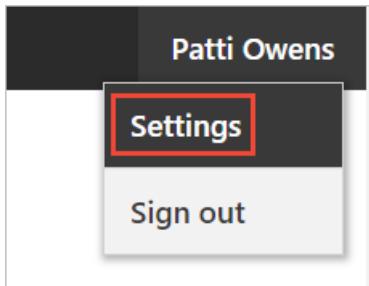
Get LUIS key

Access to the prediction endpoint is provided with an endpoint key. For the purposes of this quickstart, use the free starter key associated with your LUIS account.

1. Sign in using your LUIS account.



2. Select your name in the top right menu, then select **Settings**.



3. Copy the value of the **Authoring key**. You will use it later in the quickstart.

The authoring key allows free unlimited requests to the authoring API and up to 1000 queries to the prediction endpoint API per month for all your LUIS apps.

Get intent with browser

To understand what a LUIS prediction endpoint returns, view a prediction result in a web browser. In order to query a public app, you need your own key and the app ID. The public IoT app ID, `df67dcdb-c37d-46af-88e1-8b97951ca1c2`, is provided as part of the URL in step one.

The format of the URL for a **GET** endpoint request is:

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?subscription-key=<YOUR-KEY>&q=<user-utterance>
```

1. The endpoint of the public IoT app is in this format:

```
https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951ca1c2?  
subscription-key=<YOUR_KEY>&q=turn on the bedroom light
```

Copy the URL and substitute your key for the value of `<YOUR_KEY>`.

2. Paste the URL into a browser window and press Enter. The browser displays a JSON result that indicates that LUIS detects the `HomeAutomation.TurnOn` intent as the top intent and the `HomeAutomation.Room` entity with the value `bedroom`.

```
{  
    "query": "turn on the bedroom light",  
    "topScoringIntent": {  
        "intent": "HomeAutomation.TurnOn",  
        "score": 0.809439957  
    },  
    "entities": [  
        {  
            "entity": "bedroom",  
            "type": "HomeAutomation.Room",  
            "startIndex": 12,  
            "endIndex": 18,  
            "score": 0.8065475  
        }  
    ]  
}
```

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

```
{  
    "query": "turn off the living room light",  
    "topScoringIntent": {  
        "intent": "HomeAutomation.TurnOff",  
        "score": 0.984057844  
    },  
    "entities": [  
        {  
            "entity": "living room",  
            "type": "HomeAutomation.Room",  
            "startIndex": 13,  
            "endIndex": 23,  
            "score": 0.9619945  
        }  
    ]  
}
```

Get intent programmatically

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 to a file called `quickstart-call-endpoint.py`:

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

headers = {
    # Request headers includes endpoint key
    # You can use the authoring key instead of the endpoint key.
    # The authoring key allows 1000 endpoint queries a month.
    'Ocp-Apim-Subscription-Key': 'YOUR-KEY',
}

params = urllib.urlencode({
    # Text to analyze
    'q': 'turn on the left light',
    # Optional request parameters, set to default values
    'verbose': 'false',
})

# HTTP Request
try:
    # LUIS endpoint HOST for westus region
    conn = httplib.HTTPSConnection('westus.api.cognitive.microsoft.com')

    # LUIS endpoint path
    # includes public app ID
    conn.request("GET", "/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951ca1c2?%s" % params, "{body}",
    headers)

    response = conn.getresponse()
    data = response.read()

    # print HTTP response to screen
    print(data)
    conn.close()
except Exception as e:
    print("[Errno {0}] {1}".format(e errno, e.strerror))

#####

```

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

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

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

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

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

#####

```

2. Replace the value of the `Ocp-Apim-Subscription-Key` field with your LUIS endpoint key.
3. Install dependencies with `pip install requests`.
4. Run the script with `python ./quickstart-call-endpoint.py`. It displays the same JSON that you saw earlier in the browser window.

LUIS keys

This quickstart uses the authoring key for convenience. The key is primarily for authoring the model but does allow a small number (1000) of endpoint requests. When you are ready for more endpoint requests in a test, stage or production environment, create a **Language Understanding** resource in the Azure portal and assign it to the LUIS app in the LUIS portal.

Clean up resources

Delete the python file.

Next steps

[Add utterances](#)

Quickstart: Change model using C#

7/29/2019 • 6 minutes to read • [Edit Online](#)

In this quickstart, add example utterances to a Travel Agent app and train the app. Example utterances are conversational user text mapped to an intent. By providing example utterances for intents, you teach LUIS what kinds of user-supplied text belongs to which intent.

For more information, see the technical documentation for the [add example utterance to intent, train](#), and [training status](#) APIs.

For this article, you need a free [LUIS](#) account.

Prerequisites

- Your LUIS [authoring key](#).
- Import the [TravelAgent app](#) from the cognitive-services-language-understanding GitHub repository.
- The LUIS [application ID](#) for the imported TravelAgent app. The application ID is shown in the application dashboard.
- The [utterances.json](#) file containing the example utterances to import.
- The [version ID](#) within the application that receives the utterances. The default ID is "0.1".
- Latest [Visual Studio Community edition](#).
- C# programming language installed.
- [JsonFormatterPlus](#) and [CommandLine](#) NuGet packages

NOTE

The complete solution including an example `utterances.json` file are available from the [cognitive-services-language-understanding GitHub repository](#).

Example utterances JSON file

The example utterances file, **utterances.json**, follows a specific format.

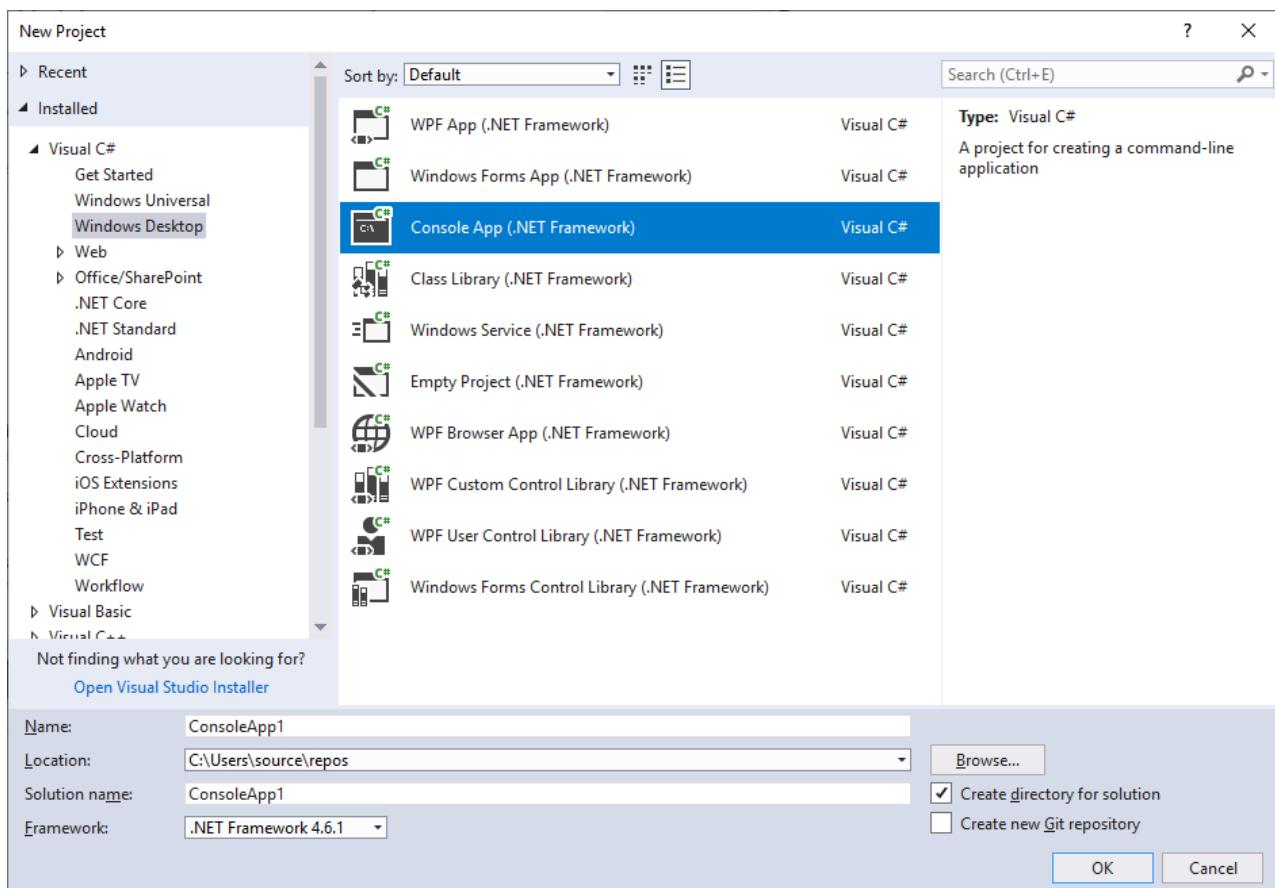
The `text` field contains the text of the example utterance. The `intentName` field must correspond to the name of an existing intent in the LUIS app. The `entityLabels` field is required. If you don't want to label any entities, provide an empty array.

If the `entityLabels` array is not empty, the `startCharIndex` and `endCharIndex` need to mark the entity referred to in the `entityName` field. The index is zero-based, meaning 6 in the top example refers to the "S" of Seattle and not the space before the capital S. If you begin or end the label at a space in the text, the API call to add the utterances fails.

```
[
  {
    "text": "go to Seattle today",
    "intentName": "BookFlight",
    "entityLabels": [
      {
        "entityName": "Location::LocationTo",
        "startCharIndex": 6,
        "endCharIndex": 12
      }
    ]
  },
  {
    "text": "purple dogs are difficult to work with",
    "intentName": "BookFlight",
    "entityLabels": []
  }
]
```

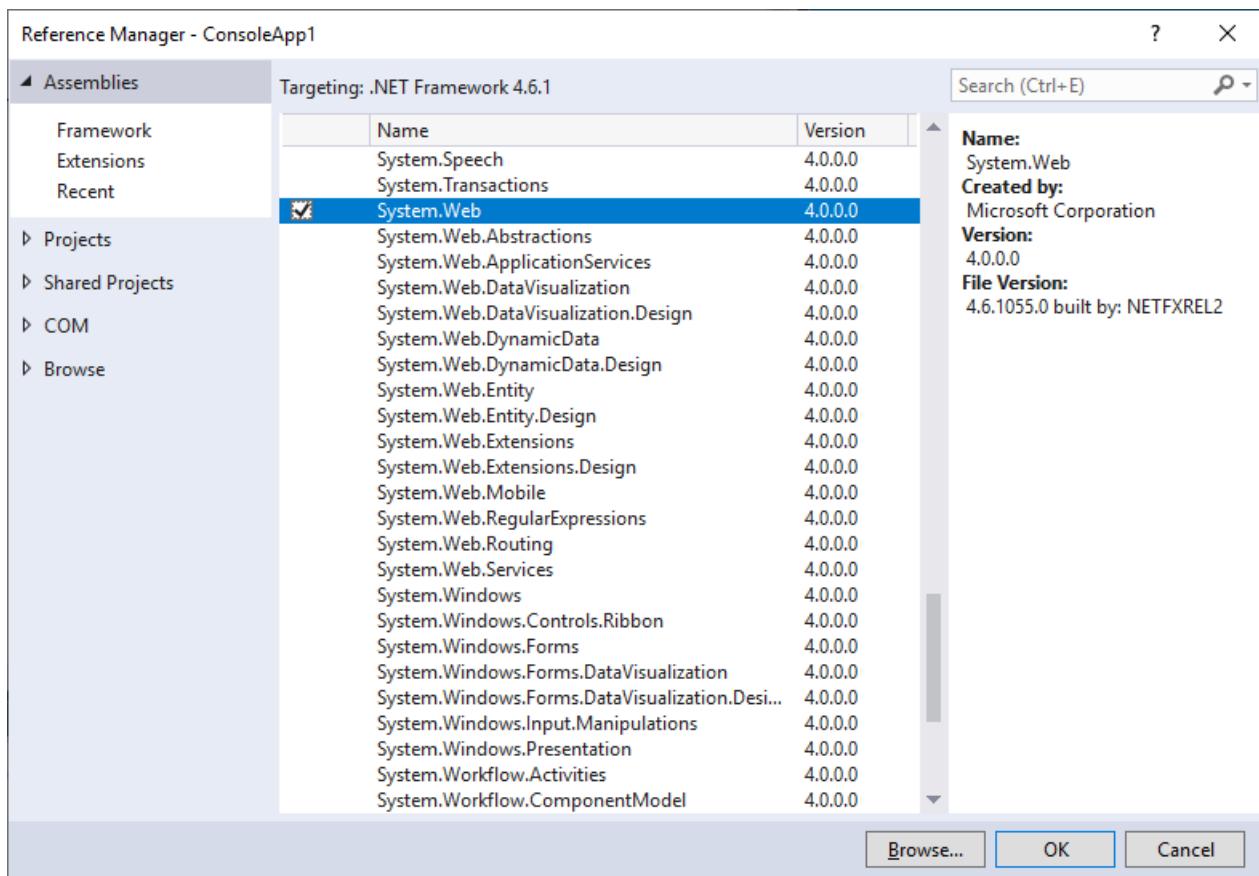
Create quickstart code

In Visual Studio, create a new **Windows Classic Desktop Console** app using the .NET Framework. Name the project `ConsoleApp1`.



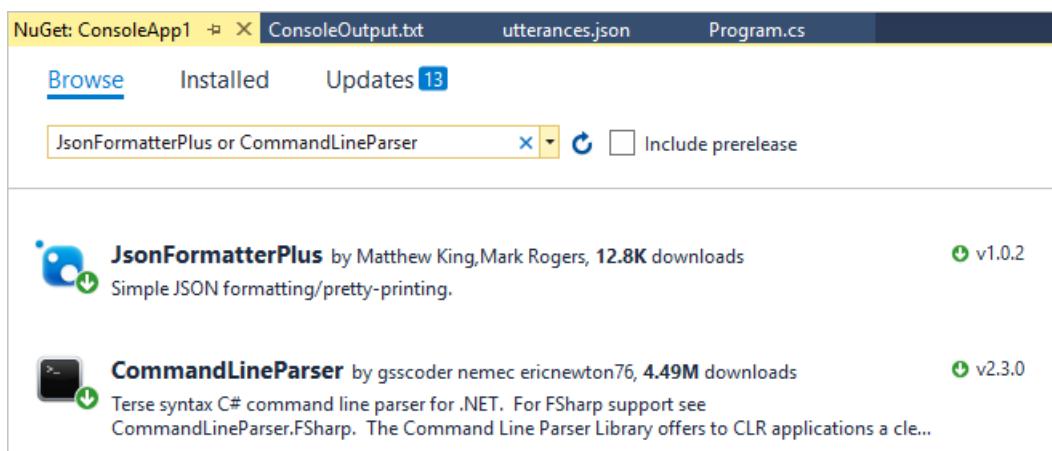
Add the System.Web dependency

The Visual Studio project needs **System.Web**. In the Solution Explorer, right-click on **References** and select **Add Reference** from the Assemblies section.



Add other dependencies

The Visual Studio project needs **JsonFormatterPlus** and **CommandLineParser**. In the Solution Explorer, right-click on **References** and select **Manage NuGet Packages....** Browse for and add each of the two packages.



Write the C# code

The **Program.cs** file should be:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}

```

Update the dependencies so that are:

```

using System;
using System.IO;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using System.Collections.Generic;
using System.Linq;

// 3rd party NuGet packages
using JsonFormatterPlus;
using CommandLine;

```

Add the LUIS IDs and strings to the **Program** class.

```

// NOTE: Replace this example LUIS application ID with the ID of your LUIS application.
static string appID = "YOUR-APP-ID";

// NOTE: Replace this example LUIS application version number with the version number of your LUIS
application.
static string appVersion = "0.1";

// NOTE: Replace this example LUIS authoring key with a valid key.
static string authoringKey = "YOUR-AUTHORING-KEY";

// Uses westus region
static string host = "https://westus.api.cognitive.microsoft.com";
static string path = "/luis/api/v2.0/apps/" + appID + "/versions/" + appVersion + "/";

```

Add class to manage command-line parameters to the **Program** class.

```
// parse command line options
public class Options
{
    [Option('v', "verbose", Required = false, HelpText = "Set output to verbose messages.")]
    public bool Verbose { get; set; }

    [Option('t', "train", Required = false, HelpText = "Train model.")]
    public bool Train { get; set; }

    [Option('s', "status", Required = false, HelpText = "Get training status.")]
    public bool Status { get; set; }

    [Option('a', "add", Required = false, HelpText = "Add example utterances to model.")]
    public IEnumerable<string> Add{ get; set; }
}
```

Add the GET request method to the **Program** class.

```
async static Task<HttpResponseMessage> SendGet(string uri)
{
    using (var client = new HttpClient())
    using (var request = new HttpRequestMessage())
    {
        request.Method = HttpMethod.Get;
        request.RequestUri = new Uri(uri);
        request.Headers.Add("Ocp-Apim-Subscription-Key", authoringKey);
        return await client.SendAsync(request);
    }
}
```

Add the POST request method to the **Program** class.

```
async static Task<HttpResponseMessage> SendPost(string uri, string requestBody)
{
    using (var client = new HttpClient())
    using (var request = new HttpRequestMessage())
    {
        request.Method = HttpMethod.Post;
        request.RequestUri = new Uri(uri);

        if (!String.IsNullOrEmpty(requestBody))
        {
            request.Content = new StringContent(requestBody, Encoding.UTF8, "text/json");
        }

        request.Headers.Add("Ocp-Apim-Subscription-Key", authoringKey);
        return await client.SendAsync(request);
    }
}
```

Add example utterances from file method to the **Program** class.

```
async static Task AddUtterances(string input_file)
{
    string uri = host + path + "examples";
    string requestBody = File.ReadAllText(input_file);

    var response = await SendPost(uri, requestBody);
    var result = await response.Content.ReadAsStringAsync();
    Console.WriteLine("Added utterances.");
    Console.WriteLine(JsonFormatter.Format(result));
}
```

After the changes are applied to the model, train the model. Add method to the **Program** class.

```
async static Task Train()
{
    string uri = host + path + "train";

    var response = await SendPost(uri, null);
    var result = await response.Content.ReadAsStringAsync();
    Console.WriteLine("Sent training request.");
    Console.WriteLine(JsonFormatter.Format(result));

}
```

Training may not complete immediately, check status to verify training is complete. Add method to the **Program** class.

```
async static Task Status()
{
    var response = await SendGet(host + path + "train");
    var result = await response.Content.ReadAsStringAsync();
    Console.WriteLine("Requested training status.");
    Console.WriteLine(JsonFormatter.Format(result));
}
```

To manage command-line arguments, add the main code. Add method to the **Program** class.

```

        static void Main(string[] args)
        {

            // Parse commandline options
            // For example:
            // ConsoleApp1.exe --add utterances.json --train --status
            Parser.Default.ParseArguments<Options>(args)
                .WithParsed<Options>(o =>
                {

                    // add example utterances
                    if (o.Add != null && o.Add.GetEnumerator().MoveNext())
                    {
                        AddUtterances(o.Add.FirstOrDefault()).Wait();
                    }

                    // request training
                    if (o.Train)
                    {
                        Train().Wait();
                    }

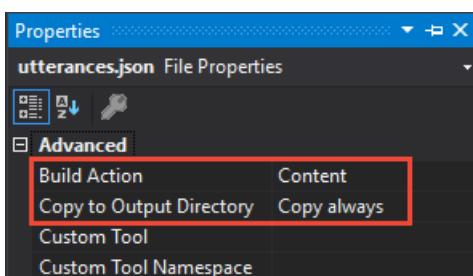
                    // get training status
                    if (o.Status)
                    {
                        Status().Wait();
                    }
                });
            }

        }
    }
}

```

Copy utterances.json to output directory

In the Solution Explorer, add the `utterances.json` by right-clicking in the Solution Explorer's project name, then selecting **Add**, then selecting **Existing item**. Select the `utterances.json` file. This adds the file to the project. Then it needs to be added to the output directory. Right-click the `utterances.json` and select **Properties**. In the properties windows, mark the **Build Action** of `Content`, and the **Copy to Output Directory** of `Copy Always`.



Build code

Build the code in Visual Studio.

Run code

In the project's /bin/Debug directory, run the application from a command line.

```
ConsoleApp1.exe --add utterances.json --train --status
```

This command-line displays the results of calling the add utterances API.

The `response` array for adding the example utterances indicates success or failure for each example utterance with the `hasError` property. The following JSON response shows both utterances were added successfully.

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

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

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

The following JSON shows the result of a successful request for training status. Each modelID is an intent. Each intent has to be trained on all the utterances to correctly identify utterances to do belong to the intent as well as utterances that do not belong to the intent.

```
[
  {
    "modelId": "0c694cf9-8c32-44b8-9ea0-3d30a7d901ca",
    "details": {
      "statusId": 3,
      "status": "InProgress",
      "exampleCount": 48
    }
  },
  {
    "modelId": "10e53836-ade4-494e-9531-3bd6a944c510",
    "details": {
      "statusId": 3,
      "status": "InProgress",
      "exampleCount": 48
    }
  },
  {
    "modelId": "21e48732-a512-4c33-b5ed-8ea629465269",
    "details": {
      "statusId": 3,
      "status": "InProgress",
      "exampleCount": 48
    }
  },
  {
    "modelId": "edee15b1-9999-45c2-bbab-591d3a643033",
    "details": {
      "statusId": 3,
```

```

        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 48
        }
    },
    {
        "modelId": "aa78e06e-df81-4bb2-b2d9-a2fbb2f81c54",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 48
        }
    },
    {
        "modelId": "e39bb7bd-b417-41a9-a24f-caf4c47fc62c",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 48
        }
    },
    {
        "modelId": "3782eac7-db84-4d66-ba00-0598dff848ee",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 48
        }
    },
    {
        "modelId": "a941d926-cb0f-47a8-ab7e-deba4378b96f",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 48
        }
    },
    {
        "modelId": "8137f40e-ce6d-40a5-881f-dfd46a05f7e0",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 48
        }
    },
    {
        "modelId": "dc08f95a-58b4-4064-a210-03fe34f75a3c",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 48
        }
    },
    {
        "modelId": "4fabdbed-5697-4562-8c7d-36e174efff2e",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 48
        }
    }
]

```

Clean up resources

When you are done with the quickstart, remove all the files created in this quickstart.

Next steps

[Build a LUIS app programmatically](#)

Quickstart: Change model using Go

7/26/2019 • 4 minutes to read • [Edit Online](#)

In this quickstart, add example utterances to a Travel Agent app and train the app. Example utterances are conversational user text mapped to an intent. By providing example utterances for intents, you teach LUIS what kinds of user-supplied text belongs to which intent.

For more information, see the technical documentation for the [add example utterance to intent, train](#), and [training status](#) APIs.

For this article, you need a free [LUIS](#) account.

Prerequisites

- Your LUIS [authoring key](#).
- Import the [TravelAgent app](#) from the cognitive-services-language-understanding GitHub repository.
- The LUIS [application ID](#) for the imported TravelAgent app. The application ID is shown in the application dashboard.
- The [utterances.json](#) file containing the example utterances to import.
- The [version ID](#) within the application that receives the utterances. The default ID is "0.1".
- [Go](#) programming language installed.
- [VSCode](#)

NOTE

The complete solution including an example `utterances.json` file are available from the [cognitive-services-language-understanding GitHub repository](#).

Example utterances JSON file

The example utterances file, **utterances.json**, follows a specific format.

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

If the `entityLabels` array is not empty, the `startCharIndex` and `endCharIndex` need to mark the entity referred to in the `entityName` field. The index is zero-based, meaning 6 in the top example refers to the "S" of Seattle and not the space before the capital S. If you begin or end the label at a space in the text, the API call to add the utterances fails.

```
[
  [
    {
      "text": "go to Seattle today",
      "intentName": "BookFlight",
      "entityLabels": [
        {
          "entityName": "Location::LocationTo",
          "startCharIndex": 6,
          "endCharIndex": 12
        }
      ]
    },
    {
      "text": "purple dogs are difficult to work with",
      "intentName": "BookFlight",
      "entityLabels": []
    }
  ]
]
```

Create quickstart code

1. Create `add-utterances.go` with VSCode.

2. Add dependencies.

```
// dependencies
package main
import (
  "fmt"
  "net/http"
  "io/ioutil"
  "log"
  "strings"
)
```

3. Add generic HTTP request function, which includes passing authoring key in header.

```
// generic HTTP request
// includes setting header with authoring key
func httpRequest(httpVerb string, url string, authoringKey string, body string){

  client := &http.Client{}

  request, err := http.NewRequest(httpVerb, url, strings.NewReader(body))
  request.Header.Add("Ocp-Apim-Subscription-Key", authoringKey)

  fmt.Println("body")
  fmt.Println(body)

  response, err := client.Do(request)
  if err != nil {
    log.Fatal(err)
  } else {
    defer response.Body.Close()
    contents, err := ioutil.ReadAll(response.Body)
    if err != nil {
      log.Fatal(err)
    }
    fmt.Println("  ", response.StatusCode)
    fmt.Println(string(contents))
  }
}
```

4. Add example utterances from JSON file.

```
// get utterances from file and add to model
func addUtterance(authoringKey string, appID string, version string, fileOfLabeledExampleUtterances
string){

    exampleUtterancesAsBytes, err := ioutil.ReadFile(fileOfLabeledExampleUtterances) // just pass the
file name
    if err != nil {
        fmt.Println(err)
    }
    fmt.Println(string(exampleUtterancesAsBytes))

    // NOTE: region is westus
    var authoringUrl =
fmt.Sprintf("https://westus.api.cognitive.microsoft.com/luis/api/v2.0/apps/%s/versions/%s/examples",
appID, version)

    httpRequest("POST", authoringUrl, authoringKey, (string(exampleUtterancesAsBytes)))
}
```

5. Request training. Uses a helper function to set the VERB for the same route as training status.

```
func requestTraining(authoringKey string, appID string, version string){

    trainApp("POST", authoringKey, appID, version)
}
func trainApp(httpVerb string, authoringKey string, appID string, version string){

    var authoringUrl =
fmt.Sprintf("https://westus.api.cognitive.microsoft.com/luis/api/v2.0/apps/%s/versions/%s/train", appID,
version)

    httpRequest(httpVerb,authoringUrl, authoringKey, "")
}
```

6. Request training status. Uses a helper function to set the VERB for the same route as request training.

```
func getTrainingStatus(authoringKey string, appID string, version string){

    trainApp("GET", authoringKey, appID, version)
}
```

7. Add main function to handle command-line parsing.

```

// main function
func main() {

    // NOTE: change to your app ID
    var appID = "YOUR-APP-ID"

    // NOTE: change to your authoring key
    var authoringKey = "YOUR-AUTHORING-KEY"

    var version = "0.1"

    var exampleUtterances = "utterances.json"

    fmt.Println("add example utterances requested")
    addUtterance(authoringKey, appID, version, exampleUtterances)

    fmt.Println("training selected")
    requestTraining(authoringKey, appID, version)

    fmt.Println("training status selected")
    getTrainingStatus(authoringKey, appID, version)

}

```

Add an utterance from the command line, train, and get status

1. From a command prompt in the directory where you created the Go file, enter `go build add-utterances.go` to compile the Go file. The command prompt does not return any information for a successful build.
2. Run the Go application from the command line by entering the following text in the command prompt:

```

add-utterances -appID <your-app-id> -authoringKey <add-your-authoring-key> -version <your-version-id> -
region westus -utteranceFile utterances.json

```

Replace `<add-your-authoring-key>` with the value of your authoring key (also known as the starter key).

Replace `<your-app-id>` with the value of your app ID. Replace `<your-version-id>` with the value of your version. Default version is `0.1`.

This command-prompt displays the results:

```

add example utterances requested
[
  {
    "text": "go lang 1",
    "intentName": "None",
    "entityLabels": []
  },
  {
    "text": "go lang 2",
    "intentName": "None",
    "entityLabels": []
  }
]
201
[
  {
    "value": {
      "ExampleId": 77783998,
      "UtteranceText": "go lang 1"
    },

```

```
        "hasError": false
    },
    {
        "value": {
            "ExampleId": 77783999,
            "UtteranceText": "go lang 2"
        },
        "hasError": false
    }
]
training selected
202
{"statusId":9,"status":"Queued"}
training status selected
200
[
    {
        "modelId": "c52d6509-9261-459e-90bc-b3c872ee4a4b",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 24
        }
    },
    {
        "modelId": "5119cbe8-97a1-4c1f-85e6-6449f3a38d77",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 24
        }
    },
    {
        "modelId": "01e6b6bc-9872-47f9-8a52-da510cddfafe",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 24
        }
    },
    {
        "modelId": "33b409b2-32b0-4b0c-9e91-31c6cfaf93fb",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 24
        }
    },
    {
        "modelId": "1fb210be-2a19-496d-bb72-e0c2dd35cbc1",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 24
        }
    },
    {
        "modelId": "3d098beb-a1aa-423f-a0ae-ce08ced216d6",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 24
        }
    },
    {
        "modelId": "cce854f8-8f8f-4ed9-a7df-44dfa562f62",
        "details": {
            "statusId": 3,
            "status": "InProgress".

```

```
        "exampleCount": 24
    },
},
{
    "modelId": "4d97bf0d-5213-4502-9712-2d6e77c96045",
    "details": {
        "statusId": 3,
        "status": "InProgress",
        "exampleCount": 24
    }
}
]
```

This response includes the HTTP status code for each of the three HTTP calls as well as any JSON response returned in the body of the response.

Clean up resources

When you are done with the quickstart, remove all the files created in this quickstart.

Next steps

[Build an app with a custom domain](#)

Quickstart: Change model using Java

7/26/2019 • 7 minutes to read • [Edit Online](#)

In this quickstart, add example utterances to a Travel Agent app and train the app. Example utterances are conversational user text mapped to an intent. By providing example utterances for intents, you teach LUIS what kinds of user-supplied text belongs to which intent.

For more information, see the technical documentation for the [add example utterance to intent, train](#), and [training status](#) APIs.

For this article, you need a free [LUIS](#) account.

Prerequisites

- Your LUIS [authoring key](#).
- Import the [TravelAgent app](#) from the cognitive-services-language-understanding GitHub repository.
- The LUIS [application ID](#) for the imported TravelAgent app. The application ID is shown in the application dashboard.
- The [utterances.json](#) file containing the example utterances to import.
- The [version ID](#) within the application that receives the utterances. The default ID is "0.1".
- [JDK SE](#) (Java Development Kit, Standard Edition)
- [Google's GSON JSON library](#).

NOTE

The complete solution including an example `utterances.json` file are available from the [cognitive-services-language-understanding GitHub repository](#).

Example utterances JSON file

The example utterances file, **utterances.json**, follows a specific format.

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

If the `entityLabels` array is not empty, the `startCharIndex` and `endCharIndex` need to mark the entity referred to in the `entityName` field. The index is zero-based, meaning 6 in the top example refers to the "S" of Seattle and not the space before the capital S. If you begin or end the label at a space in the text, the API call to add the utterances fails.

```
[
  [
    {
      "text": "go to Seattle today",
      "intentName": "BookFlight",
      "entityLabels": [
        {
          "entityName": "Location::LocationTo",
          "startCharIndex": 6,
          "endCharIndex": 12
        }
      ]
    },
    {
      "text": "purple dogs are difficult to work with",
      "intentName": "BookFlight",
      "entityLabels": []
    }
  ]
]
```

Create quickstart code

1. Add the Java dependencies to the file named `AddUtterances.java`.

```
import java.io.*;
import java.net.*;
import java.util.*;
import com.google.gson.*;
```

2. Create `AddUtterances` class. This class contains all code snippets that follow.

```
public class AddUtterances {
  // Insert code here
}
```

3. Add the LUIS constants to the class. Copy the following code and change to your authoring key, application ID, and version ID.

```
// Enter information about your LUIS application and key below
static final String LUIS_APP_ID      = "YOUR-APP-ID";
static final String LUIS_APP_VERSION = "0.1";
static final String LUIS_AUTHORING_ID = "YOUR-AUTHORING-KEY";

// Update the host if your LUIS subscription is not in the West US region
static final String LUIS_BASE        = "https://westus.api.cognitive.microsoft.com";

// File names for utterance and result files
static final String UTTERANCE_FILE   = "./utterances.json";

static final String UTF8 = "UTF-8";
```

4. Add the method to call into the LUIS API to the AddUtterances class.

```
//
// LUIS Client class
// Contains the functionality for adding utterances to a LUIS application
//
static class LuisClient{

  private final String PATH = "/luis/api/v2.0/apps/{app_id}/versions/{app_version}";
```

```

// endpoint method names
private final String TRAIN      = "/train";
private final String EXAMPLES = "/examples";
private final String APP_INFO = "/";

// HTTP verbs
private final String GET  = "GET";
private final String POST = "POST";

// Null string value for use in resolving method calls
private final String NO_DATA = null;

// Member variables
private final String key;
private final String host;
private final String path;

LuisClient(String host, String app_id, String app_version, String key) throws Exception {
    this.path = PATH.replace("{app_id}", app_id).replace("{app_version}", app_version);
    this.host = host;
    this.key = key;

    // Test configuration by getting the application info
    // {
    //     "version": "0.1",
    //     "createdDateTime": "2018-08-21T13:24:18Z",
    //     "lastModifiedDateTime": "2018-08-24T15:29:32Z",
    //     "intentsCount": 5,
    //     "entitiesCount": 8,
    //     "endpointHitsCount": 0,
    //     "trainingStatus": "InProgress"
    //}
    this.get(APP_INFO).raiseForStatus();
}

private LuisResponse call(String endpoint, String method, byte[] data) throws Exception {

    // initialize HTTP connection
    URL url = new URL(this.host + this.path + endpoint);

    HttpURLConnection conn = (HttpURLConnection)url.openConnection();
    conn.setRequestMethod(method);
    conn.setRequestProperty("Ocp-Apim-Subscription-Key", key);

    // handle POST request
    if (method.equals(POST)) {
        if (data == null)
            data = new byte[] {};
            // make zero-length body for POST w/o data
        conn.setDoOutput(true);
        conn.setRequestProperty("Content-Type", "application/json");
        conn.setRequestProperty("Content-Length", Integer.toString(data.length));
        try (OutputStream ostream = conn.getOutputStream()) {
            ostream.write(data, 0, data.length);
        }
    }

    // Get response from API call. If response is an HTTP error, the JSON
    // response is on the error string. Otherwise, it's on the input string.
    InputStream stream;
    try {
        stream = conn.getInputStream();
    } catch (IOException ex) {
        stream = conn.getErrorStream();
    }
    String body = new Scanner(stream, UTF8).useDelimiter("\A").next();

    return new LuisResponse(body, conn.getResponseCode(), conn.getResponseMessage());
}

```

```

}

// Overload of call() with String data parameter
private LuisResponse call(String endpoint, String method, String data) throws Exception {
    byte[] bytes = null;
    if (data != null)
        bytes = data.getBytes(UTF8);
    return call(endpoint, method, bytes);
}

// Overload of call() with InputStream data parameter
private LuisResponse call(String endpoint, String method, InputStream stream) throws Exception {
    String data = new Scanner(stream, UTF8).useDelimiter("\A").next();
    return call(endpoint, method, data);
}

// Shortcut for GET requests
private LuisResponse get(String endpoint) throws Exception {
    return call(endpoint, GET, NO_DATA);
}

// Shortcut for POST requests -- InputStream data
private LuisResponse post(String endpoint, InputStream data) throws Exception {
    return call(endpoint, POST, data);
}

// Shortcut for POST requests -- no data
private LuisResponse post(String endpoint) throws Exception {
    return call(endpoint, POST, NO_DATA);
}

// Call to add utterances
public LuisResponse addUtterances(String filename) throws Exception {
    try (FileInputStream stream = new FileInputStream(filename)) {
        return post(EXAMPLES, stream);
    }
}

public LuisResponse train() throws Exception {
    return post(TRAIN);
}

public LuisResponse status() throws Exception {
    return get(TRAIN);
}

}

```

5. Add the method for the HTTP response from the LUIS API to the AddUtterances class.

```

//  

// LUIS Response class  

// Represents a response from the LUIS client. All methods return  

// the instance so method calls can be chained.  

//  

static class LuisResponse {  

    private final String    body;  

    private final int       status;  

    private final String    reason;  

    private JsonElement    data;  

    LuisResponse(String body, int status, String reason) {  

        JsonParser parser = new JsonParser();  

        try {  

            this.data = parser.parse(body);  

        }  

        catch (JsonSyntaxException ex) {  

            this.data = parser.parse("{ \"message\": \"Invalid JSON response\" }");  

        }  

        this.body   = new GsonBuilder().setPrettyPrinting().create().toJson(data);  

        this.status = status;  

        this.reason = reason;  

        System.out.println(this.body);  

    }  

    LuisResponse raiseForStatus() throws StatusException {  

        if (this.status < 200 || this.status > 299) {  

            throw new StatusException(this);  

        }  

        return this;  

    }  

}

```

6. Add exception handling to the AddUtterances class.

```

// LUIS Status Exception class
// Represents an exception raised by the LUIS client for HTTP status errors
// Includes details extracted from the JSON response and the HTTP status
//
static class StatusException extends Exception {

    private String details = "";
    private final int status;

    StatusException(LuisResponse response) {
        super(String.format("%d %s", response.status, response.reason));
        JSONObject jsonInfo = (JSONObject)response.data;
        if (jsonInfo.has("error"))
            jsonInfo = (JSONObject)jsonInfo.get("error");
        if (jsonInfo.has("message"))
            this.details = jsonInfo.get("message").getAsString();
        this.status = response.status;
    }

    String getDetails() {
        return this.details;
    }

    int getStatus() {
        return this.status;
    }
}

static void printExceptionMsg(Exception ex) {
    System.out.println(String.format("%s: %s",
        ex.getClass().getSimpleName(), ex.getMessage()));

    StackTraceElement caller = ex.getStackTrace()[1];
    System.out.println(String.format("    in %s (line %d?)",
        caller.getFileName(), caller.getLineNumber()));
    if (ex instanceof StatusException)
        System.out.println(((StatusException)ex).getDetails());
}

```

7. Add the main function to the AddUtterances class.

```

// -----
// 
// Command-line entry point
// 
public static void main(String[] args) {

    LuisClient luis = null;

    try {
        luis = new LuisClient(LUIS_BASE, LUIS_APP_ID,
                             LUIS_APP_VERSION, LUIS_AUTHORIZING_ID);
    } catch (Exception ex) {
        printExceptionMsg(ex);
        System.exit(0);
    }

    try {

        System.out.println("Adding utterance(s).");
        luis.addUtterances(UTTERANCE_FILE)
            .raiseForStatus();

        System.out.println("Requesting training.");
        luis.train()
            .raiseForStatus();

        System.out.println("Requested training. Requesting training status.");
        luis.status()
            .raiseForStatus();

    } catch (Exception ex) {
        printExceptionMsg(ex);
    }
}

```

Build code

Compile AddUtterance with the dependencies

```
> javac -classpath gson-2.8.2.jar AddUtterances.java
```

Run code

Calling `AddUtterance` with no arguments adds the LUIS utterances to the app, without training it.

```
> java -classpath .;gson-2.8.2.jar AddUtterances
```

This command-line displays the results of calling the add utterances API.

The `response` array for adding the example utterances indicates success or failure for each example utterance with the `hasError` property. The following JSON response shows both utterances were added successfully.

```

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

```

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

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

The following JSON shows the result of a successful request for training status. Each modelID is an intent. Each intent has to be trained on all the utterances to correctly identify utterances to do belong to the intent as well as utterances that do not belong to the intent.

```
[
    {
        "modelId": "0c694cf9-8c32-44b8-9ea0-3d30a7d901ca",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 48
        }
    },
    {
        "modelId": "10e53836-ade4-494e-9531-3bd6a944c510",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 48
        }
    },
    {
        "modelId": "21e48732-a512-4c33-b5ed-8ea629465269",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 48
        }
    },
    {
        "modelId": "edee15b1-9999-45c2-bbab-591d3a643033",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 48
        }
    }
]
```

```
        },
        {
            "modelId": "aa78e06e-df81-4bb2-b2d9-a2fbb2f81c54",
            "details": {
                "statusId": 3,
                "status": "InProgress",
                "exampleCount": 48
            }
        },
        {
            "modelId": "e39bb7bd-b417-41a9-a24f-caf4c47fc62c",
            "details": {
                "statusId": 3,
                "status": "InProgress",
                "exampleCount": 48
            }
        },
        {
            "modelId": "3782eac7-db84-4d66-ba00-0598dfffb48ee",
            "details": {
                "statusId": 3,
                "status": "InProgress",
                "exampleCount": 48
            }
        },
        {
            "modelId": "a941d926-cb0f-47a8-ab7e-deba4378b96f",
            "details": {
                "statusId": 3,
                "status": "InProgress",
                "exampleCount": 48
            }
        },
        {
            "modelId": "8137f40e-ce6d-40a5-881f-dfd46a05f7e0",
            "details": {
                "statusId": 3,
                "status": "InProgress",
                "exampleCount": 48
            }
        },
        {
            "modelId": "dc08f95a-58b4-4064-a210-03fe34f75a3c",
            "details": {
                "statusId": 3,
                "status": "InProgress",
                "exampleCount": 48
            }
        },
        {
            "modelId": "4fabdbed-5697-4562-8c7d-36e174efff2e",
            "details": {
                "statusId": 3,
                "status": "InProgress",
                "exampleCount": 48
            }
        }
    ]
}
```

Clean up resources

When you are done with the quickstart, remove all the files created in this quickstart.

Next steps

[Build a LUIS app programmatically](#)

Quickstart: Change model using Node.js

7/26/2019 • 5 minutes to read • [Edit Online](#)

In this quickstart, add example utterances to a Travel Agent app and train the app. Example utterances are conversational user text mapped to an intent. By providing example utterances for intents, you teach LUIS what kinds of user-supplied text belongs to which intent.

For more information, see the technical documentation for the [add example utterance to intent, train](#), and [training status](#) APIs.

For this article, you need a free [LUIS](#) account.

Prerequisites

- Your LUIS [authoring key](#).
- Import the [TravelAgent app](#) from the cognitive-services-language-understanding GitHub repository.
- The LUIS [application ID](#) for the imported TravelAgent app. The application ID is shown in the application dashboard.
- The [utterances.json](#) file containing the example utterances to import.
- The [version ID](#) within the application that receives the utterances. The default ID is "0.1".
- Latest [Node.js](#) with NPM.
- NPM dependencies for this article: [request](#), [request-promise](#), [fs-extra](#).
- [Visual Studio Code](#).

NOTE

The complete solution including an example `utterances.json` file are available from the [cognitive-services-language-understanding GitHub repository](#).

Example utterances JSON file

The example utterances file, **utterances.json**, follows a specific format.

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

If the `entityLabels` array is not empty, the `startCharIndex` and `endCharIndex` need to mark the entity referred to in the `entityName` field. The index is zero-based, meaning 6 in the top example refers to the "S" of Seattle and not the space before the capital S. If you begin or end the label at a space in the text, the API call to add the utterances fails.

```
[  
  [  
    {  
      "text": "go to Seattle today",  
      "intentName": "BookFlight",  
      "entityLabels": [  
        {  
          "entityName": "Location::LocationTo",  
          "startCharIndex": 6,  
          "endCharIndex": 12  
        }  
      ]  
    },  
    {  
      "text": "purple dogs are difficult to work with",  
      "intentName": "BookFlight",  
      "entityLabels": []  
    }  
  ]
```

Create quickstart code

Add the NPM dependencies to the file named `add-utterances.js`.

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

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

```
// To run this sample, change these constants.  
  
// Authoring key, available in luis.ai under Account Settings  
const LUIS_authoringKey = "YOUR-AUTHORING-KEY";  
  
// ID of your LUIS app to which you want to add an utterance  
const LUIS_appId = "YOUR-APP-ID";  
  
// The version number of your LUIS app  
const LUIS_versionId = "0.1";
```

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

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

Add method and object for `addUtterance` function.

```

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

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

    try {

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

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

        let results = await utterancePromise;

        console.log(JSON.stringify(results));

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

```

Add method and object for `train` function.

```

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

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

    try {

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

        let results = await trainingPromise;
        console.log(JSON.stringify(results));

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

```

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

```

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

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

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

    } catch (err) {
        throw err;
    }
}

```

Add the **main** code that chooses which action.

```

var main = async() =>{
    try{

        console.log("Add utterances complete.");
        await addUtterance(configAddUtterance);

        console.log("Train");
        configTrain.method = 'POST';
        await train(configTrain, false);

        console.log("Train status.");
        configTrain.method = 'GET';
        await train(configTrain, true);

        console.log("process done");

    }catch(err){
        throw err;
    }
}

// MAIN
main();

```

Install dependencies

Create `package.json` file with the following text:

```
{
  "name": "node",
  "version": "1.0.0",
  "description": "",
  "main": "add-utterances.js",
  "scripts": {
    "start": "node add-utterances.js",
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "fs-extra": "^5.0.0",
    "request": "^2.83.0",
    "request-promise": "^4.2.2"
  }
}
```

On the command-line, from the directory that has the `package.json`, install dependencies with NPM: `npm install`.

Run code

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

Calling `npm start` adds the utterances, trains, and gets training status.

```
> npm start
```

This command-line displays the results of calling the add utterances API.

The `response` array for adding the example utterances indicates success or failure for each example utterance with the `hasError` property. The following JSON response shows both utterances were added successfully.

```

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

```

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

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

The following JSON shows the result of a successful request for training status. Each modelID is an intent. Each intent has to be trained on all the utterances to correctly identify utterances to do belong to the intent as well as utterances that do not belong to the intent.

```
[
    {
        "modelId": "0c694cf9-8c32-44b8-9ea0-3d30a7d901ca",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 48
        }
    },
    {
        "modelId": "10e53836-ade4-494e-9531-3bd6a944c510",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 48
        }
    },
    {
        "modelId": "21e48732-a512-4c33-b5ed-8ea629465269",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 48
        }
    },
    {
        "modelId": "edee15b1-9999-45c2-bbab-591d3a643033",
        "details": {
            "statusId": 3,
            "status": "InProgress",
            "exampleCount": 48
        }
    }
]
```

```
        },
        {
            "modelId": "aa78e06e-df81-4bb2-b2d9-a2fbb2f81c54",
            "details": {
                "statusId": 3,
                "status": "InProgress",
                "exampleCount": 48
            }
        },
        {
            "modelId": "e39bb7bd-b417-41a9-a24f-caf4c47fc62c",
            "details": {
                "statusId": 3,
                "status": "InProgress",
                "exampleCount": 48
            }
        },
        {
            "modelId": "3782eac7-db84-4d66-ba00-0598dfffb48ee",
            "details": {
                "statusId": 3,
                "status": "InProgress",
                "exampleCount": 48
            }
        },
        {
            "modelId": "a941d926-cb0f-47a8-ab7e-deba4378b96f",
            "details": {
                "statusId": 3,
                "status": "InProgress",
                "exampleCount": 48
            }
        },
        {
            "modelId": "8137f40e-ce6d-40a5-881f-dfd46a05f7e0",
            "details": {
                "statusId": 3,
                "status": "InProgress",
                "exampleCount": 48
            }
        },
        {
            "modelId": "dc08f95a-58b4-4064-a210-03fe34f75a3c",
            "details": {
                "statusId": 3,
                "status": "InProgress",
                "exampleCount": 48
            }
        },
        {
            "modelId": "4fabdbed-5697-4562-8c7d-36e174efff2e",
            "details": {
                "statusId": 3,
                "status": "InProgress",
                "exampleCount": 48
            }
        }
    ]
}
```

Clean up resources

When you are done with the quickstart, remove all the files created in this quickstart.

Next steps

[Build a LUIS app programmatically](#)

Quickstart: Change model using Python

7/26/2019 • 4 minutes to read • [Edit Online](#)

In this quickstart, add example utterances to a Travel Agent app and train the app. Example utterances are conversational user text mapped to an intent. By providing example utterances for intents, you teach LUIS what kinds of user-supplied text belongs to which intent.

For more information, see the technical documentation for the [add example utterance to intent, train](#), and [training status](#) APIs.

For this article, you need a free [LUIS](#) account.

Prerequisites

- Your LUIS [authoring key](#).
- Import the [TravelAgent app](#) from the cognitive-services-language-understanding GitHub repository.
- The LUIS [application ID](#) for the imported TravelAgent app. The application ID is shown in the application dashboard.
- The [utterances.json](#) file containing the example utterances to import.
- The [version ID](#) within the application that receives the utterances. The default ID is "0.1".
- [Python 3.6](#) or later.
- [Visual Studio Code](#)

NOTE

The complete solution including an example `utterances.json` file are available from the [cognitive-services-language-understanding GitHub repository](#).

Example utterances JSON file

The example utterances file, **utterances.json**, follows a specific format.

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

If the `entityLabels` array is not empty, the `startCharIndex` and `endCharIndex` need to mark the entity referred to in the `entityName` field. The index is zero-based, meaning 6 in the top example refers to the "S" of Seattle and not the space before the capital S. If you begin or end the label at a space in the text, the API call to add the utterances fails.

```
[
  [
    {
      "text": "go to Seattle today",
      "intentName": "BookFlight",
      "entityLabels": [
        {
          "entityName": "Location::LocationTo",
          "startCharIndex": 6,
          "endCharIndex": 12
        }
      ]
    },
    {
      "text": "purple dogs are difficult to work with",
      "intentName": "BookFlight",
      "entityLabels": []
    }
  ]
]
```

Create quickstart code

1. Copy the following code snippet into file named `add-utterances-3-6.py`:

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

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

# Authoring key, available in luis.ai under Account Settings
LUIS_authoringKey = "YOUR-AUTHORING-KEY"

# ID of your LUIS app to which you want to add an utterance
LUIS_APP_ID = "YOUR-APP-ID"

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

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

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

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

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

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

    # Encoding
    UTF8 = "UTF8"

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

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

```

result = ""

def __init__(self, host, app_id, app_version, key):
    self.key = key
    self.host = host
    self.path = self.PATH.format(app_id=app_id, app_version=app_version)

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

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

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

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

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

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

if __name__ == "__main__":
    luis = LUISClient(LUIS_HOST, LUIS_APP_ID, LUIS_APP_VERSION,
                      LUIS_authoringKey)

    try:
        print("Adding utterance(s).")
        luis.add_utterances().raise_for_status()

        print("Requesting training.")
        luis.train().raise_for_status()

        print("Requesting training status.")
        luis.status().raise_for_status()

    except Exception as ex:
        luis.print()    # JSON response may have more details
        print("{0.__name__}: {1}".format(type(ex), ex))

```

Run code

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

Add utterances from the command-line

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

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

The following response returns when the utterances are added to the model.

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

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

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

```
Requested training status.
[
  {
    "modelId": "eb2f117c-e10a-463e-90ea-1a0176660acc",
    "details": {
      "statusId": 0,
      "status": "Success",
      "exampleCount": 33,
      "trainingDateTime": "2017-11-20T18:09:11Z"
    }
  },
  {
    "modelId": "c1bdfbfc-e110-402e-b0cc-2af412289fb",
    "details": {
      "statusId": 0,
      "status": "Success",
      "exampleCount": 33,
      "trainingDateTime": "2017-11-20T18:09:11Z"
    }
  },
  {
    "modelId": "863023ec-2c96-4d68-9c44-34c1cbde8bc9",
    "details": {
      "statusId": 0,
      "status": "Success",
      "exampleCount": 33,
      "trainingDateTime": "2017-11-20T18:09:11Z"
    }
  }
]
```

```
},
{
  "modelId": "82702162-73ba-4ae9-a6f6-517b5244c555",
  "details": {
    "statusId": 0,
    "status": "Success",
    "exampleCount": 33,
    "trainingDateTime": "2017-11-20T18:09:11Z"
  }
},
{
  "modelId": "37121f4c-4853-467f-a9f3-6dfc8cad2763",
  "details": {
    "statusId": 0,
    "status": "Success",
    "exampleCount": 33,
    "trainingDateTime": "2017-11-20T18:09:11Z"
  }
},
{
  "modelId": "de421482-753e-42f5-a765-ad0a60f50d69",
  "details": {
    "statusId": 0,
    "status": "Success",
    "exampleCount": 33,
    "trainingDateTime": "2017-11-20T18:09:11Z"
  }
},
{
  "modelId": "80f58a45-86f2-4e18-be3d-b60a2c88312e",
  "details": {
    "statusId": 0,
    "status": "Success",
    "exampleCount": 33,
    "trainingDateTime": "2017-11-20T18:09:11Z"
  }
},
{
  "modelId": "c9eb9772-3b18-4d5f-a1e6-e0c31f91b390",
  "details": {
    "statusId": 0,
    "status": "Success",
    "exampleCount": 33,
    "trainingDateTime": "2017-11-20T18:09:11Z"
  }
},
{
  "modelId": "2afec2ff-7c01-4423-bb0e-e5f6935afae8",
  "details": {
    "statusId": 0,
    "status": "Success",
    "exampleCount": 33,
    "trainingDateTime": "2017-11-20T18:09:11Z"
  }
},
{
  "modelId": "95a81c87-0d7b-4251-8e07-f28d180886a1",
  "details": {
    "statusId": 0,
    "status": "Success",
    "exampleCount": 33,
    "trainingDateTime": "2017-11-20T18:09:11Z"
  }
}
]
```

Clean up resources

When you are done with the quickstart, remove all the files created in this quickstart.

Next steps

[Build a LUIS app programmatically](#)

Tutorial: Build LUIS app to determine user intentions

7/26/2019 • 7 minutes to read • [Edit Online](#)

In this tutorial, you create a custom Human Resources (HR) app that predicts a user's intention based on the utterance (text).

In this tutorial, you learn how to:

- Create a new app
- Create intents
- Add example utterances
- Train app
- Publish app
- Get intent from endpoint

For this article, you can use the free [LUIS](#) account in order to author your LUIS application.

User intentions as intents

The purpose of the app is to determine the intention of conversational, natural language text:

Are there any new positions in the Seattle office?

These intentions are categorized into **Intents**.

This app has a few intents.

INTENT	PURPOSE
ApplyForJob	Determine if user is applying for a job.
GetJobInformation	Determine if user is looking for information about jobs in general or a specific job.
None	Determine if user is asking something app is not supposed to answer. This intent is provided as part of app creation and can't be deleted.

Create a new app

1. Sign in to the LUIS portal with the URL of <https://www.luis.ai>.
2. Select **Create new app**.

The screenshot shows the LUIS Applications List Page. At the top, there's a navigation bar with links for Language Understanding, My apps, Docs, Pricing, Support, and About. A user profile for Patti Owens is on the right. Below the navigation is a section titled "My Apps" with a question mark icon. It includes buttons for "Create new app" and "Import new app", and a search bar labeled "Search apps ...". There are filter options for "Name", "Culture", "Created date", and "Endpoint hits". A message says "You haven't created any apps yet".

3. In the pop-up dialog, enter the name **HumanResources** and keep the default culture, **English**. Leave the description empty.

The dialog box is titled "Create new app". It has fields for "Name (Required)" containing "HumanResources", "Culture (Required)" set to "English", and a "Description" field with placeholder text "Type app description". At the bottom are "Done" and "Cancel" buttons.

4. Select **Done**.

Create intent for job information

1. Select **Create new intent**. Enter the new intent name **GetJobInformation**. This intent is predicted when a user wants information about open jobs in the company.

The dialog box is titled "Create new intent". It has a field for "Intent name (Required)" containing "GetJobInformation". At the bottom are "Done" and "Cancel" buttons.

2. Select **Done**.

3. Add several example utterances to this intent that you expect a user to ask:

EXAMPLE UTTERANCES

Any new jobs posted today?

Are there any new positions in the Seattle office?

Are there any remote worker or telecommute jobs open for engineers?

Is there any work with databases?

I'm looking for a co-working situation in the tampa office.

Is there an internship in the san francisco office?

Is there any part-time work for people in college?

Looking for a new situation with responsibilities in accounting

Looking for a job in new york city for bilingual speakers.

Looking for a new situation with responsibilities in accounting.

New jobs?

Show me all the jobs for engineers that were added in the last 2 days.

Today's job postings?

What accounting positions are open in the london office?

What positions are available for Senior Engineers?

Where is the job listings

App Assets

Intents

Entities

Improve app performance

Review endpoint utterances

Phrase lists

Patterns

GetJobInformation ↗

Labelled entities:

Example utterance

Where is the job listings

what positions are available for senior engineers ? -1.00

what accounting positions are open in the london office ? -1.00

today ' s job postings ? -1.00

show me all the jobs for engineers that were added in the last 2 days . -1.00

new jobs ? -1.00

looking for a new situation with responsibilities in accounting . -1.00

looking for a job in new york city for bilingual speakers . -1.00

looking for a new situation with responsibilities in accounting -1.00

is there any part - time work for people in college ? -1.00

is there an internship in the san francisco office ? -1.00

Score ?

1 2 Next >

By providing *example utterances*, you are training LUIS about what kinds of utterances should be

predicted for this intent.

These few utterances are for demonstration purposes only. A real-world app should have at least 15 utterances of varying length, word order, tense, grammatical correctness, punctuation, and word count.

Add example utterances to the None intent

The client application needs to know if an utterance is not meaningful or appropriate for the application. The **None** intent is added to each application as part of the creation process to determine if an utterance can't be answered by the client application.

If LUIS returns the **None** intent for an utterance, your client application can ask if the user wants to end the conversation or give more directions for continuing the conversation.

Caution

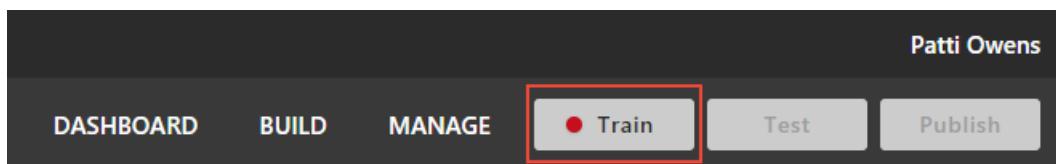
Do not leave the **None** intent empty.

1. Select **Intents** from the left panel.
2. Select the **None** intent. Add three utterances that your user might enter but are not relevant to your Human Resources app:

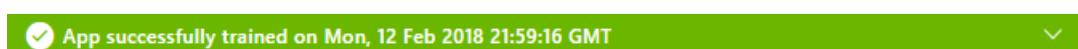
EXAMPLE UTTERANCES
Barking dogs are annoying
Order a pizza for me
Penguins in the ocean

Train the app before testing or publishing

1. In the top right side of the LUIS website, select the **Train** button.



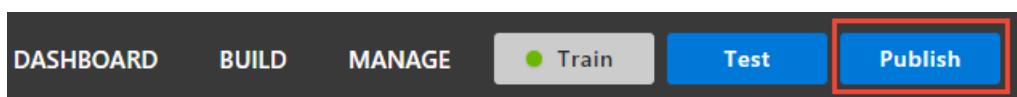
2. Training is complete when you see the green status bar at the top of the website confirming success.



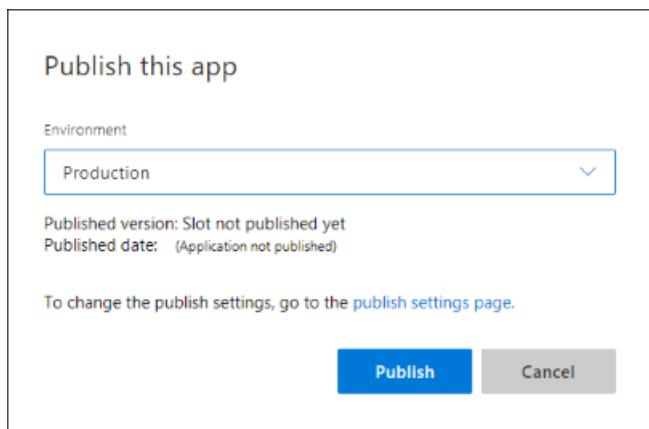
Publish the app to query from the endpoint

In order to receive a LUIS prediction in a chat bot or other client application, you need to publish the app to the endpoint.

1. Select **Publish** in the top right navigation.



2. Select the **Production** slot and the **Publish** button.



3. Publishing is complete when you see the green status bar at the top of the website confirming success.

Publishing complete. Refer to the list of endpoints to access your endpoint URL

4. Select the **endpoints** link in the green status bar to go to the **Keys and endpoints** page. The endpoint URLs are listed at the bottom.

Get intent prediction from the endpoint

1. In the **Manage** section (top right menu), on the **Keys and endpoints** page (left menu), select the **endpoint** URL at the bottom of the page. This action opens another browser tab with the endpoint URL in the address bar.

The endpoint URL looks like

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?verbose=true&subscription-key=<YOUR_KEY>&<optional-name-value-pairs>&q=<user-utterance-text>
```

2. Go to the end of the URL in the address bar and enter

I'm looking for a job with Natural Language Processing. The last query string parameter is `q`, the utterance **query**. This utterance is not the same as any of the example utterances. It is a good test and should return the `GetJobInformation` intent as the top scoring intent.

```
{
  "query": "I'm looking for a job with Natural Language Processing",
  "topScoringIntent": {
    "intent": "GetJobInformation",
    "score": 0.9923871
  },
  "intents": [
    {
      "intent": "GetJobInformation",
      "score": 0.9923871
    },
    {
      "intent": "None",
      "score": 0.007810574
    }
  ],
  "entities": []
}
```

The `verbose=true` querystring parameter means include **all the intents** in the app's query results. The entities array is empty because this app currently does not have any entities.

The JSON result identifies the top scoring intent as `topScoringIntent` property. All scores are between 1

and 0, with the better score being close to 1.

Create intent for job applications

Return to the LUIS portal and create a new intent to determine if the user utterance is about applying for a job.

1. Select **Build** from the top, right menu to return to app building.
2. Select **Intents** from the left menu to get to the list of intents.
3. Select **Create new intent** and enter the name `ApplyForJob`.

Create new intent

Intent name (Required)

ApplyForJob

Done Cancel

4. Add several utterances to this intent that you expect a user to ask for, such as:

EXAMPLE UTTERANCES

Fill out application for Job 123456

Here is my c.v. for position 654234

Here is my resume for the part-time receptionist post.

I'm applying for the art desk job with this paperwork.

I'm applying for the summer college internship in Research and Development in San Diego

I'm requesting to submit my resume to the temporary position in the cafeteria.

I'm submitting my resume for the new Autocar team in Columbus, OH

I want to apply for the new accounting job

Job 456789 accounting internship paperwork is here

Job 567890 and my paperwork

My papers for the tulsa accounting internship are attached.

My paperwork for the holiday delivery position

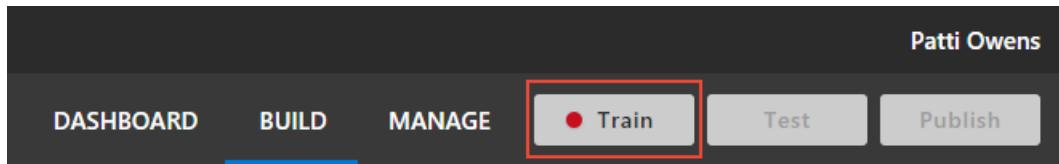
Please send my resume for the new accounting job in seattle

Submit resume for engineering position

This is my c.v. for post 234123 in Tampa.

Train again

1. In the top right side of the LUIS website, select the **Train** button.



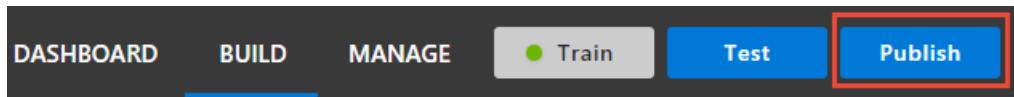
2. Training is complete when you see the green status bar at the top of the website confirming success.



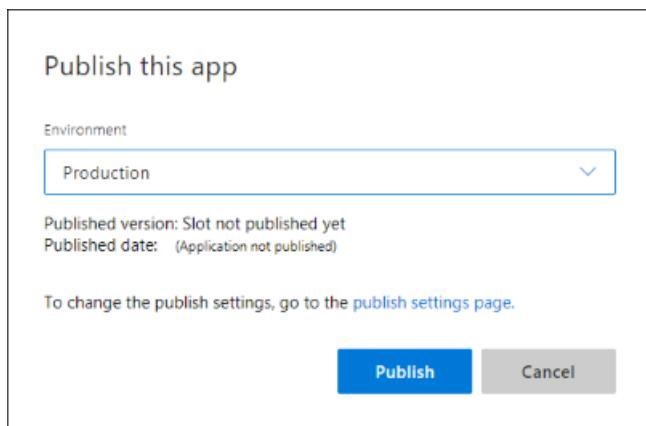
Publish again

In order to receive a LUIS prediction in a chat bot or other client application, you need to publish the app to the endpoint.

1. Select **Publish** in the top right navigation.



2. Select the **Production** slot and the **Publish** button.



3. Publishing is complete when you see the green status bar at the top of the website confirming success.



4. Select the **endpoints** link in the green status bar to go to the **Keys and endpoints** page. The endpoint URLs are listed at the bottom.

Get intent prediction again

1. In the **Manage** section (top right menu), on the **Keys and endpoints** page (left menu), select the **endpoint** URL at the bottom of the page. This action opens another browser tab with the endpoint URL in the address bar.

The endpoint URL looks like

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?verbose=true&subscription-key=<YOUR_KEY>&optional-name-value-pairs>&q=<user-utterance-text>
```

2. In the new browser window, enter `Can I submit my resume for job 235986` at the end of the URL.

```
{
  "query": "Can I submit my resume for job 235986",
  "topScoringIntent": {
    "intent": "ApplyForJob",
    "score": 0.9634406
  },
  "intents": [
    {
      "intent": "ApplyForJob",
      "score": 0.9634406
    },
    {
      "intent": "GetJobInformation",
      "score": 0.0171300638
    },
    {
      "intent": "None",
      "score": 0.00670867041
    }
  ],
  "entities": []
}
```

The results include the new intent **ApplyForJob** as well as the existing intents.

Client-application next steps

After LUIS returns the JSON response, LUIS is done with this request. LUIS doesn't provide answers to user utterances, it only identifies what type of information is being asked for in natural language. The conversational follow-up is provided by the client application such as an Azure Bot.

Clean up resources

When no longer needed, delete the LUIS app. To do so, select **My apps** from the top left menu. Select the ellipsis (...) to the right of the app name in the app list, select **Delete**. On the pop-up dialog **Delete app?**, select **Ok**.

Related information

- [Types of entities](#)
- [How to train](#)
- [How to publish](#)
- [How to test in LUIS portal](#)
- [Azure Bot](#)

Next steps

This tutorial created the Human Resources (HR) app, created 2 intents, added example utterances to each intent, added example utterances to the None intent, trained, published, and tested at the endpoint. These are the basic steps of building a LUIS model.

Continue with this app, [adding a simple entity and phrase list](#).

[Add prebuilt intents and entities to this app](#)

Tutorial: Identify common intents and entities

8/21/2019 • 4 minutes to read • [Edit Online](#)

In this tutorial, add prebuilt intents and entities to a Human Resources tutorial app to quickly gain intent prediction and data extraction. You do not need to mark any utterances with prebuilt entities because the entity is detected automatically.

Prebuilt models (domains, intents, and entities) help you build your model quickly.

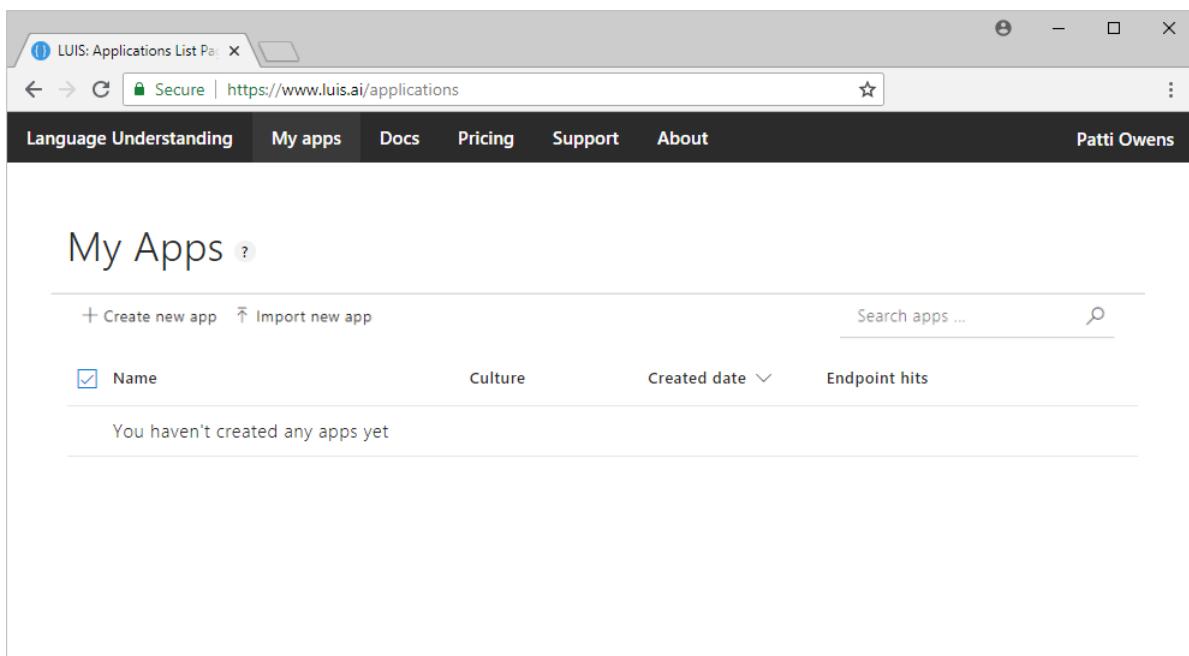
In this tutorial, you learn how to:

- Create new app
- Add prebuilt intents
- Add prebuilt entities
- Train
- Publish
- Get intents and entities from endpoint

For this article, you can use the free [LUIS](#) account in order to author your LUIS application.

Create a new app

1. Sign in to the LUIS portal with the URL of <https://www.luis.ai>.
2. Select **Create new app**.



3. In the pop-up dialog, enter the name `HumanResources` and keep the default culture, **English**. Leave the description empty.

Create new app

Name (Required)
HumanResources

Culture (Required)
English

** Culture is the language that your app understands and speaks, not the interface language.

Description
Type app description

Done **Cancel**

The screenshot shows a 'Create new app' dialog box. It has fields for 'Name (Required)' containing 'HumanResources', 'Culture (Required)' set to 'English', and a note about culture. There's a 'Description' field with placeholder text 'Type app description'. At the bottom are 'Done' and 'Cancel' buttons.

4. Select **Done**.

Add prebuilt intents to help with common user intentions

LUIS provides several prebuilt intents to help with common user intentions.

1. Make sure your Human Resources app is in the **Build** section of LUIS. You can change to this section by selecting **Build** on the top, right menu bar.
2. Select **Add prebuilt domain intent**.
3. Search for **utilities**.
4. Select all the intents and select **Done**. These intents are helpful to determine where, in the conversation, the user is and what they are asking to do.

Add prebuilt entities to help with common data type extraction

LUIS provides several prebuilt entities for common data extraction.

1. Select **Entities** from the left navigation menu.
2. Select **Add prebuilt entity** button.
3. Select the following entities from the list of prebuilt entities then select **Done**:
 - **PersonName**
 - **GeographyV2**

Add prebuilt entities

When you add a built-in entity, its predictions will be available to you while labeling utterances.

The screenshot shows a dialog box titled "Add prebuilt entities". At the top is a search bar labeled "Search built-in entities" with a magnifying glass icon. Below the search bar is a list of entities with checkboxes:

- datetimeV2**
Dates and times, resolved to a canonical form
June 23, 1976, Jul 11 2012, 7 AM, 6:49 PM, tomorrow at 7 AM
- keyPhrase**
Automatically extract key phrases to quickly identify the main talking points
Paris, cinema, soccer
- personName**
A person's partial or full name
carol, john, lili
- geographyV2**

At the bottom are two buttons: "Done" (blue) and "Cancel" (gray).

These entities will help you add name and place recognition to your client application.

Add example utterances to the None intent

The client application needs to know if an utterance is not meaningful or appropriate for the application. The **None** intent is added to each application as part of the creation process to determine if an utterance can't be answered by the client application.

If LUIS returns the **None** intent for an utterance, your client application can ask if the user wants to end the conversation or give more directions for continuing the conversation.

Caution

Do not leave the **None** intent empty.

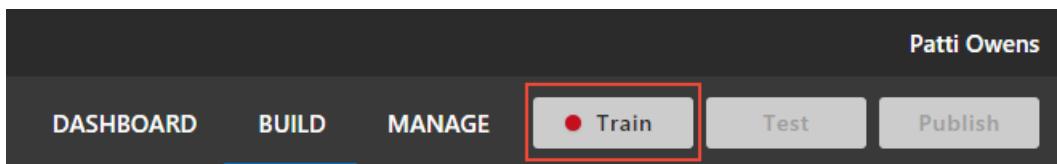
1. Select **Intents** from the left panel.
2. Select the **None** intent. Add three utterances that your user might enter but are not relevant to your Human Resources app:

The screenshot shows a text input field labeled "EXAMPLE UTTERANCES". It contains three entries separated by horizontal lines:

- Barking dogs are annoying
- Order a pizza for me
- Penguins in the ocean

Train the app so the changes to the intent can be tested

1. In the top right side of the LUIS website, select the **Train** button.



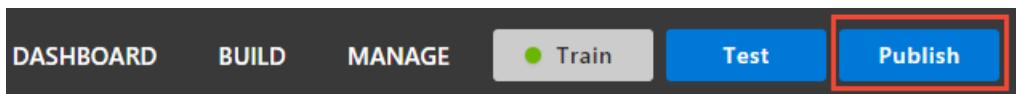
2. Training is complete when you see the green status bar at the top of the website confirming success.



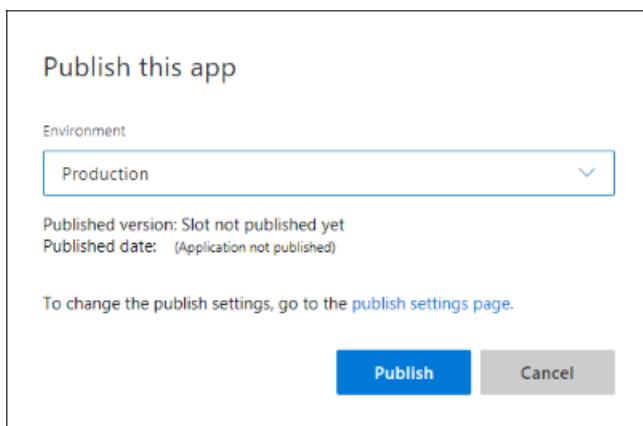
Publish the app so the trained model is queryable from the endpoint

In order to receive a LUIS prediction in a chat bot or other client application, you need to publish the app to the endpoint.

1. Select **Publish** in the top right navigation.



2. Select the **Production** slot and the **Publish** button.



3. Publishing is complete when you see the green status bar at the top of the website confirming success.



4. Select the **endpoints** link in the green status bar to go to the **Keys and endpoints** page. The endpoint URLs are listed at the bottom.

Get intent and entity prediction from endpoint

1. In the **Manage** section (top right menu), on the **Keys and endpoints** page (left menu), select the **endpoint** URL at the bottom of the page. This action opens another browser tab with the endpoint URL in the address bar.

The endpoint URL looks like

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?verbose=true&subscription-key=<YOUR_KEY>&<optional-name-value-pairs>&q=<user-utterance-text>
```

2. Go to the end of the URL in the browser address bar and enter

I want to cancel my trip to Seattle to see Bob Smith. The last query string parameter is q, the utterance **query**.

```
{
```

```
"query": "I want to cancel my trip to Seattle to see Bob Smith.",  
"topScoringIntent": {  
    "intent": "Utilities.ReadAloud",  
    "score": 0.100361854  
},  
"intents": [  
    {  
        "intent": "Utilities.ReadAloud",  
        "score": 0.100361854  
    },  
    {  
        "intent": "Utilities.Stop",  
        "score": 0.08102781  
    },  
    {  
        "intent": "Utilities.SelectNone",  
        "score": 0.0398852825  
    },  
    {  
        "intent": "Utilities.Cancel",  
        "score": 0.0277276486  
    },  
    {  
        "intent": "Utilities.SelectedItem",  
        "score": 0.0220712926  
    },  
    {  
        "intent": "Utilities.StartOver",  
        "score": 0.0145813478  
    },  
    {  
        "intent": "None",  
        "score": 0.012434179  
    },  
    {  
        "intent": "Utilities.Escalate",  
        "score": 0.0122632384  
    },  
    {  
        "intent": "Utilities.ShowNext",  
        "score": 0.008534077  
    },  
    {  
        "intent": "Utilities.ShowPrevious",  
        "score": 0.00547111453  
    },  
    {  
        "intent": "Utilities.SelectAny",  
        "score": 0.00152912608  
    },  
    {  
        "intent": "Utilities.Repeat",  
        "score": 0.0005556819  
    },  
    {  
        "intent": "Utilities.FinishTask",  
        "score": 0.000169488427  
    },  
    {  
        "intent": "Utilities.Confirm",  
        "score": 0.000149565312  
    },  
    {  
        "intent": "Utilities.GoBack",  
        "score": 0.000141017343  
    },  
    {  
        "intent": "Utilities.Reject",  
        "score": 6.27324E-06
```

```
        },
    ],
    "entities": [
        {
            "entity": "seattle",
            "type": "builtin.geographyV2.city",
            "startIndex": 28,
            "endIndex": 34
        },
        {
            "entity": "bob smith",
            "type": "builtin.personName",
            "startIndex": 43,
            "endIndex": 51
        }
    ]
}
```

The result predicted the Utilities.Cancel intent with 80% confidence and extracted the city and person name data.

Clean up resources

When no longer needed, delete the LUIS app. To do so, select **My apps** from the top left menu. Select the ellipsis (...) to the right of the app name in the app list, select **Delete**. On the pop-up dialog **Delete app?**, select **Ok**.

Related information

Learn more about prebuilt models:

- [Prebuilt domains](#): these are common domains that reduce overall LUIS app authoring
- Prebuilt intents: these are the individual intents of the common domains. You can add intents individually instead of adding the entire domain.
- [Prebuilt entities](#): these are common data types useful to most LUIS apps.

Learn more about working with your LUIS app:

- [How to train](#)
- [How to publish](#)
- [How to test in LUIS portal](#)

Next steps

By adding prebuilt intents and entities, the client application can determine common user intentions and extract common datatypes.

[Add a regular expression entity to the app](#)

Tutorial: Get sentiment of utterance

7/26/2019 • 5 minutes to read • [Edit Online](#)

In this tutorial, create an app that demonstrates how to determine positive, negative, and neutral sentiment from utterances. Sentiment is determined from the entire utterance.

In this tutorial, you learn how to:

- Create a new app
- Add sentiment analysis as publish setting
- Train app
- Publish app
- Get sentiment of utterance from endpoint

For this article, you can use the free [LUIS](#) account in order to author your LUIS application.

Sentiment analysis is a publish setting

The following utterances show examples of sentiment:

SENTIMENT	SCORE	UTTERANCE
positive	0.91	John W. Smith did a great job on the presentation in Paris.
positive	0.84	The Seattle engineers did fabulous work on the Parker sales pitch.

Sentiment analysis is a publish setting that applies to every utterance. You do not have to find the words indicating sentiment in the utterance and mark them.

Because it is a publish setting, you do not see it on the intents or entities pages. You can see it in the [interactive test](#) pane or when testing at the endpoint URL.

Create a new app

1. Sign in to the LUIS portal with the URL of <https://www.luis.ai>.
2. Select **Create new app**.

The screenshot shows the LUIS Applications List Page. At the top, there are navigation links for Language Understanding, My apps, Docs, Pricing, Support, and About. A user profile for Patti Owens is visible on the right. The main section is titled "My Apps" with a question mark icon. Below it are buttons for "Create new app" and "Import new app". A search bar labeled "Search apps ..." with a magnifying glass icon is also present. The table header includes columns for Name, Culture, Created date, and Endpoint hits, with "Name" having a checked checkbox. The message "You haven't created any apps yet" is displayed below the table.

3. In the pop-up dialog, enter the name **HumanResources** and keep the default culture, **English**. Leave the description empty.

The dialog box is titled "Create new app". It has three input fields: "Name (Required)" containing "HumanResources", "Culture (Required)" set to "English", and "Description" which is empty. Below the fields is a note: "Culture is the language that your app understands and speaks, not the interface language." At the bottom are two buttons: "Done" (highlighted in blue) and "Cancel".

4. Select **Done**.

Add PersonName prebuilt entity

1. Select **Build** from the navigation menu.
2. Select **Entities** from the left navigation menu.
3. Select **Add prebuilt entity** button.
4. Select the following entity from the list of prebuilt entities then select **Done**:
 - **PersonName**

Add prebuilt entities

When you add a built-in entity, its predictions will be available to you while labeling utterances.

datetimeV2
Dates and times, resolved to a canonical form
June 23, 1976, Jul 11 2012, 7 AM, 6:49 PM, tomorrow at 7 AM

keyPhrase
Automatically extract key phrases to quickly identify the main talking points
Paris, cinema, soccer

personName
A person's partial or full name
carol, john, lili

geographyV2

Done Cancel

Create an intent to determine employee feedback

Add a new intent to capture employee feedback from members of the company.

1. Select **Intents** from the left panel.
2. Select **Create new intent**.
3. Name the new intent name **EmployeeFeedback**.

Create new intent

Intent name (Required)

Done **Cancel**

4. Add several utterances that indicate an employee doing something well or an area that needs improvement:

UTTERANCES

John Smith did a nice job of welcoming back a co-worker from maternity leave

Jill Jones did a great job of comforting a co-worker in her time of grief.

Bob Barnes didn't have all the required invoices for the paperwork.

Todd Thomas turned in the required forms a month late with no signatures

UTTERANCES

Katherine Kelly didn't make it to the important marketing off-site meeting.

Denise Dillard missed the meeting for June reviews.

Mark Mathews rocked the sales pitch at Harvard

Walter Williams did a great job on the presentation at Stanford

Select the **View options**, select **Show entity values** to see the names.

The screenshot shows the LUIS interface for the 'EmployeeFeedback' intent. On the left, there's a sidebar with sections like 'App Assets', 'Intents' (which is selected), 'Entities', 'Improve app performance', 'Review endpoint utterances', 'Phrase lists', and 'Patterns'. The main area has a title 'EmployeeFeedback' with a pencil icon. Below it, 'Labelled entities:' is listed. There are buttons for 'Edit', 'Reassign intent', 'Add as pattern', 'Delete utterance(s)', 'Search', 'Filter', and 'Show entity values' (which is highlighted with a red box). A section for 'Example utterance' allows users to enter sample user input. A table lists several examples with their scores: 'walter williams' (-1.00), 'mark mathews' (-1.00), 'denise dillard' (-1.00), 'katherine kelly' (-1.00), 'todd thomas' (-1.00), 'bob barnes' (-1.00), 'jill jones' (-1.00), and 'john smith' (-1.00).

Add example utterances to the None intent

The client application needs to know if an utterance is not meaningful or appropriate for the application. The **None** intent is added to each application as part of the creation process to determine if an utterance can't be answered by the client application.

If LUIS returns the **None** intent for an utterance, your client application can ask if the user wants to end the conversation or give more directions for continuing the conversation.

Caution

Do not leave the **None** intent empty.

1. Select **Intents** from the left panel.
2. Select the **None** intent. Add three utterances that your user might enter but are not relevant to your Human Resources app:

EXAMPLE UTTERANCES

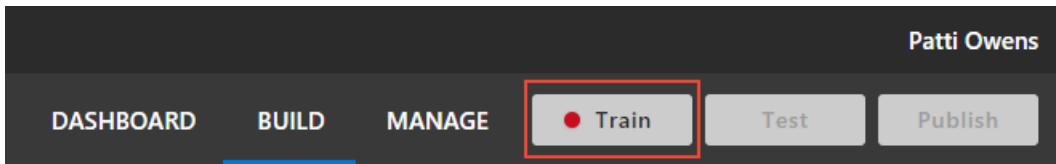
Barking dogs are annoying

Order a pizza for me

Penguins in the ocean

Train the app so the changes to the intent can be tested

1. In the top right side of the LUIS website, select the **Train** button.



2. Training is complete when you see the green status bar at the top of the website confirming success.



Configure app to include sentiment analysis

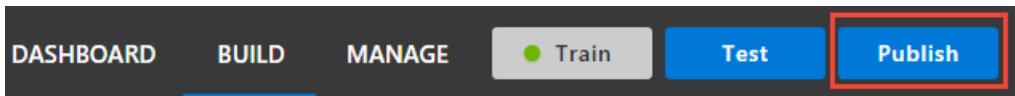
1. Select **Manage** in the top right navigation, then select **Publish settings** from the left menu.
2. Select **Use sentiment analysis to determine if a user's utterance is positive, negative, or neutral.** to enable this setting.

A screenshot of the LUIS website's Publish settings page. On the left, there's a sidebar with options like Application Settings, Publish Settings (which is selected and highlighted with a blue border), Versions, and Collaborators. The main content area is titled "Publish settings". It contains a section for "External services" with two checkboxes: "Use sentiment analysis to determine if a user's utterance is positive, negative, or neutral." (highlighted with a red box) and "Enable speech priming to allow a single request to receive audio and return LUIS prediction JSONobjects." A note at the bottom says "Ready to turn this into a bot? You can empower a bot with natural language using this app. Learn more about how to integrate LUIS with a bot.".

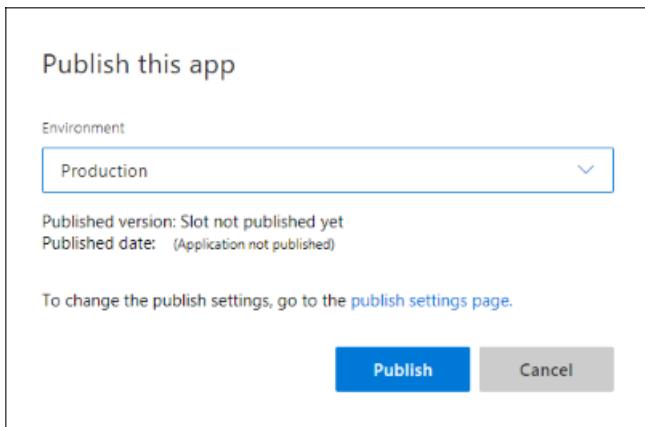
Publish the app so the trained model is queryable from the endpoint

In order to receive a LUIS prediction in a chat bot or other client application, you need to publish the app to the endpoint.

1. Select **Publish** in the top right navigation.



2. Select the **Production** slot and the **Publish** button.



3. Publishing is complete when you see the green status bar at the top of the website confirming success.



Publishing complete. Refer to the list of endpoints to access your endpoint URL

4. Select the **endpoints** link in the green status bar to go to the **Keys and endpoints** page. The endpoint URLs are listed at the bottom.

Get the sentiment of an utterance from the endpoint

1. In the **Manage** section (top right menu), on the **Keys and endpoints** page (left menu), select the **endpoint** URL at the bottom of the page. This action opens another browser tab with the endpoint URL in the address bar.

The endpoint URL looks like

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?verbose=true&subscription-key=<YOUR_KEY>&<optional-name-value-pairs>&q=<user-utterance-text>
```

2. Go to the end of the URL in the address and enter the following utterance:

```
Jill Jones work with the media team on the public portal was amazing
```

The last querystring parameter is `q`, the utterance **query**. This utterance is not the same as any of the labeled utterances so it is a good test and should return the `EmployeeFeedback` intent with the sentiment analysis extracted.

```
{
  "query": "Jill Jones work with the media team on the public portal was amazing",
  "topScoringIntent": {
    "intent": "EmployeeFeedback",
    "score": 0.9616192
  },
  "intents": [
    {
      "intent": "EmployeeFeedback",
      "score": 0.9616192
    },
    {
      "intent": "None",
      "score": 0.09347677
    }
  ],
  "entities": [
    {
      "entity": "jill jones",
      "type": "builtin.personName",
      "startIndex": 0,
      "endIndex": 9
    }
  ],
  "sentimentAnalysis": {
    "label": "positive",
    "score": 0.8694164
  }
}
```

The sentimentAnalysis is positive with a score of 86%.

Try another utterance by removing the value for in the address bar of the browser:

The sentiment score indicates a negative sentiment by returning a low score .

Clean up resources

When no longer needed, delete the LUIS app. To do so, select **My apps** from the top left menu. Select the ellipsis (...) to the right of the app name in the app list, select **Delete**. On the pop-up dialog **Delete app?**, select **Ok**.

Related information

- Sentiment analysis is provided by Cognitive Service [Text Analytics](#). The feature is restricted to [Text Analytics supported languages](#).
- [How to train](#)
- [How to publish](#)
- [How to test in LUIS portal](#)

Next steps

This tutorial adds sentiment analysis as a publish setting to extract sentiment values from the utterance as a whole.

[Review endpoint utterances in the HR app](#)

Tutorial: Get well-formatted data from the utterance

7/26/2019 • 5 minutes to read • [Edit Online](#)

In this tutorial, create an app to extract consistently-formatted data from an utterance using the **Regular Expression** entity.

In this tutorial, you learn how to:

- Create a new app
- Add intent
- Add regular expression entity
- Train
- Publish
- Get intents and entities from endpoint

For this article, you can use the free [LUIS](#) account in order to author your LUIS application.

Regular expression entities

This app's use of the regular expression entity is to pull out well-formatted Human Resources (HR) form numbers from an utterance. While the utterance's intent is always determined with machine-learning, this specific entity type is not machine-learned.

Example utterances include:

EXAMPLE UTTERANCES

Where is HRF-123456?

Who authored HRF-123234?

HRF-456098 is published in French?

HRF-456098

HRF-456098 date?

A regular expression is a good choice for this type of data when:

- the data is well-formatted.

Create a new app

1. Sign in to the LUIS portal with the URL of <https://www.luis.ai>.
2. Select **Create new app**.

Luis: Applications List Page

Secure | https://www.luis.ai/applications

Language Understanding My apps Docs Pricing Support About Patti Owens

My Apps

+ Create new app ⌂ Import new app Search apps ...

Name Culture Created date ↴ Endpoint hits

You haven't created any apps yet

3. In the pop-up dialog, enter the name **HumanResources** and keep the default culture, **English**. Leave the description empty.

Create new app

Name (Required)
HumanResources

Culture (Required)
English

** Culture is the language that your app understands and speaks, not the interface language.

Description
Type app description

Done Cancel

4. Select **Done**.

Create intent for finding form

1. Make sure your Human Resources app is in the **Build** section of LUIS. You can change to this section by selecting **Build** on the top, right menu bar.
2. Select **Create new intent**.
3. Enter **FindForm** in the pop-up dialog box then select **Done**.

Create new intent

Intent name (Required)
FindForm

Done Cancel

4. Add example utterances to the intent.

EXAMPLE UTTERANCES
What is the URL for hrf-123456?
Where is hrf-345678?
When was hrf-456098 updated?
Did John Smith update hrf-234639 last week?
How many versions of hrf-345123 are there?
Who needs to authorize form hrf-123456?
How many people need to sign off on hrf-345678?
hrf-234123 date?
author of hrf-546234?
title of hrf-456234?

App Assets

- Intents
- Entities

Improve app performance

- Review endpoint utterances
- Phrase lists
- Patterns

FindForm ↗

Labelled entities:

Example utterance Score ? Enter an example of what a user might say and hit Enter

title of hrf - 456234 ?	-1.00
author of hrf - 546234 ?	-1.00
hrf - 234123 date ?	-1.00
how many people need to sign off on hrf - 345678 ?	-1.00
who needs to authorize form hrf - 123456 ?	-1.00
how many versions of hrf - 345123 are there ?	-1.00
did john smith update hrf - 234639 last week ?	-1.00
when was hrf - 456098 updated ?	-1.00
where is hrf - 345678 ?	-1.00
what is the url for hrf - 123456 ?	-1.00

These few utterances are for demonstration purposes only. A real-world app should have at least 15 utterances of varying length, word order, tense, grammatical correctness, punctuation, and word count.

Use the regular expression entity for well-formatted data

The regular expression entity to match the form number is `hrf-[0-9]{6}`. This regular expression matches the literal characters `hrf-` but ignores case and culture variants. It matches digits 0-9, for 6 digits exactly.

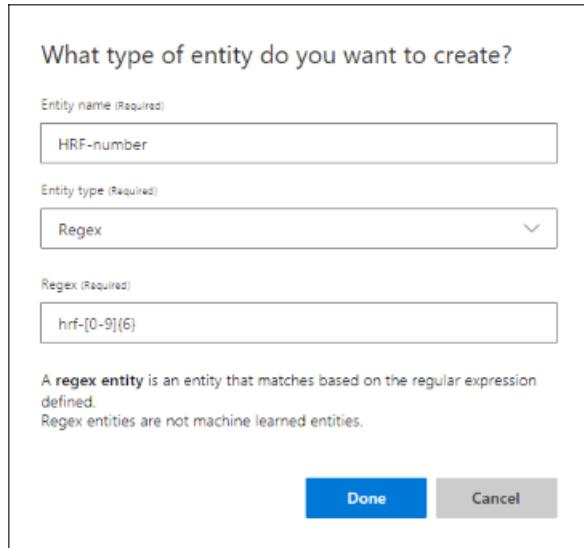
HRF stands for `human resources form`.

Luis tokenizes the utterance when it is added to an intent. The tokenization for these utterances adds spaces before and after the hyphen, `Where is HRF - 123456?` The regular expression is applied to the utterance in its raw

form, before it is tokenized. Because it is applied to the *raw* form, the regular expression doesn't have to deal with word boundaries.

Create a regular expression entity to tell LUIS what an HRF-number format is in the following steps:

1. Select **Entities** in the left panel.
2. Select **Create new entity** button on the Entities Page.
3. In the pop-up dialog, enter the new entity name **HRF-number**, select **Regex** as the entity type, enter **hrf-[0-9]{6}** as the **Regex** value, and then select **Done**.



4. Select **Intents** from the left menu, then **FindForm** intent to see the regular expression labeled in the utterances.

The screenshot shows the "FindForm" intent page. At the top, there's a header "FindForm" with a pencil icon, followed by "Labelled entities:" and a toolbar with icons for Edit, Reassign intent, Add as pattern, Delete utterance(s), Search, Filter, and View options. Below this is a table of utterances:

Example utterance	Score
title of HRF-number ?	-1.00
author of HRF-number ?	-1.00
HRF-number date ?	-1.00
how many people need to sign off on HRF-number ?	-1.00
who needs to authorize form HRF-number ?	-1.00
how many versions of HRF-number are there ?	-1.00
did john smith update HRF-number last week ?	-1.00
when was HRF-number updated ?	-1.00
where is HRF-number ?	-1.00
what is the url for HRF-number ?	-1.00

Because the entity is not a machine-learned entity, the entity is applied to the utterances and displayed in the LUIS website as soon as it is created.

Add example utterances to the None intent

The client application needs to know if an utterance is not meaningful or appropriate for the application. The **None** intent is added to each application as part of the creation process to determine if an utterance can't be answered by the client application.

If LUIS returns the **None** intent for an utterance, your client application can ask if the user wants to end the conversation or give more directions for continuing the conversation.

Caution

Do not leave the **None** intent empty.

1. Select **Intents** from the left panel.
2. Select the **None** intent. Add three utterances that your user might enter but are not relevant to your Human Resources app:

EXAMPLE UTTERANCES

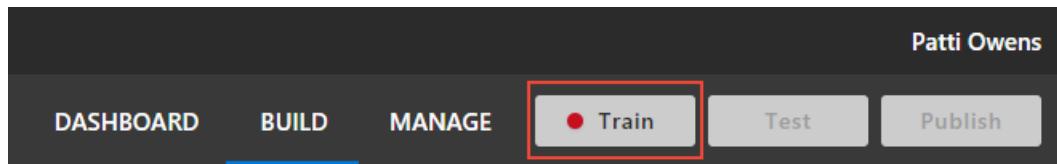
Barking dogs are annoying

Order a pizza for me

Penguins in the ocean

Train the app before testing or publishing

1. In the top right side of the LUIS website, select the **Train** button.



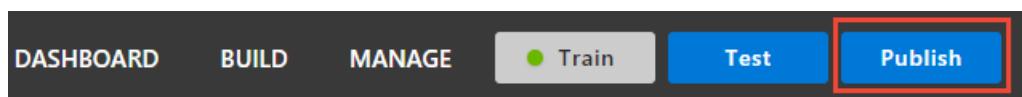
2. Training is complete when you see the green status bar at the top of the website confirming success.



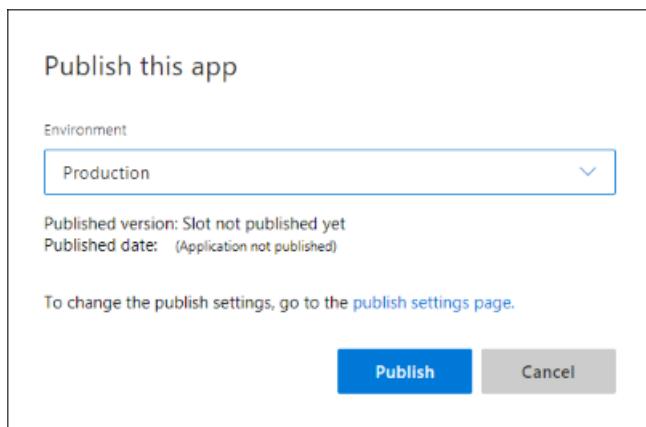
Publish the app to query from the endpoint

In order to receive a LUIS prediction in a chat bot or other client application, you need to publish the app to the endpoint.

1. Select **Publish** in the top right navigation.



2. Select the **Production** slot and the **Publish** button.



3. Publishing is complete when you see the green status bar at the top of the website confirming success.

Publishing complete. Refer to the list of endpoints to access your endpoint URL

4. Select the **endpoints** link in the green status bar to go to the **Keys and endpoints** page. The endpoint URLs are listed at the bottom.

Get intent and entity prediction from endpoint

1. In the **Manage** section (top right menu), on the **Keys and endpoints** page (left menu), select the **endpoint** URL at the bottom of the page. This action opens another browser tab with the endpoint URL in the address bar.

The endpoint URL looks like

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?verbose=true&subscription-key=<YOUR_KEY>&<optional-name-value-pairs>&q=<user-utterance-text>
```

2. Go to the end of the URL in the address and enter the following utterance:

```
When were HRF-123456 and hrf-234567 published in the last year?
```

The last querystring parameter is `q`, the utterance **query**. This utterance is not the same as any of the labeled utterances so it is a good test and should return the `FindForm` intent with the two form numbers of `HRF-123456` and `hrf-234567`.

```
{
  "query": "When were HRF-123456 and href-234567 published in the last year?",
  "topScoringIntent": {
    "intent": "FindForm",
    "score": 0.9988884
  },
  "intents": [
    {
      "intent": "FindForm",
      "score": 0.9988884
    },
    {
      "intent": "None",
      "score": 0.00204812363
    }
  ],
  "entities": [
    {
      "entity": "href-123456",
      "type": "HRF-number",
      "startIndex": 10,
      "endIndex": 19
    },
    {
      "entity": "href-234567",
      "type": "HRF-number",
      "startIndex": 25,
      "endIndex": 34
    }
  ]
}
```

By using a regular expression entity, LUIS extracts named data, which is more programmatically helpful to the client application receiving the JSON response.

Clean up resources

When no longer needed, delete the LUIS app. To do so, select **My apps** from the top left menu. Select the ellipsis (...) to the right of the app name in the app list, select **Delete**. On the pop-up dialog **Delete app?**, select **Ok**.

Related information

- [Regular expression entity concepts](#)
- [How to train](#)
- [How to publish](#)
- [How to test in LUIS portal](#)

Next steps

This tutorial created a new intent, added example utterances, then created a regular expression entity to extract well-formatted data from the utterances. After training, and publishing the app, a query to the endpoint identified the intention and returned the extracted data.

[Learn about the list entity](#)

Tutorial: Get exact text-matched data from an utterance

7/26/2019 • 5 minutes to read • [Edit Online](#)

In this tutorial, understand how to get entity data that matches a predefined list of items.

In this tutorial, you learn how to:

- Create app
- Add intent
- Add list entity
- Train
- Publish
- Get intents and entities from endpoint

For this article, you can use the free [LUIS](#) account in order to author your LUIS application.

What is a list entity?

A list entity is an exact text match to the words in the utterance.

Each item on the list can include a list of synonyms. For the human resources app, a company department can be identified by several key pieces of information such as an official name, common acronyms, and billing department codes.

The Human Resources app needs to determine the department an employee is transferring to.

A list entity is a good choice for this type of data when:

- The data values are a known set.
- The set doesn't exceed the maximum LUIS [boundaries](#) for this entity type.
- The text in the utterance is an exact match with a synonym or the canonical name. LUIS doesn't use the list beyond exact text matches. Stemming, plurals, and other variations are not resolved with just a list entity. To manage variations, consider using a [pattern](#) with the optional text syntax.

Create a new app

1. Sign in to the LUIS portal with the URL of <https://www.luis.ai>.
2. Select **Create new app**.

Luis: Applications List Page

Secure | https://www.luis.ai/applications

Language Understanding My apps Docs Pricing Support About Patti Owens

My Apps

+ Create new app ⌂ Import new app

Search apps ...

Name Culture Created date Endpoint hits

You haven't created any apps yet

3. In the pop-up dialog, enter the name `HumanResources` and keep the default culture, **English**. Leave the description empty.

Create new app

Name (Required)
HumanResources

Culture (Required)
English

** Culture is the language that your app understands and speaks, not the interface language.

Description
Type app description

Done Cancel

4. Select **Done**.

Create an intent to transfer employees to a different department

1. Make sure your Human Resources app is in the **Build** section of LUIS. You can change to this section by selecting **Build** on the top, right menu bar.
2. Select **Create new intent**.
3. Enter `TransferEmployeeToDepartment` in the pop-up dialog box then select **Done**.

Create new intent

Intent name (Required)

Done
Cancel

4. Add example utterances to the intent.

EXAMPLE UTTERANCES
move John W. Smith to the accounting department
transfer Jill Jones from to R&D
Dept 1234 has a new member named Bill Bradstreet
Place John Jackson in Engineering
move Debra Doughtery to Inside Sales
mv Jill Jones to IT
Shift Alice Anderson to DevOps
Carl Chamerlin to Finance
Steve Standish to 1234
Tanner Thompson to 3456

TransferEmployeeToDepartment	
Labelled entities:	
<input type="checkbox"/> Example utterance	Score ?
<input type="text" value="Enter an example of what a user might say and hit Enter"/>	
tanner thompson to 3456	-1.00
steve standish to 1234	-1.00
carl chamerlin to finance	-1.00
shift alice anderson to devops	-1.00
mv jill jones to it	-1.00
move debra doughtery to inside sales	-1.00
place john jackson in engineering	-1.00
dept 1234 has a new member named bill bradstreet	-1.00
transfer jill jones from to r & d	-1.00
move john w. smith to the accounting department	-1.00

These few utterances are for demonstration purposes only. A real-world app should have at least 15 utterances of varying length, word order, tense, grammatical correctness, punctuation, and word count.

Department list entity

Now that the **TransferEmployeeToDepartment** intent has example utterances, LUIS needs to understand what is a department.

The primary, *canonical*, name for each item is the department name. Examples of the synonyms of each canonical name are:

CANONICAL NAME	SYNONYMS
Accounting	acct acctg 3456
Development Operations	Devops 4949
Engineering	eng enging 4567
Finance	fin 2020
Information Technology	IT 2323
Inside Sales	isale insale 1414
Research and Development	R&D 1234

1. Select **Entities** in the left panel.
2. Select **Create new entity**.
3. In the entity pop-up dialog, enter **Department** for the entity name, and **List** for entity type. Select **Done**.

What type of entity do you want to create?

Entity name (Required)

Entity type (Required)

A **list entity** is a fixed list of values. Each value is itself a list of synonyms or other forms the value may take. For example, a list entity named PacificStates include the values Washington, Oregon, California. The Washington value then includes both "Washington" and the abbreviation "WA". Unlike other entity types, additional values for list entities aren't discovered during training. This entity type is identified in utterances by the direct matching of utterance text to the defined values, rather than learning from context.

Done **Cancel**

4. On the Department entity page, enter **Accounting** as the new value.
5. For Synonyms, add the synonyms from the previous table.
6. Continue adding all the canonical names and their synonyms.

Add example utterances to the None intent

The client application needs to know if an utterance is not meaningful or appropriate for the application. The **None** intent is added to each application as part of the creation process to determine if an utterance can't be answered by the client application.

If LUIS returns the **None** intent for an utterance, your client application can ask if the user wants to end the conversation or give more directions for continuing the conversation.

Caution

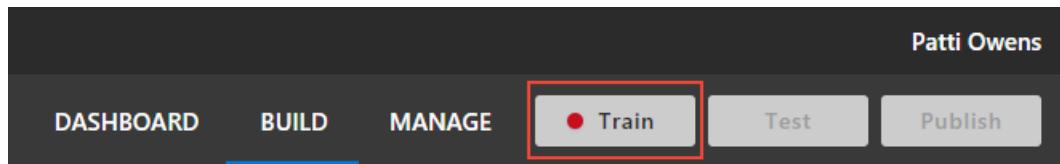
Do not leave the **None** intent empty.

1. Select **Intents** from the left panel.
2. Select the **None** intent. Add three utterances that your user might enter but are not relevant to your Human Resources app:

EXAMPLE UTTERANCES
Barking dogs are annoying
Order a pizza for me
Penguins in the ocean

Train the app so the changes to the intent can be tested

1. In the top right side of the LUIS website, select the **Train** button.



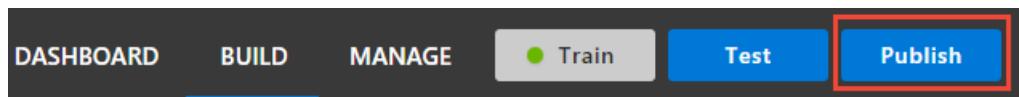
2. Training is complete when you see the green status bar at the top of the website confirming success.



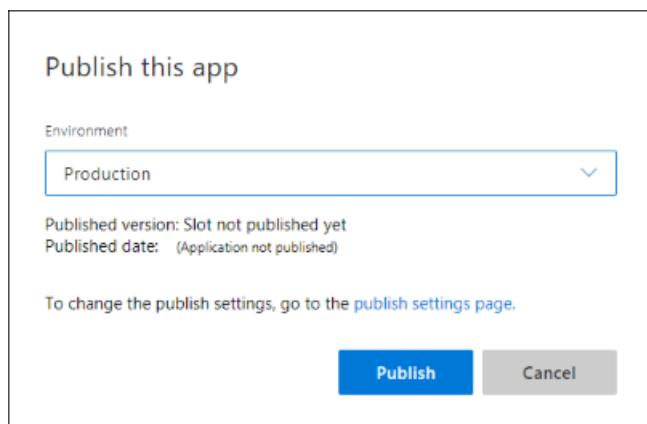
Publish the app so the trained model is queryable from the endpoint

In order to receive a LUIS prediction in a chat bot or other client application, you need to publish the app to the endpoint.

1. Select **Publish** in the top right navigation.



2. Select the **Production** slot and the **Publish** button.



3. Publishing is complete when you see the green status bar at the top of the website confirming success.



4. Select the **endpoints** link in the green status bar to go to the **Keys and endpoints** page. The endpoint URLs are listed at the bottom.

Get intent and entity prediction from endpoint

1. In the **Manage** section (top right menu), on the **Keys and endpoints** page (left menu), select the **endpoint** URL at the bottom of the page. This action opens another browser tab with the endpoint URL in the address bar.

The endpoint URL looks like

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?verbose=true&subscription-key=<YOUR_KEY>&<optional-name-value-pairs>&q=<user-utterance-text>
```

2. Go to the end of the URL in the address and enter `shift Joe Smith to IT`. The last querystring parameter is `q`, the utterance `query`. This utterance is not the same as any of the labeled utterances so it is a good test and should return the `TransferEmployeeToDepartment` intent with `Department` extracted.

```
{  
  "query": "shift Joe Smith to IT",  
  "topScoringIntent": {  
    "intent": "TransferEmployeeToDepartment",  
    "score": 0.9775754  
  },  
  "intents": [  
    {  
      "intent": "TransferEmployeeToDepartment",  
      "score": 0.9775754  
    },  
    {  
      "intent": "None",  
      "score": 0.0154493852  
    }  
  ],  
  "entities": [  
    {  
      "entity": "it",  
      "type": "Department",  
      "startIndex": 19,  
      "endIndex": 20,  
      "resolution": {  
        "values": [  
          "Information Technology"  
        ]  
      }  
    }  
  ]  
}
```

Clean up resources

When no longer needed, delete the LUIS app. To do so, select **My apps** from the top left menu. Select the ellipsis (...) to the right of the app name in the app list, select **Delete**. On the pop-up dialog **Delete app?**, select **Ok**.

Related information

- [List entity](#) conceptual information
- [How to train](#)
- [How to publish](#)
- [How to test in LUIS portal](#)

Next steps

This tutorial created a new intent, added example utterances, then created a list entity to extract exact text matches from utterances. After training, and publishing the app, a query to the endpoint identified the intention and returned the extracted data.

Continue with this app, [adding a composite entity](#).

[Add prebuilt entity with a role to the app](#)

Tutorial: Extract contextually related data from an utterance

7/26/2019 • 5 minutes to read • [Edit Online](#)

In this tutorial, find related pieces of data based on context. For example, an origin and destination locations for a transfer from one city to another. Both pieces of data may be required and they are related to each other.

A role can be used with any prebuilt or custom entity type, and used in both example utterances and patterns.

In this tutorial, you learn how to:

- Create new app
- Add intent
- Get origin and destination information using roles
- Train
- Publish
- Get intents and entity roles from endpoint

For this article, you can use the free [LUIS](#) account in order to author your LUIS application.

Related data

This app determines where an employee is to be moved from the origin city to the destination city. It uses a GeographyV2 prebuilt entity to identify the city names and it uses roles to determine the location types (origin and destination) within the utterance.

A role should be used when the entity data to extract:

- Is related to each other in the context of the utterance.
- Uses specific word choice to indicate each role. Examples of these words include: from/to, leavingheaded to, away from/toward.
- Both roles are frequently in the same utterance, allowing LUIS to learn from this frequent contextual usage.
- Need to be grouped and processed by client app as a unit of information.

Create a new app

1. Sign in to the LUIS portal with the URL of <https://www.luis.ai>.
2. Select **Create new app**.

The screenshot shows the LUIS Applications List page. At the top, there's a navigation bar with links for Language Understanding, My apps, Docs, Pricing, Support, and About. On the right, it shows the user's name, Patti Owens. Below the navigation, the title 'My Apps' is displayed with a question mark icon. Underneath, there are buttons for '+ Create new app' and 'Import new app'. A search bar with the placeholder 'Search apps ...' and a magnifying glass icon is also present. The main content area has filters: 'Name' (with a checked checkbox), 'Culture', 'Created date' (with a dropdown arrow), and 'Endpoint hits'. A message below the filters says 'You haven't created any apps yet'.

3. In the pop-up dialog, enter the name `HumanResources` and keep the default culture, **English**. Leave the description empty.

The dialog box is titled 'Create new app'. It contains fields for 'Name (Required)' with 'HumanResources' entered, 'Culture (Required)' with 'English' selected, and a 'Description' field with 'Type app description' placeholder text. At the bottom are 'Done' and 'Cancel' buttons.

4. Select **Done**.

Create an intent to move employees between cities

1. Make sure your Human Resources app is in the **Build** section of LUIS. You can change to this section by selecting **Build** on the top, right menu bar.
2. Select **Create new intent**.
3. Enter `MoveEmployeeToCity` in the pop-up dialog box then select **Done**.

The dialog box is titled 'Create new intent'. It has a single input field for 'Intent name (Required)' containing 'MoveEmployeeToCity'. At the bottom are 'Done' and 'Cancel' buttons.

4. Add example utterances to the intent.

EXAMPLE UTTERANCES

move John W. Smith leaving Seattle headed to Orlando

transfer Jill Jones from Seattle to Cairo

Place John Jackson away from Tampa, coming to Atlanta

move Debra Doughtery to Tulsa from Chicago

mv Jill Jones leaving Cairo headed to Tampa

Shift Alice Anderson to Oakland from Redmond

Carl Chamerlin from San Francisco to Redmond

Transfer Steve Standish from San Diego toward Bellevue

lift Tanner Thompson from Kansas city and shift to Chicago

MoveEmployeeToCity ↗

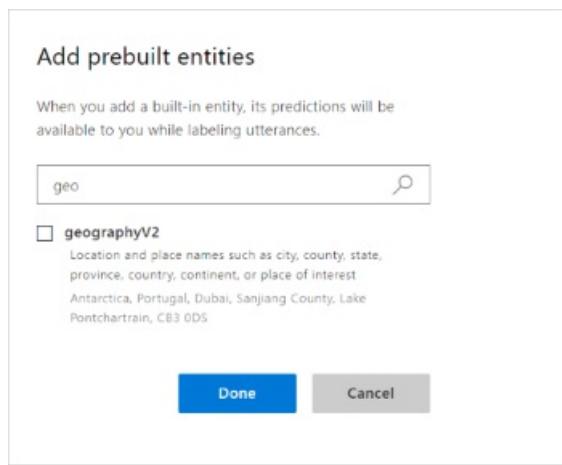
Labelled entities:

Edit		Reassign intent	Add as pattern	Delete utterance(s)	Search	Filter	View options
<input type="checkbox"/>	Example utterance				Score ?		
lift Tanner Thompson from Kansas city and shift to Chicago							
transfer steve standish from san diego toward bellevue -1.00							
carl chamerlin from san francisco to redmond -1.00							
shift alice anderson to oakland from redmond -1.00							
mv jill jones leaving cairo headed to tampa -1.00							
move debra doughtery to tulsa from chicago -1.00							
place john jackson away from tampa , coming to atlanta -1.00							
transfer jill jones from seattle to cairo -1.00							
move john w . smith leaving seattle headed to orlando -1.00							

Add prebuilt entity geographyV2

The prebuilt entity, geographyV2, extracts location information, including city names. Since the utterances have two city names, relating to each other in context, use roles to extract that context.

1. Select **Entities** from the left-side navigation.
2. Select **Add prebuilt entity**, then select **geo** in the search bar to filter the prebuilt entities.



3. Select the checkbox and select **Done**.
4. In the **Entities** list, select the **geographyV2** to open the new entity.
5. Add two roles, **Origin**, and **Destination**.

geographyV2
Entity type: Prebuilt

Delete Entity

Roles

Destination

Role name

Origin

6. Select **Intents** from the left-side navigation, then select the **MoveEmployeeToCity** intent. Notice the city names are labeled with the prebuilt entity **geographyV2**.
7. In the first utterance of the list, select the origin location. A drop-down menu appears. Select **geographyV2** in the list, then follow the menu across to select **Origin**.
8. Use the method from the previous step to mark all roles of locations in all the utterances.

Add example utterances to the None intent

The client application needs to know if an utterance is not meaningful or appropriate for the application. The **None** intent is added to each application as part of the creation process to determine if an utterance can't be answered by the client application.

If LUIS returns the **None** intent for an utterance, your client application can ask if the user wants to end the conversation or give more directions for continuing the conversation.

Caution

Do not leave the **None** intent empty.

1. Select **Intents** from the left panel.
2. Select the **None** intent. Add three utterances that your user might enter but are not relevant to your Human Resources app:

EXAMPLE UTTERANCES

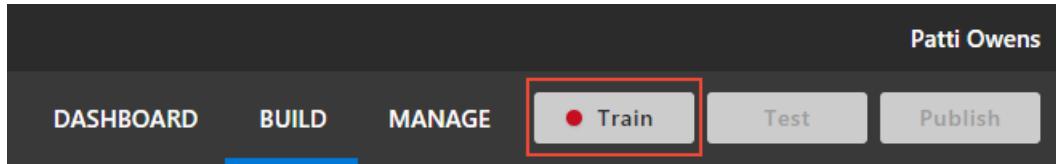
Barking dogs are annoying

Order a pizza for me

Penguins in the ocean

Train the app so the changes to the intent can be tested

1. In the top right side of the LUIS website, select the **Train** button.



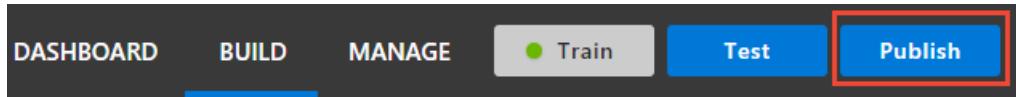
2. Training is complete when you see the green status bar at the top of the website confirming success.



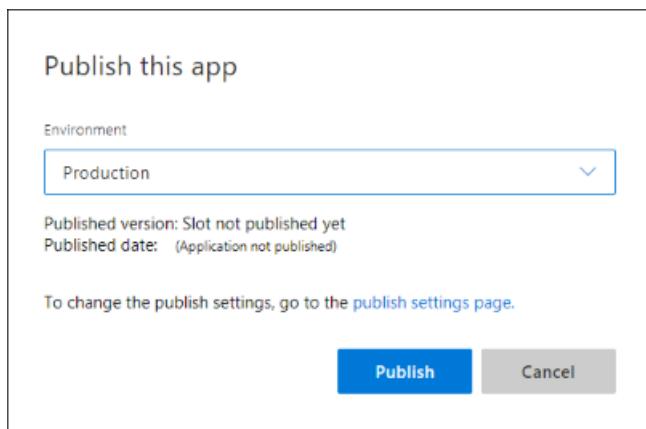
Publish the app so the trained model is queryable from the endpoint

In order to receive a LUIS prediction in a chat bot or other client application, you need to publish the app to the endpoint.

1. Select **Publish** in the top right navigation.



2. Select the **Production** slot and the **Publish** button.



3. Publishing is complete when you see the green status bar at the top of the website confirming success.



4. Select the **endpoints** link in the green status bar to go to the **Keys and endpoints** page. The endpoint URLs are listed at the bottom.

Get intent and entity prediction from endpoint

1. In the **Manage** section (top right menu), on the **Keys and endpoints** page (left menu), select the **endpoint**

URL at the bottom of the page. This action opens another browser tab with the endpoint URL in the address bar.

The endpoint URL looks like

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?verbose=true&subscription-key=<YOUR_KEY>&<optional-name-value-pairs>&q=<user-utterance-text>
```

2. Go to the end of the URL in the address bar and enter `Please move Carl Chamerlin from Tampa to Portland`.

The last querystring parameter is `q`, the utterance **query**. This utterance is not the same as any of the labeled utterances so it is a good test and should return the `MoveEmployee` intent with the entity extracted.

```
{
  "query": "Please move Carl Chamerlin from Tampa to Portland",
  "topScoringIntent": {
    "intent": "MoveEmployeeToCity",
    "score": 0.979823351
  },
  "intents": [
    {
      "intent": "MoveEmployeeToCity",
      "score": 0.979823351
    },
    {
      "intent": "None",
      "score": 0.0156363435
    }
  ],
  "entities": [
    {
      "entity": "geographyV2",
      "role": "Destination",
      "startIndex": 41,
      "endIndex": 48,
      "score": 0.6044041
    },
    {
      "entity": "geographyV2",
      "role": "Origin",
      "startIndex": 32,
      "endIndex": 36,
      "score": 0.739491045
    }
  ]
}
```

The correct intent is predicted and the entities array has both the origin and destination roles in the corresponding **entities** property.

Clean up resources

When no longer needed, delete the LUIS app. To do so, select **My apps** from the top left menu. Select the ellipsis (...) to the right of the app name in the app list, select **Delete**. On the pop-up dialog **Delete app?**, select **Ok**.

Related information

- [Entities concepts](#)
- [Roles concepts](#)
- [Prebuilt entities list](#)
- [How to train](#)

- [How to publish](#)
- [How to test in LUIS portal](#)
- [Roles](#)

Next steps

This tutorial created a new intent and added example utterances for the contextually learned data of origin and destination locations. Once the app is trained and published, a client-application can use that information to create a move ticket with the relevant information.

[Learn how to add a composite entity](#)

Tutorial: Group and extract related data

8/9/2019 • 5 minutes to read • [Edit Online](#)

In this tutorial, add a composite entity to bundle extracted data of various types into a single containing entity. By bundling the data, the client application can easily extract related data in different data types.

The purpose of the composite entity is to group related entities into a parent category entity. The information exists as separate entities before a composite is created.

The composite entity is a good fit for this type of data because the data:

- Are related to each other.
- Use a variety of entity types.
- Need to be grouped and processed by client app as a unit of information.

In this tutorial, you learn how to:

- Import example app
- Create intent
- Add composite entity
- Train
- Publish
- Get intents and entities from endpoint

For this article, you can use the free [LUIS](#) account in order to author your LUIS application.

Import example app

1. Download and save the [app JSON file](#) from the List entity tutorial.
2. Import the JSON into a new app.
3. From the **Manage** section, on the **Versions** tab, clone the version, and name it `composite`. Cloning is a great way to play with various LUIS features without affecting the original version. Because the version name is used as part of the URL route, the name can't contain any characters that are not valid in a URL.

Composite entity

In this app, the department name is defined in the **Department** list entity and includes synonyms.

The **TransferEmployeeToDepartment** intent has example utterances to request an employee be moved to a new department.

Example utterances for this intent include:

EXAMPLE UTTERANCES

move John W. Smith to the accounting department

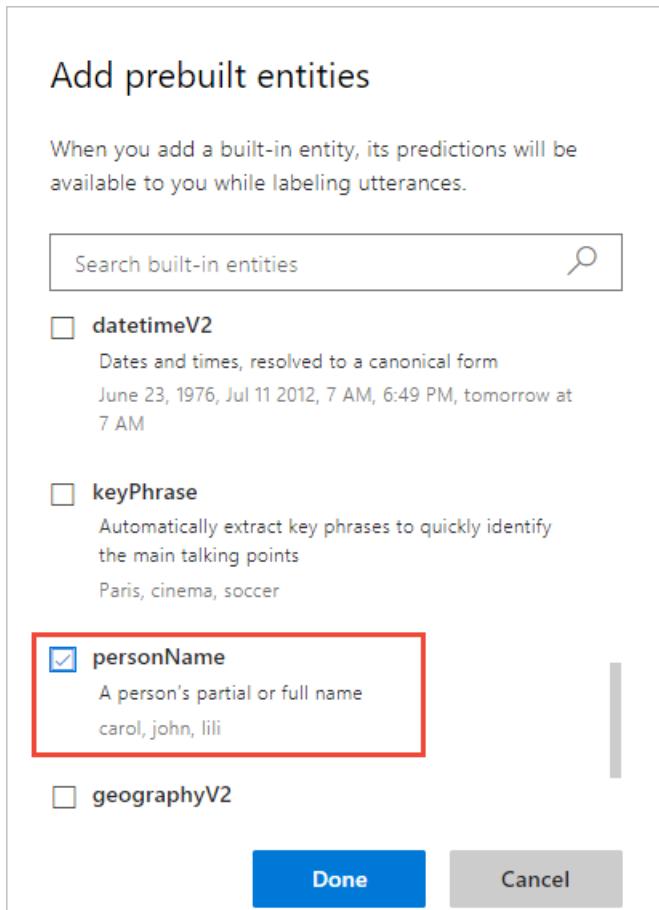
transfer Jill Jones from to R&D

The move request should include the department name, and the employee name.

Add the PersonName prebuilt entity to help with common data type extraction

LUIS provides several prebuilt entities for common data extraction.

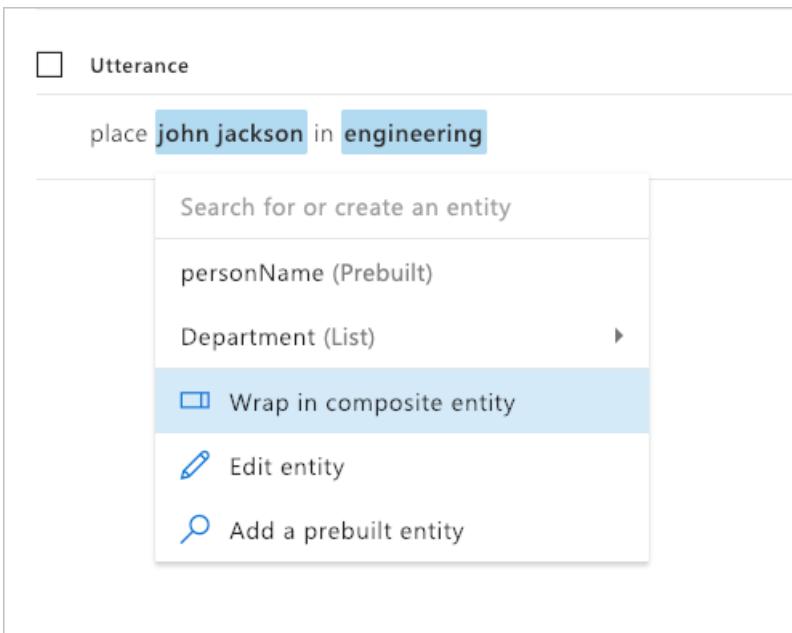
1. Select **Build** from the top navigation, then select **Entities** from the left navigation menu.
2. Select **Manage prebuilt entity** button.
3. Select **PersonName** from the list of prebuilt entities then select **Done**.



This entity helps you add name recognition to your client application.

Create composite entity from example utterances

1. Select **Intents** from the left navigation.
2. Select **TransferEmployeeToDepartment** from the intents list.
3. In the utterance `place John Jackson in engineering`, select the personName entity, `John Jackson`, then select **Wrap in composite entity** in the pop-up menu list for the following utterance.



4. Then immediately select the last entity, `engineering` in the utterance. A green bar is drawn under the selected words indicating a composite entity. In the pop-up menu, enter the composite name `TransferEmployeeInfo` then select enter.



5. In **What type of entity do you want to create?**, all the fields required are in the list: `personName` and `Department`. Select **Done**. Notice that the prebuilt entity, `personName`, was added to the composite entity. If you could have a prebuilt entity appear between the beginning and ending tokens of a composite entity, the composite entity must contain those prebuilt entities. If the prebuilt entities are not included, the composite entity is not correctly predicted but each individual element is.

What type of entity do you want to create?

Entity name (Required)

Entity type (Required)

Child entity

personName

Child entity

Department

+ Add a child entity

Use a **composite entity** to represent an object that has parts. The composite entity is made up of entities that form the whole.
For example, a composite entity called TicketsOrder in a travel app can be composed of three child entities that describe attributes of the tickets to order: Number, PassengerCategory and TravelClass.

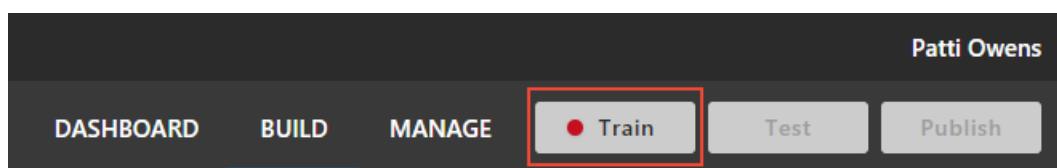
Done **Cancel**

Label example utterances with composite entity

1. In each example utterance, select the left-most entity that should be in the composite. Then select **Wrap in composite entity**.
2. Select the last word in the composite entity then select **TransferEmployeeInfo** from the pop-up menu.
3. Verify all utterances in the intent are labeled with the composite entity.

Train the app so the changes to the intent can be tested

1. In the top right side of the LUIS website, select the **Train** button.



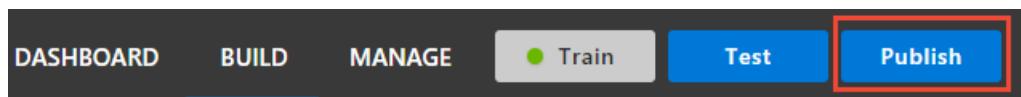
2. Training is complete when you see the green status bar at the top of the website confirming success.



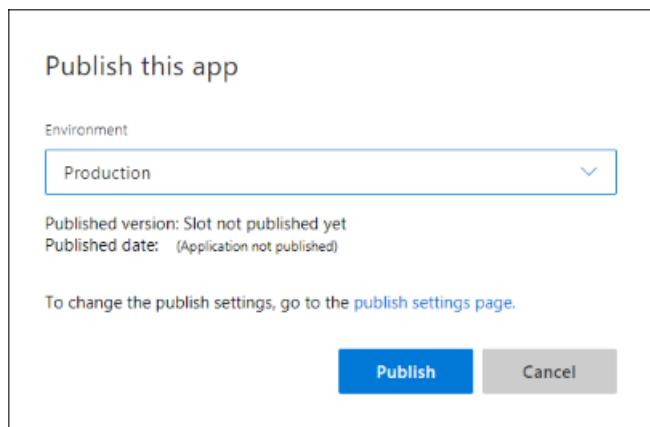
Publish the app so the trained model is queryable from the endpoint

In order to receive a LUIS prediction in a chat bot or other client application, you need to publish the app to the endpoint.

1. Select **Publish** in the top right navigation.



2. Select the **Production** slot and the **Publish** button.



3. Publishing is complete when you see the green status bar at the top of the website confirming success.

Publishing complete. Refer to the list of endpoints to access your endpoint URL

4. Select the **endpoints** link in the green status bar to go to the **Keys and endpoints** page. The endpoint URLs are listed at the bottom.

Get intent and entity prediction from endpoint

1. In the **Manage** section (top right menu), on the **Keys and endpoints** page (left menu), select the **endpoint** URL at the bottom of the page. This action opens another browser tab with the endpoint URL in the address bar.

The endpoint URL looks like

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?verbose=true&subscription-key=<YOUR_KEY>&<optional-name-value-pairs>&q=<user-utterance-text>
```

2. Go to the end of the URL in the address and enter `Move Jill Jones to DevOps`. The last querystring parameter is `q`, the utterance query.

Since this test is to verify the composite is extracted correctly, a test can either include an existing sample utterance or a new utterance. A good test is to include all the child entities in the composite entity.

```
{
  "query": "Move Jill Jones to DevOps",
  "topScoringIntent": {
    "intent": "TransferEmployeeToDepartment",
    "score": 0.9882747
  },
  "intents": [
    {
      "intent": "TransferEmployeeToDepartment",
      "score": 0.9882747
    },
    {
      "intent": "None",
      "score": 0.00925369747
    }
  ],
  "entities": [
    {
      "entity": "jill jones",
      "type": "builtin.personName",
      "startIndex": 5,
      "endIndex": 14
    },
    {
      "entity": "devops",
      "type": "Department",
      "startIndex": 19,
      "endIndex": 24,
      "resolution": {
        "values": [
          "Development Operations"
        ]
      }
    },
    {
      "entity": "jill jones to devops",
      "type": "TransferEmployeeInfo",
      "startIndex": 5,
      "endIndex": 24,
      "score": 0.9607566
    }
  ],
  "compositeEntities": [
    {
      "parentType": "TransferEmployeeInfo",
      "value": "jill jones to devops",
      "children": [
        {
          "type": "builtin.personName",
          "value": "jill jones"
        },
        {
          "type": "Department",
          "value": "devops"
        }
      ]
    }
  ]
}
```

This utterance returns a composite entities array. Each entity is given a type and value. To find more precision for each child entity, use the combination of type and value from the composite array item to find the corresponding item in the entities array.

Clean up resources

When no longer needed, delete the LUIS app. To do so, select **My apps** from the top left menu. Select the ellipsis (...) to the right of the app name in the app list, select **Delete**. On the pop-up dialog **Delete app?**, select **Ok**.

Related information

- [List entity tutorial](#)
- [Composite entity](#) conceptual information
- [How to train](#)
- [How to publish](#)
- [How to test in LUIS portal](#)

Next steps

This tutorial created a composite entity to encapsulate existing entities. This allows the client application to find a group of related data in different datatypes to continue the conversation. A client application for this Human Resources app could ask what day and time the move needs to begin and end. It could also ask about other logistics of the move such as a physical phone.

[Learn how to add a simple entity with a phrase list](#)

Tutorial: Extract names with simple entity and a phrase list

8/20/2019 • 8 minutes to read • [Edit Online](#)

In this tutorial, extract machine-learned data of employment job name from an utterance using the **Simple** entity. To increase the extraction accuracy, add a phrase list of terms specific to the simple entity.

The simple entity detects a single data concept contained in words or phrases.

In this tutorial, you learn how to:

- Import example app
- Add simple entity
- Add phrase list to boost signal words
- Train
- Publish
- Get intents and entities from endpoint

For this article, you can use the free [LUIS](#) account in order to author your LUIS application.

Simple entity

This tutorial adds a new simple entity to extract the job name. The purpose of the simple entity in this LUIS app is to teach LUIS what a job name is and where it can be found in an utterance. The part of the utterance that is the job name can change from utterance to utterance based on word choice and utterance length. LUIS needs examples of job names across all intents that use job names.

The simple entity is a good fit for this type of data when:

- Data is a single concept.
- Data is not well-formatted such as a regular expression.
- Data is not common such as a prebuilt entity of phone number or data.
- Data is not matched exactly to a list of known words, such as a list entity.
- Data does not contain other data items such as a composite entity or contextual roles.

Consider the following utterances from a chat bot:

UTTERANCE	EXTRACTABLE JOB NAME
I want to apply for the new accounting job.	accounting
Submit my resume for the engineering position.	engineering
Fill out application for job 123456	123456

The job name is difficult to determine because a name can be a noun, verb, or a phrase of several words. For example:

JOBs

engineer

software engineer

senior software engineer

engineering team lead

air traffic controller

motor vehicle operator

ambulance driver

tender

extruder

millwright

This LUIS app has job names in several intents. By labeling these words in all the intents' utterances, LUIS learns more about what a job name is and where it is found in utterances.

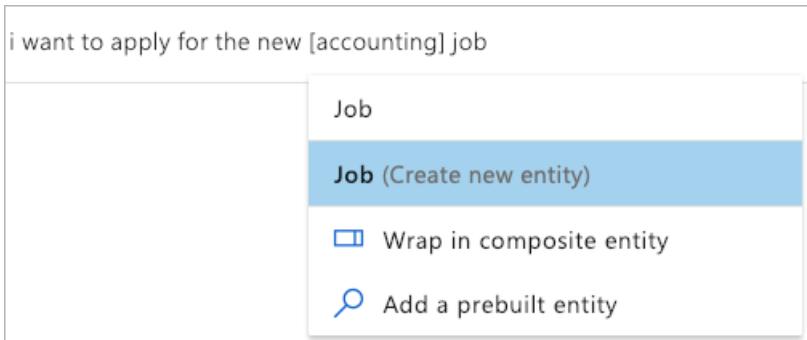
Once the entities are marked in the example utterances, it is important to add a phrase list to boost the signal of the simple entity. A phrase list is **not** used as an exact match and does not need to be every possible value you expect.

Import example app

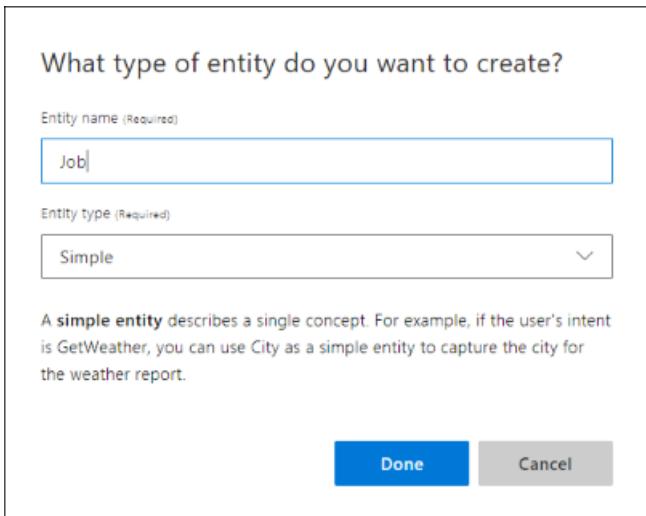
1. Download and save the [app JSON file](#) from the Intents tutorial.
2. Import the JSON into a new app.
3. From the **Manage** section, on the **Versions** tab, clone the version, and name it `simple`. Cloning is a great way to play with various LUIS features without affecting the original version. Because the version name is used as part of the URL route, the name can't contain any characters that are not valid in a URL.

Mark entities in example utterances of an intent

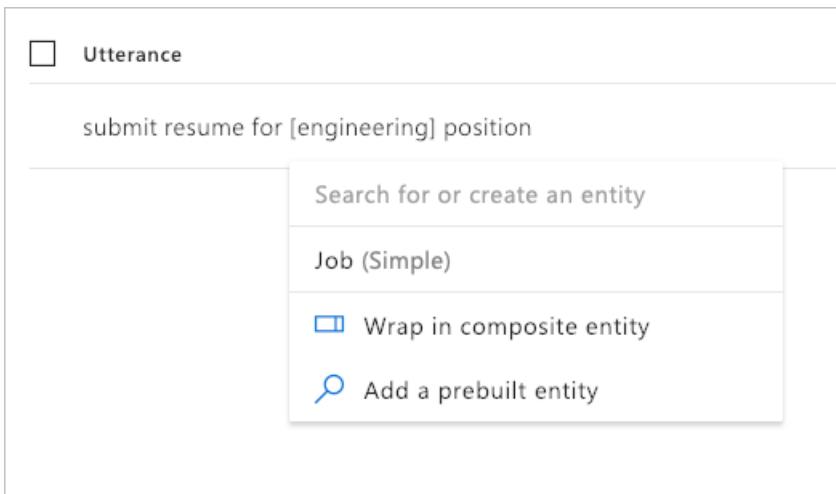
1. Make sure your Human Resources app is in the **Build** section of LUIS. You can change to this section by selecting **Build** on the top, right menu bar.
2. On the **Intents** page, select **ApplyForJob** intent.
3. In the utterance, `I want to apply for the new accounting job`, select `accounting`, enter `Job` in the top field of the pop-up menu, then select **Create new entity** in the pop-up menu.



4. In the pop-up window, verify the entity name and type and select **Done**.



5. In the remaining utterances, mark the job-related words with **Job** entity by selecting the word or phrase, then selecting **Job** from the pop-up menu.



Add more example utterances and mark entity

Simple entities need many examples in order to have a high confidence of prediction.

1. Add more utterances and mark the job words or phrases as **Job** entity.

UTTERANCE	JOB ENTITY
I'm applying for the Program Manager desk in R&D	Program Manager
Here is my line cook application.	line cook

UTTERANCE	JOB ENTITY
My resume for camp counselor is attached.	camp counselor
This is my c.v. for administrative assistant.	administrative assistant
I want to apply for the management job in sales.	management, sales
This is my resume for the new accounting position.	accounting
My application for barback is included.	barback
I'm submitting my application for roofer and framer.	roofer, framer
My c.v. for bus driver is here.	bus driver
I'm a registered nurse. Here is my resume.	registered nurse
I would like to submit my paperwork for the teaching position I saw in the paper.	teaching
This is my c.v. for the stocker post in fruits and vegetables.	stocker
Apply for tile work.	tile
Attached resume for landscape architect.	landscape architect
My curriculum vitae for professor of biology is enclosed.	professor of biology
I would like to apply for the position in photography.	photography

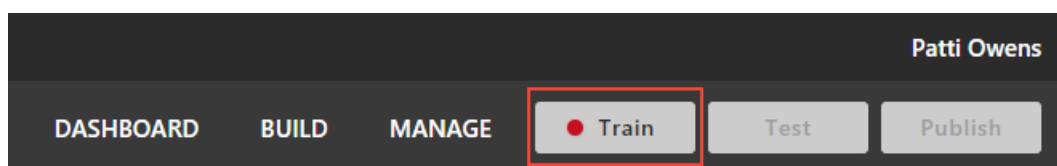
Mark job entity in other intents

1. Select **Intents** from the left menu.
2. Select **GetJobInformation** from the list of intents.
3. Label the jobs in the example utterances

If there are more example utterances in one intent than another intent, that intent has a higher likelihood of being the highest predicted intent.

Train the app so the changes to the intent can be tested

1. In the top right side of the LUIS website, select the **Train** button.



2. Training is complete when you see the green status bar at the top of the website confirming success.



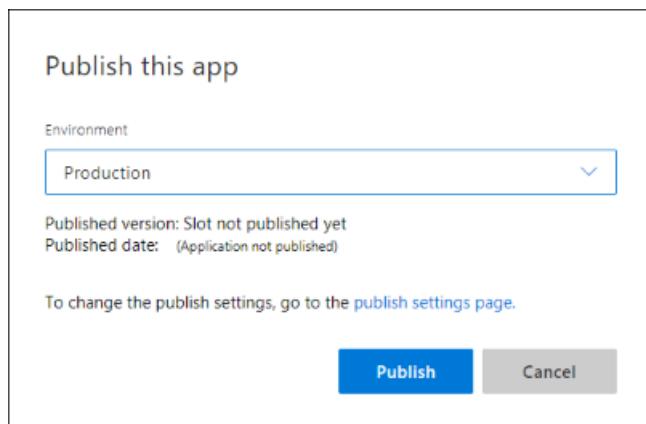
Publish the app so the trained model is queryable from the endpoint

In order to receive a LUIS prediction in a chat bot or other client application, you need to publish the app to the endpoint.

1. Select **Publish** in the top right navigation.



2. Select the **Production** slot and the **Publish** button.



3. Publishing is complete when you see the green status bar at the top of the website confirming success.

Publishing complete. Refer to the list of endpoints to access your endpoint URL

4. Select the **endpoints** link in the green status bar to go to the **Keys and endpoints** page. The endpoint URLs are listed at the bottom.

Get intent and entity prediction from endpoint

1. In the **Manage** section (top right menu), on the **Keys and endpoints** page (left menu), select the **endpoint** URL at the bottom of the page. This action opens another browser tab with the endpoint URL in the address bar.

The endpoint URL looks like

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?verbose=true&subscription-key=<YOUR_KEY>&<optional-name-value-pairs>&q=<user-utterance-text>
```

2. Go to the end of the URL in the address and enter `Here is my c.v. for the engineering job`. The last querystring parameter is `q`, the utterance **query**. This utterance is not the same as any of the labeled utterances so it is a good test and should return the `ApplyForJob` utterances.

```
{  
  "query": "Here is my c.v. for the engineering job",  
  "topScoringIntent": {  
    "intent": "ApplyForJob",  
    "score": 0.98052007  
  },  
  "intents": [  
    {  
      "intent": "ApplyForJob",  
      "score": 0.98052007  
    },  
    {  
      "intent": "GetJobInformation",  
      "score": 0.03424581  
    },  
    {  
      "intent": "None",  
      "score": 0.0015820954  
    }  
],  
  "entities": [  
    {  
      "entity": "engineering",  
      "type": "Job",  

```

LUIS found the correct intent, **ApplyForJob**, and extracted the correct entity, **Job**, with a value of `engineering`.

Names are tricky

The LUIS app found the correct intent with high confidence and it extracted the job name, but names are tricky. Try the utterance `This is the lead welder paperwork`.

In the following JSON, LUIS responds with the correct intent, `ApplyForJob`, but didn't extract the `lead welder` job name.

```
{  
  "query": "This is the lead welder paperwork",  
  "topScoringIntent": {  
    "intent": "ApplyForJob",  
    "score": 0.860295951  
  },  
  "intents": [  
    {  
      "intent": "ApplyForJob",  
      "score": 0.860295951  
    },  
    {  
      "intent": "GetJobInformation",  
      "score": 0.07265678  
    },  
    {  
      "intent": "None",  
      "score": 0.00482481951  
    }  
  ],  
  "entities": []  
}
```

Because a name can be anything, LUIS predicts entities more accurately if it has a phrase list of words to boost the signal.

To boost signal of the job-related words, add a phrase list of job-related words

Open the [jobs-phrase-list.csv](#) from the Azure-Samples GitHub repository. The list is over 1,000 job words and phrases. Look through the list for job words that are meaningful to you. If your words or phrases are not on the list, add your own.

1. In the **Build** section of the LUIS app, select **Phrase lists** found under the **Improve app performance** menu.
2. Select **Create new phrase list**.
3. Name the new phrase list `JobNames` and copy the list from `jobs-phrase-list.csv` into the **Values** text box.

Add Phrase list

Name (Required)

Job

Value (Required)

ce management analyst, workforce planning intern, writer, x-ray technician, yardmaster, youth initiatives lead advisor

Phrase list values

Related Values

Add values and click 'Recommend' to get suggestions ...

Add all Recommend

These values are interchangeable

Done

Cancel

If you want more words added to the phrase list, select **Recommend** then review the new **Related Values** and add any that are relevant.

Make sure to keep the **These values are interchangeable** checked because these values should all be treated as synonyms for jobs. Learn more about interchangeable and noninterchangeable [phrase list concepts](#).

4. Select **Done** to activate the phrase list.

Add Phrase list

Name (Required)

Job

Value (Required)

Type comma separated values and press enter ...

Phrase list values

.net developer .net software developer
accessibility outreach coordinator
accessibility program manager
account executive account manager
accountable manager
accountable project manager accountant
accounting investment analyst actor

Related Values

engineer secretary consultant
graphic designer cook
marketing manager software engineer
teacher sales manager administrator

Add all Recommend

These values are interchangeable

Done

Cancel

5. Train and publish the app again to use phrase list.
6. Requery at the endpoint with the same utterance: `This is the lead welder paperwork.`

The JSON response includes the extracted entity:

```
{  
  "query": "This is the lead welder paperwork.",  
  "topScoringIntent": {  
    "intent": "ApplyForJob",  
    "score": 0.983076453  
  },  
  "intents": [  
    {  
      "intent": "ApplyForJob",  
      "score": 0.983076453  
    },  
    {  
      "intent": "GetJobInformation",  
      "score": 0.0120766377  
    },  
    {  
      "intent": "None",  
      "score": 0.00248388131  
    }  
  ],  
  "entities": [  
    {  
      "entity": "lead welder",  
      "type": "Job",  
      "startIndex": 12,  
      "endIndex": 22,  
      "score": 0.8373154  
    }  
  ]  
}
```

Clean up resources

When no longer needed, delete the LUIS app. To do so, select **My apps** from the top left menu. Select the ellipsis (...) to the right of the app name in the app list, select **Delete**. On the pop-up dialog **Delete app?**, select **Ok**.

Related information

- [Intents without entities tutorial](#)
- [Simple entity conceptual information](#)
- [Phrase list conceptual information](#)
- [How to train](#)
- [How to publish](#)
- [How to test in LUIS portal](#)

Next steps

In this tutorial, the Human Resources app uses a machine-learned simple entity to find job names in utterances. Because job names can be such a wide variety of words or phrases, the app needed a phrase list to boost the job name words.

[Add a prebuilt keyphrase entity](#)

Tutorial: Fix unsure predictions by reviewing endpoint utterances

7/26/2019 • 7 minutes to read • [Edit Online](#)

In this tutorial, improve app predictions by verifying or correcting utterances received via the LUIS HTTPS endpoint that LUIS is unsure of. Some utterances may have to be verified for intent and others may need to be verified for entity. You should review endpoint utterances as a regular part of your scheduled LUIS maintenance.

This review process is another way for LUIS to learn your app domain. LUIS selected the utterances that appear in the review list. This list is:

- Specific to the app.
- Is meant to improve the app's prediction accuracy.
- Should be reviewed on a periodic basis.

By reviewing the endpoint utterances, you verify or correct the utterance's predicted intent. You also label custom entities that were not predicted or predicted incorrectly.

In this tutorial, you learn how to:

- Import example app
- Review endpoint utterances
- Update phrase list
- Train app
- Publish app
- Query endpoint of app to see LUIS JSON response

For this article, you can use the free [LUIS](#) account in order to author your LUIS application.

Import example app

Continue with the app created in the last tutorial, named **HumanResources**.

Use the following steps:

1. Download and save [app JSON file](#).
2. Import the JSON into a new app.
3. From the **Manage** section, on the **Versions** tab, clone the version, and name it `review`. Cloning is a great way to play with various LUIS features without affecting the original version. Because the version name is used as part of the URL route, the name can't contain any characters that are not valid in a URL.
4. Train and publish the new app.
5. Use the endpoint to add the following utterances. You can either do this with a [script](#) or from the endpoint in a browser. The utterances to add are:

```

/* all utterances up through the Sentiment tutorial in series */
const endpointUtterances = [
    "I'm looking for a job with Natural Language Processing",
    "I want to cancel on March 3",
    "When were HRF-123456 and hrf-234567 published in the last year?",
    "shift 123-45-6789 from Z-1242 to T-54672",
    "Please relocation jill-jones@mycompany.com from x-2345 to g-23456",
    "Here is my c.v. for the programmer job",
    "This is the lead welder paperwork.",
    "does form hrf-123456 cover the new dental benefits and medical plan",
    "Jill Jones work with the media team on the public portal was amazing",
];

```

If you have all the versions of the app, through the series of tutorials, you may be surprised to see that the **Review endpoint utterances** list doesn't change, based on the version. There is a single pool of utterances to review, regardless of which version you are actively editing or which version of the app was published at the endpoint.

Review endpoint utterances

1. Make sure your Human Resources app is in the **Build** section of LUIS. You can change to this section by selecting **Build** on the top, right menu bar.
2. Select **Review endpoint utterances** from the left navigation. The list is filtered for the **ApplyForJob** intent.

Utterance	Aligned intent	Add/Delete
Employee work with the keyPhrase on the keyPhrase was amazing	EmployeeFeedback (0....)	
i want to cancel on datetimeV2	Utilities.Cancel (0.425)	
i 'm looking for a keyPhrase with keyPhrase	ApplyForJob (0.199)	
does form HRF-number cover the keyPhrase and keyPhrase	FindForm (0.884)	
here is my c . v. for the Job job	ApplyForJob (0.949)	
shift Employee from Locations::Origin to Locations::Destination	MoveEmployee (0.764)	

3. Toggle the **Entities view** to see the labeled entities.

The screenshot shows the Dialogflow interface with the 'Review endpoint utterances' section selected. The left sidebar has 'App Assets' and 'Improve app performance' sections, with 'Review endpoint utterances' highlighted. The main area shows a table of utterances with their aligned intents and checkboxes for adding or deleting them. A 'Get Started' button is visible at the bottom right.

This utterance, `I'm looking for a job with Natural Language Processing`, is not in the correct intent.

The reason the utterance was mispredicted is that the **ApplyForJob** intent has 21 utterances compared to the 7 utterances in **GetJobInformation**. The intent with more utterances will have a higher prediction. It is important that the quantity and quality of the utterances across intents is balanced.

- To align this utterance, select the correct intent and mark the Job entity within it. Add the changed utterance to the app by selecting the green checkbox.

UTTERANCE	CORRECT INTENT	MISSING ENTITIES
<code>I'm looking for a job with Natural Language Processing</code>	GetJobInfo	Job - "Natural Language Process"

Adding the utterance moves the utterance from the **Review endpoint utterances** to the **GetJobInformation** intent. The endpoint utterance is now an example utterance for that intent.

Along with aligning this utterance correctly, more utterances should be added to the **GetJobInformation** intent. That is left as an exercise for you to complete on your own. Each intent, except for the **None** intent, should have roughly the same number of example utterances. The **None** intent should have 10% of the total utterances in the app.

- Review the remaining utterances in this intent, labeling utterances and correcting the **Aligned intent**, if these are incorrect.
- The list should no longer have those utterances. If more utterances appear, continue to work through the list, correcting intents and labeling any missing entities, until the list is empty.
- Select the next intent in the Filter list, then continue correcting utterances and labeling entities. Remember the last step of each intent is to either select **Add to aligned intent** on the utterance row or check the box by each intent and select **Add selected** above the table.

Continue until all intents and entities in the filter list have an empty list. This is a very small app. The review process takes only a few minutes.

Update phrase list

Keep the phrase list current with any newly discovered job names.

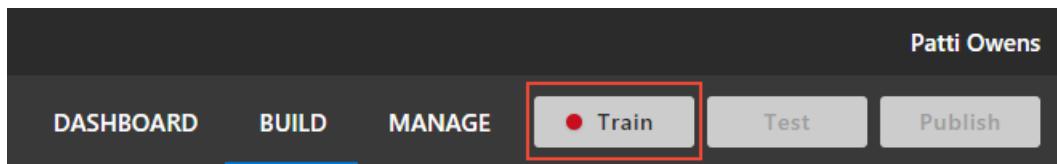
- Select **Phrase lists** from left navigation.

2. Select the **Jobs** phrase list.
3. Add **Natural Language Processing** as a value then select **Save**.

Train

LUIS doesn't know about the changes until it is trained.

1. In the top right side of the LUIS website, select the **Train** button.



2. Training is complete when you see the green status bar at the top of the website confirming success.

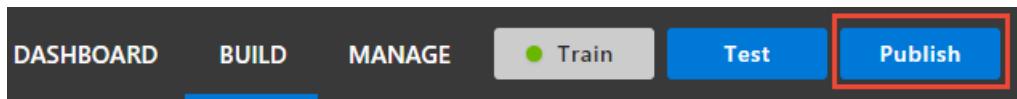


Publish

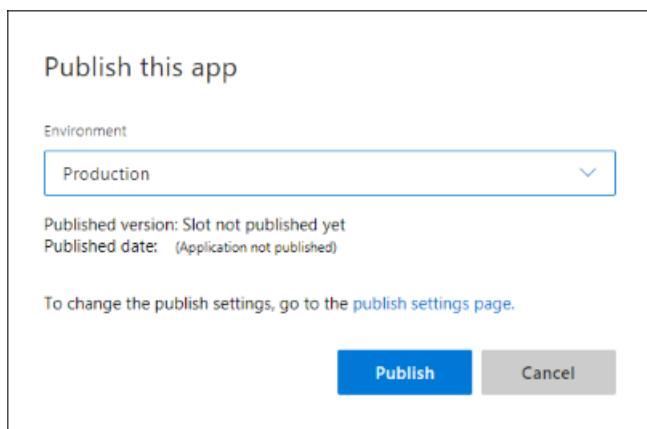
If you imported this app, you need to select **Sentiment analysis**.

In order to receive a LUIS prediction in a chat bot or other client application, you need to publish the app to the endpoint.

1. Select **Publish** in the top right navigation.



2. Select the **Production** slot and the **Publish** button.



3. Publishing is complete when you see the green status bar at the top of the website confirming success.



4. Select the **endpoints** link in the green status bar to go to the **Keys and endpoints** page. The endpoint URLs are listed at the bottom.

Get intent and entities from endpoint

Try an utterance close to the corrected utterance.

1. In the **Manage** section (top right menu), on the **Keys and endpoints** page (left menu), select the **endpoint** URL at the bottom of the page. This action opens another browser tab with the endpoint URL in the address bar.

The endpoint URL looks like

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?verbose=true&subscription-key=<YOUR_KEY>&<optional-name-value-pairs>&q=<user-utterance-text>
```

2. Go to the end of the URL in the address and enter

Are there any natural language processing jobs in my department right now? . The last querystring parameter is **q**, the utterance **query**.

```
{
  "query": "are there any natural language processing jobs in my department right now?",
  "topScoringIntent": {
    "intent": "GetJobInformation",
    "score": 0.9247605
  },
  "intents": [
    {
      "intent": "GetJobInformation",
      "score": 0.9247605
    },
    {
      "intent": "ApplyForJob",
      "score": 0.129989788
    },
    {
      "intent": "FindForm",
      "score": 0.006438211
    },
    {
      "intent": "EmployeeFeedback",
      "score": 0.00408575451
    },
    {
      "intent": "Utilities.StartOver",
      "score": 0.00194211153
    },
    {
      "intent": "None",
      "score": 0.00166400627
    },
    {
      "intent": "Utilities.Help",
      "score": 0.00118593348
    },
    {
      "intent": "MoveEmployee",
      "score": 0.0007885918
    },
    {
      "intent": "Utilities.Cancel",
      "score": 0.0006373631
    },
    {
      "intent": "Utilities.Stop",
      "score": 0.0005980781
    },
    {
      "intent": "Utilities.Confirm",
      "score": 3.719905E-05
    }
  ],
  "entities": [
```

```
{
  "entity": "right now",
  "type": "builtin.datetimeV2.datetime",
  "startIndex": 64,
  "endIndex": 72,
  "resolution": {
    "values": [
      {
        "timex": "PRESENT_REF",
        "type": "datetime",
        "value": "2018-07-05 15:23:18"
      }
    ]
  },
  {
    "entity": "natural language processing",
    "type": "Job",
    "startIndex": 14,
    "endIndex": 40,
    "score": 0.9869922
  },
  {
    "entity": "natural language processing jobs",
    "type": "builtin.keyPhrase",
    "startIndex": 14,
    "endIndex": 45
  },
  {
    "entity": "department",
    "type": "builtin.keyPhrase",
    "startIndex": 53,
    "endIndex": 62
  }
],
"sentimentAnalysis": {
  "label": "positive",
  "score": 0.8251864
}
}
```

The correct intent was predicted with a high score and the **Job** entity is detected as
 natural language processing .

Can reviewing be replaced by adding more utterances?

You may wonder why not add more example utterances. What is the purpose of reviewing endpoint utterances? In a real-world LUIS app, the endpoint utterances are from users with word choice and arrangement you haven't used yet. If you had used the same word choice and arrangement, the original prediction would have a higher percentage.

Why is the top intent on the utterance list?

Some of the endpoint utterances will have a high prediction score in the review list. You still need to review and verify those utterances. They are on the list because the next highest intent had a score too close to the top intent score. You want about 15% difference between the two top intents.

Clean up resources

When no longer needed, delete the LUIS app. To do so, select **My apps** from the top left menu. Select the ellipsis (...) to the right of the app name in the app list, select **Delete**. On the pop-up dialog **Delete app?**, select **Ok**.

Next steps

In this tutorial, you reviewed utterances submitted at the endpoint, that LUIS was unsure of. Once these utterances have been verified and moved into the correct intents as example utterances, LUIS will improve the prediction accuracy.

[Learn how to use patterns](#)

Tutorial: Batch test data sets

8/20/2019 • 9 minutes to read • [Edit Online](#)

This tutorial demonstrates how to use batch testing to find utterance prediction issues in your app and fix them.

Batch testing allows you to validate the active, trained model's state with a known set of labeled utterances and entities. In the JSON-formatted batch file, add the utterances and set the entity labels you need predicted inside the utterance.

Requirements for batch testing:

- Maximum of 1000 utterances per test.
- No duplicates.
- Entity types allowed: only machined-learned entities of simple and composite. Batch testing is only useful for machined-learned intents and entities.

When using an app other than this tutorial, do *not* use the example utterances already added to an intent.

In this tutorial, you learn how to:

- Import example app
- Create a batch test file
- Run a batch test
- Review test results
- Fix errors
- Retest the batch

For this article, you can use the free [LUIS](#) account in order to author your LUIS application.

Import example app

Continue with the app created in the last tutorial, named **HumanResources**.

Use the following steps:

1. Download and save [app JSON file](#).
2. Import the JSON into a new app.
3. From the **Manage** section, on the **Versions** tab, clone the version, and name it `batchtest`. Cloning is a great way to play with various LUIS features without affecting the original version. Because the version name is used as part of the URL route, the name can't contain any characters that are not valid in a URL.
4. Train the app.

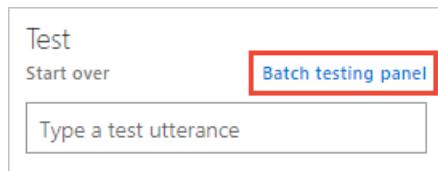
Batch file

1. Create `HumanResources-jobs-batch.json` in a text editor or [download](#) it.
2. In the JSON-formatted batch file, add utterances with the **Intent** you want predicted in the test.

```
[
  {
    "text": "Are there any janitorial jobs currently open?",
    "intent": "GetJobInformation",
    "entities": []
  },
  {
    "text": "I would like a fullstack typescript programming with azure job",
    "intent": "GetJobInformation",
    "entities": []
  },
  {
    "text": "Is there a database position open in Los Colinas?",
    "intent": "GetJobInformation",
    "entities": []
  },
  {
    "text": "Please find database jobs open today in Seattle",
    "intent": "GetJobInformation",
    "entities": []
  }
]
```

Run the batch

1. Select **Test** in the top navigation bar.
2. Select **Batch testing panel** in the right-side panel.



3. Select **Import dataset**.

A screenshot of the 'Batch testing' interface. At the top left is the title 'Batch testing'. To the right is a link 'Single testing panel'. Below the title is a large blue button labeled 'Import dataset', which is highlighted with a red box. A table follows, with columns labeled 'State', 'Name', 'Size', 'Last Run', and 'Last Result'. The table body contains the message 'You haven't created any datasets yet.'

4. Choose the file location of the `HumanResources-jobs-batch.json` file.
5. Name the dataset `intents only` and select **Done**.

Import new dataset

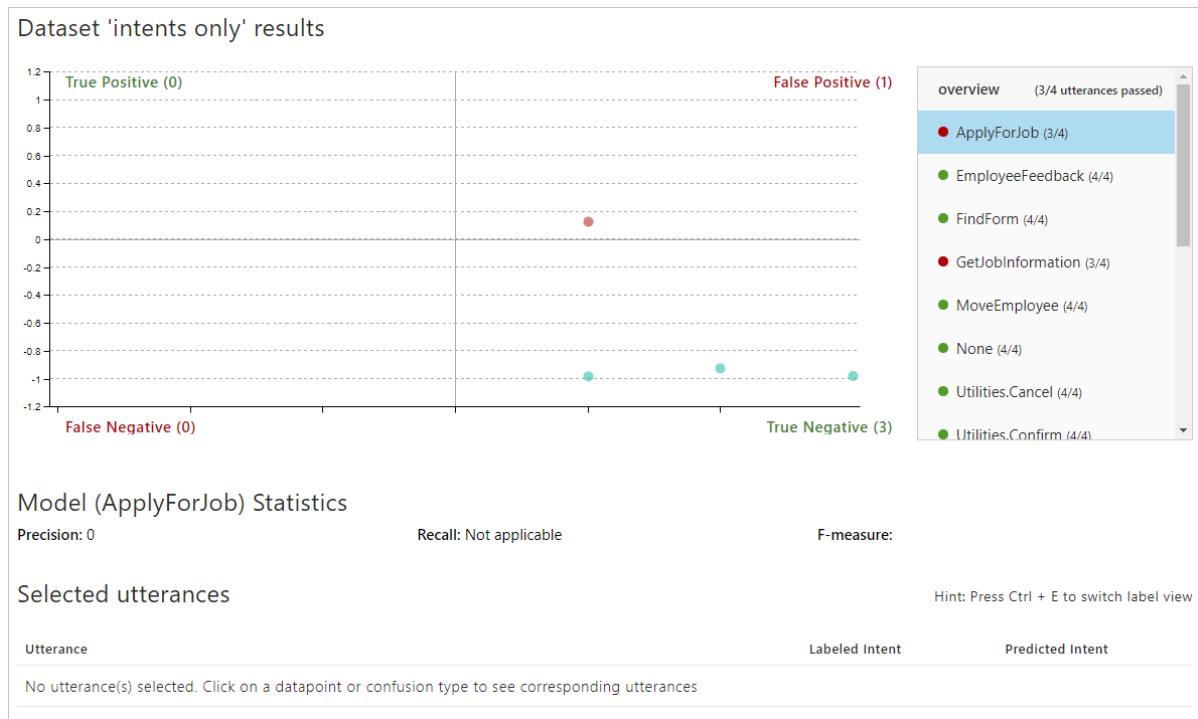
Dataset JSON file (Required)

HumanResour...-batch.json

Dataset Name (Required)

[Learn more about allowed dataset syntax](#)

6. Select the **Run** button.
7. Select **See results**.
8. Review results in the graph and legend.



Review batch results

The batch chart displays four quadrants of results. To the right of the chart is a filter. The filter contains intents and entities. When you select a [section of the chart](#) or a point within the chart, the associated utterance(s) display below the chart.

While hovering over the chart, a mouse wheel can enlarge or reduce the display in the chart. This is useful when there are many points on the chart clustered tightly together.

The chart is in four quadrants, with two of the sections displayed in red. **These are the sections to focus on.**

GetJobInformation test results

The **GetJobInformation** test results displayed in the filter show that 2 of the four predictions were successful. Select the name **False negative** at the bottom left quadrant to see the utterances below the chart.

Use the keyboard, Ctrl + E, to switch to the label view to see the exact text of the user utterance.

The utterance `Is there a database position open in Los Colinas?` is labeled as *GetJobInformation* but the current model predicted the utterance as *ApplyForJob*.

There are almost three times as many examples for **ApplyForJob** than **GetJobInformation**. This unevenness of example utterances weighs in **ApplyForJob** intent's favor, causing the incorrect prediction.

Notice that both intents have the same count of errors. An incorrect prediction in one intent affects the other intent as well. They both have errors because the utterances were incorrectly predicted for one intent, and also incorrectly not predicted for another intent.

How to fix the app

The goal of this section is to have all the utterances correctly predicted for **GetJobInformation** by fixing the app.

A seemingly quick fix would be to add these batch file utterances to the correct intent. That is not what you want to do. You want LUIS to correctly predict these utterances without adding them as examples.

You might also wonder about removing utterances from **ApplyForJob** until the utterance quantity is the same as **GetJobInformation**. That may fix the test results but would hinder LUIS from predicting that intent accurately next time.

The fix is to add more utterances to **GetJobInformation**. Remember to vary utterance length, word choice and word arrangement while still targeting the intent of finding job information, *not* applying for the job.

Add more utterances

1. Close the batch test panel by selecting the **Test** button in the top navigation panel.
2. Select **GetJobInformation** from the intents list.
3. Add more utterances that are varied for length, word choice, and word arrangement, making sure to include the terms `resume`, `c.v.`, and `apply`:

EXAMPLE UTTERANCES FOR GETJOBINFORMATION INTENT

Does the new job in the warehouse for a stocker require that I apply with a resume?

Where are the roofing jobs today?

I heard there was a medical coding job that requires a resume.

I would like a job helping college kids write their c.v.s.

Here is my resume, looking for a new post at the community college using computers.

What positions are available in child and home care?

Is there an intern desk at the newspaper?

My C.v. shows I'm good at analyzing procurement, budgets, and lost money. Is there anything for this type of work?

Where are the earth drilling jobs right now?

I've worked 8 years as an EMS driver. Any new jobs?

New food handling jobs require application?

How many new yard work jobs are available?

EXAMPLE UTTERANCES FOR GETJOBINFORMATION INTENT

Is there a new HR post for labor relations and negotiations?

I have a masters in library and archive management. Any new positions?

Are there any babysitting jobs for 13 year olds in the city today?

Do not label the **Job** entity in the utterances. This section of the tutorial is focused on intent prediction only.

4. Train the app by selecting **Train** in the top right navigation.

Verify the new model

In order to verify that the utterances in the batch test are correctly predicted, run the batch test again.

1. Select **Test** in the top navigation bar. If the batch results are still open, select **Back to list**.
2. Select the ellipsis (...) button to the right of the batch name and select **Run**. Wait until the batch test is done.
Notice that the **See results** button is now green. This means the entire batch ran successfully.
3. Select **See results**. The intents should all have green icons to the left of the intent names.

Create batch file with entities

In order to verify entities in a batch test, the entities need to be labeled in the batch JSON file.

The variation of entities for total word (**token**) count can impact the prediction quality. Make sure the training data supplied to the intent with labeled utterances includes a variety of lengths of entity.

When first writing and testing batch files, it is best to start with a few utterances and entities that you know work, as well as a few that you think may be incorrectly predicted. This helps you focus in on the problem areas quickly. After testing the **GetJobInformation** and **ApplyForJob** intents using several different Job names, which were not predicted, this batch test file was developed to see if there is a prediction problem with certain values for **Job** entity.

The value of a **Job** entity, provided in the test utterances, is usually one or two words, with a few examples being more words. If *your own* human resources app typically has job names of many words, the example utterances labeled with **Job** entity in this app would not work well.

1. Create `HumanResources-entities-batch.json` in a text editor such as [VSCode](#) or [download](#) it.
2. In the JSON-formatted batch file, add an array of objects that include utterances with the **Intent** you want predicted in the test as well as locations of any entities in the utterance. Since an entity is token-based, make sure to start and stop each entity on a character. Do not begin or end the utterance on a space. This causes an error during the batch file import.

```
[  
  {  
    "text": "I'm a registered nurse. Here is my resume.",  
    "intent": "ApplyForJob",  
    "entities": [{  
      "entity": "Job",  
      "startPos": 6,  
      "endPos": 21  
    }]  
  },  
  {  
    "text": "I'm a database analyst. Here is my resume.",  
    "intent": "ApplyForJob",  
    "entities": [{  
      "entity": "Job",  
      "startPos": 6,  
      "endPos": 21  
    }]  
  }]
```

```
"intent": "ApplyForJob",
"entities": [
    "entity": "Job",
    "startPos": 6,
    "endPos": 21
}]
},
{
    "text": "I'm a SQL Server programmer. Here is my resume.",
    "intent": "ApplyForJob",
    "entities": [
        "entity": "Job",
        "startPos": 6,
        "endPos": 26
    ]
},
{
    "text": "I'm a registered nurse. Are there any open jobs.",
    "intent": "GetJobInformation",
    "entities": [
        "entity": "Job",
        "startPos": 6,
        "endPos": 21
    ]
},
{
    "text": "I'm a database analyst. Are there any open jobs.",
    "intent": "GetJobInformation",
    "entities": [
        "entity": "Job",
        "startPos": 6,
        "endPos": 21
    ]
},
{
    "text": "Are there any open jobs for a SQL Server programmer?",
    "intent": "GetJobInformation",
    "entities": [
        "entity": "Job",
        "startPos": 30,
        "endPos": 50
    ]
},
{
    "text": "Is there any open positions for a costume designer?",
    "intent": "GetJobInformation",
    "entities": [
        "entity": "Job",
        "startPos": 34,
        "endPos": 49
    ]
},
{
    "text": "Are there any open jobs for a SQL programmer?",
    "intent": "GetJobInformation",
    "entities": [
        "entity": "Job",
        "startPos": 30,
        "endPos": 42
    ]
}
,
{
    "text": "Are there any open jobs with SQL?",
    "intent": "GetJobInformation",
    "entities": [
        "entity": "Job",
        "startPos": 29,
        "endPos": 31
    ]
}
```

```
    }]  
}  
]
```

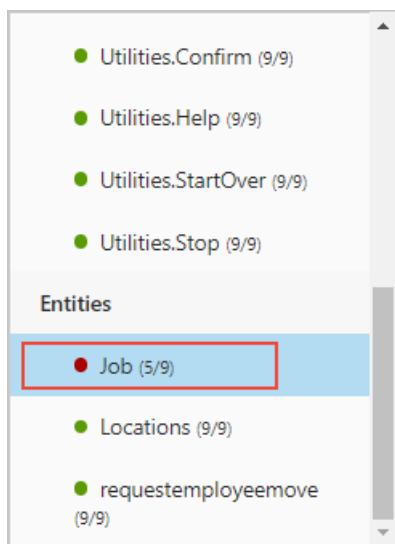
Run the batch with entities

1. Select **Test** in the top navigation bar.
2. Select **Batch testing panel** in the right-side panel.
3. Select **Import dataset**.
4. Choose the file system location of the `HumanResources-entities-batch.json` file.
5. Name the dataset `entities` and select **Done**.
6. Select the **Run** button. Wait until the test is done.
7. Select **See results**.

Review entity batch results

The chart opens with all the intents correctly predicted. Scroll down in the right-side filter to find the entity predictions with errors.

1. Select the **Job** entity in the filter.



The chart changes to display the entity predictions.

2. Select **False Negative** in the lower, left quadrant of the chart. Then use the keyboard combination control + E to switch into the token view.

[Back to list](#)

Dataset 'entities' results



Model 'Job' statistics

Precision: 1

Recall: 0.56

F-measure: 0.72

Selected utterances (False Negative)

Hint: Press Ctrl + E to switch label view

Utterance	Labeled Intent	Predicted Intent
i 'm a sql server programmer . here is my resume .	ApplyForJob (0.57)	ApplyForJob (0.57)
are there any open jobs for a sql server programmer ?	GetJobInformation (0.93)	GetJobInformation (0.93)
are there any open jobs for a sql programmer ?	GetJobInformation (0.93)	GetJobInformation (0.93)
are there any open jobs with sql ?	GetJobInformation (0.99)	GetJobInformation (0.99)

Reviewing the utterances below the chart reveals a consistent error when the Job name includes **SQL**.

Reviewing the example utterances and the Job phrase list, SQL is only used once, and only as part of a larger job name, **sql/oracle database administrator**.

Fix the app based on entity batch results

Fixing the app requires LUIS to correctly determine the variations of SQL jobs. There are several options for that fix.

- Explicitly add more example utterances, which use SQL and label those words as a Job entity.
- Explicitly add more SQL jobs to the phrase list

These tasks are left for you to do.

Adding a **pattern** before the entity is correctly predicted, is not going to fix the problem. This is because the pattern won't match until all the entities in the pattern are detected.

Clean up resources

When no longer needed, delete the LUIS app. To do so, select **My apps** from the top left menu. Select the ellipsis (...) to the right of the app name in the app list, select **Delete**. On the pop-up dialog **Delete app?**, select **Ok**.

Next steps

The tutorial used a batch test to find problems with the current model. The model was fixed and retested with the

batch file to verify the change was correct.

[Learn about patterns](#)

Tutorial: Add common pattern template utterance formats

7/26/2019 • 12 minutes to read • [Edit Online](#)

In this tutorial, use patterns to increase intent and entity prediction while providing fewer example utterances. The pattern is provided by way of a template utterance example, which includes syntax to identify entities and ignorable text. A pattern is a combination of expression matching and machine learning. The template utterance example, along with the intent utterances, give LUIS a better understanding of what utterances fit the intent.

In this tutorial, you learn how to:

- Import example app
- Create intent
- Train
- Publish
- Get intents and entities from endpoint
- Create a pattern
- Verify pattern prediction improvements
- Mark text as ignorable and nest within pattern
- Use test panel to verify pattern success

For this article, you can use the free [LUIS](#) account in order to author your LUIS application.

Import example app

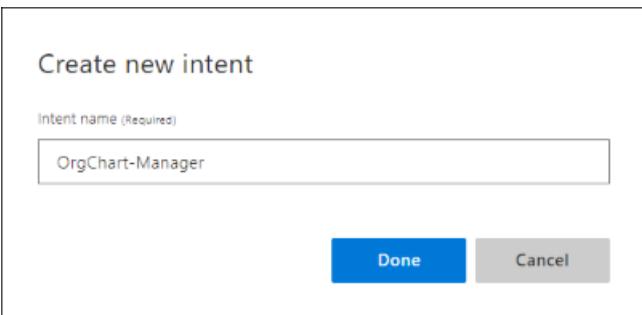
Continue with the app created in the last tutorial, named **HumanResources**.

Use the following steps:

1. Download and save [app JSON file](#).
2. Import the JSON into a new app.
3. From the **Manage** section, on the **Versions** tab, clone the version, and name it **patterns**. Cloning is a great way to play with various LUIS features without affecting the original version. Because the version name is used as part of the URL route, the name can't contain any characters that are not valid in a URL.

Create new intents and their utterances

1. Make sure your Human Resources app is in the **Build** section of LUIS. You can change to this section by selecting **Build** on the top, right menu bar.
2. On the **Intents** page, select **Create new intent**.
3. Enter **OrgChart-Manager** in the pop-up dialog box then select **Done**.



4. Add example utterances to the intent.

EXAMPLE UTTERANCES

Who is John W. Smith the subordinate of?

Who does John W. Smith report to?

Who is John W. Smith's manager?

Who does Jill Jones directly report to?

Who is Jill Jones supervisor?

OrgChart-Manager

Delete Intent

Who is Jill Jones supervisor?

Entity filters Show All Entities View

Utterance	Labeled intent	More
who does Employee directly report to ?	OrgChart-...	...
who is Employee's manager ?	OrgChart-...	...
who does Employee report to ?	OrgChart-...	...
who is Employee the subordinate of ?	OrgChart-...	...

Entities used in this intent

Don't worry if the keyPhrase entity is labeled in the utterances of the intent instead of the employee entity. Both are correctly predicted in the Test pane and at the endpoint.

5. Select **Intents** in the left navigation.
6. Select **Create new intent**.
7. Enter **OrgChart-Reports** in the pop-up dialog box then select **Done**.
8. Add example utterances to the intent.

EXAMPLE UTTERANCES

Who are John W. Smith's subordinates?

EXAMPLE UTTERANCES

Who reports to John W. Smith?

Who does John W. Smith manage?

Who are Jill Jones direct reports?

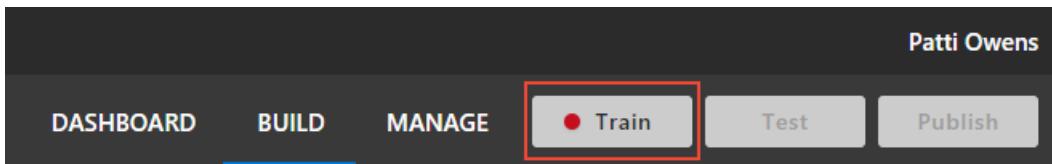
Who does Jill Jones supervise?

Caution about example utterance quantity

These few utterances are for demonstration purposes only. A real-world app should have at least 15 utterances of varying length, word order, tense, grammatical correctness, punctuation, and word count.

Train

1. In the top right side of the LUIS website, select the **Train** button.



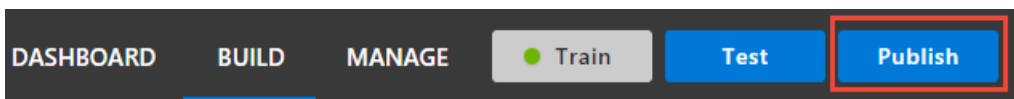
2. Training is complete when you see the green status bar at the top of the website confirming success.



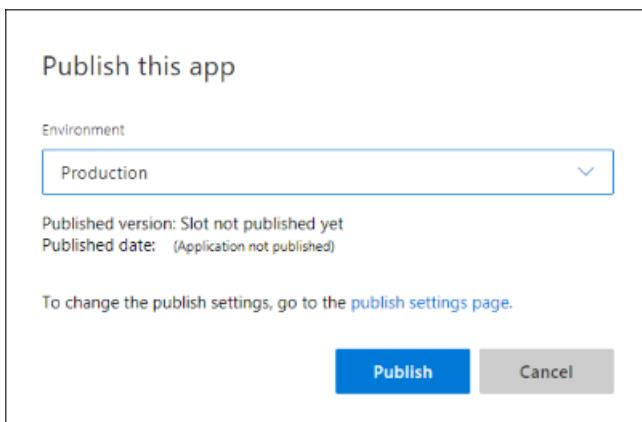
Publish

In order to receive a LUIS prediction in a chat bot or other client application, you need to publish the app to the endpoint.

1. Select **Publish** in the top right navigation.



2. Select the **Production** slot and the **Publish** button.



3. Publishing is complete when you see the green status bar at the top of the website confirming success.



Publishing complete. Refer to the list of endpoints to access your endpoint URL

4. Select the **endpoints** link in the green status bar to go to the **Keys and endpoints** page. The endpoint URLs are listed at the bottom.

Get intent and entities from endpoint

1. In the **Manage** section (top right menu), on the **Keys and endpoints** page (left menu), select the **endpoint** URL at the bottom of the page. This action opens another browser tab with the endpoint URL in the address bar.

The endpoint URL looks like

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?verbose=true&subscription-key=<YOUR_KEY>&<optional-name-value-pairs>&q=<user-utterance-text>
```

2. Go to the end of the URL in the address and enter . The last querystring parameter is , the utterance **query**.

```
{
  "query": "who is the boss of jill jones?",
  "topScoringIntent": {
    "intent": "OrgChart-Manager",
    "score": 0.353984952
  },
  "intents": [
    {
      "intent": "OrgChart-Manager",
      "score": 0.353984952
    },
    {
      "intent": "OrgChart-Reports",
      "score": 0.214128986
    },
    {
      "intent": "EmployeeFeedback",
      "score": 0.08434003
    },
    {
      "intent": "MoveEmployee",
      "score": 0.019131
    },
    {
      "intent": "GetJobInformation",
      "score": 0.004819009
    },
    {
      "intent": "Utilities.Confirm",
      "score": 0.0043958663
    },
    {
      "intent": "Utilities.StartOver",
      "score": 0.00312064588
    },
    {
      "intent": "Utilities.Cancel",
      "score": 0.002265454
    },
    {
      "intent": "Utilities.Help",
      "score": 0.00133465114
    },
    {
      "intent": "None",
    }
  ]
}
```

```

        "score": 0.0011388344
    },
    {
        "intent": "Utilities.Stop",
        "score": 0.00111166481
    },
    {
        "intent": "FindForm",
        "score": 0.0008900076
    },
    {
        "intent": "ApplyForJob",
        "score": 0.0007836131
    }
],
"entities": [
{
    "entity": "jill jones",
    "type": "Employee",
    "startIndex": 19,
    "endIndex": 28,
    "resolution": {
        "values": [
            "Employee-45612"
        ]
    }
},
{
    "entity": "boss of jill jones",
    "type": "builtin.keyPhrase",
    "startIndex": 11,
    "endIndex": 28
}
]
}

```

Did this query succeed? For this training cycle it did succeed. The scores of the two top intents are close. Because LUIS training is not exactly the same each time, there is a bit of variation, these two scores could invert on the next training cycle. The result is that the wrong intent could be returned.

Use patterns to make the correct intent's score significantly higher in percentage and farther from the next highest score.

Leave this second browser window open. You use it again later in the tutorial.

Template utterances

Because of the nature of the Human Resource domain, there are a few common ways of asking about employee relationships in organizations. For example:

UTTERANCES

Who does Jill Jones report to?

Who reports to Jill Jones?

These utterances are too close to determine the contextual uniqueness of each without providing many utterance examples. By adding a pattern for an intent, LUIS learns common utterance patterns for an intent without supplying many utterance examples.

Template utterance examples for this intent include:

TEMPLATE UTTERANCES EXAMPLES	SYNTAX MEANING
Who does {Employee} report to[?]	interchangeable {Employee}, ignore [?]
Who reports to {Employee}[?]	interchangeable {Employee}, ignore [?]

The `{Employee}` syntax marks the entity location within the template utterance as well as which entity it is. The optional syntax, `[?]`, marks words, or punctuation that are optional. LUIS matches the utterance, ignoring the optional text inside the brackets.

While the syntax looks like regular expressions, it is not regular expressions. Only the curly bracket, `{}`, and square bracket, `[]`, syntax is supported. They can be nested up to two levels.

In order for a pattern to be matched to an utterance, the entities within the utterance have to match the entities in the template utterance first. However, the template doesn't help predict entities, only intents.

While patterns allow you to provide fewer example utterances, if the entities are not detected, the pattern does not match.

Add the patterns for the OrgChart-Manager intent

1. Select **Build** in the top menu.
2. In the left navigation, under **Improve app performance**, select **Patterns** from the left navigation.
3. Select the **OrgChart-Manager** intent, then enter the following template utterances:

TEMPLATE UTTERANCES
Who is {Employee} the subordinate of[?]
Who does {Employee} report to[?]
Who is {Employee}'s manager[?]
Who does {Employee} directly report to[?]
Who is {Employee}'s supervisor[?]
Who is the boss of {Employee}[?]

Entities with roles use syntax that includes the role name, and are covered in a [separate tutorial for roles](#).

If you type the template utterance, LUIS helps you fill in the entity when you enter the left curly bracket, `{`.

The screenshot shows the Microsoft Bot Framework's Patterns page. At the top, there is a dropdown menu set to 'OrgChart-Manager'. Below it, a search bar contains the placeholder 'Who is {}'. A red box highlights the word 'Employee' in the search bar. To the right of the search bar, there are 'Entity filters' and 'Intent filters' dropdowns, along with a magnifying glass icon. Below the search bar, a list of entities is shown: 'Job', 'number', 'Employee' (which is also highlighted with a red box), 'keyPhrase', and 'Locations'. Further down, several intent templates are listed under the heading 'Labeled intent ?': 'who does Employee directly report to[?]', 'who is Employee['s] manager[?]', 'who does Employee report to[?]', and 'who is Employee the subordinate of[?]'. Each template has an 'OrgChart-' dropdown and a '...' button.

4. While still on the Patterns page, select the **OrgChart-Reports** intent, then enter the following template utterances:

TEMPLATE UTTERANCES
Who are {Employee}[s] subordinates[?]
Who reports to {Employee}[?]
Who does {Employee} manage[?]
Who are {Employee} direct reports[?]
Who does {Employee} supervise[?]
Who does {Employee} boss[?]

Query endpoint when patterns are used

Now that the patterns are added to the app, train, publish and query the app at the prediction runtime endpoint.

1. Train and publish the app again.
2. Switch browser tabs back to the endpoint URL tab.
3. Go to the end of the URL in the address and enter `Who is the boss of Jill Jones?` as the utterance. The last querystring parameter is `q`, the utterance **query**.

```
{
  "query": "who is the boss of jill jones?",
  "topScoringIntent": {
    "intent": "OrgChart-Manager",
    "score": 0.9999989
  },
  "intents": [
    {
      "intent": "OrgChart-Manager",
      "score": 0.9999989
    },
    {
      ...
    }
  ]
}
```

```
        "intent": "OrgChart-Reports",
        "score": 7.616303E-05
    },
    {
        "intent": "EmployeeFeedback",
        "score": 7.84204349E-06
    },
    {
        "intent": "GetJobInformation",
        "score": 1.20674213E-06
    },
    {
        "intent": "MoveEmployee",
        "score": 7.91245157E-07
    },
    {
        "intent": "None",
        "score": 3.875E-09
    },
    {
        "intent": "Utilities.StartOver",
        "score": 1.49E-09
    },
    {
        "intent": "Utilities.Confirm",
        "score": 1.34545453E-09
    },
    {
        "intent": "Utilities.Help",
        "score": 1.34545453E-09
    },
    {
        "intent": "Utilities.Stop",
        "score": 1.34545453E-09
    },
    {
        "intent": "Utilities.Cancel",
        "score": 1.225E-09
    },
    {
        "intent": "FindForm",
        "score": 1.123077E-09
    },
    {
        "intent": "ApplyForJob",
        "score": 5.625E-10
    }
],
"entities": [
    {
        "entity": "jill jones",
        "type": "Employee",
        "startIndex": 19,
        "endIndex": 28,
        "resolution": {
            "values": [
                "Employee-45612"
            ]
        },
        "role": ""
    },
    {
        "entity": "boss of jill jones",
        "type": "builtin.keyPhrase",
        "startIndex": 11,
        "endIndex": 28
    }
]
```

The intent prediction is now significantly more confident.

Working with optional text and prebuilt entities

The previous pattern template utterances in this tutorial had a few examples of optional text such as the possessive use of the letter s, `'s`, and the use of the question mark, `?`. Suppose you need to allow for current and future dates in the utterance text.

Example utterances are:

INTENT	EXAMPLE UTTERANCES WITH OPTIONAL TEXT AND PREBUILT ENTITIES
OrgChart-Manager	Who was Jill Jones manager on March 3?
OrgChart-Manager	Who is Jill Jones manager now?
OrgChart-Manager	Who will be Jill Jones manager in a month?
OrgChart-Manager	Who will be Jill Jones manager on March 3?

Each of these examples uses a verb tense, `was`, `is`, `will be`, as well as a date, `March 3`, `now`, and `in a month`, that LUIS needs to predict correctly. Notice that the last two examples use almost the same text except for `in` and `on`.

Example template utterances that allow for this optional information:

INTENT	EXAMPLE UTTERANCES WITH OPTIONAL TEXT AND PREBUILT ENTITIES
OrgChart-Manager	who was {Employee}['s] manager [[on]{datetimeV2}]?
OrgChart-Manager	who is {Employee}['s] manager [[on]{datetimeV2}]?

The use of the optional syntax of square brackets, `[]`, makes this optional text easy to add to the template utterance and can be nested up to a second level, `[[[]]]`, and include entities or text.

Question: Why are all the `w` letters, the first letter in each template utterance, lowercase? Shouldn't they be optionally upper or lowercase? The utterance submitted to the query endpoint, by the client application, is converted into lowercase. The template utterance can be uppercase or lowercase and the endpoint utterance can also be either. The comparison is always done after the conversion to lowercase.

Question: Why isn't prebuilt number part of the template utterance if March 3 is predicted both as number `3` and date `March 3`? The template utterance contextually is using a date, either literally as in `March 3` or abstracted as `in a month`. A date can contain a number but a number may not necessarily be seen as a date. Always use the entity that best represents the type you want returned in the prediction JSON results.

Question: What about poorly phrased utterances such as `Who will {Employee}['s] manager be on March 3?`. Grammatically different verb tenses such as this where the `will` and `be` are separated need to be a new template utterance. The existing template utterance will not match it. While the intent of the utterance hasn't changed, the word placement in the utterance has changed. This change impacts the prediction in LUIS. You can group and or the verb-tenses to combine these utterances.

Remember: entities are found first, then the pattern is matched.

Edit the existing pattern template utterance

1. On the LUIS website, select **Build** in the top menu then select **Patterns** in the left menu.
2. Search for the existing template utterance, `who is {Employee}'s manager[?]`, and select the ellipsis (...) to the right, then select **Edit** from the pop-up menu.
3. Change the template utterance to: `who is {Employee}'s manager [[on]{datetimeV2}?]?`

Add new pattern template utterances

1. While still in the **Patterns** section of **Build**, add several new pattern template utterances. Select **OrgChart-Manager** from the Intent drop-down menu and enter each of the following template utterances:

INTENT	EXAMPLE UTTERANCES WITH OPTIONAL TEXT AND PREBUILT ENTITIES
OrgChart-Manager	<code>who was {Employee}'s manager [[on]{datetimeV2}?]?</code>
OrgChart-Manager	<code>who will be {Employee}'s manager [[in]{datetimeV2}?]?</code>
OrgChart-Manager	<code>who will be {Employee}'s manager [[on]{datetimeV2}?]?</code>

2. Train the app.
3. Select **Test** at the top of the panel to open the testing panel.
4. Enter several test utterances to verify that the pattern is matched and the intent score is significantly high.

After you enter the first utterance, select **Inspect** under the result so you can see all the prediction results. Each utterance should have the **OrgChart-Manager** intent and should extract the values for the entities of Employee and datetimeV2.

UTTERANCE
Who will be Jill Jones manager
who will be jill jones's manager
Who will be Jill Jones's manager?
who will be Jill jones manager on March 3
Who will be Jill Jones manager next Month
Who will be Jill Jones manager in a month?

All of these utterances found the entities inside, therefore they match the same pattern, and have a high prediction score.

Use the OR operator and groups

Several of the previous template utterances are very close. Use the **group** `()` and **OR** `|` syntax to reduce the

template utterances.

The following 2 patterns can combine into a single pattern using the group `()` and OR `|` syntax.

INTENT	EXAMPLE UTTERANCES WITH OPTIONAL TEXT AND PREBUILT ENTITIES
OrgChart-Manager	<code>who will be {Employee}'s manager [[in] {datetimeV2}]?</code>
OrgChart-Manager	<code>who will be {Employee}'s manager [[on] {datetimeV2}]?</code>

The new template utterance will be:

```
who ( was | is | will be ) {Employee}'s manager [[(in)|on)]{datetimeV2}] .
```

This uses a **group** around the required verb tense and the optional `in` and `on` with an **or** pipe between them.

1. On the **Patterns** page, select the **OrgChart-Manager** filter. Narrow the list by searching for `manager`.

Pattern	Labeled intent ?
who will be Employee's manager [[on] datetimeV2?]	OrgChart-... ▾
who will be Employee's manager [[in] datetimeV2?]	OrgChart-... ▾
who was Employee's manager [[on] datetimeV2?]	OrgChart-... ▾
who is Employee's manager [[on] datetimeV2?]	OrgChart-... ▾

2. Keep one version of the template utterance (to edit in next step) and delete the other variations.

3. Change the template utterance to:

```
who ( was | is | will be ) {Employee}'s manager [[(in)|on)]{datetimeV2}] .
```

4. Train the app.

5. Use the Test pane to test versions of the utterance:

UTTERANCES TO ENTER IN TEST PANE
Who is Jill Jones manager this month
Who is Jill Jones manager on July 5th
Who was Jill Jones manager last month
Who was Jill Jones manager on July 5th
Who will be Jill Jones manager in a month

UTTERANCES TO ENTER IN TEST PANE

```
Who will be Jill Jones manager on July 5th
```

Use the utterance beginning and ending anchors

The pattern syntax provides beginning and ending utterance anchor syntax of a caret, `^`. The beginning and ending utterance anchors can be used together to target very specific and possibly literal utterance or used separately to target intents.

Clean up resources

When no longer needed, delete the LUIS app. To do so, select **My apps** from the top left menu. Select the ellipsis (...) to the right of the app name in the app list, select **Delete**. On the pop-up dialog **Delete app?**, select **Ok**.

Next steps

This tutorial adds two intents for utterances that were difficult to predict with high accuracy without having many example utterances. Adding patterns for these allowed LUIS to better predict the intent with a significantly higher score. Marking entities and ignorable text allowed LUIS to apply the pattern to a wider variety of utterances.

[Learn how to use roles with a pattern](#)

Tutorial: Extract contextually related patterns using roles

7/26/2019 • 9 minutes to read • [Edit Online](#)

In this tutorial, use a pattern to extract data from a well-formatted template utterance. The template utterance uses a **simple entity** and **roles** to extract related data such as origin location and destination location. When using patterns, fewer example utterances are needed for the intent.

In this tutorial, you learn how to:

- Import example app
- Create new entities
- Create new intent
- Train
- Publish
- Get intents and entities from endpoint
- Create pattern with roles
- Create phrase list of Cities
- Get intents and entities from endpoint

For this article, you can use the free [LUIS](#) account in order to author your LUIS application.

Using roles in patterns

The purpose of roles is to extract contextually related entities in an utterance. In the utterance,

`Move new employee Robert Williams from Sacramento and San Francisco`, the origin city, and destination city values are related to each other and use common language to denote each location.

The name of the new employee, Billy Patterson, is not part of the list entity **Employee** yet. The new employee name is extracted first, in order to send the name to an external system to create the company credentials. After the company credentials are created, the employee credentials are added to the list entity **Employee**.

The new employee and family need to be moved from the current city to a city where the fictitious company is located. Because a new employee can come from any city, the locations need to be discovered. A set list such as a list entity would not work because only the cities in the list would be extracted.

The role names associated with the origin and destination cities need to be unique across all entities. An easy way to make sure the roles are unique is to tie them to the containing entity through a naming strategy. The **NewEmployeeRelocation** entity is a simple entity with two roles: **NewEmployeeReloOrigin** and **NewEmployeeReloDestination**. Relo is short for relocation.

Because the example utterance `Move new employee Robert Williams from Sacramento and San Francisco` has only machine-learned entities, it is important to provide enough example utterances to the intent so the entities are detected.

While patterns allow you to provide fewer example utterances, if the entities are not detected, the pattern does not match.

If you have difficulty with simple entity detection because it is a name such as a city, consider adding a phrase list of similar values. This helps the detection of the city name by giving LUIS an additional signal about that type of word or phrase. Phrase lists only help the pattern by helping with entity detection, which is necessary for the

pattern to match.

Import example app

Continue with the app created in the last tutorial, named **HumanResources**.

Use the following steps:

1. Download and save [app JSON file](#).
2. Import the JSON into a new app.
3. From the **Manage** section, on the **Versions** tab, clone the version, and name it `roles`. Cloning is a great way to play with various LUIS features without affecting the original version. Because the version name is used as part of the URL route, the name can't contain any characters that are not valid in a URL.

Create new entities

1. Make sure your Human Resources app is in the **Build** section of LUIS. You can change to this section by selecting **Build** on the top, right menu bar.
2. Select **Entities** from the left navigation.
3. Select **Create new entity**.
4. In the pop-up window, enter `NewEmployee` as a **Simple** entity.
5. Select **Create new entity**.
6. In the pop-up window, enter `NewEmployeeRelocation` as a **Simple** entity.
7. Select **NewEmployeeRelocation** from the list of entities.
8. Enter the first role as `NewEmployeeReloOrigin` and select enter.
9. Enter the second role as `NewEmployeeReloDestination` and select enter.

Create new intent

Labeling the entities in these steps may be easier if the prebuilt keyPhrase entity is removed before beginning then added back after you are done with the steps in this section.

1. Select **Intents** from the left navigation.
2. Select **Create new intent**.
3. Enter `NewEmployeeRelocationProcess` as the intent name in the pop-up dialog box.
4. Enter the following example utterances, labeling the new entities. The entity and role values are in bold. Remember to switch to the **Tokens View** if you find it easier to label the text.

You don't specify the role of the entity when labeling in the intent. You do that later when creating the pattern.

UTTERANCE	NEWEMPLOYEE	NEWEMPLOYEERELOCATION
Move Bob Jones from Seattle to Los Colinas	Bob Jones	Seattle, Los Colinas

UTTERANCE	NEWEMPLOYEE	NEWEMPLOYEELOCATION
Move Dave C. Cooper from Redmond to New York City	Dave C. Cooper	Redmond, New York City
Move Jim Paul Smith from Toronto to West Vancouver	Jim Paul Smith	Toronto, West Vancouver
Move J. Benson from Boston to Staines-upon-Thames	J. Benson	Boston, Staines-upon-Thames
Move Travis "Trav" Hinton from Castelo Branco to Orlando	Travis "Trav" Hinton	Castelo Branco, Orlando
Move Trevor Nottington III from Aranda de Duero to Boise	Trevor Nottington III	Aranda de Duero, Boise
Move Dr. Greg Williams from Orlando to Ellicott City	Dr. Greg Williams	Orlando, Ellicott City
Move Robert "Bobby" Gregson from Kansas City to San Juan Capistrano	Robert "Bobby" Gregson	Kansas City, San Juan Capistrano
Move Patti Owens from Bellevue to Rockford	Patti Owens	Bellevue, Rockford
Move Janet Bartlet from Tuscan to Santa Fe	Janet Bartlet	Tuscan, Santa Fe

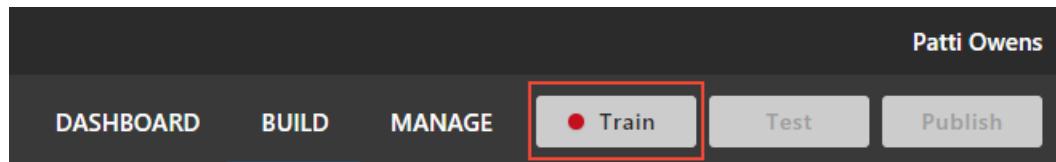
The employee name has a variety of prefix, word count, syntax, and suffix. This is important for LUIS to understand the variations of a new employee name. The city names also have a variety of word count and syntax. This variety is important to teach LUIS how these entities may appear in a user's utterance.

If either entity had been of the same word count and no other variations, you would teach LUIS that this entity only has that word count and no other variations. LUIS would not be able to correctly predict a broader set of variations because it was not shown any.

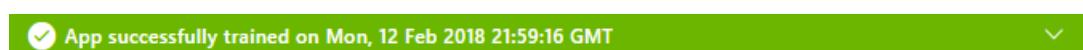
If you removed the keyPhrase entity, add it back to the app now.

Train

- In the top right side of the LUIS website, select the **Train** button.



- Training is complete when you see the green status bar at the top of the website confirming success.

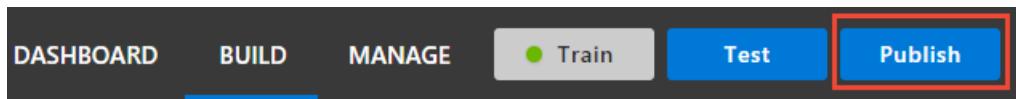


Publish

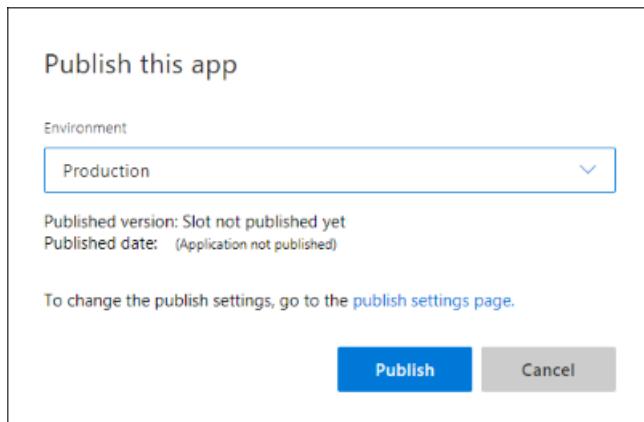
In order to receive a LUIS prediction in a chat bot or other client application, you need to publish the app to the

endpoint.

1. Select **Publish** in the top right navigation.



2. Select the **Production** slot and the **Publish** button.



3. Publishing is complete when you see the green status bar at the top of the website confirming success.

Publishing complete. Refer to the list of endpoints to access your endpoint URL

4. Select the **endpoints** link in the green status bar to go to the **Keys and endpoints** page. The endpoint URLs are listed at the bottom.

Get intent and entities from endpoint

1. In the **Manage** section (top right menu), on the **Keys and endpoints** page (left menu), select the **endpoint** URL at the bottom of the page. This action opens another browser tab with the endpoint URL in the address bar.

The endpoint URL looks like

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?verbose=true&subscription-key=<YOUR_KEY>&<optional-name-value-pairs>&q=<user-utterance-text>
```

2. Go to the end of the URL in the address and enter `Move Wayne Berry from Miami to Mount Vernon`. The last querystring parameter is `q`, the utterance **query**.

```
{
  "query": "Move Wayne Berry from Newark to Columbus",
  "topScoringIntent": {
    "intent": "NewEmployeeRelocationProcess",
    "score": 0.514479756
  },
  "intents": [
    {
      "intent": "NewEmployeeRelocationProcess",
      "score": 0.514479756
    },
    {
      "intent": "Utilities.Confirm",
      "score": 0.017118983
    },
    {
      "intent": "MoveEmployee",
      "score": 0.009982505
    }
  ]
}
```

```

},
{
  "intent": "GetJobInformation",
  "score": 0.008637771
},
{
  "intent": "ApplyForJob",
  "score": 0.007115978
},
{
  "intent": "Utilities.StartOver",
  "score": 0.006120186
},
{
  "intent": "Utilities.Cancel",
  "score": 0.00452428637
},
{
  "intent": "None",
  "score": 0.00400899537
},
{
  "intent": "OrgChart-Reports",
  "score": 0.00240071164
},
{
  "intent": "Utilities.Help",
  "score": 0.001770991
},
{
  "intent": "EmployeeFeedback",
  "score": 0.001697356
},
{
  "intent": "OrgChart-Manager",
  "score": 0.00168116146
},
{
  "intent": "Utilities.Stop",
  "score": 0.00163952739
},
{
  "intent": "FindForm",
  "score": 0.00112958835
}
],
"entities": [
{
  "entity": "wayne berry",
  "type": "NewEmployee",
  "startIndex": 5,
  "endIndex": 15,
  "score": 0.629158735
},
{
  "entity": "newark",
  "type": "NewEmployeeRelocation",
  "startIndex": 22,
  "endIndex": 27,
  "score": 0.638941
}
]
}

```

The intent prediction score is only about 50%. If your client application requires a higher number, this needs to be fixed. The entities were not predicted either.

One of the locations was extracted but the other location was not.

Patterns will help the prediction score, however, the entities must be correctly predicted before the pattern matches the utterance.

Pattern with roles

1. Select **Build** in the top navigation.
2. Select **Patterns** in the left navigation.
3. Select **NewEmployeeRelocationProcess** from the **Select an intent** drop-down list.
4. Enter the following pattern:

```
move {NewEmployee} from {NewEmployeeRelocation:NewEmployeeReloOrigin} to  
{NewEmployeeRelocation:NewEmployeeReloDestination}[.]
```

If you train, publish, and query the endpoint, you may be disappointed to see that the entities are not found, so the pattern didn't match, therefore the prediction didn't improve. This is a consequence of not enough example utterances with labeled entities. Instead of adding more examples, add a phrase list to fix this problem.

Cities phrase list

Cities, like people's names are tricky in that they can be any mix of words and punctuation. The cities of the region and world are known, so LUIS needs a phrase list of cities to begin learning.

1. Select **Phrase list** from the **Improve app performance** section of the left menu.
2. Name the list **Cities** and add the following **values** for the list:

VALUES OF PHRASE LIST
Seattle
San Diego
New York City
Los Angeles
Portland
Philadelphia
Miami
Dallas

Do not add every city in the world or even every city in the region. LUIS needs to be able to generalize what a city is from the list. Make sure to keep **These values are interchangeable** selected. This setting means the words on the list are treated as synonyms.

3. Train and publish the app.

Get intent and entities from endpoint

1. Make sure your Human Resources app is in the **Build** section of LUIS. You can change to this section by selecting **Build** on the top, right menu bar.

2. Go to the end of the URL in the address and enter `Move wayne berry from miami to mount vernon`. The last querystring parameter is `q`, the utterance **query**.

```
{  
  "query": "Move Wayne Berry from Miami to Mount Vernon",  
  "topScoringIntent": {  
    "intent": "NewEmployeeRelocationProcess",  
    "score": 0.9999999  
  },  
  "intents": [  
    {  
      "intent": "NewEmployeeRelocationProcess",  
      "score": 0.9999999  
    },  
    {  
      "intent": "Utilities.Confirm",  
      "score": 1.49678385E-06  
    },  
    {  
      "intent": "MoveEmployee",  
      "score": 8.240291E-07  
    },  
    {  
      "intent": "GetJobInformation",  
      "score": 6.3131273E-07  
    },  
    {  
      "intent": "None",  
      "score": 4.25E-09  
    },  
    {  
      "intent": "OrgChart-Manager",  
      "score": 2.8E-09  
    },  
    {  
      "intent": "OrgChart-Reports",  
      "score": 2.8E-09  
    },  
    {  
      "intent": "EmployeeFeedback",  
      "score": 1.64E-09  
    },  
    {  
      "intent": "Utilities.StartOver",  
      "score": 1.64E-09  
    },  
    {  
      "intent": "Utilities.Help",  
      "score": 1.48181822E-09  
    },  
    {  
      "intent": "Utilities.Stop",  
      "score": 1.48181822E-09  
    },  
    {  
      "intent": "Utilities.Cancel",  
      "score": 1.35E-09  
    },  
    {  
      "intent": "FindForm",  
      "score": 1.23846156E-09  
    },  
    {  
      "intent": "ApplyForJob",  
      "score": 1.23846156E-09  
    }  
  ]  
}
```

```

        "score": 5.692308E-10
    }
],
"entities": [
{
    "entity": "wayne berry",
    "type": "builtin.keyPhrase",
    "startIndex": 5,
    "endIndex": 15
},
{
    "entity": "miami",
    "type": "builtin.keyPhrase",
    "startIndex": 22,
    "endIndex": 26
},
{
    "entity": "wayne berry",
    "type": "NewEmployee",
    "startIndex": 5,
    "endIndex": 15,
    "score": 0.9410646,
    "role": ""
},
{
    "entity": "miami",
    "type": "NewEmployeeRelocation",
    "startIndex": 22,
    "endIndex": 26,
    "score": 0.9853915,
    "role": "NewEmployeeReloOrigin"
},
{
    "entity": "mount vernon",
    "type": "NewEmployeeRelocation",
    "startIndex": 31,
    "endIndex": 42,
    "score": 0.986044347,
    "role": "NewEmployeeReloDestination"
}
],
"sentimentAnalysis": {
    "label": "neutral",
    "score": 0.5
}
}

```

The intent score is now much higher and the role names are part of the entity response.

Clean up resources

When no longer needed, delete the LUIS app. To do so, select **My apps** from the top left menu. Select the ellipsis (...) to the right of the app name in the app list, select **Delete**. On the pop-up dialog **Delete app?**, select **Ok**.

Next steps

This tutorial added an entity with roles and an intent with example utterances. The first endpoint prediction using the entity correctly predicted the intent but with a low confidence score. Only one of the two entities was detected. Next, the tutorial added a pattern that used the entity roles, and a phrase list to boost the value of the city names in the utterances. The second endpoint prediction returned a high-confidence score and found both entity roles.

[Learn best practices for LUIS apps](#)

Tutorial: Extract free-form data with Pattern.any entity

7/26/2019 • 4 minutes to read • [Edit Online](#)

In this tutorial, use the pattern.any entity to extract data from utterances where the utterances are well-formatted and where the end of the data may be easily confused with the remaining words of the utterance.

In this tutorial, you learn how to:

- Import example app
- Add example utterances to existing entity
- Create Pattern.any entity
- Create pattern
- Train
- Test new pattern

For this article, you can use the free [LUIS](#) account in order to author your LUIS application.

Using Pattern.any entity

The pattern.any entity allows you to find free-form data where the wording of the entity makes it difficult to determine the end of the entity from the rest of the utterance.

This Human Resources app helps employees find company forms.

UTTERANCE
Where is HRF-123456 ?
Who authored HRF-123234 ?
HRF-456098 is published in French?

However, each form has both a formatted name, used in the preceding table, as well as a friendly name, such as

`Request relocation from employee new to the company 2018 version 5`.

Utterances with the friendly form name look like:

UTTERANCE
Where is Request relocation from employee new to the company 2018 version 5 ?
Who authored " Request relocation from employee new to the company 2018 version 5 "?
Request relocation from employee new to the company 2018 version 5 is published in French?

The varying length includes words that may confuse LUIS about where the entity ends. Using a Pattern.any entity in a pattern allows you to specify the beginning and end of the form name so LUIS correctly extracts the form name.

TEMPLATE UTTERANCE EXAMPLE

Where is {FormName}[]?

Who authored {FormName}[]?

{FormName} is published in French[]?

Import example app

1. Download and save [app JSON file](#).
2. In the [LUIS portal](#), on the **My apps** page, import the JSON into a new app.
3. From the **Manage** section, on the **Versions** tab, clone the version, and name it `patt-any`. Cloning is a great way to play with various LUIS features without affecting the original version. Because the version name is used as part of the URL route, the name can't contain any characters that are not valid in a URL.

Add example utterances

1. Select **Build** from the top navigation, then select **Intents** from left navigation.
2. Select **FindForm** from the intents list.
3. Add some example utterances:

EXAMPLE UTTERANCE

Where is the form **What to do when a fire breaks out in the Lab** and who needs to sign it after I read it?

Where is **Request relocation from employee new to the company** on the server?

Who authored "**Health and wellness requests on the main campus**" and what is the most current version?

I'm looking for the form named "**Office move request including physical assets**".

Without a Pattern.any entity, it would be difficult for LUIS to understand where the form title ends because of the many variations of form names.

Create a Pattern.any entity

The Pattern.any entity extracts entities of varying length. It only works in a pattern because the pattern marks the beginning and end of the entity.

1. Select **Entities** in the left navigation.
2. Select **Create new entity**, enter the name `FormName`, and select **Pattern.any** as the type. Select **Done**.

You can't label the entity in an intent's example utterances because a Pattern.any is only valid in a pattern.

If you want the extracted data to include other entities such as number or datetimeV2, you need to create a composite entity that includes the Pattern.any, as well as number and datetimeV2.

Add a pattern that uses the Pattern.any

1. Select **Patterns** from the left navigation.

2. Select the **FindForm** intent.
3. Enter the following template utterances, which use the new entity:

TEMPLATE UTTERANCES
Where is the form ["{FormName}"] and who needs to sign it after I read it[?]
Where is ["{FormName}"] on the server[?]
Who authored ["{FormName}"] and what is the most current version[?]
I'm looking for the form named ["{FormName}"][.]

If you want to account for variations of the form such as single quotes instead of double quotes or a period instead of a question mark, create a new pattern for each variation.

Train the LUIS app

1. In the top right side of the LUIS website, select the **Train** button.



2. Training is complete when you see the green status bar at the top of the website confirming success.



Test the new pattern for free-form data extraction

1. Select **Test** from the top bar to open the test panel.
2. Enter the following utterance:

```
Where is the form Understand your responsibilities as a member of the community and who needs to sign it  
after I read it?
```

3. Select **Inspect** under the result to see the test results for entity and intent.

The entity `FormName` is found first, then the pattern is found, determining the intent. If you have a test result where the entities are not detected, and therefore the pattern is not found, you need to add more example utterances on the intent (not the pattern).

4. Close the test panel by selecting the **Test** button in the top navigation.

Using an explicit list

If you find that your pattern, when it includes a Pattern.any, extracts entities incorrectly, use an [explicit list](#) to correct this problem.

Clean up resources

When no longer needed, delete the LUIS app. To do so, select **My apps** from the top left menu. Select the ellipsis (...) to the right of the app name in the app list, select **Delete**. On the pop-up dialog **Delete app?**, select **Ok**.

Next steps

This tutorial added example utterances to an existing intent then created a new Pattern.any for the form name. Then the tutorial created a pattern for the existing intent with the new example utterances and entity. Interactive testing showed that the pattern and its intent were predicted because the entity was found.

[Learn how to use roles with a pattern](#)

Tutorial: Use a Web App Bot enabled with Language Understanding in C#

7/26/2019 • 8 minutes to read • [Edit Online](#)

Use C# to build a chat bot integrated with language understanding (LUIS). The bot is built with the Azure [Web app bot](#) resource and [Bot Framework version V4](#).

In this tutorial, you learn how to:

- Create a web app bot. This process creates a new LUIS app for you.
- Download the bot project created by the Web bot service
- Start bot & emulator locally on your computer
- View utterance results in bot

Prerequisites

- [Bot emulator](#)
- [Visual Studio](#)

Create a web app bot resource

1. In the [Azure portal](#), select **Create new resource**.
2. In the search box, search for and select **Web App Bot**. Select **Create**.
3. In **Bot Service**, provide the required information:

SETTING	PURPOSE	SUGGESTED SETTING
Bot name	Resource name	<code>luis-csharp-bot-</code> + <code><your-name></code> , for example, <code>luis-csharp-bot-johnsmith</code>
Subscription	Subscription where to create bot.	Your primary subscription.
Resource group	Logical group of Azure resources	Create a new group to store all resources used with this bot, name the group <code>luis-csharp-bot-resource-group</code> .
Location	Azure region - this doesn't have to be the same as the LUIS authoring or publishing region.	<code>westus</code>
Pricing tier	Used for service request limits and billing.	<code>F0</code> is the free tier.

SETTING	PURPOSE	SUGGESTED SETTING
App name	The name is used as the subdomain when your bot is deployed to the cloud (for example, <code>humanresourcesbot.azurewebsites.net</code>).	<code>luis-csharp-bot-</code> + <code><your-name></code> , for example, <code>luis-csharp-bot-johnsmith</code>
Bot template	Bot framework settings - see next table	
LUIS App location	Must be the same as the LUIS resource region	<code>westus</code>
App service plan/Location	Do not change from provided default value.	
Application Insights	Do not change from provided default value.	
Microsoft App ID and password	Do not change from provided default value.	

4. In the **Bot template**, select the following, then choose the **Select** button under these settings:

SETTING	PURPOSE	SELECTION
SDK version	Bot framework version	SDK v4
SDK language	Programming language of bot	C#
Bot	Type of bot	Basic bot

5. Select **Create**. This creates and deploys the bot service to Azure. Part of this process creates a LUIS app named `luis-csharp-bot-xxxx`. This name is based on the /Azure Bot Service app name.

The screenshot shows two overlapping windows. The left window is titled "Web App Bot" and contains fields for "Bot name" (luis-csharp-bot-johnsmith), "Subscription" (documentationteam), "Resource group" (documentationteam), "Location" (West US), "Pricing tier" (\$1 (1K Premium Msgs/Unit)), "App name" (luis-csharp-bot-johnsmith), "Bot template" (Basic Bot (C#)), "LUIS App location" (West US), "App service plan/Location" (WebApplication1-dev-asp/East US), "Application Insights" (On), and "Application Insights Location" (East US). The right window is titled "Bot template" and shows a list of templates: Echo Bot, Basic Bot, Enterprise Bot, Virtual Assistant, Language Understanding Bot, and a section for LUIS App. The "Basic Bot (C#)" template is selected.

Wait until the bot service is created before continuing.

The bot has a Language Understanding model

The bot service creation process also creates a new LUIS app with intents and example utterances. The bot provides intent mapping to the new LUIS app for the following intents:

BASIC BOT LUIS INTENTS	EXAMPLE UTTERANCE
Book flight	Travel to Paris
Cancel	bye
None	Anything outside the domain of the app.

Test the bot in Web Chat

1. While still in the Azure portal for the new bot, select **Test in Web Chat**.
2. In the **Type your message** textbox, enter the text `hello`. The bot responds with information about the bot

framework, as well as example queries for the specific LUIS model such as booking a flight to Paris.

Home > luis-csharp-bot-johnsmith - Test in Web Chat

luis-csharp-bot-johnsmith - Test in Web Chat
Web App Bot

Search (Ctrl+ /) « Test Start over

Overview
Activity log
Access control (IAM)
Tags

Bot management

Build
Test in Web Chat
Analytics
Channels
Settings
Speech priming
Bot Service pricing

App Service Settings

Configuration
All App service settings

Support + troubleshooting

New support request

luis-csharp-bot-johnsmith

What can I help you with today?
Say something like "Book a flight from Paris to Berlin on March 22, 2020"

luis-csharp-bot-johnsmith at 9:32:34 AM

Type your message...

You can use the test functionality for quickly testing your bot. For more complete testing, including debugging, download the bot code and use Visual Studio.

Download the web app bot source code

In order to develop the web app bot code, download the code and use on your local computer.

1. In the Azure portal, select **Build** from the **Bot management** section.
2. Select **Download Bot source code**.

The screenshot shows the Azure Bot Service interface. On the left, there's a sidebar with various navigation links like Overview, Activity log, Access control (IAM), Tags, Bot management, Build (which is selected and highlighted with a red box), Test in Web Chat, Analytics, Channels, Settings, Speech priming, Bot Service pricing, App Service Settings, Application Settings, All App service settings, Support + troubleshooting, and New support request. The main content area has three sections: 'Build: Get set up with local development', 'Test: Use emulator to test and refine', and 'Publish: Upload to Azure'. The 'Build' section contains links for Download Bot source code (which is highlighted with a red box), Download Emulator, and Download command line tools. Below these links is a note: 'Make quick changes to your bot code online, run build.cmd in the editor console, and see your changes instantly.' It also provides a link to 'Open online code editor'. The 'Test' section and 'Publish' section contain general instructions and links for further action.

3. When the pop-up dialog asks **Include app settings in the downloaded zip file?**, select **Yes**.
4. When the source code is zipped, a message will provide a link to download the code. Select the link.
5. Save the zip file to your local computer and extract the files. Open the project with Visual Studio.

Review code to send utterance to LUIS and get response

1. Open the **LuisHelper.cs** file. This is where the user utterance entered into the bot is sent to LUIS. The response from LUIS is returned from the method as a **BookDetails** object. When you create your own bot, you should also create your own object to return the details from LUIS.

```

// Copyright (c) Microsoft Corporation. All rights reserved.
// Licensed under the MIT License.

using System;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.AI.Luis;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;

namespace Microsoft.BotBuilderSamples
{
    public static class LuisHelper
    {
        public static async Task<BookingDetails> ExecuteLuisQuery(IConfiguration configuration, ILogger logger, ITurnContext turnContext, CancellationToken cancellationToken)
        {
            var bookingDetails = new BookingDetails();

            try
            {
                // Create the LUIS settings from configuration.
                var luisApplication = new LuisApplication(
                    configuration["LuisAppId"],
                    configuration["LuisAPIKey"],
                    "https://" + configuration["LuisAPIHostName"]
                );

                var recognizer = new LuisRecognizer(luisApplication);

                // The actual call to LUIS
                var recognizerResult = await recognizer.RecognizeAsync(turnContext, cancellationToken);

                var (intent, score) = recognizerResult.GetTopScoringIntent();
                if (intent == "Book_flight")
                {
                    // We need to get the result from the LUIS JSON which at every level returns an array.
                    bookingDetails.Destination = recognizerResult.Entities["To"]?.FirstOrDefault()?
                        ["Airport"]?.FirstOrDefault()?.FirstOrDefault()?.ToString();
                    bookingDetails.Origin = recognizerResult.Entities["From"]?.FirstOrDefault()?
                        ["Airport"]?.FirstOrDefault()?.FirstOrDefault()?.ToString();

                    // This value will be a TIMEX. And we are only interested in a Date so grab the first result and drop the Time part.
                    // TIMEX is a format that represents DateTime expressions that include some ambiguity. e.g. missing a Year.
                    bookingDetails.TravelDate = recognizerResult.Entities["datetime"]?.FirstOrDefault()?
                        ["timex"]?.FirstOrDefault()?.ToString().Split('T')[0];
                }
            }
            catch (Exception e)
            {
                logger.LogWarning($"LUIS Exception: {e.Message} Check your LUIS configuration.");
            }

            return bookingDetails;
        }
    }
}

```

2. Open **BookingDetails.cs** to view how the object abstracts the LUIS information.

```

// Copyright (c) Microsoft Corporation. All rights reserved.
// Licensed under the MIT License.

namespace Microsoft.BotBuilderSamples
{
    public class BookingDetails
    {
        public string Destination { get; set; }

        public string Origin { get; set; }

        public string TravelDate { get; set; }
    }
}

```

3. Open **Dialogs -> BookingDialog.cs** to understand how the BookingDetails object is used to manage the conversation flow. Travel details are asked in steps, then the entire booking is confirmed and finally repeated back to the user.

```

// Copyright (c) Microsoft Corporation. All rights reserved.
// Licensed under the MIT License.

using System.Threading;
using System.Threading.Tasks;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Recognizers.Text.DataTypes.TimexExpression;

namespace Microsoft.BotBuilderSamples.Dialogs
{
    public class BookingDialog : CancelAndHelpDialog
    {
        public BookingDialog()
            : base(nameof(BookingDialog))
        {

            AddDialog(new TextPrompt(nameof(TextPrompt)));
            AddDialog(new ConfirmPrompt(nameof(ConfirmPrompt)));
            AddDialog(new DateResolverDialog());
            AddDialog(new WaterfallDialog(nameof(WaterfallDialog)), new WaterfallStep[]
            {
                DestinationStepAsync,
                OriginStepAsync,
                TravelDateStepAsync,
                ConfirmStepAsync,
                FinalStepAsync,
            }));
        }

        // The initial child Dialog to run.
        InitialDialogId = nameof(WaterfallDialog);
    }

    private async Task<DialogTurnResult> DestinationStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
    {
        var bookingDetails = (BookingDetails)stepContext.Options;

        if (bookingDetails.Destination == null)
        {
            return await stepContext.PromptAsync(nameof(TextPrompt), new PromptOptions { Prompt =
MessageFactory.Text("Where would you like to travel to?") }, cancellationToken);
        }
        else
        {
            return await stepContext.NextAsync(bookingDetails.Destination, cancellationToken);
        }
    }
}

```

```

        }

        private async Task<DialogTurnResult> OriginStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    var bookingDetails = (BookingDetails)stepContext.Options;

    bookingDetails.Destination = (string)stepContext.Result;

    if (bookingDetails.Origin == null)
    {
        return await stepContext.PromptAsync(nameof(TextPrompt), new PromptOptions { Prompt =
MessageFactory.Text("Where are you traveling from?") }, cancellationToken);
    }
    else
    {
        return await stepContext.NextAsync(bookingDetails.Origin, cancellationToken);
    }
}

private async Task<DialogTurnResult> TravelDateStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    var bookingDetails = (BookingDetails)stepContext.Options;

    bookingDetails.Origin = (string)stepContext.Result;

    if (bookingDetails.TravelDate == null || IsAmbiguous(bookingDetails.TravelDate))
    {
        return await stepContext.BeginDialogAsync(nameof(DateResolverDialog),
bookingDetails.TravelDate, cancellationToken);
    }
    else
    {
        return await stepContext.NextAsync(bookingDetails.TravelDate, cancellationToken);
    }
}

private async Task<DialogTurnResult> ConfirmStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    var bookingDetails = (BookingDetails)stepContext.Options;

    bookingDetails.TravelDate = (string)stepContext.Result;

    var msg = $"Please confirm, I have you traveling to: {bookingDetails.Destination} from:
{bookingDetails.Origin} on: {bookingDetails.TravelDate}";

    return await stepContext.PromptAsync(nameof(ConfirmPrompt), new PromptOptions { Prompt =
MessageFactory.Text(msg) }, cancellationToken);
}

private async Task<DialogTurnResult> FinalStepAsync(WaterfallStepContext stepContext,
CancellationToken cancellationToken)
{
    if ((bool)stepContext.Result)
    {
        var bookingDetails = (BookingDetails)stepContext.Options;

        return await stepContext.EndDialogAsync(bookingDetails, cancellationToken);
    }
    else
    {
        return await stepContext.EndDialogAsync(null, cancellationToken);
    }
}

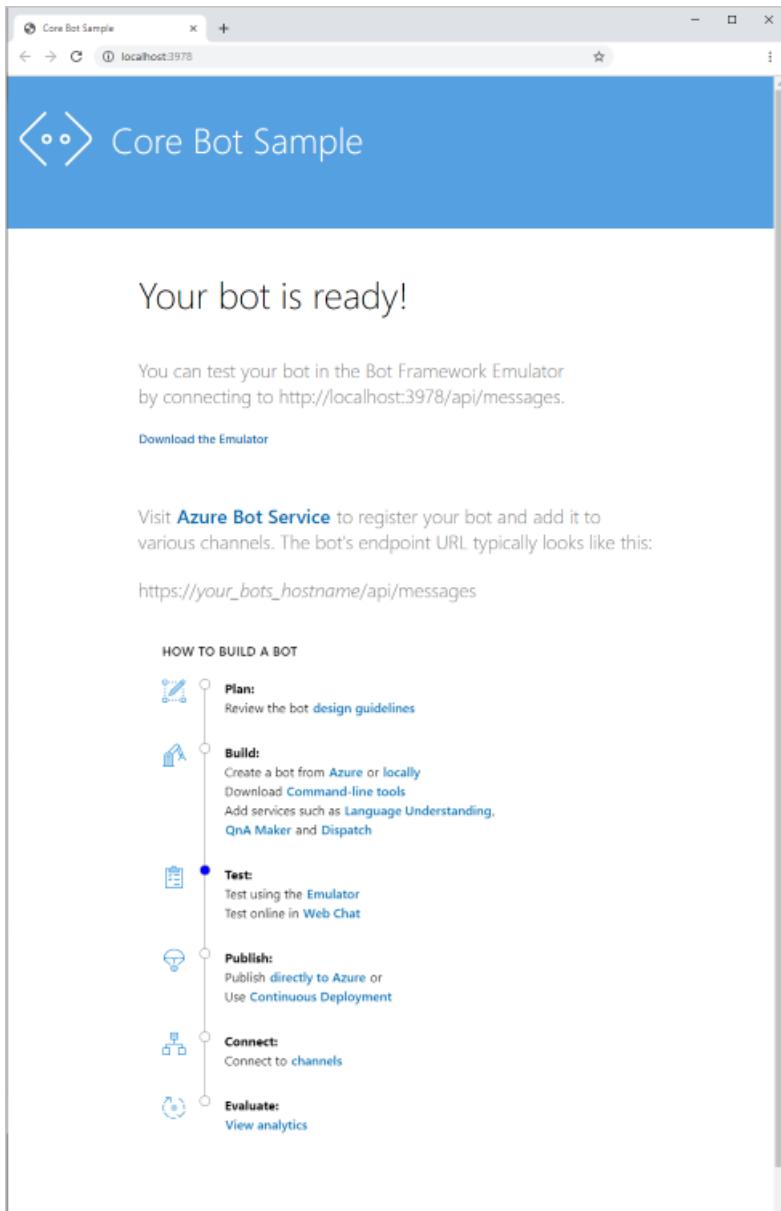
private static bool IsAmbiguous(string timex)
{
    var timexProperty = new TimexProperty(timex);
}

```

```
        return !timexProperty.Types.Contains(Constants.TimexTypes.Definite);
    }
}
```

Start the bot code in Visual Studio

In Visual Studio, start the bot. A browser window opens with the web app bot's web site at <http://localhost:3978/>. A home page displays with information about your bot.



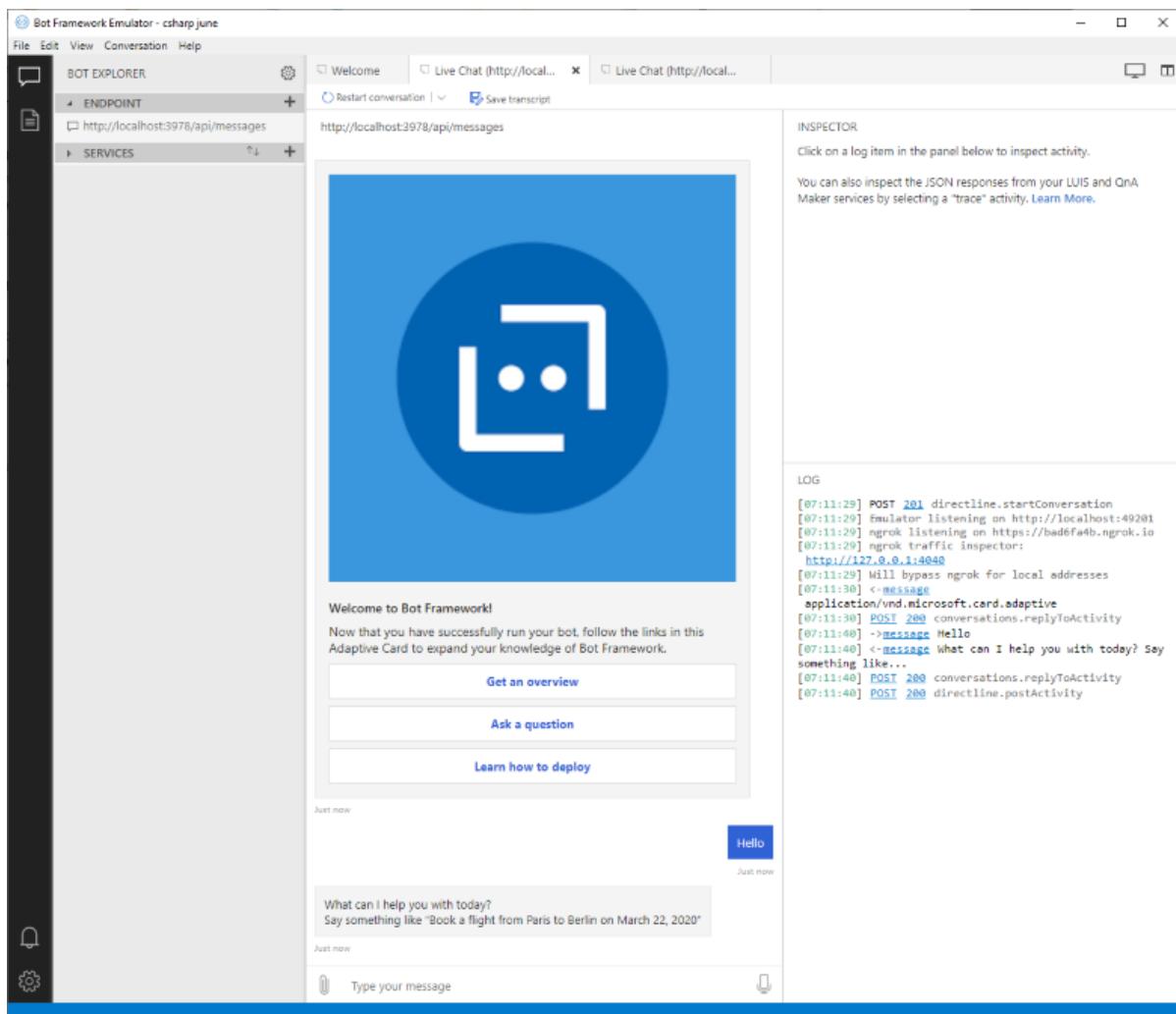
Use the bot emulator to test the bot

1. Begin the Bot Emulator and select **Open Bot**.
2. In the **Open a bot** pop-up dialog, enter your bot URL, such as <http://localhost:3978/api/messages>. The `/api/messages` route is the web address for the bot.
3. Enter the **Microsoft App ID** and **Microsoft App password**, found in the **appsettings.json** file in the root of the bot code you downloaded.

Optionally, you can create a new bot configuration and copy the `appId` and `appPassword` from the **appsettings.json** file in the Visual Studio project for the bot. The name of the bot configuration file should be the same as the bot name.

```
{
  "name": "<bot name>",
  "description": "<bot description>",
  "services": [
    {
      "type": "endpoint",
      "appId": "<appId from appsettings.json>",
      "appPassword": "<appPassword from appsettings.json>",
      "endpoint": "http://localhost:3978/api/messages",
      "id": "<don't change this value>",
      "name": "http://localhost:3978/api/messages"
    }
  ],
  "padlock": "",
  "version": "2.0",
  "overrides": null,
  "path": "<local path to .bot file>"
}
```

4. In the bot emulator, enter `Hello` and get the same response for the basic bot as you received in the **Test in Web Chat**.



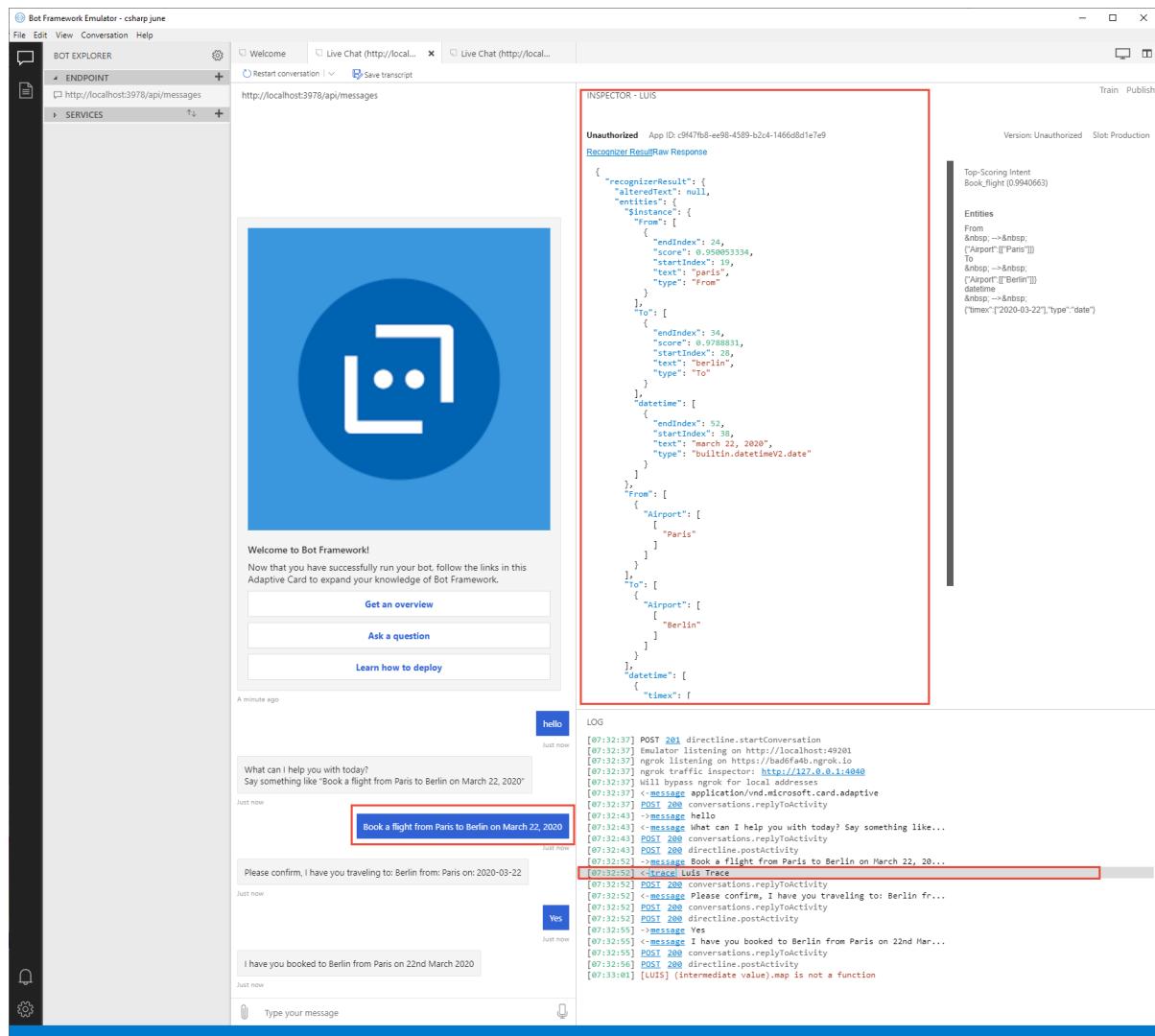
Ask bot a question for the Book Flight intent

1. In the bot emulator, book a flight by entering the following utterance:

```
Book a flight from Paris to Berlin on March 22, 2020
```

The bot emulator asks to confirm.

2. Select **Yes**. The bot responds with a summary of its actions.
3. From the log of the bot emulator, select the line that includes **Luis Trace**. This displays the JSON response from LUIS for the intent and entities of the utterance.



More information about bots

For more information about using this service with bots, begin with the following resources:

RESOURCE	PURPOSE
Azure Bot service	The Azure Bot service provides a complete cloud-hosted web service with a bot endpoint. The service uses Bot framework , which is available in several languages.
Bot Framework	The Microsoft Bot Framework is a comprehensive platform for building enterprise-grade conversational AI experiences.
Bot Framework Emulator	The Bot Framework Emulator is a cross-platform desktop application that allows bot developers to test and debug bots built using the Bot Framework SDK. You can use the Bot Framework Emulator to test bots running locally on your machine or to connect to bots running remotely.

RESOURCE	PURPOSE
Bot tools	The Bot Framework tools are a collection of cross-platform command line tools designed to cover end-to-end bot development workflow.
Bot builder samples	Full-developed bot samples are designed to illustrate scenarios you'll need to implement to build great bots.

Next steps

See more [samples](#) with conversational bots.

[Build a Language Understanding app with a custom subject domain](#)

Add LUIS results to Application Insights from a Bot in C#

7/26/2019 • 5 minutes to read • [Edit Online](#)

This tutorial adds bot and Language Understanding information to [Application Insights](#) telemetry data storage. Once you have that data, you can query it with the Kusto language or Power BI to analyze, aggregate, and report on intents, and entities of the utterance in real-time. This analysis helps you determine if you should add or edit the intents and entities of your LUIS app.

In this tutorial, you learn how to:

- Capture bot and Language understanding data in Application Insights
- Query Application Insights for Language Understanding data

Prerequisites

- An Azure bot service bot, created with Application Insights enabled.
- Downloaded bot code from the previous bot [tutorial](#).
- [Bot emulator](#)
- [Visual Studio Code](#)

All of the code in this tutorial is available on the [Azure-Samples Language Understanding GitHub repository](#).

Add Application Insights to web app bot project

Currently, the Application Insights service, used in this web app bot, collects general state telemetry for the bot. It does not collect LUIS information.

In order to capture the LUIS information, the web app bot needs the [Microsoft.ApplicationInsights](#) NuGet package installed and configured.

1. From Visual Studio, add the dependency to the solution. In the **Solution Explorer**, right-click on the project name and select **Manage NuGet Packages....** The NuGet Package manager shows a list of installed packages.
2. Select **Browse** then search for [Microsoft.ApplicationInsights](#).
3. Install the package.

Capture and send LUIS query results to Application Insights

1. Open the `LuisHelper.cs` file and replace the contents with the following code. The **LogToApplicationInsights** method capture the bot and LUIS data and sends it to Application Insights as a Trace event named `LUIS`.

```
// Copyright (c) Microsoft Corporation. All rights reserved.  
// Licensed under the MIT License.  
  
using System;  
using System.Linq;  
using System.Threading;  
using System.Threading.Tasks;  
using Microsoft.Bot.Builder;  
using Microsoft.Bot.Builder.AI.Luis;  
using Microsoft.Extensions.Configuration;
```

```

using Microsoft.Extensions.Logging;
using Microsoft.ApplicationInsights;
using System.Collections.Generic;

namespace Microsoft.BotBuilderSamples
{
    public static class LuisHelper
    {
        public static async Task<BookingDetails> ExecuteLuisQuery(IConfiguration configuration, ILogger logger, ITurnContext turnContext, CancellationToken cancellationToken)
        {
            var bookingDetails = new BookingDetails();

            try
            {
                // Create the LUIS settings from configuration.
                var luisApplication = new LuisApplication(
                    configuration["LuisAppId"],
                    configuration["LuisAPIKey"],
                    "https://" + configuration["LuisAPIHostName"]
                );

                var recognizer = new LuisRecognizer(luisApplication);

                // The actual call to LUIS
                var recognizerResult = await recognizer.RecognizeAsync(turnContext, cancellationToken);

                LuisHelper.LogToApplicationInsights(configuration, turnContext, recognizerResult);

                var (intent, score) = recognizerResult.GetTopScoringIntent();
                if (intent == "Book_flight")
                {
                    // We need to get the result from the LUIS JSON which at every level returns an array.
                    bookingDetails.Destination = recognizerResult.Entities["To"]?.FirstOrDefault()?
                    ["Airport"]?.FirstOrDefault()?.ToString();
                    bookingDetails.Origin = recognizerResult.Entities["From"]?.FirstOrDefault()?
                    ["Airport"]?.FirstOrDefault()?.ToString();

                    // This value will be a TIMEX. And we are only interested in a Date so grab the first result and drop the Time part.
                    // TIMEX is a format that represents DateTime expressions that include some ambiguity. e.g. missing a Year.
                    bookingDetails.TravelDate = recognizerResult.Entities["datetime"]?.FirstOrDefault()?
                    ["timex"]?.FirstOrDefault()?.ToString().Split('T')[0];
                }
                catch (Exception e)
                {
                    logger.LogWarning($"LUIS Exception: {e.Message} Check your LUIS configuration.");
                }

                return bookingDetails;
            }
            public static void LogToApplicationInsights(IConfiguration configuration, ITurnContext turnContext, RecognizerResult result)
            {
                // Create Application Insights object
                TelemetryClient telemetry = new TelemetryClient();

                // Set Application Insights Instrumentation Key from App Settings
                telemetry.InstrumentationKey = configuration["BotDevAppInsightsKey"];

                // Collect information to send to Application Insights
                Dictionary<string, string> logProperties = new Dictionary<string, string>();

                logProperties.Add("BotConversation", turnContext.Activity.Conversation.Name);
                logProperties.Add("Bot_userId", turnContext.Activity.Conversation.Id);
            }
        }
    }
}

```

```

        logProperties.Add("LUIS_query", result.Text);
        logProperties.Add("LUIS_topScoringIntent_Name", result.GetTopScoringIntent().intent);
        logProperties.Add("LUIS_topScoringIntentScore",
            result.GetTopScoringIntent().score.ToString());

        // Add entities to collected information
        int i = 1;
        if (result.Entities.Count > 0)
        {
            foreach (var item in result.Entities)
            {
                logProperties.Add("LUIS_entities_" + i++ + "_" + item.Key, item.Value.ToString());
            }
        }

        // Send to Application Insights
        telemetry.TrackTrace("LUIS", ApplicationInsights.DataContracts.SeverityLevel.Information,
            logProperties);

    }
}

```

Add Application Insights instrumentation key

In order to add data to application insights, you need the instrumentation key.

1. In a browser, in the [Azure portal](#), find your bot's **Application Insights** resource. Its name will have most of the bot's name, then random characters at the end of the name, such as `luis-csharp-bot-johnsmithxqowom`.
2. On the Application Insights resource, on the **Overview** page, copy the **Instrumentation Key**.
3. In Visual Studio, open the **appsettings.json** file at the root of the bot project. This file holds all your environment variables.
4. Add a new variable, `BotDevAppInsightsKey` with the value of your instrumentation key. The value in should be in quotes.

Build and start the bot

1. In Visual Studio, build and run the bot.
2. Start the bot emulator and open the bot. This [step](#) is provided in the previous tutorial.
3. Ask the bot a question. This [step](#) is provided in the previous tutorial.

View LUIS entries in Application Insights

Open Application Insights to see the LUIS entries. It can take a few minutes for the data to appear in Application Insights.

1. In the [Azure portal](#), open the bot's Application Insights resource.
2. When the resource opens, select **Search** and search for all data in the last **30 minutes** with the event type of **Trace**. Select the trace named **LUIS**.
3. The bot and LUIS information is available under **Custom Properties**.

The screenshot shows the Application Insights Log Analytics interface. On the left, there's a summary bar with '1 All', '1 Traces', and '0 Events'. A blue button labeled 'View timeline' is visible. Below this is a search/filter bar with dropdowns for 'Component | Call' and 'All [Component | Call]'. A large text area displays a single trace entry:

```
| INFORMATION 9:55:54.041 AM |  
LUIS
```

To the right, under 'Trace Properties', are the following details:

Trace Properties		Show all
Event time	6/16/2019, 9:55:54 AM	
Device type	PC	...
Message	LUIS	...
Severity level	Information	...

Under 'Custom Properties', there are several entries:

Custom Property Name	Value	...
LUIS_entities_4_datetime	[{"type": "date", "timex": "2020-03-22"}]	...
LUIS_topScoringIntent_Name	Book_flight	...
LUIS_entities_2_From	[{"\$instance": {"Airport": [{"startIndex": 19, "endIndex": 24, "text": "paris", "type": "Airport"}]}], ... [show more]	...
LUIS_topScoringIntent_Score	0.9940663	...
LUIS_query	Book a flight from Paris to Berlin on March 22, 2020	...
Bot_userId	8e94ed31-9057-11e9-b3a8-3fa35487ab7e livechat	...
LUIS_entities_3_To	[{"\$instance": {"Airport": [{"startIndex": 28, "endIndex": 34, "text": "berlin", "type": "Airport"}]}], ... [show more]	...
LUIS_entities_1_Instance	{"From": [{"startIndex": 19, "endIndex": 24, "text": "paris", "type": "From", "score": 0.95005334}], "To": [{"startIndex": 28, ... [show more]	...

At the bottom, a 'Related Items' section is partially visible.

Query Application Insights for intent, score, and utterance

Application Insights gives you the power to query the data with the [Kusto](#) language, as well as export it to [Power BI](#).

1. Select **Log (Analytics)**. A new window opens with a query window at the top and a data table window below that. If you have used databases before, this arrangement is familiar. The query represents your previous filtered data. The **CustomDimensions** column has the bot and LUIS information.
2. To pull out the top intent, score, and utterance, add the following just above the last line (the `|top...` line) in the query window:

```
| extend topIntent = tostring(customDimensions.LUIS_topScoringIntent_Name)
| extend score = todouble(customDimensions.LUIS_topScoringIntentScore)
| extend utterance = tostring(customDimensions.LUIS_query)
```

3. Run the query. The new columns of topIntent, score, and utterance are available. Select topIntent column to sort.

Learn more about the [Kusto query language](#) or export the data to Power BI.

Learn more about Bot Framework

Learn more about [Bot Framework](#).

Next steps

Other information you may want to add to the application insights data includes app ID, version ID, last model

change date, last train date, last publish date. These values can either be retrieved from the endpoint URL (app ID and version ID), or from an authoring API call then set in the web app bot settings and pulled from there.

If you are using the same endpoint subscription for more than one LUIS app, you should also include the subscription ID and a property stating that it is a shared key.

[Learn more about example utterances](#)

Tutorial: Use a Web App Bot enabled with Language Understanding in Node.js

6/24/2019 • 8 minutes to read • [Edit Online](#)

Use Node.js to build a chat bot integrated with language understanding (LUIS). The bot is built with the Azure [Web app bot](#) resource and [Bot Framework version V4](#).

In this tutorial, you learn how to:

- Create a web app bot. This process creates a new LUIS app for you.
- Download the bot project created by the Web bot service
- Start bot & emulator locally on your computer
- View utterance results in bot

Prerequisites

- [Bot emulator](#)
- [Visual Studio Code](#)

Create a web app bot resource

1. In the [Azure portal](#), select **Create new resource**.
2. In the search box, search for and select **Web App Bot**. Select **Create**.
3. In **Bot Service**, provide the required information:

SETTING	PURPOSE	SUGGESTED SETTING
Bot name	Resource name	<code>luis-nodejs-bot-</code> + <code><your-name></code> , for example, <code>luis-nodejs-bot-johnsmith</code>
Subscription	Subscription where to create bot.	Your primary subscription.
Resource group	Logical group of Azure resources	Create a new group to store all resources used with this bot, name the group <code>luis-nodejs-bot-resource-group</code> .
Location	Azure region - this doesn't have to be the same as the LUIS authoring or publishing region.	<code>westus</code>
Pricing tier	Used for service request limits and billing.	<code>F0</code> is the free tier.

SETTING	PURPOSE	SUGGESTED SETTING
App name	The name is used as the subdomain when your bot is deployed to the cloud (for example, <code>humanresourcesbot.azurewebsites.net</code>).	<code>luis-nodejs-bot-</code> + <code><your-name></code> , for example, <code>luis-nodejs-bot-johnsmith</code>
Bot template	Bot framework settings - see next table	
LUIS App location	Must be the same as the LUIS resource region	<code>westus</code>
App service plan/Location	Do not change from provided default value.	
Application Insights	Do not change from provided default value.	
Microsoft App ID and password	Do not change from provided default value.	

4. In the **Bot template**, select the following, then choose the **Select** button under these settings:

SETTING	PURPOSE	SELECTION
SDK version	Bot framework version	SDK v4
SDK language	Programming language of bot	Node.js
Bot	Type of bot	Basic bot

5. Select **Create**. This creates and deploys the bot service to Azure. Part of this process creates a LUIS app named `luis-nodejs-bot-xxxx`. This name is based on the /Azure Bot Service app name.

The screenshot shows two overlapping windows. On the left is the 'Web App Bot' configuration window, which includes fields for Bot name (luis-nodejs-bot-john-smith), Subscription (documentationteam), Resource group (documentationteam), Location (West US), Pricing tier (F0 (10K Premium Messages)), App name (luis-nodejs-bot-john-smith), Bot template (Basic Bot (NodeJS)), LUIS App location (West US), App service plan/Location (AppSvcPlan/East US), Application Insights (On), Application Insights Location (East US), and Microsoft App ID and password (Auto create App ID and password). At the bottom are 'Create' and 'Automation options' buttons. On the right is the 'Bot template' selection window, titled 'Choose a template'. It lists several templates: Echo Bot (NodeJS, description: Simple bot that echoes back the user's message), Basic Bot (NodeJS, description: This bot template contains the following services: Language Understanding and Bot Analytics), Enterprise Bot (description: This template includes all Basic Bot services, as well as: CosmosDB, Dispatch, QnA Maker, Authentication, Content Moderator, App Insights, PowerBI and example dialogs. Download the [Enterprise Bot template](#) and follow the instructions in the Readme.md.), Virtual Assistant (description: This includes all Enterprise Bot services and provides additional capabilities to enable creation of a Virtual Assistant including Linked Accounts and Skills. Download the [Virtual Assistant template](#) and follow the instructions in the Readme.md.), and Language Understanding Bot (description: Create a Language Understanding bot from a [new or existing LUIS app](#)). An 'OK' button is at the bottom of the template list.

Wait until the bot service is created before continuing.

The bot has a Language Understanding model

The bot service creation process also creates a new LUIS app with intents and example utterances. The bot provides intent mapping to the new LUIS app for the following intents:

BASIC BOT LUIS INTENTS	EXAMPLE UTTERANCE
Book flight	Travel to Paris
Cancel	bye
None	Anything outside the domain of the app.

Test the bot in Web Chat

1. While still in the Azure portal for the new bot, select **Test in Web Chat**.
2. In the **Type your message** textbox, enter the text `hello`. The bot responds with information about the bot

framework, as well as example queries for the specific LUIS model such as booking a flight to Paris.

Home > luis-nodejs-bot-johnsmith - Test in Web Chat

luis-nodejs-bot-johnsmith - Test in Web Chat
Web App Bot

Search (Ctrl+ /) « Test Start over

Overview
Activity log
Access control (IAM)
Tags

Bot management

Build
Test in Web Chat
Analytics
Channels
Settings
Speech priming
Bot Service pricing

App Service Settings

Configuration
All App service settings

Support + troubleshooting

New support request

Welcome to Bot Framework!

Now that you have successfully run your bot, follow the links in this Adaptive Card to expand your knowledge of Bot Framework.

Get an overview
Ask a question
Learn how to deploy

luis-nodejs-bot-johnsmith

What can I help you with today?
Say something like "Book a flight from Paris to Berlin on March 22, 2020"

luis-nodejs-bot-johnsmith at 9:35:10 AM

Type your message...

You can use the test functionality for quickly testing your bot. For more complete testing, including debugging, download the bot code and use Visual Studio.

Download the web app bot source code

In order to develop the web app bot code, download the code and use on your local computer.

1. In the Azure portal, select **Build** from the **Bot management** section.
2. Select **Download Bot source code**.

The screenshot shows the Azure Bot Service interface. On the left, there's a sidebar with various navigation options like Overview, Activity log, Access control (IAM), Tags, Bot management, Build (which is selected and highlighted with a red box), Test in Web Chat, Analytics, Channels, Settings, Speech priming, Bot Service pricing, App Service Settings, Application Settings, All App service settings, Support + troubleshooting, and New support request. The main content area has three columns: 'Build: Get set up with local development' (with a 'Download Bot source code' button highlighted with a red box), 'Test: Use emulator to test and refine' (with 'Download Emulator' and 'Download command line tools' buttons), and 'Publish: Upload to Azure' (with information about publishing to Azure and continuous deployment).

3. When the pop-up dialog asks **Include app settings in the downloaded zip file?**, select **Yes**.
4. When the source code is zipped, a message will provide a link to download the code. Select the link.
5. Save the zip file to your local computer and extract the files. Open the project with Visual Studio.

Review code to send utterance to LUIS and get response

1. Open the **dialogs -> luisHelper.js** file. This is where the user utterance entered into the bot is sent to LUIS. The response from LUIS is returned from the method as a **bookDetails** JSON object. When you create your own bot, you should also create your own object to return the details from LUIS.

```
// Copyright (c) Microsoft Corporation. All rights reserved.
// Licensed under the MIT License.

const { LuisRecognizer } = require('botbuilder-ai');

class LuisHelper {
    /**
     * Returns an object with preformatted LUIS results for the bot's dialogs to consume.
     * @param {*} logger
     * @param {TurnContext} context
     */
    static async executeLuisQuery(logger, context) {
        const bookingDetails = {};

        try {
            const recognizer = new LuisRecognizer({
                applicationId: process.env.LuisAppId,
                endpointKey: process.env.LuisAPIKey,
                endpoint: `https://${process.env.LuisAPIHostName}`
            }, {}, true);

            const recognizerResult = await recognizer.recognize(context);

            const intent = LuisRecognizer.topIntent(recognizerResult);

            bookingDetails.intent = intent;

            if (intent === 'Book_flight') {
                // We need to get the result from the LUIS JSON which at every level returns an array

                bookingDetails.destination = LuisHelper.parseCompositeEntity(recognizerResult, 'To', 'Airport');
                bookingDetails.origin = LuisHelper.parseCompositeEntity(recognizerResult, 'From', 'Airport');
            }
        }
    }
}
```

```

        // This value will be a TIMEX. And we are only interested in a Date so grab the first
        result and drop the Time part.
        // TIMEX is a format that represents DateTime expressions that include some ambiguity.
        e.g. missing a Year.
            bookingDetails.travelDate = LuisHelper.parseDatetimeEntity(recognizerResult);
        }
    } catch (err) {
        logger.warn(`LUIS Exception: ${ err } Check your LUIS configuration`);
    }
    return bookingDetails;
}

static parseCompositeEntity(result, compositeName, entityName) {
    const compositeEntity = result.entities[compositeName];
    if (!compositeEntity || !compositeEntity[0]) return undefined;

    const entity = compositeEntity[0][entityName];
    if (!entity || !entity[0]) return undefined;

    const entityValue = entity[0][0];
    return entityValue;
}

static parseDatetimeEntity(result) {
    const datetimeEntity = result.entities['datetime'];
    if (!datetimeEntity || !datetimeEntity[0]) return undefined;

    const timex = datetimeEntity[0]['timex'];
    if (!timex || !timex[0]) return undefined;

    const datetime = timex[0].split('T')[0];
    return datetime;
}
}

module.exports.LuisHelper = LuisHelper;

```

2. Open **dialogs -> bookingDialog.js** to understand how the BookingDetails object is used to manage the conversation flow. Travel details are asked in steps, then the entire booking is confirmed and finally repeated back to the user.

```

// Copyright (c) Microsoft Corporation. All rights reserved.
// Licensed under the MIT License.

const { TimexProperty } = require('@microsoft/recognizers-text-data-types-timex-expression');
const { ConfirmPrompt, TextPrompt, WaterfallDialog } = require('botbuilder-dialogs');
const { CancelAndHelpDialog } = require('./cancelAndHelpDialog');
const { DateResolverDialog } = require('./dateResolverDialog');

const CONFIRM_PROMPT = 'confirmPrompt';
const DATE_RESOLVER_DIALOG = 'dateResolverDialog';
const TEXT_PROMPT = 'textPrompt';
const WATERFALL_DIALOG = 'waterfallDialog';

class BookingDialog extends CancelAndHelpDialog {
    constructor(id) {
        super(id || 'bookingDialog');

        this.addDialog(new TextPrompt(TEXT_PROMPT))
            .addDialog(new ConfirmPrompt(CONFIRM_PROMPT))
            .addDialog(new DateResolverDialog(DATE_RESOLVER_DIALOG))
            .addDialog(new WaterfallDialog(WATERFALL_DIALOG, [
                this.destinationStep.bind(this),
                this.originStep.bind(this),
                this.travelDateStep.bind(this),
                this.confirmStep.bind(this),
            ]));
    }
}

```

```

        this.finalStep.bind(this)
    ]));

    this.initialDialogId = WATERFALL_DIALOG;
}

/**
 * If a destination city has not been provided, prompt for one.
 */
async destinationStep(stepContext) {
    const bookingDetails = stepContext.options;

    if (!bookingDetails.destination) {
        return await stepContext.prompt(TEXT_PROMPT, { prompt: 'To what city would you like to
travel?' });
    } else {
        return await stepContext.next(bookingDetails.destination);
    }
}

/**
 * If an origin city has not been provided, prompt for one.
 */
async originStep(stepContext) {
    const bookingDetails = stepContext.options;

    // Capture the response to the previous step's prompt
    bookingDetails.destination = stepContext.result;
    if (!bookingDetails.origin) {
        return await stepContext.prompt(TEXT_PROMPT, { prompt: 'From what city will you be
travelling?' });
    } else {
        return await stepContext.next(bookingDetails.origin);
    }
}

/**
 * If a travel date has not been provided, prompt for one.
 * This will use the DATE_RESOLVER_DIALOG.
 */
async travelDateStep(stepContext) {
    const bookingDetails = stepContext.options;

    // Capture the results of the previous step
    bookingDetails.origin = stepContext.result;
    if (!bookingDetails.travelDate || this.isAmbiguous(bookingDetails.travelDate)) {
        return await stepContext.beginDialog(DATE_RESOLVER_DIALOG, { date: bookingDetails.travelDate
});
    } else {
        return await stepContext.next(bookingDetails.travelDate);
    }
}

/**
 * Confirm the information the user has provided.
 */
async confirmStep(stepContext) {
    const bookingDetails = stepContext.options;

    // Capture the results of the previous step
    bookingDetails.travelDate = stepContext.result;
    const msg = `Please confirm, I have you traveling to: ${ bookingDetails.destination } from: ${
bookingDetails.origin } on: ${ bookingDetails.travelDate }.`;

    // Offer a YES/NO prompt.
    return await stepContext.prompt(CONFIRM_PROMPT, { prompt: msg });
}

```

```

        /*
         * Complete the interaction and end the dialog.
         */
        async finalStep(stepContext) {
            if (stepContext.result === true) {
                const bookingDetails = stepContext.options;

                return await stepContext.endDialog(bookingDetails);
            } else {
                return await stepContext.endDialog();
            }
        }

        isAmbiguous(timex) {
            const timexProperty = new TimexProperty(timex);
            return !timexProperty.types.has('definite');
        }
    }

    module.exports.BookingDialog = BookingDialog;

```

Install dependencies and start the bot code in Visual Studio

1. In VSCode, from the integrated terminal, install dependencies with the command `npm install`.
2. Also from the integrated terminal, start the bot with the command `npm start`.

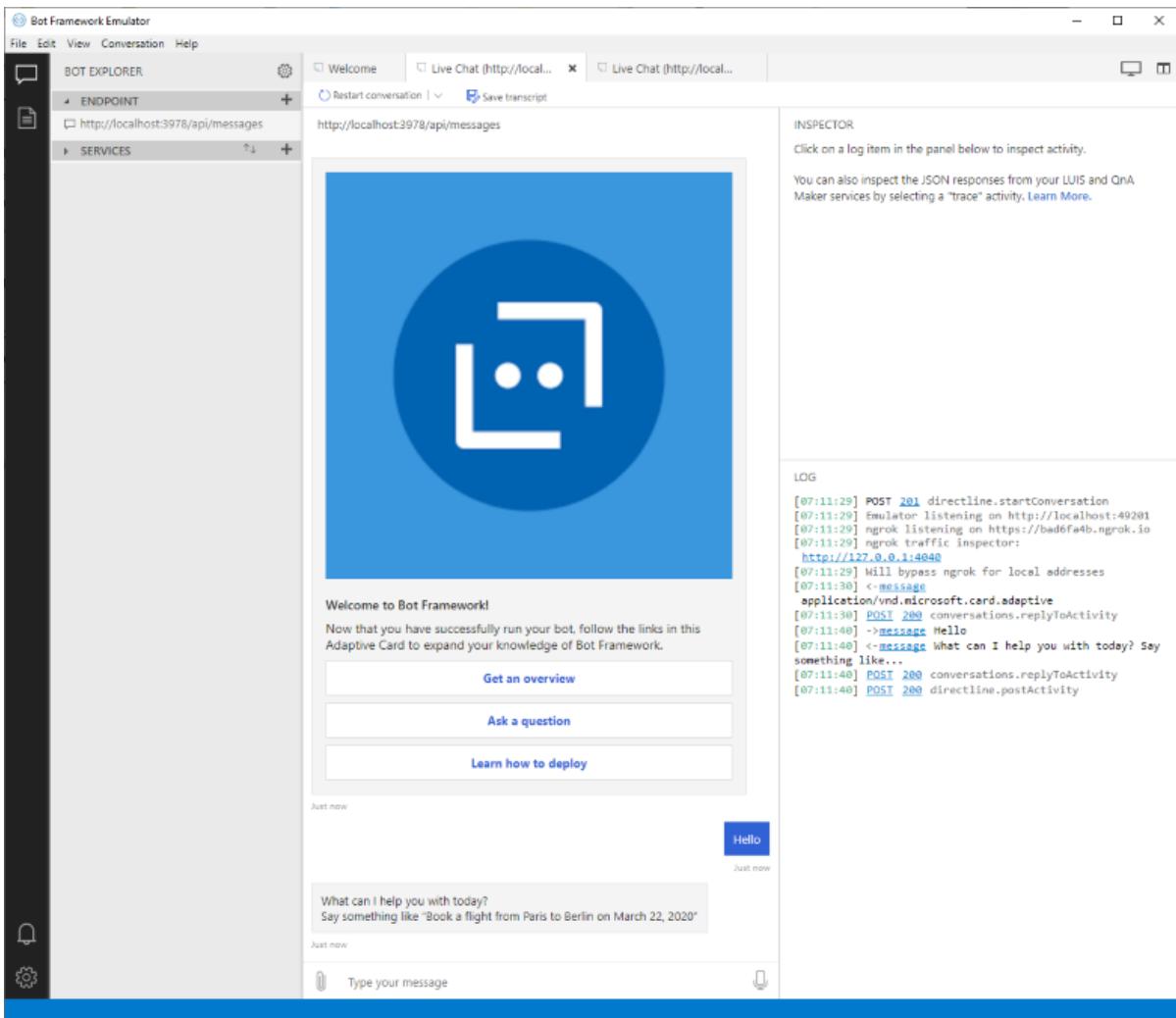
Use the bot emulator to test the bot

1. Begin the Bot Emulator and select **Open Bot**.
2. In the **Open a bot** pop-up dialog, enter your bot URL, such as `http://localhost:3978/api/messages`. The `/api/messages` route is the web address for the bot.
3. Enter the **Microsoft App ID** and **Microsoft App password**, found in the `.env` file in the root of the bot code you downloaded.

Optionally, you can create a new bot configuration and copy the `MicrosoftAppId` and `MicrosoftAppPassword` from the `.env` file in the Visual Studio project for the bot. The name of the bot configuration file should be the same as the bot name.

```
{
    "name": "<bot name>",
    "description": "<bot description>",
    "services": [
        {
            "type": "endpoint",
            "appId": "<appId from .env>",
            "appPassword": "<appPassword from .env>",
            "endpoint": "http://localhost:3978/api/messages",
            "id": "<don't change this value>",
            "name": "http://localhost:3978/api/messages"
        }
    ],
    "padlock": "",
    "version": "2.0",
    "overrides": null,
    "path": "<local path to .bot file>"
}
```

4. In the bot emulator, enter `Hello` and get the same response for the basic bot as you received in the **Test in Web Chat**.



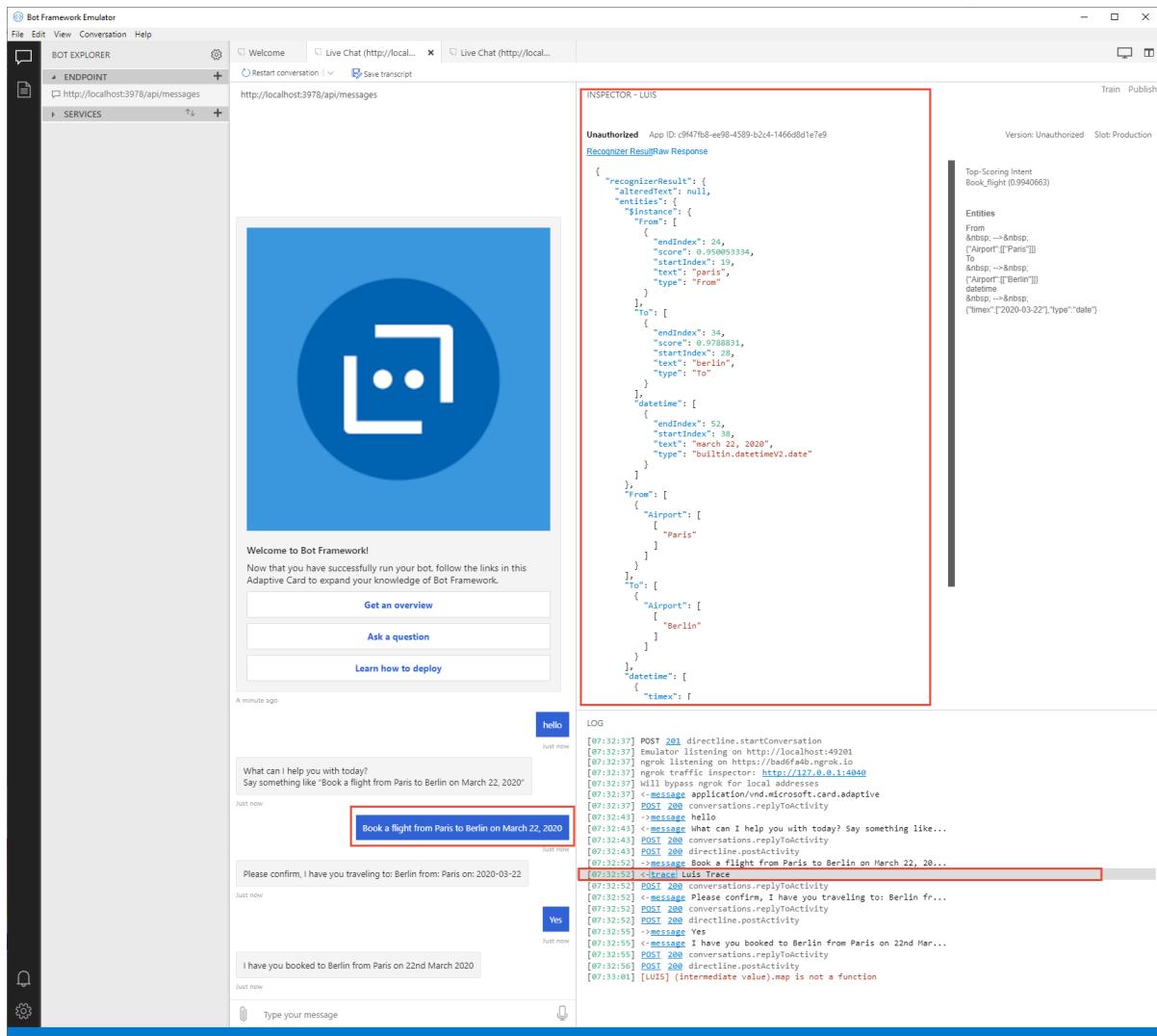
Ask bot a question for the Book Flight intent

1. In the bot emulator, book a flight by entering the following utterance:

```
Book a flight from Paris to Berlin on March 22, 2020
```

The bot emulator asks to confirm.

2. Select **Yes**. The bot responds with a summary of its actions.
3. From the log of the bot emulator, select the line that includes **Luis Trace**. This displays the JSON response from LUIS for the intent and entities of the utterance.



More information about bots

For more information about using this service with bots, begin with the following resources:

RESOURCE	PURPOSE
Azure Bot service	The Azure Bot service provides a complete cloud-hosted web service with a bot endpoint. The services uses Bot framework , which is available in several languages.
Bot Framework	The Microsoft Bot Framework is a comprehensive platform for building enterprise-grade conversational AI experiences.
Bot Framework Emulator	The Bot Framework Emulator is a cross-platform desktop application that allows bot developers to test and debug bots built using the Bot Framework SDK. You can use the Bot Framework Emulator to test bots running locally on your machine or to connect to bots running remotely.
Bot tools	The Bot Framework tools are a collection of cross-platform command line tools designed to cover end-to-end bot development workflow.
Bot builder samples	Full-developed bot samples are designed to illustrate scenarios you'll need to implement to build great bots.

Next steps

See more [samples](#) with conversational bots.

[Build a Language Understanding app with a custom subject domain](#)

Add LUIS results to Application Insights from a Bot in Node.js

7/26/2019 • 5 minutes to read • [Edit Online](#)

This tutorial adds bot and Language Understanding information to [Application Insights](#) telemetry data storage. Once you have that data, you can query it with the Kusto language or Power BI to analyze, aggregate, and report on intents, and entities of the utterance in real-time. This analysis helps you determine if you should add or edit the intents and entities of your LUIS app.

In this tutorial, you learn how to:

- Capture bot and Language understanding data in Application Insights
- Query Application Insights for Language Understanding data

Prerequisites

- An Azure bot service bot, created with Application Insights enabled.
- Downloaded bot code from the previous bot [tutorial](#).
- [Bot emulator](#)
- [Visual Studio Code](#)

All of the code in this tutorial is available on the [Azure-Samples Language Understanding GitHub repository](#).

Add Application Insights to web app bot project

Currently, the Application Insights service, used in this web app bot, collects general state telemetry for the bot. It does not collect LUIS information.

In order to capture the LUIS information, the web app bot needs the [Application Insights](#) NPM package installed and configured.

1. In the VSCode integrated terminal, at the root for the bot project, add the following NPM packages using the command shown:

```
npm install applicationinsights && npm install underscore
```

The **underscore** package is used to flatten the LUIS JSON structure so it is easier to see and use in Application Insights.

Capture and send LUIS query results to Application Insights

1. In VSCode, create a new file **appInsightsLog.js** and add the following code:

```

const appInsights = require('applicationinsights');
const _ = require("underscore");

// Log LUIS results to Application Insights
// must flatten as name/value pairs
var appInsightsLog = (botContext,luisResponse) => {

    appInsights.setup(process.env.MicrosoftApplicationInsightsInstrumentationKey).start();
    const appInsightsClient = appInsights.defaultClient;

    // put bot context and LUIS results into single object
    var data = Object.assign({}, {'botContext': botContext._activity}, {'luisResponse': luisResponse});

    // Flatten data into name/value pairs
    flatten = (x, result, prefix) => {
        if(_.isObject(x)) {
            _.each(x, (v, k) => {
                flatten(v, result, prefix ? prefix + '_' + k : k)
            })
        } else {
            result["LUIS_" + prefix] = x
        }
        return result;
    }

    // call fn to flatten data
    var flattenedData = flatten(data, {});

    // ApplicationInsights Trace
    console.log(JSON.stringify(flattenedData));

    // send data to Application Insights
    appInsightsClient.trackTrace({message: "LUIS", severity:
    appInsights.Contracts.SeverityLevel.Information, properties: flattenedData});
}

module.exports.appInsightsLog = appInsightsLog;

```

This file takes the bot context and the luis response, flattens both objects and inserts them into a **Trace** event in application insights. The event's name is **LUIS**.

2. Open the **dialogs** folder, then the **luisHelper.js** file. Include the new **appInsightsLog.js** as a required file and capture the bot context and LUIS response. The complete code for this file is:

```

// Copyright (c) Microsoft Corporation. All rights reserved.
// Licensed under the MIT License.

const { LuisRecognizer } = require('botbuilder-ai');
const { appInsightsLog } = require('../appInsightsLog');

class LuisHelper {
    /**
     * Returns an object with preformatted LUIS results for the bot's dialogs to consume.
     * @param {*} logger
     * @param {TurnContext} context
     */
    static async executeLuisQuery(logger, context) {
        const bookingDetails = {};

        try {
            const recognizer = new LuisRecognizer({
                applicationId: process.env.LuisAppId,
                endpointKey: process.env.LuisAPIKey,
                endpoint: `https://${process.env.LuisAPIHostName}``,
            }, {}, true);

```

```

const recognizerResult = await recognizer.recognize(context);

// APPINSIGHT: Log results to Application Insights
appInsightsLog(context,recognizerResult);

const intent = LuisRecognizer.topIntent(recognizerResult);

bookingDetails.intent = intent;

if (intent === 'Book_flight') {
    // We need to get the result from the LUIS JSON which at every level returns an array

    bookingDetails.destination = LuisHelper.parseCompositeEntity(recognizerResult, 'To',
'Airport');
    bookingDetails.origin = LuisHelper.parseCompositeEntity(recognizerResult, 'From',
'Airport');

    // This value will be a TIMEX. And we are only interested in a Date so grab the first
result and drop the Time part.
    // TIMEX is a format that represents DateTime expressions that include some ambiguity.
e.g. missing a Year.
    bookingDetails.travelDate = LuisHelper.parseDatetimeEntity(recognizerResult);
}
} catch (err) {
    logger.warn(`LUIS Exception: ${ err } Check your LUIS configuration`);
}
return bookingDetails;
}

static parseCompositeEntity(result, compositeName, entityName) {
    const compositeEntity = result.entities[compositeName];
    if (!compositeEntity || !compositeEntity[0]) return undefined;

    const entity = compositeEntity[0][entityName];
    if (!entity || !entity[0]) return undefined;

    const entityValue = entity[0][0];
    return entityValue;
}

static parseDatetimeEntity(result) {
    const datetimeEntity = result.entities['datetime'];
    if (!datetimeEntity || !datetimeEntity[0]) return undefined;

    const timex = datetimeEntity[0]['timex'];
    if (!timex || !timex[0]) return undefined;

    const datetime = timex[0].split('T')[0];
    return datetime;
}

module.exports.LuisHelper = LuisHelper;

```

Add Application Insights instrumentation key

In order to add data to application insights, you need the instrumentation key.

1. In a browser, in the [Azure portal](#), find your bot's **Application Insights** resource. Its name will have most of the bot's name, then random characters at the end of the name, such as `luis-nodejs-bot-johnsmithxqowom`.
2. On the Application Insights resource, on the **Overview** page, copy the **Instrumentation Key**.
3. In VSCode, open the `.env` file at the root of the bot project. This file holds all your environment variables.
4. Add a new variable, `MicrosoftApplicationInsightsInstrumentationKey` with the value of your instrumentation key.

Do not put the value in quotes.

Start the bot

1. From the VSCode integrated terminal, start the bot:

```
npm start
```

2. Start the bot emulator and open the bot. This [step](#) is provided in the previous tutorial.
3. Ask the bot a question. This [step](#) is provided in the previous tutorial.

View LUIS entries in Application Insights

Open Application Insights to see the LUIS entries. It can take a few minutes for the data to appear in Application Insights.

1. In the [Azure portal](#), open the bot's Application Insights resource.
2. When the resource opens, select **Search** and search for all data in the last **30 minutes** with the event type of **Trace**. Select the trace named **Luis**.
3. The bot and LUIS information is available under **Custom Properties**.

The screenshot shows the Azure Application Insights search interface. On the left, there's a sidebar with 'End-to-end transaction' and a timeline dropdown showing 1 All, 1 Traces, 0 Events. Below that is a filter dropdown set to 'All [Component | Call]'. A large blue box highlights a trace entry: 'I INFORMATION 12:24:52.728 PM | LUIS'. To the right, the main pane shows a 'TRACE Information' card with details: Event time (6/16/2019, 12:24:52 PM), Device type (PC), Message (LUIS), and Severity level (Information). Below this is a 'Trace Properties' table with columns for name and value. At the bottom is a 'Custom Properties' table with several entries, each with a '...' button. The properties listed include LUIS_luisResponse_text, LUIS_luisResponse_alternateText, LUIS_luisResponse_intents_Book_flight_score, LUIS_luisResponse_entities_Instance_From_0_startIndex, LUIS_luisResponse_luisResult_topScoringIntentScore, LUIS_luisResponse_luisResult_entities_0_entity, LUIS_luisResponse_luisResult_entities_0_type, LUIS_luisResponse_luisResult_entities_0_startIndex, and LUIS_luisResponse_luisResult_entities_0_endIndex.

Query Application Insights for intent, score, and utterance

Application Insights gives you the power to query the data with the [Kusto](#) language, as well as export it to [Power BI](#).

1. Select **Log (Analytics)**. A new window opens with a query window at the top and a data table window below that. If you have used databases before, this arrangement is familiar. The query represents your previous filtered data. The **CustomDimensions** column has the bot and LUIS information.
2. To pull out the top intent, score, and utterance, add the following just above the last line (the `|top...` line) in the query window:

```
| extend topIntent = tostring(customDimensions.LUIS_luisResponse_luisResult_topScoringIntent_intent)  
| extend score = todouble(customDimensions.LUIS_luisResponse_luisResult_topScoringIntent_score)  
| extend utterance = tostring(customDimensions.LUIS_luisResponse_text)
```

3. Run the query. The new columns of topIntent, score, and utterance are available. Select topIntent column to sort.

Learn more about the [Kusto query language](#) or export the data to Power BI.

Next steps

Other information you may want to add to the application insights data includes app ID, version ID, last model change date, last train date, last publish date. These values can either be retrieved from the endpoint URL (app ID and version ID), or from an authoring API call then set in the web app bot settings and pulled from there.

If you are using the same endpoint subscription for more than one LUIS app, you should also include the subscription ID and a property stating that it is a shared key.

[Learn more about example utterances](#)

Best practices for building a language understanding app with Cognitive Services

7/30/2019 • 7 minutes to read • [Edit Online](#)

Use the app authoring process to build your LUIS app:

- Build language model
- Add a few training example utterances (10-15 per intent)
- Publish
- Test from endpoint
- Add features

Once your app is [published](#), use the authoring cycle to add features, publish, and test from endpoint. Do not begin the next authoring cycle by adding more example utterances. That does not let LUIS learn your model with real-world user utterances.

In order for LUIS to be efficient at its job of learning, do not expand the utterances until the current set of both example and endpoint utterances are returning confident, high prediction scores. Improve scores using [active learning](#), [patterns](#), and [phrase lists](#).

Do and Don't

The following list includes best practices for LUIS apps:

DO	DON'T
Define distinct intents	Add many example utterances to intents
Find a sweet spot between too generic and too specific for each intent	Use LUIS as a training platform
Build your app iteratively	Add many example utterances of the same format, ignoring other formats
Add phrase lists and patterns in later iterations	Mix the definition of intents and entities
Balance your utterances across all intents except the None intent. Add example utterances to None intent	Create phrase lists with all possible values
Leverage the suggest feature for active learning	Add too many patterns
Monitor the performance of your app	Train and publish with every single example utterance added
Use versions for each app iteration	

Do define distinct intents

Make sure the vocabulary for each intent is just for that intent and not overlapping with a different intent. For example, if you want to have an app that handles travel arrangements such as airline flights and hotels, you can

choose to have these subject areas as separate intents or the same intent with entities for specific data inside the utterance.

If the vocabulary between two intents is the same, combine the intent, and use entities.

Consider the following example utterances:

EXAMPLE UTTERANCES

Book a flight

Book a hotel

"Book a flight" and "Book a hotel" use the same vocabulary of "book a ". This format is the same so it should be the same intent with the different words of flight and hotel as extracted entities.

For more information:

- Concept: [Concepts about intents in your LUIS app](#)
- Tutorial: [Build LUIS app to determine user intentions](#)
- How to: [Add intents to determine user intention of utterances](#)

Do find sweet spot for intents

Use prediction data from LUIS to determine if your intents are overlapping. Overlapping intents confuse LUIS. The result is that the top scoring intent is too close to another intent. Because LUIS does not use the exact same path through the data for training each time, an overlapping intent has a chance of being first or second in training. You want the utterance's score for each intention to be farther apart so this flip/flop doesn't happen. Good distinction for intents should result in the expected top intent every time.

Do build the app iteratively

Keep a separate set of utterances that isn't used as [example utterances](#) or endpoint utterances. Keep improving the app for your test set. Adapt the test set to reflect real user utterances. Use this test set to evaluate each iteration or version of the app.

Developers should have three sets of data. The first is the example utterances for building the model. The second is for testing the model at the endpoint. The third is the blind test data used in [batch testing](#). This last set isn't used in training the application nor sent on the endpoint.

For more information:

- Concept: [Authoring cycle for your LUIS app](#)

Do add phrase lists and patterns in later iterations

A best practice is to not apply these practices before your app has been tested. You should understand how the app behaves before adding [phrase lists](#) and [patterns](#) because these features are weighted more heavily than example utterances and will skew confidence.

Once you understand how your app behaves without these, add each of these features as they apply to your app. You do not need to add these features with each [iteration](#) or change the features with each version.

There is no harm adding them in the beginning of your model design but it is easier to see how each feature changes results after the model is tested with utterances.

A best practice is to test via the [endpoint](#) so that you get the added benefit of [active learning](#). The [interactive](#)

[testing pane](#) is also a valid test methodology.

Phrase lists

Phrase lists allow you to define dictionaries of words related to your app domain. Seed your phrase list with a few words then use the suggest feature so LUIS knows about more words in the vocabulary specific to your app. A Phrase List improves intent detection and entity classification by boosting the signal associated with words or phrases that are significant to your app.

Don't add every word to the vocabulary since the phrase list isn't an exact match.

For more information:

- Concept: [Phrase list features in your LUIS app](#)
- How-to: [Use phrase lists to boost signal of word list](#)

Patterns

Real user utterances from the endpoint, very similar to each other, may reveal patterns of word choice and placement. The pattern feature takes this word choice and placement along with regular expressions to improve your prediction accuracy. A regular expression in the pattern allows for words and punctuation you intend to ignore while still matching the pattern.

Use pattern's [optional syntax](#) for punctuation so punctuation can be ignored. Use the [explicit list](#) to compensate for pattern.any syntax issues.

For more information:

- Concept: [Patterns improve prediction accuracy](#)
- How-to: [How to add Patterns to improve prediction accuracy](#)

Balance your utterances across all intents

In order for LUIS predictions to be accurate, the quantity of example utterances in each intent (except for the None intent), must be relatively equal.

If you have an intent with 100 example utterances and an intent with 20 example utterances, the 100-utterance intent will have a higher rate of prediction.

Do add example utterances to None intent

This intent is the fallback intent, indicated everything outside your application. Add one example utterance to the None intent for every 10 example utterances in the rest of your LUIS app.

For more information:

- Concept: [Understand what good utterances are for your LUIS app](#)

Do leverage the suggest feature for active learning

Use [active learning](#)'s **Review endpoint utterances** on a regular basis, instead of adding more example utterances to intents. Because the app is constantly receiving endpoint utterances, this list is growing and changing.

For more information:

- Concept: [Concepts for enabling active learning by reviewing endpoint utterances](#)
- Tutorial: [Tutorial: Fix unsure predictions by reviewing endpoint utterances](#)
- How-to: [How to review endpoint utterances in LUIS portal](#)

Do monitor the performance of your app

Monitor the prediction accuracy using a [batch test](#) set.

Don't add many example utterances to intents

After the app is published, only add utterances from active learning in the iterative process. If utterances are too similar, add a pattern.

Don't use LUIS as a training platform

Luis is specific to a language model's domain. It isn't meant to work as a general natural language training platform.

Don't add many example utterances of the same format, ignoring other formats

Luis expects variations in an intent's utterances. The utterances can vary while having the same overall meaning. Variations can include utterance length, word choice, and word placement.

DON'T USE SAME FORMAT	DO USE VARYING FORMAT
Buy a ticket to Seattle Buy a ticket to Paris Buy a ticket to Orlando	Buy 1 ticket to Seattle Reserve two seats on the red eye to Paris next Monday I would like to book 3 tickets to Orlando for spring break

The second column uses different verbs (buy, reserve, book), different quantities (1, two, 3), and different arrangements of words but all have the same intention of purchasing airline tickets for travel.

Don't mix the definition of intents and entities

Create an intent for any action your bot will take. Use entities as parameters that make that action possible.

For a chatbot that will book airline flights, create a **BookFlight** intent. Do not create an intent for every airline or every destination. Use those pieces of data as [entities](#) and mark them in the example utterances.

Don't create phrase lists with all the possible values

Provide a few examples in the [phrase lists](#) but not every word. Luis generalizes and takes context into account.

Don't add many patterns

Don't add too many [patterns](#). Luis is meant to learn quickly with fewer examples. Don't overload the system unnecessarily.

Don't train and publish with every single example utterance

Add 10 or 15 utterances before training and publishing. That allows you to see the impact on prediction accuracy. Adding a single utterance may not have a visible impact on the score.

Do use versions for each app iteration

Each authoring cycle should be within a new [version](#), cloned from an existing version. Luis has no limit for versions. A version name is used as part of the API route so it is important to pick characters allowed in a URL as

well as keeping within the 10 character count for a version. Develop a version name strategy to keep your versions organized.

For more information:

- Concept: [Understand how and when to use a LUIS version](#)
- How-to: [Use versions to edit and test without impacting staging or production apps](#)

Next steps

- Learn how to [plan your app](#) in your LUIS app.

Use Cognitive Services with natural language processing (NLP) to enrich bot conversations

8/5/2019 • 4 minutes to read • [Edit Online](#)

Cognitive Services provides two natural language processing services, [Language Understanding](#) and [QnA Maker](#), each with a different purpose. Understand when to use each service and how they compliment each other.

Natural language processing (NLP) allows your client application, such as a chat bot, to work with your users, using natural language. A user enters a sentence or phrase. The user's text can have poor grammar, spelling, and punctuation. The Cognitive Service can work through the user sentence anyway, returning information the chat bot needs to help the user.

Cognitive Services with NLP

Language Understanding (LUIS) and QnA Maker provide NLP. The client application submits natural language text. The service takes the text, processes it, and returns a result.

When to use each service

Language Understanding (LUIS) and QnA Maker solve different issues. LUIS determines the intent of a user's text (known as an utterance), while QnA Maker determines the answer to a user's text (known as a query).

In order to pick the correct service, you need to understand the user text coming from the client application, and what information the client application needs to get from the Cognitive Service.

If your chat bot receives the text `How do I get to the Human Resources building on the Seattle North campus?`, use the chart below to understand how each service works with the text.

SERVICE	CLIENT APPLICATION DETERMINES
LUIS	Determines user's intention of text - the service doesn't return the answer to the question. For example, this text is classified as matching the <code>FindLocation</code> intent.
QnA Maker	Returns the answer to the question from a custom knowledge base. For example, this text is determined as a question with the static text answer of <code>Get on the #9 bus and get off at Franklin street.</code>

When do you use LUIS?

Use LUIS when you need to know the intention of the utterance as part of a process in the chat bot. Continuing with the example text, `How do I get to the Human Resources building on the Seattle North campus?`, once you know the user's intention is to find a location, you can pass details about the utterance (pulled out with entities) to another service, such as a transportation server, to get the answer.

You don't need to combine LUIS and QnA Maker to determine intent.

You might combine the two services for this utterance, if the chat bot needs to process the text based on intentions and entities (using LUIS) as well as find the specific static text answer (using QnA Maker).

When do you use QnA Maker?

Use QnA Maker when you have a static knowledge base of answers. This knowledge base is custom to your needs, which you've built with documents such as PDFs and URLs.

Continuing with the example utterance,

How do I get to the Human Resources building on the Seattle North campus? , send the text, as a query, to your published QnA Maker service and receive the best answer.

You don't need to combine LUIS and QnA Maker to determine the answer to the question.

You might combine the two services for this utterance, if the chat bot needs to process the text based on intentions and entities (using LUIS) as well as find the answer (using QnA Maker).

Use both services when your knowledge base is incomplete

If you are building your QnA Maker knowledge base but know the subject domain is changing (such as timely information), you could combine LUIS and QnA Maker services. This allows you to use the information in your knowledge base but also use LUIS to determine a user's intention. Once the client application has the intention, it can request relevant information from another source.

Your client application would need to monitor both LUIS and QnA Maker responses for scores. If the score from QnA Maker is below some arbitrary threshold, use the intent and entity information returned from LUIS to pass the information on to a third-party service.

Continuing with the example text, How do I get to the Human Resources building on the Seattle North campus? , suppose that QnA Maker returns a low confidence score. Use the intent returned from LUIS, FindLocation and any extracted entities, such as Human Resources building and Seattle North campus , to send this information to a mapping or search service for another answer.

You can present this third-party answer to the user for validation. Once you have the user's approval, you can go back to QnA Maker to add the information to grow your knowledge.

Use both services when your chat bot needs more information

If your chat bot needs more information than either service provides, to continue through a decision tree, use both services and process both responses in the client application.

Use the Bot framework **Dispatch CLI** tool to help build a process to work with both services. This tool builds a top LUIS app of intents that dispatches between LUIS and QnA Maker as child apps.

Use the Bot builder sample, **NLP with dispatch**, in **C#** or **Node.js**, to implement this type of chat bot.

Best practices

Implement best practices for each service:

- [LUIS](#) best practices
- [QnA Maker](#) best practices

See also

- [Language Understanding \(LUIS\)](#)
- [QnA Maker](#)
- [Dispatch CLI](#)
- [Bot framework samples](#)

- [Azure bot service](#)
- [Azure bot emulator](#)
- [Bot framework web chat](#)

Plan your LUIS app with subject domain, intents and entities

7/30/2019 • 2 minutes to read • [Edit Online](#)

To plan your app, identify your subject-area domain. This includes possible intents and entities that are relevant to your application.

Identify your domain

A LUIS app is 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. Identifying the domain helps you find words or phrases that are important to your domain.

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

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 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.

Create example utterances for each intent

Once you have determined the intents, create 15 to 30 example utterances for each intent. To begin with, do not have fewer than this number or create too many utterances for each intent. Each utterance should be different from the previous utterance. A good variety in the utterances includes overall word count, word choice, verb tense, and punctuation.

Review [utterances](#) for more information.

Identify your entities

In the example utterances, identify the entities you want extracted. To book a flight, you need information like the destination, date, airline, ticket category, and travel class. Create entities for these data types and then mark the [entities](#) in the example utterances 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.

Next steps

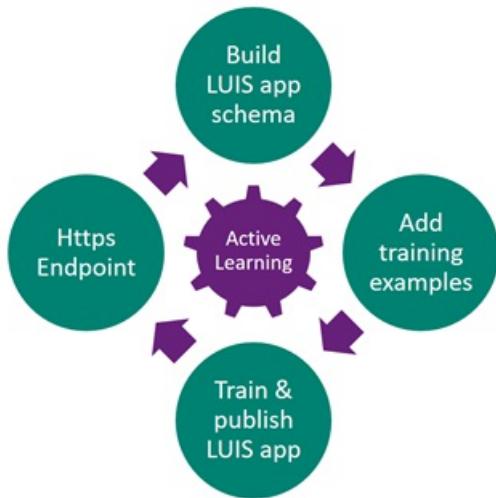
After your app is trained, published, and gets endpoint utterances, plan to implement prediction improvements with [active learning](#), [phrase lists](#), and [patterns](#).

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

Authoring cycle for your LUIS app

7/30/2019 • 2 minutes to read • [Edit Online](#)

LUIS learns best in an iterative cycle of model changes, utterance examples, publishing, and gathering data from endpoint queries.



Building a LUIS model

The model's purpose is to figure out what the user is asking for (the intention or intent) and what parts of the question provide details (entities) that help determine the answer.

The model needs to be specific to the app domain in order to determine words and phrases that are relevant as well as typical word ordering.

The model requires intents, and *should have* entities.

Add training examples

LUIS needs example utterances in the intents. The examples need enough variation of word choice and word order to be able to determine which intent the utterance is meant for. Each example utterance needs to have any required data labeled as entities.

You instruct LUIS to ignore utterances that are not relevant to your app's domain by assigning the utterance to the **None** intent. Any words or phrases you do not need pulled out of an utterance do not need to be labeled. There is no label for words or phrases to ignore.

Train and publish the app

Once you have 15 to 30 different utterances in each intent, with the required entities labeled, you need to [train](#) then [publish](#). From the publish success notification, use the link to get your endpoints. Make sure you create and publish your app so that it is available in the [endpoint regions](#) you need.

HTTPS endpoint testing

You can test your LUIS app from the HTTPS endpoint. Testing from the endpoint allows LUIS to choose any utterances with low-confidence for [review](#).

Recycle

When you are done with a cycle of authoring, you can begin again. Start with [reviewing endpoint utterances](#) LUIS marked with low-confidence. Check these utterances for both intent and entity. Once you review utterances, the review list should be empty.

Consider [cloning](#) the current version into a new version, then begin your authoring changes in the new version.

Batch testing

[Batch testing](#) is a way to see how many example utterances are scored by LUIS. The examples should be new to LUIS and should be correctly labeled with intent and entities you want LUIS to find. The test results indicate how well LUIS would perform on that set of utterances.

Next steps

Learn concepts about [collaboration](#).

Collaborating with other authors

7/26/2019 • 2 minutes to read • [Edit Online](#)

LUIS apps require a single owner and optional collaborators allowing multiple people to author a single app.

LUIS account

A LUIS account is associated with a single [Microsoft Live](#) account. Each LUIS account is given a free [authoring key](#) to use for authoring all the LUIS apps the account has access to.

A LUIS account may have many LUIS apps.

See [Azure Active Directory tenant user](#) to learn more about Active Directory user accounts.

LUIS app owner

The account that creates an app is the owner and each app has a single owner. The owner is listed on the app [Settings](#) page. The owner receives email when the endpoint quota reaches 75% of the monthly limit.

Authorization roles

LUIS doesn't support different roles for owners and collaborators with one exception. The owner is the only account that can delete the app.

If you are interested in controlling access to the model, consider slicing the model into smaller LUIS apps, where each smaller app has a more limited set of collaborators. Use [Dispatch](#) to allow a parent LUIS app to manage the coordination between parent and child apps.

Transfer ownership

LUIS doesn't provide transfer of ownership, however any collaborator can export the app, and then create an app by importing it. Be aware the new app has a different App ID. The new app needs to be trained, published, and the new endpoint used.

LUIS app collaborators

An app owner can add collaborators to an app. The owner needs to add the collaborator's email address on app [Settings](#). The collaborator has full access to the app. If the collaborator deletes the app, the app is removed from the collaborator's account but remains in the owner's account.

If you want to share multiple apps with collaborators, each app needs the collaborator's email added.

Managing multiple authors

The [LUIS](#) website doesn't currently offer transaction-level authoring. You can allow authors to work on independent versions from a base version. Two different methods are described in the following sections.

Manage multiple versions inside the same app

Begin by [cloning](#), from a base version, for each author.

Each author makes changes to their own version of the app. Once each author is satisfied with the model, export

the new versions to JSON files.

Exported apps are JSON-formatted files, which can be compared for changes. Combine the files to create a single JSON file of the new version. Change the **versionId** property in the JSON to signify the new merged version. Import that version into the original app.

This method allows you to have one active version, one stage version, and one published version. You can compare the results in the interactive testing pane across the three versions.

Manage multiple versions as apps

[Export](#) the base version. Each author imports the version. The person that imports the app is the owner of the version. When they are done modifying the app, export the version.

Exported apps are JSON-formatted files, which can be compared with the base export for changes. Combine the files to create a single JSON file of the new version. Change the **versionId** property in the JSON to signify the new merged version. Import that version into the original app.

Collaborator roles vs entity roles

[Entity roles](#) apply to the data model of the LUIS app. Collaborator roles apply to levels of authoring access.

Next steps

Understand [versioning](#) concepts.

See [App Settings](#) to learn how to manage collaborators in your LUIS app.

See [Add email to access list](#) with the Authoring APIs.

Understand how and when to use a LUIS version

7/29/2019 • 2 minutes to read • [Edit Online](#)

Versions, in LUIS, are similar to versions in traditional programming. Each version is a snapshot in time of the app. Before you make changes to the app, create a new version. It is easier to go back to the exact version, then to try to remove intents and utterances to a previous state.

Create different models of the same app with [versions](#).

Version ID

The version ID consists of characters, digits or '.' and cannot be longer than 10 characters.

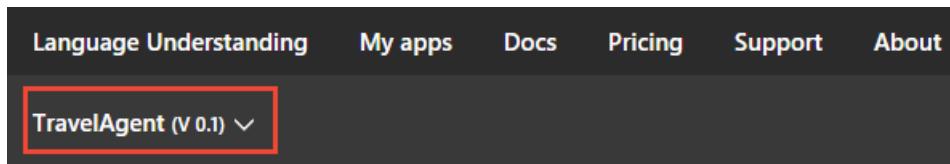
Initial version

The initial version (0.1) is the default active version.

Active version

To [set a version](#) as the active means it is currently edited and tested in the [LUIS](#) website. Set a version as active to access its data, make updates, as well as to test and publish it.

The name of the currently active version is displayed in the top, left panel after the app name.



Versions and publishing slots

You publish to either the stage and product slots. Each slot can have a different version or the same version. This is useful for verifying changes between model versions via the endpoint, which is available to bots or other LUIS calling applications.

Clone a version

Clone a version to create a copy of an existing version and save it as a new version. Clone a version to use the same content of the existing version as a starting point for the new version. Once you clone a version, the new version becomes the **active** version.

Import and export a version

You can import a version at the app level. That version becomes the active version and used the version ID in the "versionId" property of the app file. You can also import at the version level into an existing app. The new version becomes the active version.

You can export a version at the app level or you can export a version at the version level. The only difference is that the app-level exported version is the currently active version while at the version level, you can choose any version to export on the [Settings](#) page.

The exported file does not contain machine-learned information because the app is retrained after it is imported.

The exported file does not contain collaborators -- you need to add these back once the version is imported into the new app.

Export each version as app backup

In order to back up your LUIS app, export each version on the [Settings](#) page.

Delete a version

You can delete all versions except the active version from the Versions list on Settings page.

Version availability at the endpoint

Trained versions are not automatically available at your app [endpoint](#). You must [publish](#) or republish a version in order for it to be available at your app endpoint. You can publish to **Staging** and **Production**, giving you up to two versions of the app available at the endpoint. If you need more versions of the app available at an endpoint, you should export the version and reimport to a new app. The new app has a different app ID.

Collaborators

The owner and all [collaborators](#) have full access to all versions of the app.

Next steps

See how to add [versioning](#) on the app settings page.

Learn how to design [intents](#) into the model.

Concepts about intents in your LUIS app

7/29/2019 • 5 minutes 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 **utterance**.

Define a set of intents that corresponds to actions users want to take in your application. For example, a travel app defines several intents:

TRAVEL APP INTENTS	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" "Did the Lakers win?"

All applications come with the predefined intent, "**None**", which is the fallback intent.

Prebuilt domains provide intents

In addition to intents that you define, you can use prebuilt intents from one of the prebuilt domains. For more information, see [Use prebuilt domains in LUIS apps](#) to learn about how to customize intents from a prebuilt domain for use in your app.

Return all intents' scores

You assign an utterance to a single intent. When LUIS receives an utterance on the endpoint, it returns the one top intent for that utterance. If you want scores for all intents for the utterance, you can provide `verbose=true` flag on the query string of the API [endpoint call](#).

Intent compared to entity

The intent represents action the chatbot should take for the user and is based on the entire utterance. The entity represents words or phrases contained inside the utterance. An utterance can have only one top scoring intent but it can have many entities.

Create an intent when the user's *intention* would trigger an action in your client application, like a call to the `checkweather()` function. Then create an entity to represent parameters required to execute the action.

EXAMPLE INTENT	ENTITY	ENTITY IN EXAMPLE UTTERANCES
----------------	--------	------------------------------

EXAMPLE INTENT	ENTITY	ENTITY IN EXAMPLE UTTERANCES
CheckWeather	{ "type": "location", "entity": "seattle" } { "type": "builtin.datetimeV2.date", "entity": "tomorrow", "resolution": "2018-05-23" }	What's the weather like in Seattle tomorrow ?
CheckWeather	{ "type": "date_range", "entity": "this weekend" }	Show me the forecast for this weekend

Custom intents

Similarly intentioned [utterances](#) correspond to a single intent. Utterances in your intent can use any [entity](#) in the app since entities are not intent-specific.

Prebuilt domain intents

[Prebuilt domains](#) have intents with utterances.

None intent

The **None** intent is important to every app and shouldn't have zero utterances.

None intent is fallback for app

The **None** intent is a catch-all or fallback intent. It is used to teach LUIS utterances that are not important in the app domain (subject area). The **None** intent should have between 10 and 20 percent of the total utterances in the application. Do not leave the None empty.

None intent helps conversation direction

When an utterance is predicted as the None intent and returned to the chatbot with that prediction, the bot can ask more questions or provide a menu to direct the user to valid choices in the chatbot.

No utterances in None intent skews predictions

If you do not add any utterances for the **None** intent, LUIS forces an utterance that is outside the domain into one of the domain intents. This will skew the prediction scores by teaching LUIS the wrong intent for the utterance.

Add utterances to the None intent

The **None** intent is created but left empty on purpose. Fill it with utterances that are outside of your domain. A good utterance for **None** is something completely outside the app as well as the industry the app serves. For example, a travel app should not use any utterances for **None** that can relate to travel such as reservations, billing, food, hospitality, cargo, inflight entertainment.

What type of utterances are left for the None intent? Start with something specific that your bot shouldn't answer such "What kind of dinosaur has blue teeth?" This is a very specific question far outside of a travel app.

None is a required intent

The **None** intent is a required intent and can't be deleted or renamed.

Negative intentions

If you want to determine negative and positive intentions, such as "I **want** a car" and "I **don't** want a car", you

can create two intents (one positive, and one negative) and add appropriate utterances for each. Or you can create a single intent and mark the two different positive and negative terms as an entity.

Intents and patterns

If you have example utterances, which can be defined in part or whole as a regular expression, consider using the [regular expression entity](#) paired with a [pattern](#).

Using a regular expression entity guarantees the data extraction so that the pattern is matched. The pattern matching guarantees an exact intent is returned.

Intent balance

The app domain intents should have a balance of utterances across each intent. Do not have one intent with 10 utterances and another intent with 500 utterances. This is not balanced. If you have this situation, review the intent with 500 utterances to see if many of the intents can be reorganized into a [pattern](#).

The **None** intent is not included in the balance. That intent should contain 10% of the total utterances in the app.

Intent limits

Review [limits](#) to understand how many intents you can add to a model.

If you need more than the maximum number of intents

First, consider whether your system is using too many intents.

Can multiple intents be combined into single intent with entities

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

Dispatcher model

Learn more about combining LUIS and QnA maker apps with the [dispatch model](#).

Request help for apps with significant number of intents

If reducing the number of intents or dividing your intents into multiple apps doesn't work for you, contact support. If your Azure subscription includes support services, contact [Azure technical support](#).

Next steps

- Learn more about [entities](#), which are important words relevant to intents
- Learn how to [add and manage intents](#) in your LUIS app.
- Review intent [best practices](#)

Entity types and their purposes in LUIS

7/26/2019 • 9 minutes to read • [Edit Online](#)

Entities extract data from the utterance. Entity types give you predictable extraction of data. There are two types of entities: machine-learned and non-machine-learned. It is important to know which type of entity you are working with in utterances.

Entity compared to intent

The entity represents a word or phrase inside the utterance that you want extracted. An utterance can include many entities or none at all. A client application may need the entity to perform its task or use it as a guide of several choices to present to the user.

An entity:

- Represents a class including a collection of similar objects (places, things, people, events or concepts).
- Describes information relevant to the intent

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

By comparison, the intent represents the prediction of the entire utterance.

Entities help with data extraction only

You label or mark entities for the purpose of entity extraction only, it does not help with intent prediction.

Entities represent data

Entities are data you want to pull from the utterance. This can be a name, date, product name, or any group of words.

UTTERANCE	ENTITY	DATA
Buy 3 tickets to New York	Prebuilt number Location.Destination	3 New York
Buy a ticket from New York to London on March 5	Location.Origin Location.Destination Prebuilt datetimeV2	New York London March 5, 2018

Entities are optional but highly recommended

While intents are required, entities are optional. You do not need to create entities for every concept in your app, but only for those required for the client application to take action.

If your utterances do not have details your bot needs to continue, you do not need to add them. As your app matures, you can add them later.

If you're not sure how you would use the information, add a few common prebuilt entities such as [datetimeV2](#), [ordinal](#), [email](#), and [phone number](#).

Label for word meaning

If the word choice or word arrangement is the same, but doesn't mean the same thing, do not label it with the entity.

The following utterances, the word **fair** is a homograph. It is spelled the same but has a different meaning:

UTTERANCE
What kind of county fairs are happening in the Seattle area this summer?
Is the current rating for the Seattle review fair?

If you wanted an event entity to find all event data, label the word **fair** in the first utterance, but not in the second.

Entities are shared across intents

Entities are shared among intents. They don't belong to any single intent. Intents and entities can be semantically associated but it isn't an exclusive relationship.

In the utterance "Book me a ticket to Paris", "Paris" is an entity referring to location. By recognizing the entities that are mentioned in the user's utterance, LUIS helps your client application choose the specific actions to take to fulfill the user's request.

Mark entities in None intent

All intents, including the **None** intent, should have marked entities, when possible. This helps LUIS learn more about where the entities are in the utterances and what words are around the entities.

Entity status for predictions

The LUIS portal tells you when the entity in an example utterance is either different from the marked entity or is too close to another entity and therefore unclear. This is indicated by a red underline in the example utterance.

For more information, see [Entity Status predictions](#).

Types of entities

Luis offers many types of entities. Choose the entity based on how the data should be extracted and how it should be represented after it is extracted.

Entities can be extracted with machine-learning, which allows LUIS to continue learning about how the entity appears in the utterance. Entities can be extracted without machine-learning, matching either exact text or a regular expression. Entities in patterns can be extracted with a mixed implementation.

Once the entity is extracted, the entity data can be represented as a single unit of information or combined with other entities to form a unit of information the client-application can use.

MACHINE-LEARNED	CAN MARK	TUTORIAL	EXAMPLE RESPONSE	ENTITY TYPE	PURPOSE

MACHINE-LEARNED	CAN MARK	TUTORIAL	EXAMPLE RESPONSE	ENTITY TYPE	PURPOSE
✓	✓	✓	✓	Composite	Grouping of entities, regardless of entity type.
		✓	✓	List	List of items and their synonyms extracted with exact text match.
Mixed		✓	✓	Pattern.any	Entity where end of entity is difficult to determine.
		✓	✓	Prebuilt	Already trained to extract various kinds of data.
		✓	✓	Regular Expression	Uses regular expression to match text.
✓	✓	✓	✓	Simple	Contains a single concept in word or phrase.

Only Machine-learned entities need to be marked in the example utterances. Machine-learned entities work best when tested via [endpoint queries](#) and [reviewing endpoint utterances](#).

Pattern.any entities need to be marked in the Pattern template examples, not the intent user examples.

Mixed entities use a combination of entity detection methods.

Machine-learned entities use context

Machine-learned entities learn from context in the utterance. This makes variation of placement in example utterances significant.

Non-machine-learned entities don't use context

The following non-machine learned entities do not take utterance context into account when matching entities:

- Prebuilt entities
- Regex entities
- List entities

These entities do not require labeling or training the model. Once you add or configure the entity, the entities are extracted. The tradeoff is that these entities can be overmatched, where if context was taken into account, the match would not have been made.

This happens with list entities on new models frequently. You build and test your model with a list entity but when you publish your model and receive queries from the endpoint, you realize your model is overmatching due to lack of context.

If you want to match words or phrases and take context into account, you have two options. The first is to use a simple entity paired with a phrase list. The phrase list will not be used for matching but instead will help signal relatively similar words (interchangeable list). If you must have an exact match instead of a phrase list's variations, use a list entity with a role, described below.

Context with non-machine-learned entities

If you want context of the utterance to matter for non-machine learned entities, you should use [roles](#).

If you have a non-machine-learned entity, such as [prebuilt entities](#), [regex](#) entities or [list](#) entities, which is matching beyond the instance you want, consider creating one entity with two roles. One role will capture what you are looking for, and one role will capture what you are not looking for. Both versions will need to be labeled in example utterances.

Composite entity

A [composite entity](#) is made up of other entities, such as prebuilt entities, simple, regular expression, and list entities. The separate entities form a whole entity.

List entity

[List entities](#) represent a fixed, closed set of related words along with their synonyms. LUIS does not discover additional values for list entities. Use the **Recommend** feature to see suggestions for new words based on the current list. If there is more than one list entity with the same value, each entity is returned in the endpoint query.

Pattern.any entity

[Pattern.any](#) is a variable-length placeholder used only in a pattern's template utterance to mark where the entity begins and ends.

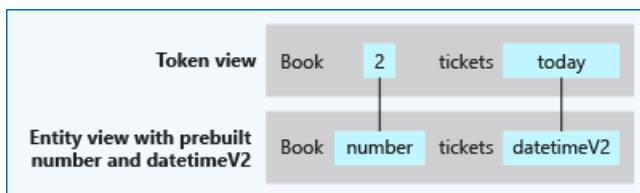
Prebuilt entity

Prebuilt entities are built-in types that represent common concepts such as email, URL, and phone number. Prebuilt entity names are reserved. [All prebuilt entities](#) that are added to the application are returned in the endpoint prediction query if they are found in the utterance.

The entity is a good fit when:

- The data matches a common use case supported by prebuilt entities for your language culture.

Prebuilt entities can be added and removed at any time.



Tutorial

Example JSON response for entity

Some of these prebuilt entities are defined in the open-source [Recognizers-Text](#) project. If your specific culture or entity isn't currently supported, contribute to the project.

Troubleshooting prebuilt entities

In the LUIS portal, if a prebuilt entity is tagged instead of your custom entity, you have a few choices of how to

fix this.

The prebuilt entities added to the app will *always* be returned, even if the utterance should extract custom entities for the same text.

Change tagged entity in example utterance

If the prebuilt entity is the same text or tokens as the custom entity, select the text in the example utterance and change the tagged utterance.

If the prebuilt entity is tagged with more text or tokens than your custom entity, you have a couple of choices of how to fix this:

- [Remove example utterance](#) method
- [Remove prebuilt entity](#) method

Remove example utterance to fix tagging

Your first choice is to remove the example utterance.

1. Delete the example utterance.
2. Retrain the app.
3. Add back just the word or phrase that is the entity, which is marked as a prebuilt entity, as a complete example utterance. The word or phrase will still have the prebuilt entity marked.
4. Select the entity in the example utterance on the **Intent** page, and change to your custom entity and train again. This should prevent LUIS from marking this exact text as the prebuilt entity in any example utterances that use that text.
5. Add the entire original example utterance back to the Intent. The custom entity should continue to be marked instead of the prebuilt entity. If the custom entity is not marked, you need to add more examples of that text in utterances.

Remove prebuilt entity to fix tagging

1. Remove the prebuilt entity from the app.
2. On the **Intent** page, mark the custom entity in the example utterance.
3. Train the app.
4. Add the prebuilt entity back to the app and train the app. This fix assumes the prebuilt entity isn't part of a composite entity.

Regular expression entity

A [regular expression entity](#) extracts an entity based on a regular expression pattern you provide.

Simple entity

A [simple entity](#) is a machine-learned value. It can be a word or phrase.

Entity limits

Review [limits](#) to understand how many of each type of entity you can add to a model.

If you need more than the maximum number of entities

You might need to use composite entities in combination with entity roles.

Composite entities represent parts of a whole. For example, a composite entity named PlaneTicketOrder might have child entities Airline, Destination, DepartureCity, DepartureDate, and PlaneTicketClass.

Luis also provides the list entity type that isn't machine-learned but allows your Luis app to specify a fixed

list of values. See [LUIS Boundaries](#) reference to review limits of the List entity type.

If you've considered these entities and still need more than the limit, contact support. To do so, gather detailed information about your system, go to the [LUIS](#) website, and then select **Support**. If your Azure subscription includes support services, contact [Azure technical support](#).

Next steps

Learn concepts about good [utterances](#).

See [Add entities](#) to learn more about how to add entities to your LUIS app.

Understand what good utterances are for your LUIS app

7/30/2019 • 5 minutes 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 example utterances 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 utterances that you think users will enter. Include utterances, which mean the same thing but are constructed in a variety of different ways:

- Utterance length - short, medium, and long for your client-application
- Word and phrase length
- Word placement - entity at beginning, middle, and end of utterance
- Grammar
- Pluralization
- Stemming
- Noun and verb choice
- Punctuation - a good variety using correct, incorrect, and no grammar

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 a ticket to Paris for me", 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 use [Bing Spell Check](#) to correct user input before passing it to LUIS.

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 correct for the typical user of your client application. They may not have domain experience. 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 create varied sentence patterns, you will still repeat some vocabulary.

Take these example utterances:

EXAMPLE UTTERANCES

how do I get a computer?

EXAMPLE UTTERANCES

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", isn't varied. Use alternatives such as desktop computer, laptop, workstation, or even just machine. LUIS intelligently infers synonyms from context, but when you create utterances for training, it's still better to vary them.

Example utterances in each intent

Each intent needs to have example utterances, at least 15. If you have an intent that does not have any example utterances, you will not be able to train LUIS. If you have an intent with one or very few example utterances, LUIS will not accurately predict the intent.

Add small groups of 15 utterances for each authoring iteration

In each iteration of the model, do not add a large quantity of utterances. Add utterances in quantities of 15. [Train](#), [publish](#), and [test](#) again.

Luis builds effective models with utterances that are carefully selected by the LUIS model author. Adding too many utterances isn't valuable because it introduces confusion.

It is better to start with a few utterances, then [review endpoint utterances](#) for correct intent prediction and entity extraction.

Utterance normalization

Utterance normalization is the process of ignoring the effects of punctuation and diacritics during training and prediction.

Utterance normalization for diacritics and punctuation

Utterance normalization is defined when you create or import the app because it is a setting in the app JSON file. The utterance normalization settings are turned off by default.

Diacritics are marks or signs within the text, such as:

í î § Ÿ š ÿ ü

If your app turns normalization on, scores in the **Test** pane, batch tests, and endpoint queries will change for all utterances using diacritics or punctuation.

Turn on utterance normalization for diacritics or punctuation to your LUIS JSON app file in the `settings` parameter.

```
"settings": [
    {"name": "NormalizePunctuation", "value": "true"},
    {"name": "NormalizeDiacritics", "value": "true"}
]
```

Normalizing **punctuation** means that before your models get trained and before your endpoint queries get predicted, punctuation will be removed from the utterances.

Normalizing **diacritics** replaces the characters with diacritics in utterances with regular characters. For example:
Je parle français becomes Je parle francais .

Normalization doesn't mean you will not see punctuation and diacritics in your example utterances or prediction responses, merely that they will be ignored during training and prediction.

Punctuation marks

Punctuation is a separate token in LUIS. An utterance that contains a period at the end versus an utterance that does not contain a period at the end are two separate utterances and may get two different predictions.

If punctuation is not normalized, LUIS doesn't ignore punctuation marks, by default, because some client applications may place significance on these marks. Make sure your example utterances use both punctuation and no punctuation in order for both styles to return the same relative scores.

Make sure the model handles punctuation either in the [example utterances](#) (having and not having punctuation) or in the [patterns](#) where it is easier to ignore punctuation with the special syntax:

I am applying for the {Job} position[.]

If punctuation has no specific meaning in your client application, consider [ignoring punctuation](#) by normalizing punctuation.

Ignoring words and punctuation

If you want to ignore specific words or punctuation in patterns, use a [pattern](#) with the *ignore* syntax of square brackets, [].

Training utterances

Training is generally non-deterministic: the utterance prediction could vary slightly across versions or apps. You can remove non-deterministic training by updating the [version settings](#) API with the UseAllTrainingData name/value pair to use all training data.

Testing utterances

Developers should start testing their LUIS application with real traffic by sending utterances to the [prediction endpoint](#) URL. These utterances are used to improve the performance of the intents and entities with [Review utterances](#). Tests submitted with the LUIS website testing pane are not sent through the endpoint, and so do not contribute to active learning.

Review utterances

After your model is trained, published, and receiving [endpoint](#) queries, [review the utterances](#) suggested by LUIS. LUIS selects endpoint utterances that have low scores for either the intent or entity.

Best practices

Review [best practices](#) and apply them as part of your regular authoring cycle.

Next steps

See [Add example utterances](#) for information on training a LUIS app to understand user utterances.

Entity roles for contextual subtypes

7/30/2019 • 2 minutes to read • [Edit Online](#)

Roles allow entities to have named subtypes. A role can be used with any prebuilt or custom entity type, and used in both example utterances and patterns.

Machine-learned entity example of roles

In the utterance "buy a ticket from **New York** to **London**, both New York and London are cities but each has a different meaning in the sentence. New York is the origin city and London is the destination city.

buy a ticket from New York to London

Roles give a name to those differences:

ENTITY TYPE	ENTITY NAME	ROLE	PURPOSE
Simple	Location	origin	where the plane leaves from
Simple	Location	destination	where the plane lands

Non-machine-learned entity example of roles

In the utterance "Schedule the meeting from 8 to 9", both the numbers indicate a time but each time has a different meaning in the utterance. Roles provide the name for the differences.

Schedule the meeting from 8 to 9

ENTITY TYPE	ROLE NAME	VALUE
Prebuilt datetimeV2	Starttime	8
Prebuilt datetimeV2	Endtime	9

Are multiple entities in an utterance the same thing as roles?

Multiple entities can exist in an utterance and can be extracted without using roles. If the context of the sentence indicates which version of the entity has a value, then a role should be used.

Don't use roles for duplicates without meaning

If the utterance includes a list of locations, `I want to travel to Seattle, Cairo, and London.`, this is a list where each item doesn't have an additional meaning.

Use roles if duplicates indicate meaning

If the utterance includes a list of locations with meaning,

`I want to travel from Seattle, with a layover in London, landing in Cairo.`, this meaning of origin, layover, and destination should be captured with roles.

Roles can indicate order

If the utterance changed to indicate order that you wanted to extract,

I want to first start with Seattle, second London, then third Cairo, you can extract in a couple of ways. You can tag the tokens that indicate the role, `first start with`, `second`, `third`. You could also use the prebuilt entity **Ordinal** and the **GeographyV2** prebuilt entity in a composite entity to capture the idea of order and place.

How are roles used in example utterances?

When an entity has a role, and the entity is marked in an example utterance, you have the choice of selecting just the entity, or selecting the entity and role.

The following example utterances use entities and roles:

TOKEN VIEW	ENTITY VIEW
I'm interesting in learning more about Seattle	I'm interested in learning more about {Location}
Buy a ticket from Seattle to New York	Buy a ticket from {Location:Origin} to {Location:Destination}

How are roles used in patterns?

In a pattern's template utterance, roles are used within the utterance:

PATTERN WITH ENTITY ROLES
<code>buy a ticket from {Location:origin} to {Location:destination}</code>

Role syntax in patterns

The entity and role are surrounded in parentheses, `{}`. The entity and the role are separated by a colon.

Entity roles versus collaborator roles

Entity roles apply to the data model of the LUIS app. [Collaborator](#) roles apply to levels of authoring access.

Roles in batch testing

Caution

Entity roles are not supported in batch testing.

Next steps

- Use a [hands-on tutorial](#) using entity roles with non-machine-learned entities
- Learn how to add [roles](#)

Prebuilt domain, intent, and entity models

7/29/2019 • 2 minutes to read • [Edit Online](#)

Prebuilt models provide domains, intents, utterances, and entities. You can start your app with a prebuilt domain or add a relevant domain to your app later.

Types of prebuilt models

There are 3 types of prebuilt models LUIS provides. Each model can be added to your app at any time.

MODEL TYPE	INCLUDES
Domain	Intents, utterances, entities
Intents	Intents, utterances
Entities	Entities only

Prebuilt domains

Language Understanding (LUIS) provides *prebuilt domains*, which are prebuilt sets of [intents](#) and [entities](#) that work together for domains or common categories of client applications.

The prebuilt domains are trained and ready to add to your LUIS app. The intents and entities in a prebuilt domain are fully customizable once you've added them to your app.

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

TIP

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

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.

You can find a full listing of the prebuilt domains in the [Prebuilt domains reference](#).

Prebuilt intents

LUIS provides prebuilt intents and their utterances. Intents can be added without adding the whole domain. Adding an intent is the process of adding an intent and its utterances. Both the intent name and the utterance list can be modified.

Prebuilt entities

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

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

NOTE

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

Next steps

Learn how to [add prebuilt entities](#) to your app.

Prediction scores indicate prediction accuracy for intent and entities

7/30/2019 • 2 minutes to read • [Edit Online](#)

A prediction score indicates the degree of confidence LUIS has for prediction results, based on a user utterance.

A prediction score is between zero (0) and one (1). An example of a highly confident LUIS score is 0.99. An example of a score of low confidence is 0.01.

SCORE VALUE	CONFIDENCE
1	definite match
0.99	high confidence
0.01	low confidence
0	definite failure to match

When an utterance results in a low-confidence score, LUIS highlights that in the [LUIS](#) website **Intent** page, with the identified **labeled-intent** outlined with red.

The screenshot shows the LUIS Intent page interface. On the left, there's a checkbox labeled "Utterance". Below it, an input field contains the utterance "test a number **number** pencil on a flight to seattle in **datetimeV2**". To the right of the input field, a red-bordered box highlights the prediction "BookFlight 0.42". Above the input field, there's a link labeled "Labeled intent ?". On the far right, there's a three-dot menu icon.

Top-scoring intent

Every utterance prediction returns a top-scoring intent. This prediction is a numerical comparison of prediction scores. The top 2 scores can have a very small difference between them. LUIS doesn't indicate this proximity other than returning the top score.

Return prediction score for all intents

A test or endpoint result can include all intents. This configuration is set on the endpoint with the `verbose=true` query string name/value pair.

Review intents with similar scores

Reviewing the score for all intents is a good way to verify that not only is the correct intent identified, but that the next identified intent's score is significantly lower consistently for utterances.

If multiple intents have close prediction scores, based on the context of an utterance, LUIS may switch between the intents. To fix this situation, continue to add utterances to each intent with a wider variety of contextual differences or you can have the client application, such as a chat bot, make programmatic choices about how to handle the 2 top intents.

The 2 intents, which are too-closely scored, may invert due to non-deterministic training. The top score could become the second top and the second top score could become the first top score. In order to prevent this situation, add example utterances to each of the top two intents for that utterance with word choice and context

that differentiates the 2 intents. The two intents should have about the same number of example utterances. A rule of thumb for separation to prevent inversion due to training, is a 15% difference in scores.

You can turn off the non-deterministic training by [training with all data](#).

Differences with predictions between different training sessions

When you train the same model in a different app, and the scores are not the same, this difference is because there is non-deterministic training (an element of randomness). Secondly, any overlap of an utterance to more than one intent means the top intent for the same utterance can change based on training.

If your chat bot requires a specific LUIS score to indicate confidence in an intent, you should use the score difference between the top two intents. This situation provides flexibility for variations in training.

E (exponent) notation

Prediction scores can use exponent notation, *appearing* above the 0-1 range, such as `9.910309E-07`. This score is an indication of a very **small** number.

E NOTATION SCORE	ACTUAL SCORE
9.910309E-07	.0000009910309

Punctuation

[Learn more](#) about how to use or ignore punctuation.

Next steps

See [Add entities](#) to learn more about how to add entities to your LUIS app.

Concepts for enabling active learning by reviewing endpoint utterances

7/29/2019 • 2 minutes to read • [Edit Online](#)

Active learning is one of three strategies to improve prediction accuracy and the easiest to implement. With active learning, your review endpoint utterances for correct intent and entity. LUIS chooses endpoint utterances it is unsure of.

What is active learning

Active learning is a two-step process. First, LUIS selects utterances it receives at the app's endpoint that need validation. The second step is performed by the app owner or collaborator to validate the selected utterances for [review](#), including the correct intent and any entities within the intent. After reviewing the utterances, train and publish the app again.

Which utterances are on the review list

LUIS adds utterances to the review list when the top firing intent has a low score or the top two intents' scores are too close.

Single pool for utterances per app

The **Review endpoint utterances** list doesn't change based on the version. There is a single pool of utterances to review, regardless of which version the utterance you are actively editing or which version of the app was published at the endpoint.

Where are the utterances from

Endpoint 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 do not have any utterances to review. If no endpoint hits are received for a specific intent or entity, you do not have utterances to review that contain them.

Schedule review periodically

Reviewing suggested utterances doesn't need to be done every day but should be part of your regular maintenance of LUIS.

Delete review items programmatically

Use the [delete unlabeled utterances](#) API. Back up these utterances before deletion by [exporting the log files](#).

Next steps

- Learn how to [review](#) endpoint utterances

Phrase list features in your LUIS app

8/19/2019 • 5 minutes to read • [Edit Online](#)

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

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.

What is a phrase list feature?

A phrase list is a list of words or phrases that are significant to your app, more so than other words in utterances. A phrase list adds to the vocabulary of the app domain as an additional signal to LUIS about those words. What LUIS learns about one of them is automatically applied to the others as well. This list is not a closed [list entity](#) of exact text matches.

Phrase lists do not help with stemming so you need to add utterance examples that use a variety of stemming for any significant vocabulary words and phrases.

Phrase lists help all models

Phrase lists are not linked to a specific intent or entity but are added as a significant boost to all the intents and entities. Its purpose is to improve intent detection and entity classification.

How to use phrase lists

Create a [phrase](#) list when your app has words or phrases that are important to the app such as:

- industry terms
- slang
- abbreviations
- company-specific language
- language that is from another language but frequently used in your app
- key words and phrases in your example utterances

Once you've entered a few words or phrases, use the **Recommend** feature to find related values. Review the related values before adding to your phrase list values.

LIST TYPE	PURPOSE
Interchangeable	Synonyms or words that, when changed to another word in the list, have the same intent, and entity extraction.
Non-interchangeable	App vocabulary, specific to your app, more so than generally other words in that language.

Interchangeable lists

An *interchangeable* phrase list is for values that are synonyms. For example, if you want all bodies of water found and you have example utterances such as:

- What cities are close to the Great Lakes?

- What road runs along Lake Havasu?
- Where does the Nile start and end?

Each utterance should be determined for both intent and entities regardless of body of water:

- What cities are close to [bodyOfWater]?
- What road runs along [bodyOfWater]?
- Where does the [bodyOfWater] start and end?

Because the words or phrases for the body of water are synonymous and can be used interchangeably in the utterances, use the **Interchangeable** setting on the phrase list.

Non-interchangeable lists

A non-interchangeable phrase list is a signal that boosts detection to LUIS. The phrase list indicates words or phrases that are more significant than other words. This helps with both determining intent and entity detection. For example, say you have a subject domain like travel that is global (meaning across cultures but still in a single language). There are words and phrases that are important to the app but are not synonymous.

For another example, use a non-interchangeable phrase list 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). The non-interchangeable setting indicates that the set of rare words forms a class that LUIS should learn to recognize, but they are not synonyms or interchangeable with each other.

Do not add every possible word or phrase to a phrase list, add a few words or phrases at a time, then retrain and publish.

As the phrase list grows over time, you may find some terms have many forms (synonyms). Break these out into another phrase list that is interchangeable.

Phrase lists help identify simple interchangeable entities

Interchangeable phrase lists are a good way to tune the performance of your LUIS app. If your app has trouble predicting utterances to the correct intent, or recognizing 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.

Phrase lists help identify intents by better understanding context

A phrase list 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 "Find me driving directions to Selma, Georgia".

Adding a phrase list is an alternative to adding more example utterances to an intent.

When to use phrase lists versus list entities

While both a phrase list and [list entities](#) can impact utterances across all intents, each does this in a different way. Use a phrase list to affect intent prediction score. Use a list entity to affect entity extraction for an exact text match.

Use a phrase list

With 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, use a phrase list.

When 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, use another type of machine-learned entity such as a simple entity. Then create a phrase list of words and phrases that helps LUIS find other words similar to the entity. This list guides LUIS to recognize examples of the entity by adding additional significance to the value of those words.

Phrase lists are like domain-specific vocabulary that help with enhancing the quality of understanding of both intents and entities. A common usage of a phrase list is proper nouns such as city names. A city name can be several words including hyphens, or apostrophes.

Don't use a phrase list

A list entity explicitly defines every value an entity can take, and only identifies values 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. Examples are food items on a restaurant menu that changes infrequently. If you need an exact text match of an entity, do not use a phrase list.

Best practices

Learn [best practices](#).

Next steps

See [Add Features](#) to learn more about how to add features to your LUIS app.

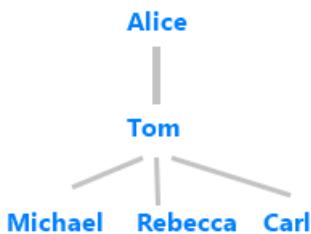
Patterns improve prediction accuracy

7/30/2019 • 6 minutes to read • [Edit Online](#)

Patterns are designed to improve accuracy when several utterances are very similar. A pattern allows you to gain more accuracy for an intent without providing many more utterances.

Patterns solve low intent confidence

Consider a Human Resources app that reports on the organizational chart in relation to an employee. Given an employee's name and relationship, LUIS returns the employees involved. Consider an employee, Tom, with a manager name Alice, and a team of subordinates named: Michael, Rebecca, and Carl.



UTTERANCES	INTENT PREDICTED	INTENT SCORE
Who is Tom's subordinate?	GetOrgChart	.30
Who is the subordinate of Tom?	GetOrgChart	.30

If an app has between 10 and 20 utterances with different lengths of sentence, different word order, and even different words (synonyms of "subordinate", "manage", "report"), LUIS may return a low confidence score. Create a pattern to help LUIS understand the importance of the word order.

Patterns solve the following situations:

- The intent score is low
- The correct intent is not the top score but too close to the top score.

Patterns are not a guarantee of intent

Patterns use a mix of prediction technologies. Setting an intent for a template utterance in a pattern is not a guarantee of the intent prediction but it is a strong signal.

Patterns do not improve machine-learned entity detection

A pattern is primarily meant to help the prediction of intents and roles. The pattern.any entity is used to extract free-form entities. While patterns use entities, a pattern does not help detect a machine-learned entity.

Do not expect to see improved entity prediction if you collapse multiple utterances into a single pattern. For Simple entities to fire, you need to add utterances or use list entities else your pattern will not fire.

Patterns use entity roles

If two or more entities in a pattern are contextually related, patterns use entity [roles](#) to extract contextual information about entities.

Prediction scores with and without patterns

Given enough example utterances, LUIS would be able to increase prediction confidence without patterns. Patterns increase the confidence score without having to provide as many utterances.

Pattern matching

A pattern is matched based on detecting the entities inside the pattern first, then validating the rest of the words and word order of the pattern. Entities are required in the pattern for a pattern to match. The pattern is applied at the token level, not the character level.

Pattern syntax

Pattern syntax is a template for an utterance. The template should contain words and entities you want to match as well as words and punctuation you want to ignore. It is **not** a regular expression.

Entities in patterns are surrounded by curly brackets, `{}`. Patterns can include entities, and entities with roles. [Pattern.any](#) is an entity only used in patterns.

Pattern syntax supports the following syntax:

FUNCTION	SYNTAX	NESTING LEVEL	EXAMPLE
entity	{ } - curly brackets	2	Where is form {entity-name}?
optional	[] - square brackets There is a limit of 3 on nesting levels of any combination of optional and grouping	2	The question mark is optional [?]
grouping	() - parentheses	2	is (a b)
or	- vertical bar (pipe) There is a limit of 2 on the vertical bars (Or) in one group	-	Where is form {{form-name-short} {form-name-long} {form-number}}?
beginning and/or end of utterance	^ - caret	-	^begin the utterance the utterance is done^ ^strict literal match of entire utterance with {number} entity^

Nesting syntax in patterns

The **optional** syntax, with square brackets, can be nested two levels. For example: `[[this]is] a new form`. This example allows for the following utterances:

NESTED OPTIONAL UTTERANCE EXAMPLE	EXPLANATION
this is a new form	matches all words in pattern
is a new form	matches outer optional word and non-optional words in pattern
a new form	matches required words only

The **grouping** syntax, with parentheses, can be nested two levels. For example:

`(({Entity1.RoleName1} | {Entity1.RoleName2}) | {Entity2})`. This feature allows any of the three entities to be matched.

If Entity1 is a Location with roles such as origin (Seattle) and destination (Cairo) and Entity 2 is a known building name from a list entity (RedWest-C), the following utterances would map to this pattern:

NESTED GROUPING UTTERANCE EXAMPLE	EXPLANATION
RedWest-C	matches outer grouping entity
Seattle	matches one of the inner grouping entities
Cairo	matches one of the inner grouping entities

Nesting limits for groups with optional syntax

A combination of **grouping** with **optional** syntax has a limit of 3 nesting levels.

ALLOWED	EXAMPLE
Yes	<code>([(test1 test2)] test3)</code>
No	<code>([[[test1] test2]] test3)</code>

Nesting limits for groups with or-ing syntax

A combination of **grouping** with **or-ing** syntax has a limit of 2 vertical bars.

ALLOWED	EXAMPLE
Yes	<code>(test1 test2 (test3 test4))</code>
No	<code>(test1 test2 test3 (test4 test5))</code>

Syntax to add an entity to a pattern template

To add an entity into the pattern template, surround the entity name with curly braces, such as

PATTERN WITH ENTITY
<code>Who does {Employee} manage?</code>

Syntax to add an entity and role to a pattern template

An entity role is denoted as `{entity:role}` with the entity name followed by a colon, then the role name. To add an entity with a role into the pattern template, surround the entity name and role name with curly braces, such as `Book a ticket from {Location:Origin} to {Location:Destination}`.

PATTERN WITH ENTITY ROLES

```
Book a ticket from {Location:Origin} to {Location:Destination}
```

Syntax to add a pattern.any to pattern template

The Pattern.any entity allows you to add an entity of varying length to the pattern. As long as the pattern template is followed, the pattern.any can be any length.

To add a **Pattern.any** entity into the pattern template, surround the Pattern.any entity with the curly braces, such as `How much does {Booktitle} cost and what format is it available in?`.

PATTERN WITH PATTERN.ANY ENTITY

```
How much does {Booktitle} cost and what format is it available in?
```

BOOK TITLES IN THE PATTERN

How much does **steal this book** cost and what format is it available in?

How much does **ask** cost and what format is it available in?

How much does **The Curious Incident of the Dog in the Night-Time** cost and what format is it available in?

The words of the book title are not confusing to LUIS because LUIS knows where the book title ends, based on the Pattern.any entity.

Explicit lists

create an [Explicit List](#) through the authoring API to allow the exception when:

- Your pattern contains a [Pattern.any](#)
- And that pattern syntax allows for the possibility of an incorrect entity extraction based on the utterance.

For example, suppose you have a pattern containing both optional syntax, `[]`, and entity syntax, `{}`, combined in a way to extract data incorrectly.

Consider the pattern '`[find] email about {subject} [from {person}]`'.

In the following utterances, the **subject** and **person** entity are extracted correctly and incorrectly:

UTTERANCE	ENTITY	CORRECT EXTRACTION
email about dogs from Chris	subject=dogs person=Chris	✓
email about the man from La Mancha	subject=the man person=La Mancha	X

In the preceding table, the subject should be `the man from La Mancha` (a book title) but because the subject includes the optional word `from`, the title is incorrectly predicted.

To fix this exception to the pattern, add `the man from la mancha` as an explicit list match for the {subject} entity using the [authoring API for explicit list](#).

Syntax to mark optional text in a template utterance

Mark optional text in the utterance using the regular expression square bracket syntax, `[]`. The optional text can nest square brackets up to two brackets only.

PATTERN WITH OPTIONAL TEXT	MEANING
<code>[find] email about {subject} [from {person}]</code>	<code>find</code> and <code>from {person}</code> are optional
<code>'Can you help me[?]</code>	The punctuation mark is optional

Punctuation marks (`?`, `!`, `.`) should be ignored and you need to ignore them using the square bracket syntax in patterns.

Pattern-only apps

You can build an app with intents that have no example utterances, as long as there's a pattern for each intent. For a pattern-only app, the pattern shouldn't contain machine-learned entities because these do require example utterances.

Best practices

Learn [best practices](#).

Next steps

Learn how to implement patterns in this tutorial

Alter utterance data before or during prediction

7/29/2019 • 2 minutes to read • [Edit Online](#)

LUIS provides ways to manipulate the utterance before or during the prediction. These include [fixing spelling](#), and fixing timezone issues for prebuilt [datetimeV2](#).

Correct spelling errors in utterance

LUIS uses [Bing Spell Check API V7](#) to correct spelling errors in the utterance. LUIS needs the key associated with that service. Create the key, then add the key as a querystring parameter at the [endpoint](#).

You can also correct spelling errors in the **Test** panel by [entering the key](#). The key is kept as a session variable in the browser for the Test panel. Add the key to the Test panel in each browser session you want spelling corrected.

Usage of the key in the test panel and at the endpoint count toward the [key usage](#) quota. LUIS implements Bing Spell Check limits for text length.

The endpoint requires two params for spelling corrections to work:

PARAM	VALUE
spellCheck	boolean
bing-spell-check-subscription-key	Bing Spell Check API V7 endpoint key

When [Bing Spell Check API V7](#) detects an error, the original utterance, and the corrected utterance are returned along with predictions from the endpoint.

```
{
  "query": "Book a flite to London?",
  "alteredQuery": "Book a flight to London?",
  "topScoringIntent": {
    "intent": "BookFlight",
    "score": 0.780123
  },
  "entities": []
}
```

List of allowed words

The Bing spell check API used in LUIS does not support a list (also called a whitelist) of words to ignore during the spell check alterations. If you need to allow a list of words or acronyms, process the utterance in the client application before sending the utterance to LUIS for intent prediction.

Change time zone of prebuilt datetimeV2 entity

When a LUIS app uses the prebuilt [datetimeV2](#) entity, a datetime value can be returned in the prediction response. The timezone of the request is used to determine the correct datetime to return. If the request is coming from a bot or another centralized application before getting to LUIS, correct the timezone LUIS uses.

Endpoint querystring parameter

The timezone is corrected by adding the user's timezone to the [endpoint](#) using the `timezoneOffset` param. The value of `timezoneOffset` should be the positive or negative number, in minutes, to alter the time.

PARAM	VALUE
timezoneOffset	positive or negative number, in minutes

Daylight savings example

If you need the returned prebuilt datetimeV2 to adjust for daylight savings time, you should use the `timezoneOffset` querystring parameter with a +/- value in minutes for the [endpoint](#) query.

Add 60 minutes:

```
https://[region].api.cognitive.microsoft.com/luis/v2.0/apps/{appId}?q=Turn the lights  
on?timezoneOffset=60&verbose={boolean}&spellCheck={boolean}&staging={boolean}&bing-spell-check-  
subscription-key={string}&log={boolean}
```

Remove 60 minutes:

```
https://[region].api.cognitive.microsoft.com/luis/v2.0/apps/{appId}?q=Turn the lights on?timezoneOffset=-60&verbose={boolean}&spellCheck={boolean}&staging={boolean}&bing-spell-check-subscription-key=  
{string}&log={boolean}
```

C# code determines correct value of timezoneOffset

The following C# code uses the [TimeZoneInfo](#) class's [FindSystemTimeZoneById](#) method to determine the correct `timezoneOffset` based on system time:

```
// Get CST zone id  
TimeZoneInfo targetZone = TimeZoneInfo.FindSystemTimeZoneById("Central Standard Time");  
  
// Get local machine's value of Now  
DateTime utcDatetime = DateTime.UtcNow;  
  
// Get Central Standard Time value of Now  
DateTime cstDatetime = TimeZoneInfo.ConvertTimeFromUtc(utcDatetime, targetZone);  
  
// Find timezoneOffset  
int timezoneOffset = (int)((cstDatetime - utcDatetime).TotalMinutes);
```

Next steps

[Correct spelling mistakes with this tutorial](#)

Convert data format of utterances

7/29/2019 • 2 minutes to read • [Edit Online](#)

LUIS provides the following conversions of a user utterance before prediction"

- Speech to text using [Cognitive Services Speech](#) service.

Speech to text

Speech to text is provided as an integration with LUIS.

Intent conversion concepts

Conversion of speech to text in LUIS allows you to send spoken utterances to an endpoint and receive a LUIS prediction response. The process is an integration of the [Speech](#) service with LUIS. Learn more about Speech to Intent with a [tutorial](#).

Key requirements

You do not need to create a [Bing Speech API](#) key for this integration. A **Language Understanding** key created in the Azure portal works for this integration. Do not use the LUIS starter key.

Pricing Tier

This integration uses a different [pricing](#) model than the usual Language Understanding pricing tiers.

Quota usage

See [Key limits](#) for information.

Next steps

[Extracting data](#)

Extract data from utterance text with intents and entities

7/26/2019 • 9 minutes to read • [Edit Online](#)

LUIS gives you the ability to get information from a user's natural language utterances. The information is extracted in a way that it can be used by a program, application, or chat bot to take action. In the following sections, learn what data is returned from intents and entities with examples of JSON.

The hardest data to extract is the machine-learned data because it isn't an exact text match. Data extraction of the machine-learned [entities](#) needs to be part of the [authoring cycle](#) until you're confident you receive the data you expect.

Data location and key usage

LUIS provides the data from the published [endpoint](#). The **HTTPS request** (POST or GET) contains the utterance as well as some optional configurations such as staging or production environments.

```
https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?subscription-key=<subscription-key>&verbose=true&timezoneOffset=0&q=book 2 tickets to paris
```

The `appID` is available on the **Settings** page of your LUIS app as well as part of the URL (after `/apps/`) when you're editing that LUIS app. The `subscription-key` is the endpoint key used for querying your app. While you can use your free authoring/starter key while you're learning LUIS, it is important to change the endpoint key to a key that supports your [expected LUIS usage](#). The `timezoneOffset` unit is minutes.

The **HTTPS response** contains all the intent and entity information LUIS can determine based on the current published model of either the staging or production endpoint. The endpoint URL is found on the [LUIS](#) website, in the **Manage** section, on the **Keys and endpoints** page.

Data from intents

The primary data is the top scoring **intent name**. Using the `MyStore` [quickstart](#), the endpoint response is:

```
{
  "query": "when do you open next?",
  "topScoringIntent": {
    "intent": "GetStoreInfo",
    "score": 0.984749258
  },
  "entities": []
}
```

DATA OBJECT	DATA TYPE	DATA LOCATION	VALUE
Intent	String	topScoringIntent.intent	"GetStoreInfo"

If your chatbot or LUIS-calling app makes a decision based on more than one intent score, return all the intents' scores by setting the querystring parameter, `verbose=true`. The endpoint response is:

```
{
  "query": "when do you open next?",
  "topScoringIntent": {
    "intent": "GetStoreInfo",
    "score": 0.984749258
  },
  "intents": [
    {
      "intent": "GetStoreInfo",
      "score": 0.984749258
    },
    {
      "intent": "None",
      "score": 0.2040639
    }
  ],
  "entities": []
}
```

The intents are ordered from highest to lowest score.

DATA OBJECT	DATA TYPE	DATA LOCATION	VALUE	SCORE
Intent	String	intents[0].intent	"GetStoreInfo"	0.984749258
Intent	String	intents[1].intent	"None"	0.0168218873

If you add prebuilt domains, the intent name indicates the domain, such as `Utilities` or `Communication` as well as the intent:

```
{
  "query": "Turn on the lights next monday at 9am",
  "topScoringIntent": {
    "intent": "Utilities.ShowNext",
    "score": 0.07842206
  },
  "intents": [
    {
      "intent": "Utilities.ShowNext",
      "score": 0.07842206
    },
    {
      "intent": "Communication.StartOver",
      "score": 0.0239675418
    },
    {
      "intent": "None",
      "score": 0.0168218873
    }
  ],
  "entities": []
}
```

DOMAIN	DATA OBJECT	DATA TYPE	DATA LOCATION	VALUE
Utilities	Intent	String	intents[0].intent	" Utilities .ShowNext"
Communication	Intent	String	intents[1].intent	Communication .Star tOver"

DOMAIN	DATA OBJECT	DATA TYPE	DATA LOCATION	VALUE
	Intent	String	intents[2].intent	"None"

Data from entities

Most chatbots and applications need more than the intent name. This additional, optional data comes from entities discovered in the utterance. Each type of entity returns different information about the match.

A single word or phrase in an utterance can match more than one entity. In that case, each matching entity is returned with its score.

All entities are returned in the **entities** array of the response from the endpoint:

```
"entities": [
  {
    "entity": "bob jones",
    "type": "Name",
    "startIndex": 0,
    "endIndex": 8,
    "score": 0.473899543
  },
  {
    "entity": "3",
    "type": "builtin.number",
    "startIndex": 16,
    "endIndex": 16,
    "resolution": {
      "value": "3"
    }
  }
]
```

Tokenized entity returned

Several [cultures](#) return the entity object with the `entity` value [tokenized](#). The startIndex and endIndex returned by LUIS in the entity object do not map to the new, tokenized value but instead to the original query in order for you to extract the raw entity programmatically.

For example, in German, the word `das Bauernbrot` is tokenized into `das bauern brot`. The tokenized value, `das bauern brot`, is returned and the original value can be programmatically determined from the startIndex and endIndex of the original query, giving you `das Bauernbrot`.

Simple entity data

A [simple entity](#) is a machine-learned value. It can be a word or phrase.

Composite entity data

A [composite entity](#) is made up of other entities, such as prebuilt entities, simple, regular expression, and list entities. The separate entities form a whole entity.

List entity data

[List entities](#) represent a fixed, closed set of related words along with their synonyms. LUIS does not discover additional values for list entities. Use the **Recommend** feature to see suggestions for new words based on the current list. If there is more than one list entity with the same value, each entity is returned in the endpoint query.

Prebuilt entity data

Prebuilt entities are discovered based on a regular expression match using the open-source [Recognizers-Text](#) project. Prebuilt entities are returned in the entities array and use the type name prefixed with `builtin::`. The following text is an example utterance with the returned prebuilt entities:

```
Dec 5th send to +1 360-555-1212
```

```
"entities": [
  {
    "entity": "dec 5th",
    "type": "builtin.datetimeV2.date",
    "startIndex": 0,
    "endIndex": 6,
    "resolution": {
      "values": [
        {
          "timex": "XXXX-12-05",
          "type": "date",
          "value": "2017-12-05"
        },
        {
          "timex": "XXXX-12-05",
          "type": "date",
          "value": "2018-12-05"
        }
      ]
    }
  },
  {
    "entity": "1",
    "type": "builtin.number",
    "startIndex": 18,
    "endIndex": 18,
    "resolution": {
      "value": "1"
    }
  },
  {
    "entity": "360",
    "type": "builtin.number",
    "startIndex": 20,
    "endIndex": 22,
    "resolution": {
      "value": "360"
    }
  },
  {
    "entity": "555",
    "type": "builtin.number",
    "startIndex": 26,
    "endIndex": 28,
    "resolution": {
      "value": "555"
    }
  },
  {
    "entity": "1212",
    "type": "builtin.number",
    "startIndex": 32,
    "endIndex": 35,
    "resolution": {
      "value": "1212"
    }
  },
]
```

```
{
  "entity": "5th",
  "type": "builtin.ordinal",
  "startIndex": 4,
  "endIndex": 6,
  "resolution": {
    "value": "5"
  }
},
{
  "entity": "1 360 - 555 - 1212",
  "type": "builtin.phonenumber",
  "startIndex": 18,
  "endIndex": 35,
  "resolution": {
    "value": "1 360 - 555 - 1212"
  }
}
]
```

Regular expression entity data

A [regular expression entity](#) extracts an entity based on a regular expression pattern you provide.

Extracting names

Getting names from an utterance is difficult because a name can be almost any combination of letters and words. Depending on what type of name you're extracting, you have several options. The following suggestions are not rules but more guidelines.

Add prebuilt PersonName and GeographyV2 entities

[PersonName](#) and [GeographyV2](#) entities are available in some [language cultures](#).

Names of people

People's name can have some slight format depending on language and culture. Use either a prebuilt [personName](#) entity or a [simple entity](#) with [roles](#) of first and last name.

If you use the simple entity, make sure to give examples that use the first and last name in different parts of the utterance, in utterances of different lengths, and utterances across all intents including the None intent. [Review endpoint utterances](#) on a regular basis to label any names that were not predicted correctly.

Names of places

Location names are set and known such as cities, counties, states, provinces, and countries/regions. Use the prebuilt entity [geographyV2](#) to extract location information.

New and emerging names

Some apps need to be able to find new and emerging names such as products or companies. These types of names are the most difficult type of data extraction. Begin with a [simple entity](#) and add a [phrase list](#). [Review endpoint utterances](#) on a regular basis to label any names that were not predicted correctly.

Pattern roles data

Roles are contextual differences of entities.

```
{
  "query": "move bob jones from seattle to redmond",
  "topScoringIntent": {
    "intent": "MoveAssetsOrPeople",
    "score": 0.9999998
  },
  "intents": [
    {
      "intent": "MoveAssetsOrPeople",
      "score": 0.9999998
    },
    {
      "intent": "None",
      "score": 1.02040713E-06
    },
    {
      "intent": "GetEmployeeBenefits",
      "score": 6.12244548E-07
    },
    {
      "intent": "GetEmployeeOrgChart",
      "score": 6.12244548E-07
    },
    {
      "intent": "FindForm",
      "score": 1.1E-09
    }
  ],
  "entities": [
    {
      "entity": "bob jones",
      "type": "Employee",
      "startIndex": 5,
      "endIndex": 13,
      "score": 0.922820568,
      "role": ""
    },
    {
      "entity": "seattle",
      "type": "Location",
      "startIndex": 20,
      "endIndex": 26,
      "score": 0.948008537,
      "role": "Origin"
    },
    {
      "entity": "redmond",
      "type": "Location",
      "startIndex": 31,
      "endIndex": 37,
      "score": 0.7047979,
      "role": "Destination"
    }
  ]
}
```

Pattern.any entity data

[Pattern.any](#) is a variable-length placeholder used only in a pattern's template utterance to mark where the entity begins and ends.

Sentiment analysis

If Sentiment analysis is configured, the LUIS json response includes sentiment analysis. Learn more about

sentiment analysis in the [Text Analytics](#) documentation.

Sentiment data

Sentiment data is a score between 1 and 0 indicating the positive (closer to 1) or negative (closer to 0) sentiment of the data.

When culture is `en-us`, the response is:

```
"sentimentAnalysis": {  
    "label": "positive",  
    "score": 0.9163064  
}
```

For all other cultures, the response is:

```
"sentimentAnalysis": {  
    "score": 0.9163064  
}
```

Key phrase extraction entity data

The key phrase extraction entity returns key phrases in the utterance, provided by [Text Analytics](#).

```
{  
    "query": "Is there a map of places with beautiful views on a favorite trail?",  
    "topScoringIntent": {  
        "intent": "GetJobInformation",  
        "score": 0.764368951  
    },  
    "intents": [  
        ...  
    ],  
    "entities": [  
        {  
            "entity": "beautiful views",  
            "type": "builtin.keyPhrase",  
            "startIndex": 30,  
            "endIndex": 44  
        },  
        {  
            "entity": "map of places",  
            "type": "builtin.keyPhrase",  
            "startIndex": 11,  
            "endIndex": 23  
        },  
        {  
            "entity": "favorite trail",  
            "type": "builtin.keyPhrase",  
            "startIndex": 51,  
            "endIndex": 64  
        }  
    ]  
}
```

Data matching multiple entities

Luis returns all entities discovered in the utterance. As a result, your chatbot may need to make decision based on the results. An utterance can have many entities in an utterance:

```
book me 2 adult business tickets to paris tomorrow on air france
```

The LUIS endpoint can discover the same data in different entities:

```
{
  "query": "book me 2 adult business tickets to paris tomorrow on air france",
  "topScoringIntent": {
    "intent": "BookFlight",
    "score": 1.0
  },
  "intents": [
    {
      "intent": "BookFlight",
      "score": 1.0
    },
    {
      "intent": "Concierge",
      "score": 0.04216196
    },
    {
      "intent": "None",
      "score": 0.03610297
    }
  ],
  "entities": [
    {
      "entity": "air france",
      "type": "Airline",
      "startIndex": 54,
      "endIndex": 63,
      "score": 0.8291798
    },
    {
      "entity": "adult",
      "type": "Category",
      "startIndex": 10,
      "endIndex": 14,
      "resolution": {
        "values": [
          "adult"
        ]
      }
    },
    {
      "entity": "paris",
      "type": "Cities",
      "startIndex": 36,
      "endIndex": 40,
      "resolution": {
        "values": [
          "Paris"
        ]
      }
    },
    {
      "entity": "tomorrow",
      "type": "builtin.datetimeV2.date",
      "startIndex": 42,
      "endIndex": 49,
      "resolution": {
        "values": [
          {
            "timex": "2018-02-21",
            "type": "date",
            "value": "2018-02-21"
          }
        ]
      }
    },
    {
      "entity": "business",
      "type": "Category",
      "startIndex": 54,
      "endIndex": 63,
      "score": 0.8291798
    }
  ]
}
```

```

    "entity": "paris",
    "type": "Location::ToLocation",
    "startIndex": 36,
    "endIndex": 40,
    "score": 0.9730773
},
{
    "entity": "2",
    "type": "builtin.number",
    "startIndex": 8,
    "endIndex": 8,
    "resolution": {
        "value": "2"
    }
},
{
    "entity": "business",
    "type": "Seat",
    "startIndex": 16,
    "endIndex": 23,
    "resolution": {
        "values": [
            "business"
        ]
    }
},
{
    "entity": "2 adult business",
    "type": "TicketSeatOrder",
    "startIndex": 8,
    "endIndex": 23,
    "score": 0.8784727
}
],
"compositeEntities": [
{
    "parentType": "TicketSeatOrder",
    "value": "2 adult business",
    "children": [
        {
            "type": "Category",
            "value": "adult"
        },
        {
            "type": "builtin.number",
            "value": "2"
        },
        {
            "type": "Seat",
            "value": "business"
        }
    ]
}
]
}

```

Data matching multiple list entities

If a word or phrase matches more than one list entity, the endpoint query returns each List entity.

For the query `when is the best time to go to red rock?`, and the app has the word `red` in more than one list, LUIS recognizes all the entities and returns an array of entities as part of the JSON endpoint response:

```
{  
  "query": "when is the best time to go to red rock?",  
  "topScoringIntent": {  
    "intent": "Calendar.Find",  
    "score": 0.06701678  
  },  
  "entities": [  
    {  
      "entity": "red",  
      "type": "Colors",  
      "startIndex": 31,  
      "endIndex": 33,  
      "resolution": {  
        "values": [  
          "Red"  
        ]  
      }  
    },  
    {  
      "entity": "red rock",  
      "type": "Cities",  
      "startIndex": 31,  
      "endIndex": 38,  
      "resolution": {  
        "values": [  
          "Destinations"  
        ]  
      }  
    }  
  ]  
}
```

Next steps

See [Add entities](#) to learn more about how to add entities to your LUIS app.

Data storage and removal in Language Understanding (LUIS) Cognitive Services

7/30/2019 • 2 minutes to read • [Edit Online](#)

LUIS stores data encrypted in an Azure data store corresponding to the region specified by the key. This data is stored for 30 days.

Export and delete app

Users have full control over [exporting](#) and [deleting](#) the app.

Utterances

Utterances can be stored in two different places.

- During **the authoring process**, utterances are created and stored in the Intent. Utterances in intents are required for a successful LUIS app. Once the app is published and receives queries at the endpoint, the endpoint request's querystring, `log=false`, determines if the endpoint utterance is stored. If the endpoint is stored, it becomes part of the active learning utterances found in the **Build** section of the portal, in the **Review endpoint utterances** section.
- When you **review endpoint utterances**, and add an utterance to an intent, the utterance is no longer stored as part of the endpoint utterances to be reviewed. It is added to the app's intents.

Delete example utterances from an intent

Delete example utterances used for training [LUIS](#). If you delete an example utterance from your LUIS app, it is removed from the LUIS web service and is unavailable for export.

Delete utterances in review from active learning

You can delete utterances from the list of user utterances that LUIS suggests in the [Review endpoint utterances page](#). Deleting utterances from this list prevents them from being suggested, but doesn't delete them from logs.

If you don't want active learning utterances, you can [disable active learning](#). Disabling active learning also disables logging.

Disable logging utterances

[Disabling active learning](#) is disables logging.

Delete an account

If you delete an account, all apps are deleted, along with their example utterances and logs. The data is retained for 60 days before the account and data are deleted permanently.

Deleting account is available from the **Settings** page. Select your account name in the top right navigation bar to get to the **Settings** page.

Data inactivity as an expired subscription

For the purposes of data retention and deletion, an inactive LUIS app may at *Microsoft's discretion* be treated as an expired subscription. An app is considered inactive if it meets the following criteria for the last 90 days:

- Has had **no** calls made to it.

- Has not been modified.
- Does not have a current key assigned to it.
- Has not had a user sign in to it.

Next steps

[Learn about exporting and deleting an app](#)

Testing example utterances in LUIS

7/29/2019 • 2 minutes to read • [Edit Online](#)

Testing is the process of providing sample utterances to LUIS and getting a response of LUIS-recognized intents and entities.

You can [test](#) LUIS interactively, one utterance at a time, or provide a [batch](#) of utterances. With testing, you compare the current [active](#) model to the published model.

What is a score in testing?

See [Prediction score](#) concepts to learn more about prediction scores.

Interactive testing

Interactive testing is done from the **Test** panel of the website. You can enter an utterance to see how intents and entities are identified and scored. If LUIS isn't predicting the intents and entities as you expect on an utterance in the testing pane, copy it to the **Intent** page as a new utterance. Then label the parts of that utterance, and train LUIS.

Batch testing

See [batch testing](#) if you are testing more than one utterance at a time.

Endpoint testing

You can test using the [endpoint](#) with a maximum of two versions of your app. With your main or live version of your app set as the **production** endpoint, add a second version to the **staging** endpoint. This approach gives you three versions of an utterance: the current model in the Test pane of the [LUIS](#) website, and the two versions at the two different endpoints.

All endpoint testing counts toward your usage quota.

Do not log tests

If you test against an endpoint, and do not want the utterance logged, remember to use the `logging=false` query string configuration.

Where to find utterances

LUIS stores all logged utterances in the query log, available for download on the [LUIS](#) website **Apps** list page, as well as the [LUIS authoring APIs](#).

Any utterances LUIS is unsure of are listed in the [Review endpoint utterances](#) page of the [LUIS](#) website.

The screenshot shows the LUIS application interface under the 'App Assets' section. On the left sidebar, there are sections for 'App Assets' (Intents, Entities), 'Improve app performance' (Review endpoint utterances, highlighted with a red box), and 'Phrase lists' (Patterns). At the bottom of the sidebar is a 'PREVIEW' button next to 'Prebuilt Domains'. The main area is titled 'Review endpoint utterances' with a help icon. It includes a 'Filter list by intent or entity' input field containing 'GetCurrentTemperature', a toggle switch for 'Entities view', and buttons for 'Add all selected utterances to their aligned intents', 'Add selected', and 'Delete'. A table lists three utterances: 1) '~ ! @' aligned to 'None 0.54' with a checkmark and delete icon. 2) '!!!!' aligned to 'None 0.45' with a checkmark and delete icon. 3) 'turn on all the lights' aligned to 'TurnAllOn 1' with a checkmark and delete icon.

Remember to train

Remember to [train](#) LUIS after you make changes to the model. Changes to the LUIS app are not seen in testing until the app is trained.

Best practices

Learn [best practices](#).

Next steps

- Learn more about [testing](#) your utterances.

Batch testing with 1000 utterances in LUIS portal

7/30/2019 • 3 minutes to read • [Edit Online](#)

Batch testing validates your [active](#) trained model to measure its prediction accuracy. A batch test helps you view the accuracy of each intent and entity in your current trained model, displaying results with a chart. Review the batch test results to take appropriate action to improve accuracy, such as adding more example utterances to an intent if your app frequently fails to identify the correct intent.

Group data for batch test

It is important that utterances used for batch testing are new to LUIS. If you have a data set of utterances, divide the utterances into three sets: example utterances added to an intent, utterances received from the published endpoint, and utterances used to batch test LUIS after it is trained.

A data set of utterances

Submit a batch file of utterances, known as a *data set*, for batch testing. The data set is a JSON-formatted file containing a maximum of 1,000 labeled **non-duplicate** utterances. You can test up to 10 data sets in an app. If you need to test more, delete a data set and then add a new one.

RULES

*No duplicate utterances

1000 utterances or less

*Duplicates are considered exact string matches, not matches that are tokenized first.

Entities allowed in batch tests

All custom entities in the model appear in the batch test entities filter even if there are no corresponding entities in the batch file data.

Batch file format

The batch file consists of utterances. Each utterance must have an expected intent prediction along with any [machine-learned entities](#) you expect to be detected.

Batch syntax template for intents with entities

Use the following template to start your batch file:

```
[  
  {  
    "text": "example utterance goes here",  
    "intent": "intent name goes here",  
    "entities":  
      [  
        {  
          "entity": "entity name 1 goes here",  
          "startPos": 14,  
          "endPos": 23  
        },  
        {  
          "entity": "entity name 2 goes here",  
          "startPos": 14,  
          "endPos": 23  
        }  
      ]  
  }  
]
```

The batch file uses the **startPos** and **endPos** properties to note the beginning and end of an entity. The values are zero-based and should not begin or end on a space. This is different from the query logs, which use `startIndex` and `endIndex` properties.

Roles in batch testing

Caution

Entity roles are not supported in batch testing.

Batch syntax template for intents without entities

Use the following template to start your batch file without entities:

```
[  
  {  
    "text": "example utterance goes here",  
    "intent": "intent name goes here",  
    "entities": []  
  }  
]
```

If you do not want to test entities, include the `entities` property and set the value as an empty array, `[]`.

Common errors importing a batch

Common errors include:

- More than 1,000 utterances
- An utterance JSON object that doesn't have an `entities` property. The property can be an empty array.
- Word(s) labeled in multiple entities
- Entity label starting or ending on a space.

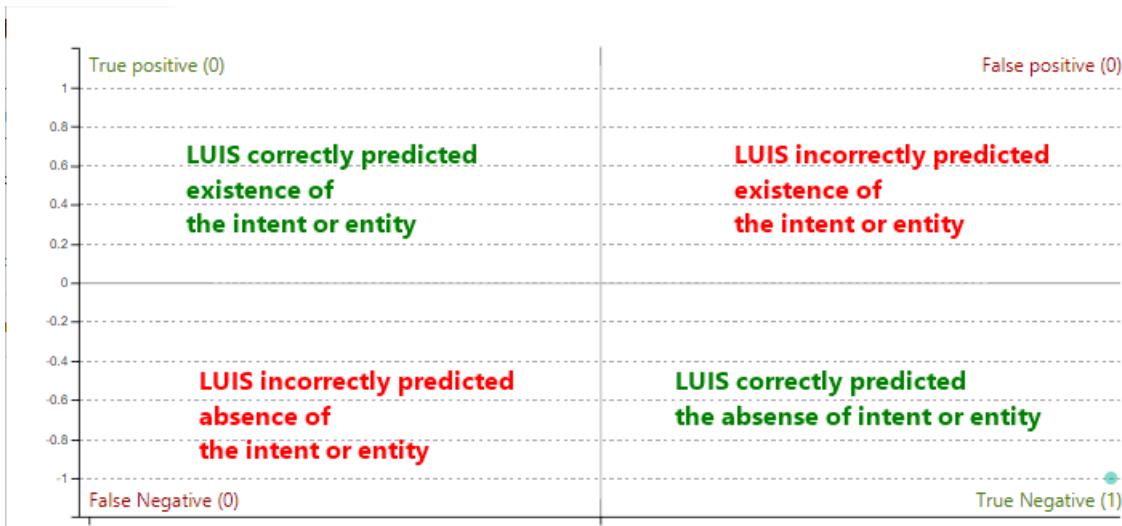
Batch test state

Luis tracks the state of each data set's last test. This includes the size (number of utterances in the batch), last run date, and last result (number of successfully predicted utterances).

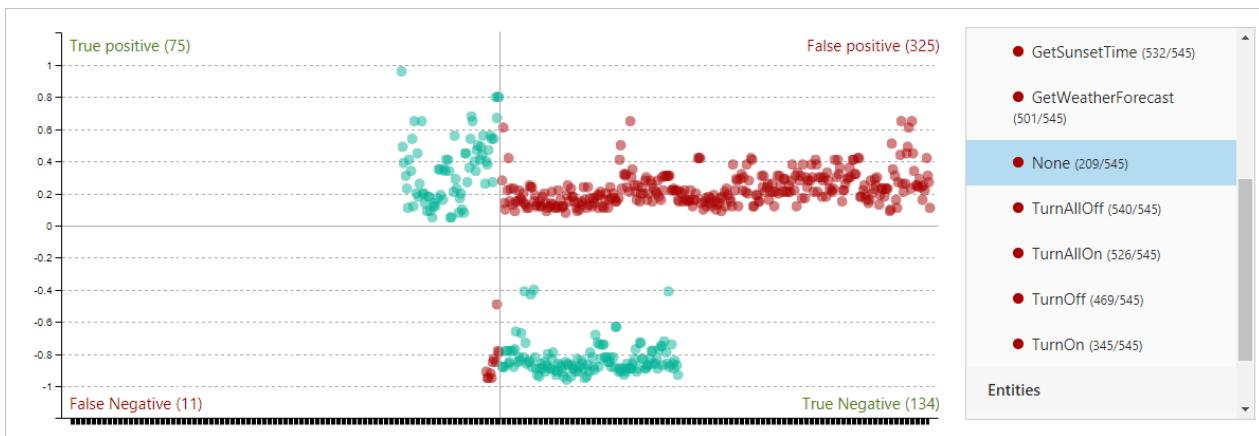
Batch test results

The batch test result is a scatter graph, known as an error matrix. This graph is a 4-way comparison of the utterances in the batch file and the current model's predicted intent and entities.

Data points on the **False Positive** and **False Negative** sections indicate errors, which should be investigated. If all data points are on the **True Positive** and **True Negative** sections, then your app's accuracy is perfect on this data set.



This chart helps you find utterances that LUIS predicts incorrectly based on its current training. The results are displayed per region of the chart. Select individual points on the graph to review the utterance information or select region name to review utterance results in that region.



Errors in the results

Errors in the batch test indicate intents that are not predicted as noted in the batch file. Errors are indicated in the two red sections of the chart.

The false positive section indicates that an utterance matched an intent or entity when it shouldn't have. The false negative indicates an utterance did not match an intent or entity when it should have.

Fixing batch errors

If there are errors in the batch testing, you can either add more utterances to an intent, and/or label more utterances with the entity to help LUIS make the discrimination between intents. If you have added utterances, and labeled them, and still get prediction errors in batch testing, consider adding a [phrase list](#) feature with domain-specific vocabulary to help LUIS learn faster.

Next steps

- Learn how to [test a batch](#)

Enterprise strategies for a LUIS app

7/29/2019 • 3 minutes to read • [Edit Online](#)

Review these design strategies for your enterprise app.

When you expect LUIS requests beyond the quota

LUIS has a monthly quota as well as a per second quota, based on the pricing tier of the Azure resource.

If your LUIS app request rate exceeds the allowed [quota rate](#), you can:

- Spread the load to more LUIS apps with the [same app definition](#). This includes, optionally, running LUIS from a [container](#).
- Create and [assign multiple keys](#) to the app.

Use multiple apps with same app definition

Export the original LUIS app, then import the app back into separate apps. Each app has its own app ID. When you publish, instead of using the same key across all apps, create a separate key for each app. Balance the load across all apps so that no single app is overwhelmed. Add [Application Insights](#) to monitor usage.

In order to get the same top intent between all the apps, make sure the intent prediction between the first and second intent is wide enough that LUIS is not confused, giving different results between apps for minor variations in utterances.

When training these sibling apps, make sure to [train with all data](#).

Designate a single app as the master. Any utterances that are suggested for review should be added to the master app then moved back to all the other apps. This is either a full export of the app, or loading the labeled utterances from the master to the children. Loading can be done from either the [LUIS](#) website or the authoring API for a [single utterance](#) or for a [batch](#).

Schedule a periodic review, such as every two weeks, of [endpoint utterances](#) for active learning, then retrain and republish.

Assign multiple LUIS keys to same app

If your LUIS app receives more endpoint hits than your single key's quota allows, create and assign more keys to the LUIS app. Create a traffic manager or load balancer to manage the endpoint queries across the endpoint keys.

When your monolithic app returns wrong intent

If your app is meant to predict a wide variety of user utterances, consider implementing the [dispatch model](#).

Breaking up a monolithic app allows LUIS to focus detection between intents successfully instead of getting confused between intents across the parent app and child apps.

Schedule a periodic [review of endpoint utterances](#) for active learning, such as every two weeks, then retrain and republish.

When you need to have more than 500 intents

Assume you're developing an office assistant that has over 500 intents. If 200 intents relate to scheduling meetings, 200 are about reminders, 200 are about getting information about colleagues, and 200 are for sending email, group intents so that each group is in a single app, then create a top-level app containing each intent. Use the

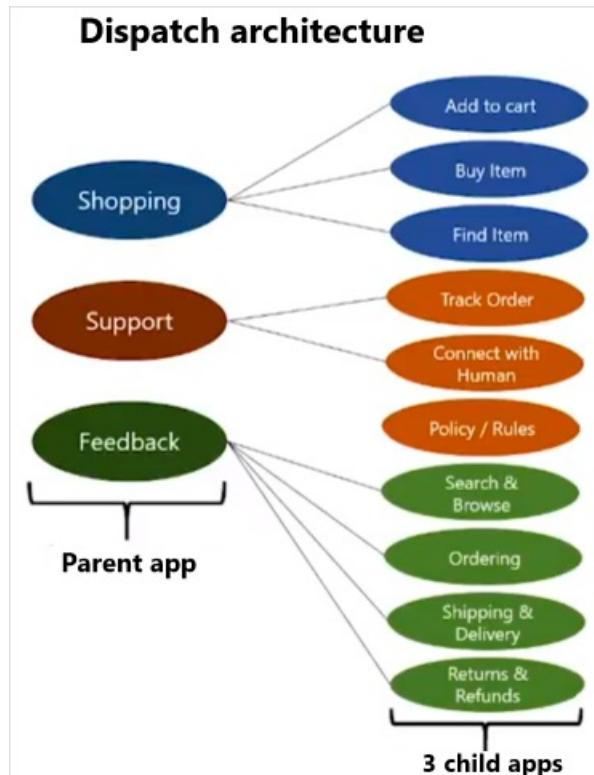
[dispatch model](#) to build the top-level app. Then change your bot to use the cascading call as shown in the [dispatch model's tutorial](#).

When you need to combine several LUIS and QnA maker apps

If you have several LUIS and QnA maker apps that need to respond to a bot, use the [dispatch model](#) to build the top-level app. Then change your bot to use the cascading call as shown in the [dispatch model's tutorial](#).

Dispatch tool and model

Use the [Dispatch](#) command-line tool, found in [BotBuilder-tools](#) to combine multiple LUIS and/or QnA Maker apps into a parent LUIS app. This approach allows you to have a parent domain including all subjects and different child subject domains in separate apps.



The parent domain is noted in LUIS with a version named `Dispatch` in the apps list.

The chat bot receives the utterance, then sends to the parent LUIS app for prediction. The top predicted intent from the parent app determines which LUIS child app is called next. The chat bot sends the utterance to the child app for a more specific prediction.

Understand how this hierarchy of calls is made from the Bot Builder v4 [dispatcher-application-tutorial](#).

Intent limits in dispatch model

A dispatch application has 500 dispatch sources, equivalent to 500 intents, as the maximum.

More information

- [Bot framework SDK](#)
- [Dispatch model tutorial](#)
- [Dispatch CLI](#)
- Dispatch model bot sample - [.NET](#), [Node.js](#)

Next steps

- Learn how to [test a batch](#)

Authoring and endpoint user access

7/26/2019 • 3 minutes to read • [Edit Online](#)

Authoring access is available for owners and collaborators. For a private app, endpoint access is available for owners and collaborators. For a public app, endpoint access is available to everyone that has their own Azure Cognitive Service or [LUIS](#) resource, and has the public app's ID.

Access to authoring

Access to the app from the [LUIS](#) website or the [authoring APIs](#) is controlled by the owner of the app.

The owner and all collaborators have access to author the app.

AUTHORING ACCESS INCLUDES	NOTES
Add or remove endpoint keys	
Exporting version	
Export endpoint logs	
Importing version	
Make app public	When an app is public, anyone with an authoring or endpoint key can query the app.
Modify model	
Publish	
Review endpoint utterances for active learning	
Train	

Access to endpoint

Access to query the endpoint is controlled by a setting on the **Application Information** page in the **Manage** section.

Application Information

- Keys and Endpoints
- Publish Settings
- Versions
- Collaborators

Application Information

Application ID ae270639-789d-4d21-ac21-87e673492535

Display Name **versions-test**

(Required)

Description

Culture en-us

Make this app public so that non-contributors can query this app with a valid key.

Not public

PRIVATE ENDPOINT	PUBLIC ENDPOINT
Available to owner and collaborators	Available to owner, collaborators, and anyone else that knows app ID

Private app endpoint security

A private app's endpoint is only available to the following:

KEY AND USER	EXPLANATION
Owner's authoring key	Up to 1000 endpoint hits
Collaborators' authoring keys	Up to 1000 endpoint hits
Any key assigned to LUIS by an author or collaborator	Based on key usage tier

Microsoft user accounts

Authors and collaborators can assign keys to a private LUIS app. The Microsoft user account that creates the LUIS key in the Azure portal needs to be either the app owner or an app collaborator. You can't assign a key to a private app from another Azure account.

See [Azure Active Directory tenant user](#) to learn more about Active Directory user accounts.

Public app endpoint access

Once an app is configured as public, *any* valid LUIS authoring key or LUIS endpoint key can query your app, as long as the key has not used the entire endpoint quota.

A user who is not an owner or collaborator, can only access a public app if given the app ID. LUIS doesn't have a public *market* or other way to search for a public app.

A public app is published in all regions so that a user with a region-based LUIS resource key can access the app in whichever region is associated with the resource key.

Microsoft user accounts

Authors and collaborators can add keys to LUIS on the Publish page. The Microsoft user account that creates the LUIS key in the Azure portal needs to be either the app owner or an app collaborator.

See [Azure Active Directory tenant user](#) to learn more about Active Directory user accounts.

Securing the endpoint

You can control who can see your LUIS endpoint key by calling it in a server-to-server environment. If you are using LUIS from a bot, the connection between the bot and LUIS is already secure. If you are calling the LUIS endpoint directly, you should create a server-side API (such as an Azure [function](#)) with controlled access (such as [AAD](#)). When the server-side API is called and authentication and authorization are verified, pass the call on to LUIS. While this strategy doesn't prevent man-in-the-middle attacks, it obfuscates your endpoint from your users, allows you to track access, and allows you to add endpoint response logging (such as [Application Insights](#)).

Security Compliance

Luis successfully completed the ISO 27001:2013 and ISO 27018:2014 audit with ZERO non-conformities (findings) in the audit report. Additionally, Luis also obtained the CSA STAR Certification with the highest possible Gold Award for the maturity capability assessment. Azure is the only major public cloud service provider to earn this certification. For more details, you can find the Luis included in the updated scope statement in Azure's main [compliance overview](#) document that is referenced on [Trust Center](#) ISO pages.

Next steps

See [Best Practices](#) to learn how to use intents and entities for the best predictions.

Authoring and query prediction endpoint keys in LUIS

7/29/2019 • 3 minutes to read • [Edit Online](#)

LUIS uses two keys: [authoring](#) and [endpoint](#). The authoring key is created for you automatically when you create your LUIS account. When you are ready to publish your LUIS app, you need to [create the endpoint key, assign it](#) to your LUIS app, and [use it with the endpoint query](#).

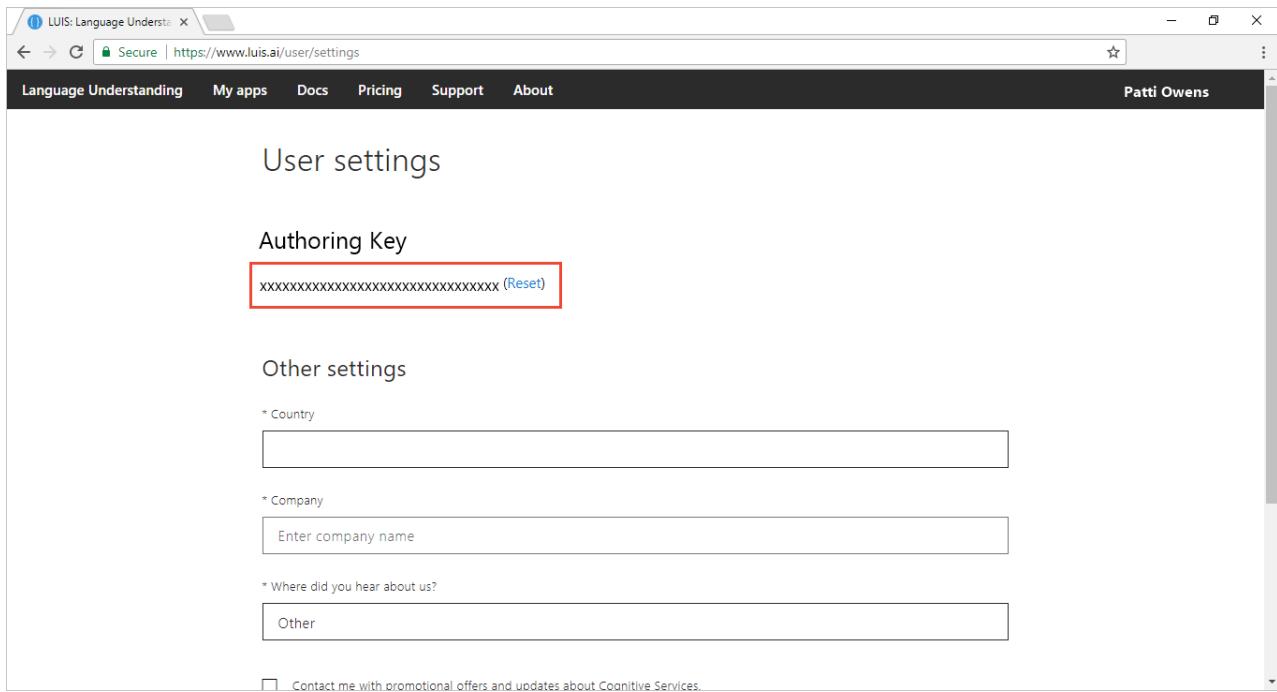
KEY	PURPOSE
Authoring key	Authoring, publishing, managing collaborators, versioning
Endpoint key	Querying

It is important to author LUIS apps in [regions](#) where you also want to publish and query.

Authoring key

An authoring key, also known as a starter key, is created automatically when you create a LUIS account and it is free. You have one authoring key across all your LUIS apps for each authoring [region](#). The authoring key is provided to author your LUIS app or to test endpoint queries.

To find the authoring Key, sign in to [LUIS](#) and click on the account name in the upper-right navigation bar to open **Account Settings**.



When you want to make **production endpoint queries**, create the Azure [LUIS subscription](#).

Caution

For convenience, many of the samples use the Authoring key since it provides a few endpoint calls in its [quota](#).

Endpoint key

When you need **production endpoint queries**, create an Azure Resource then assign it to the LUIS app.

If you are new to Cognitive Services or new to Azure, create a [temporary key](#). If you have an Azure account, create a new resource in the portal. Language Understanding endpoint queries support two Azure Resource types: [Language](#)

Understanding or the **Cognitive Service** resource. You need the name of the resource when you [assign](#) the key to the app.

When the Azure resource creation process is finished, [assign the key](#) to the app.

- The endpoint 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 endpoint key can be used for all your LUIS apps or for specific LUIS apps.
- Do not use the endpoint key for authoring LUIS apps.

Use endpoint key in query

The LUIS endpoint accepts two styles of query, both use the endpoint key, but in different places:

VERB	EXAMPLE URL AND KEY LOCATION
GET	<p><code>https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951ca1c2?subscription-key=your-endpoint-key-here&verbose=true&timezoneOffset=0&q=turn%20on%20the%20lights</code></p> <p>query string value for <code>subscription-key</code></p> <p>Change your endpoint query value for the <code>subscription-key</code> from the authoring (starter) key, to the new endpoint key in order to use the LUIS endpoint key quota rate. If you create the key, and assign the key but do not change the endpoint query value for <code>subscription-key</code>, you are not using your endpoint key quota.</p>
POST	<p><code>https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951ca1c2</code></p> <p>header value for <code>Ocp-Apim-Subscription-Key</code></p> <p>Change your endpoint query value for the <code>Ocp-Apim-Subscription-Key</code> from the authoring (starter) key, to the new endpoint key in order to use the LUIS endpoint key quota rate. If you create the key, and assign the key but do not change the endpoint query value for <code>Ocp-Apim-Subscription-Key</code>, you are not using your endpoint key quota.</p>

The app ID used in the previous URLs, `df67dcdb-c37d-46af-88e1-8b97951ca1c2`, is the public IoT app used for the [interactive demonstration](#).

API usage of Ocp-Apim-Subscription-Key

The LUIS APIs use the header, `Ocp-Apim-Subscription-Key`. The header name does not change based on which key and set of APIs you are using. Set the header to the authoring key for authoring APIs. If you are using the endpoint, set the header to the endpoint key.

You can't pass the endpoint key for authoring APIs. If you do, you get a 401 error - access denied due to invalid endpoint key.

Key limits

See [Key Limits](#) and [Azure Regions](#). The authoring key is free and used for authoring. The LUIS endpoint key has a free tier but must be created by you and associated with your LUIS app on the **Publish** page. It can't be used for authoring, but only endpoint queries.

Publishing regions are different from authoring regions. Make sure you create an app in the authoring region

corresponding to the publishing region you want.

Key limit errors

If you exceed your per second quota, you receive an HTTP 429 error. If you exceed your per month quota, you receive an HTTP 403 error. Fix these errors by getting a LUIS endpoint key, [assigning](#) the key to the app on the **Publish** page of the [LUIS](#) website.

Assignment of the endpoint key

You can [assign](#) the endpoint key in the [LUIS portal](#) or via the corresponding APIs.

Next steps

- Learn [concepts](#) about authoring and endpoint keys.

Create a new LUIS app in the LUIS portal

8/9/2019 • 2 minutes to read • [Edit Online](#)

There are a couple of ways to create a LUIS app. You can create a LUIS app in the [LUIS](#) portal, or through the LUIS authoring [APIs](#).

Using the LUIS portal

You can create a new app in the LUIS portal in several ways:

- Start with an empty app and create intents, utterances, and entities.
- Start with an empty app and add a [prebuilt domain](#).
- Import a LUIS app from a JSON file that already contains intents, utterances, and entities.

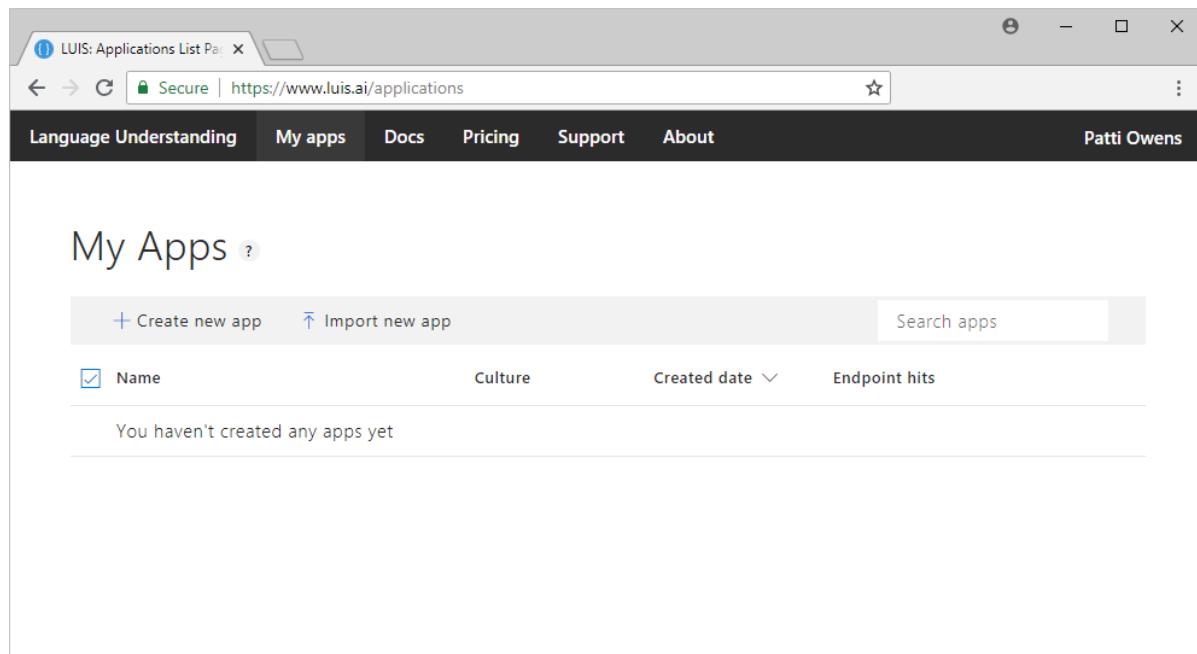
Using the authoring APIs

You can create a new app with the authoring APIs in a couple of ways:

- [Start](#) with an empty app and create intents, utterances, and entities.
- [Start](#) with a prebuilt domain.

Create new app in LUIS

1. On **My Apps** page, select **Create new app**.



2. In the dialog box, name your application "TravelAgent".

Create new app

* Name

* Culture
** Culture is the language that your app understands and speaks, not the interface language.

Description

3. Choose your application culture (for TravelAgent app, choose English), and then select **Done**.

NOTE

The culture cannot be changed once the application is created.

Import an app from file

1. On **My Apps** page, select **Import new app**.
2. In the pop-up dialog, select a valid app JSON file, and then select **Done**.

Import errors

Possible errors are:

- An app with that name already exists. To fix this, reimport the app, and set the **Optional Name** to a new name.

Export app for backup

1. On **My Apps** page, select **Export**.
2. Select **Export as JSON**. Your browser downloads the active version of the app.
3. Add this file to your backup system to archive the model.

Export app for containers

1. On **My Apps** page, select **Export**.
2. Select **Export as container** then select which published slot (production or stage) you want to export.
3. Use this file with your [LUIS container](#).

If you are interested in exporting a trained but not yet published model to use with the LUIS container, go to the **Versions** page and export from there.

Delete app

1. On **My Apps** page, select the three dots (...) at the end of the app row.
2. Select **Delete** from the menu.
3. Select **Ok** in the confirmation window.

Next steps

Your first task in the app is to [add intents](#).

Add intents to determine user intention of utterances

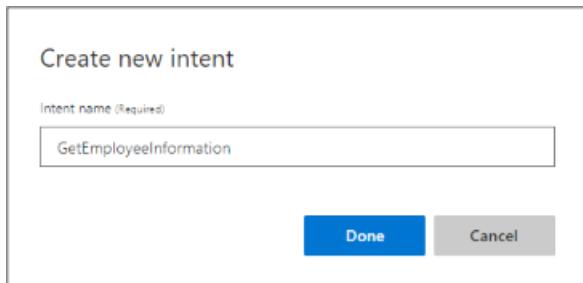
8/9/2019 • 3 minutes to read • [Edit Online](#)

Add [intents](#) to your LUIS app to identify groups of questions or commands that have the same intention.

Intents are managed from top navigation bar's **Build** section, then from the left panel's **Intents**.

Add intent

1. On the **Intents** page, select **Create new intent**.
2. In the **Create new intent** dialog box, enter the intent name, `GetEmployeeInformation`, and click **Done**.



Add an example utterance

Example utterances are text examples of user questions or commands. To teach Language Understanding (LUIS), you need to add example utterances to an intent.

1. On the **GetEmployeeInformation** intent details page, enter a relevant utterance you expect from your users, such as `Does John Smith work in Seattle?` in the text box below the intent name, and then press Enter.

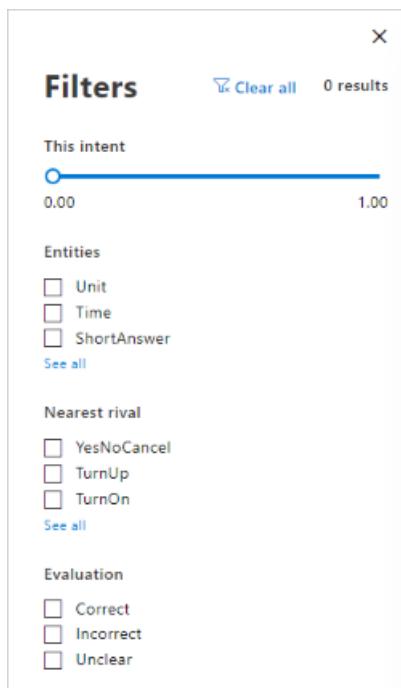
The screenshot shows the LUIS interface for managing the 'GetEmployeeInformation' intent. The intent name 'GetEmployeeInformation' is displayed prominently. Below it, there is a text input field containing the utterance 'Does John Smith work in Seattle?'. This input field is highlighted with a red rectangle. To the right of the input field is a 'Delete Intent' button. On the left side of the page, there is a sidebar with sections for 'App Assets' (Intents, Entities) and 'Improve app performance' (Review endpoint utterances, Phrase lists, Patterns). At the bottom left, there is a 'PREVIEW' section for 'Prebuilt Domains'. The main area of the page shows the intent details, including the utterance and any associated entities.

LUIS converts all utterances to lowercase and adds spaces around tokens such as hyphens.

Intent prediction errors

An example utterance in an intent might have an intent prediction error between the intent the example utterance is currently in and the prediction intent determined during training.

To find utterance prediction errors and fix them, use the **Filter** option's **Evaluation** options of **Incorrect** and **Unclear** combined with the **View** option of **Detailed view**.



When the filters and view are applied, and there are example utterances with errors, the example utterance list shows the utterances and the issues.

The screenshot shows the LUIS 'BUILD' tab for the 'Device_KeyPress' intent. The left sidebar shows 'App Assets' and 'Intents' (selected). The main area displays the intent details and a table of example utterances:

	This intent	Nearest rival	Evaluation
Example utterance	0.85	0.87	DeviceRelatedAnswer -0.02
silence	0.91	0.87	DeviceRelatedAnswer 0.04
silent	0.91	0.87	DeviceRelatedAnswer 0.04
quiet	0.91	0.87	DeviceRelatedAnswer 0.04
muted	0.91	0.87	DeviceRelatedAnswer 0.04
mute	0.99	0.87	DeviceRelatedAnswer 0.12
exit	0.94	0.87	DeviceRelatedAnswer 0.07

Each row shows the current training's prediction score for the example utterance, the nearest rival's score, which is the difference in these two scores.

Fixing intents

To learn how to fix intent prediction errors, use the [Summary Dashboard](#). The summary dashboard provides analysis for the active version's last training and offers the top suggestions to fix your model.

Add a custom entity

Once an utterance is added to an intent, you can select text from within the utterance to create a custom entity. A custom entity is a way to tag text for extraction, along with the correct intent.

See [Add entity to utterance](#) to learn more.

Entity prediction discrepancy errors

The entity is underlined in red to indicate an [entity prediction discrepancy](#). Because this is the first occurrence of an entity, there are not enough examples for LUIS to have a high-confidence that this text is tagged with the correct entity. This discrepancy is removed when the app is trained.



The text is highlighted in blue, indicating an entity.

Add a prebuilt entity

For information, see [Prebuilt entity](#).

Using the contextual toolbar

When one or more example utterances are selected in the list, by checking the box to the left of an utterance, the toolbar above the utterance list allows you to perform the following actions:

- Reassign intent: move utterance(s) to different intent
- Delete utterance(s)
- Entity filters: only show utterances containing filtered entities
- Show all/Errors only: show utterances with prediction errors or show all utterances
- Entities/Tokens view: show entities view with entity names or show raw text of utterance
- Magnifying glass: search for utterances containing specific text

Working with an individual utterance

The following actions can be performed on an individual utterance from the ellipsis menu to the right of the utterance:

- Edit: change the text of the utterance
- Delete: remove the utterance from the intent. If you still want the utterance, a better method is to move it to the **None** intent.
- Add a pattern: A pattern allows you to take a common utterance and mark replaceable text and ignorable text, thereby reducing the need for more utterances in the intent.

The **Labeled intent** column allows you to change the intent of the utterance.

Train your app after changing model with intents

After you add, edit, or remove intents, [train](#) and [publish](#) your app so that your changes are applied to endpoint queries.

Next steps

Learn more about adding [example utterances](#) with entities.

Add an entity to example utterances

8/9/2019 • 6 minutes to read • [Edit Online](#)

Example utterances are text examples of user questions or commands. To teach Language Understanding (LUIS), you need to add [example utterances](#) to an [intent](#).

Usually, you add an example utterance to an intent first, and then you create entities and label utterances on the [Intents](#) page. If you would rather create entities first, see [Add entities](#).

Marking entities in example utterances

When you select text in the example utterance to mark for an entity, an in-place pop-up menu appears. Use this menu to either create or select an entity.

Certain entity types, such as prebuilt entities and regular expression entities, cannot be tagged in the example utterance because they are tagged automatically.

Add a simple entity

In the following procedure, you create and tag a custom entity within the following utterance on the [Intents](#) page:

Are there any SQL server jobs?

1. Select `SQL server` in the utterance to label it as a simple entity. In the entity drop-down box that appears, you can either select an existing entity or add a new entity. To add a new entity, type its name `Job` in the text box, and then select **Create new entity**.

The screenshot shows the LUIS Intents page with the intent name "GetJobInformation". A sample utterance "are there any [sql server] jobs ?" is displayed. The word "[sql server]" is selected, and a context menu is open, showing options: "Job", "Job (Create new entity)" (which is highlighted), "Wrap in composite entity", and "Add a prebuilt entity".

NOTE

When selecting words to tag as entities:

- For a single word, just select it.
- For a set of two or more words, select the first word and then the final word.

2. In the **What type of entity do you want to create?** pop-up box, verify the entity name and select the **Simple** entity type, and then select **Done**.

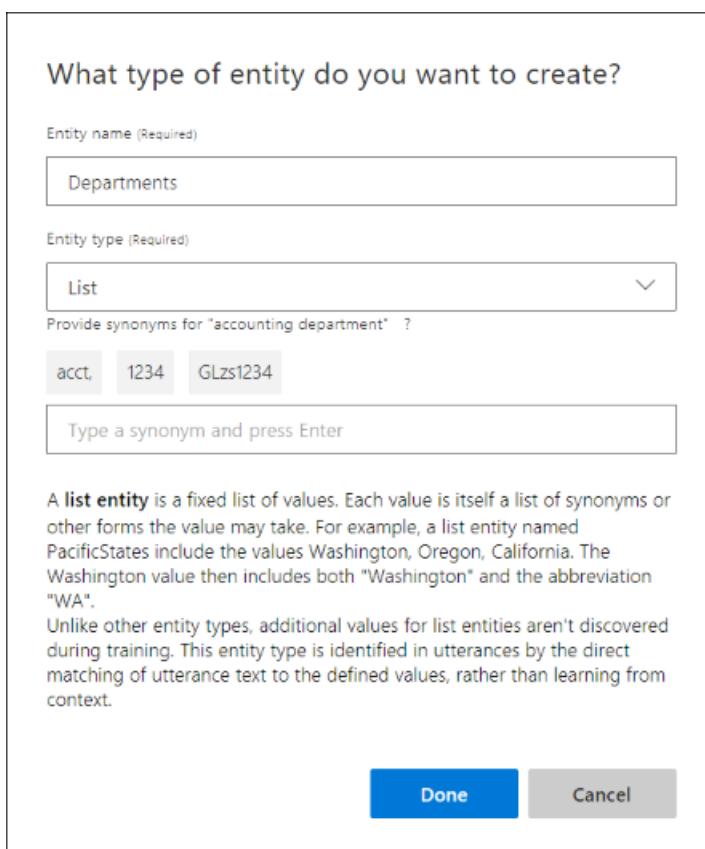
A [phrase list](#) is commonly used to boost the signal of a simple entity.

Add a list entity

List entities represent a set of exact text matches of related words in your system.

For a company's department list, you can have normalized values: `Accounting` and `Human Resources`. Each normalized name has synonyms. For a department, these synonyms can include any department acronyms, numbers, or slang. You don't have to know all the values when you create the entity. You can add more after reviewing real user utterances with synonyms.

1. In an example utterance on the **Intents** page, select the word or phrase that you want in the new list. When the entity drop-down appears, enter the name for the new list entity in the top textbox, then select **Create new entity**.
2. In the **What type of entity do you want to create?** pop-up box, name the entity and select **List** as the type. Add synonyms of this list item, then select **Done**.



You can add more list items or more item synonyms by labeling other utterances, or by editing the entity from the **Entities** in the left navigation. [Editing](#) the entities gives you the options of entering additional items with corresponding synonyms or importing a list.

Add a composite entity

Composite entities are created from existing **Entities** to form a parent entity.

Assuming the utterance, `Does John Smith work in Seattle?`, a composite utterance can return entity information of the employee name `John Smith`, and the location `Seattle` in a composite entity. The child entities must already exist in the app and be marked in the example utterance before creating the composite entity.

1. To wrap the child entities into a composite entity, select the **first** labeled entity (left-most) in the utterance for the composite entity. A drop-down list appears to show the choices for this selection.
2. Select **Wrap in composite entity** from the drop-down list.
3. Select the last word of the composite entity (right-most). Notice a green line follows the composite entity. This is the visual indicator for a composite entity and should be under all words in the composite entity from the left-most child entity to the right-most child entity.
4. Enter the composite entity name in the drop-down list.

When you wrap the entities correctly, a green line is under the entire phrase.

5. Validate the composite entity details on the **What type of entity do you want to create?** pop-up box then select **Done**.

The dialog box has a title 'What type of entity do you want to create?'. It contains the following fields:

- Entity name (Required)**: A text input field containing 'employeeinformation'.
- Entity type (Required)**: A dropdown menu showing 'Composite'.
- Child entity**: A dropdown menu showing 'personName' with a delete icon to its right.
- Child entity**: A dropdown menu showing 'Location' with a delete icon to its right.
- + Add a child entity**: A link to add more child entities.
- Use a composite entity**: A descriptive text explaining that a composite entity represents an object with parts, made up of child entities like Number, PassengerCategory, and TravelClass.
- Done** and **Cancel** buttons at the bottom.

6. The composite entity displays with both blue highlights for individual entities and a green underline for the entire composite entity.

The screenshot shows the LUIS portal interface. On the left, there's a sidebar with sections like App Assets, Intents, Entities, Improve app performance, Review endpoint utterances, Phrase lists, and Patterns. The main area is titled 'GetEmployeeInformation' with a 'Delete Intent' button. Below it is a text input field with placeholder text 'Type about 5 examples of what a user might say and hit Enter'. Underneath is a table with columns for Utterance, Entity filters, Show All, Entities View, and Labeled intent. An utterance 'does personName work in Location ?' is listed, with 'personName' and 'Location' underlined in red. A table below shows entities used in the intent: 'personName' with 1 labeled utterance and 'Location' with 1 labeled utterance.

Add entity's role to utterance

A role is a named subtype of an entity, determined by the context of the utterance. You can mark an entity within an utterance as the entity, or select a role within that entity. Any entity can have roles including custom entities that are machine-learned (simple entities and composite entities), are not machine-learned (prebuilt entities, regular expression entities, list entities).

[Learn how to mark an utterance with entity roles](#) from a hands-on tutorial.

Entity status predictions

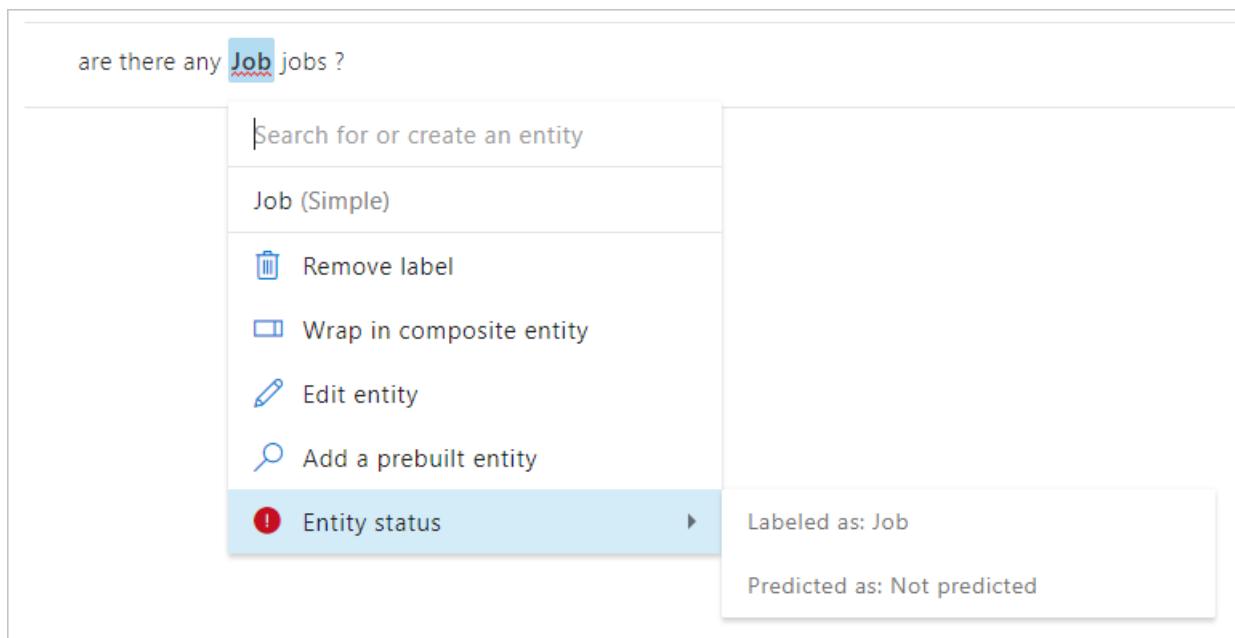
When you enter a new utterance in the LUIS portal, the utterance may have entity prediction errors. The prediction error is a difference between how an entity is labeled compared with how LUIS has predicted the entity.

This difference is visually represented in the LUIS portal with a red underline in the utterance. The red underline may appear in entity brackets or outside of brackets.

[sql server]

Select the words that are underlined in red in the utterance.

The entity box displays the **Entity status** with a red exclamation mark if there is a prediction discrepancy. To see the Entity status with information about the difference between labeled and predicted entities, select **Entity status** then select the item to the right.



The red-line can appear at any of the following times:

- When an utterance is entered but before the entity is labeled
- When the entity label is applied
- When the entity label is removed
- When more than one entity label is predicted for that text

The following solutions help resolve the entity prediction discrepancy:

ENTITY	VISUAL INDICATOR	PREDICTION	SOLUTION
Utterance entered, entity isn't labeled yet.	red underline	Prediction is correct.	Label the entity with the predicted value.
Unlabeled text	red underline	Incorrect prediction	The current utterances using this incorrect entity need to be reviewed across all intents. The current utterances have mistaught LUIS that this text is the predicted entity.
Correctly labeled text	blue entity highlight, red underline	Incorrect prediction	Provide more utterances with the correctly labeled entity in a variety of places and usages. The current utterances are either not sufficient to teach LUIS that this is the entity or similar entities appear in the same context. Similar entity should be combined into a single entity so LUIS isn't confused. Another solution is to add a phrase list to boost the significance of the words.

ENTITY	VISUAL INDICATOR	PREDICTION	SOLUTION
Incorrectly labeled text	blue entity highlight, red underline	Correct prediction	Provide more utterances with the correctly labeled entity in a variety of places and usages.

Other actions

You can perform actions on example utterances as a selected group or as an individual item. Groups of selected example utterances change the contextual menu above the list. Single items may use both the contextual menu above the list and the individual contextual ellipsis at the end of each utterance row.

Remove entity labels from utterances

You can remove machine-learned entity labels from an utterance on the Intents page. If the entity is not machine-learned, it can't be removed from an utterance. If you need to remove a non-machine-learned entity from the utterance, you need to delete the entity from the entire app.

To remove a machine-learned entity label from an utterance, select the entity in the utterance. Then select **Remove Label** in the entity drop-down box that appears.

Add a prebuilt entity label

When you add the prebuilt entities to your LUIS app, you don't need to tag utterances with these entities. To learn more about prebuilt entities and how to add them, see [Add entities](#).

Add a regular expression entity label

If you add the regular expression entities to your LUIS app, you don't need to tag utterances with these entities. To learn more about regular expression entities and how to add them, see [Add entities](#).

Create a pattern from an utterance

See [Add pattern from existing utterance on intent or entity page](#).

Add a pattern.any entity

If you add the pattern.any entities to your LUIS app, you can't label utterances with these entities. They are only valid in patterns. To learn more about pattern.any entities and how to add them, see [Add entities](#).

Train your app after changing model with utterances

After you add, edit, or remove utterances, [train](#) and [publish](#) your app for your changes to affect endpoint queries.

Next steps

After labeling utterances in your **Intents**, you can now create a [composite entity](#).

Create entities without utterances

8/9/2019 • 6 minutes to read • [Edit Online](#)

The entity represents a word or phrase inside the utterance that you want extracted. 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. You can create entities when you add an utterance to an intent or apart from (before or after) adding an utterance to an intent.

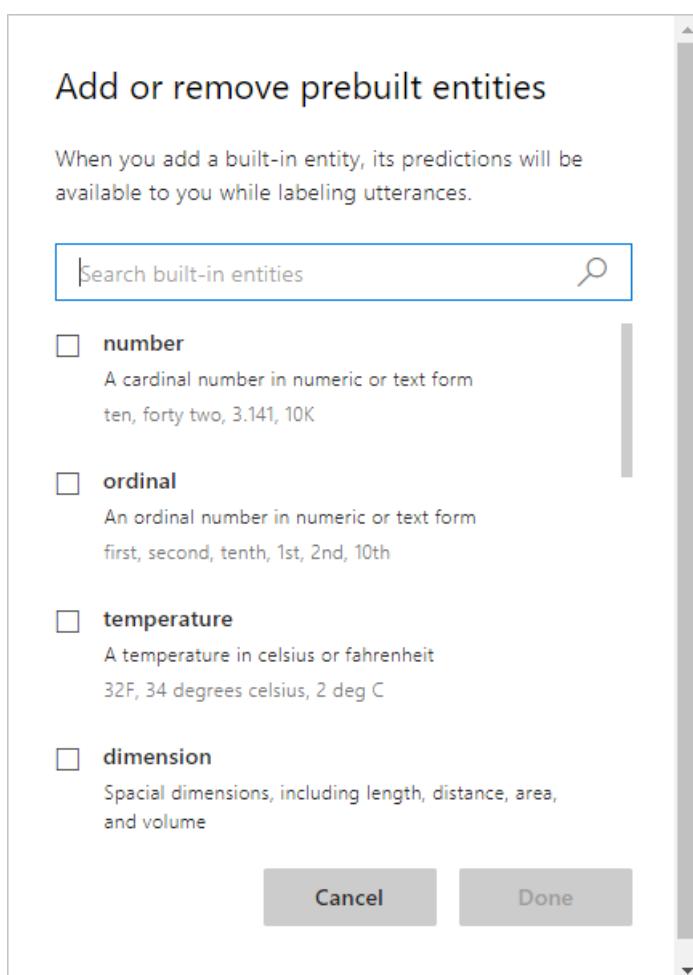
You can add, edit, or delete entities in your LUIS app through the **Entities list** on the **Entities** page. LUIS offers two main types of entities: [prebuilt entities](#), and your own [custom entities](#).

Once a machine-learned entity is created, you need to mark that entity in all the example utterance of all the intents it is in.

Add a prebuilt entity to your app

Common prebuilt entities added to an application are *number* and *datetimeV2*.

1. In your app, from the **Build** section, select **Entities** in the left panel.
2. On the **Entities** page, select **Add prebuilt entities**.
3. In **Add prebuilt entities** dialog box, select the **number** and **datetimeV2** prebuilt entities. Then select **Done**.



Add simple entities for single concepts

A simple entity describes a single concept. Use the following procedure to create an entity that extracts company department names such as *Human resources* or *Operations*.

1. In your app, select the **Build** section, then select **Entities** in the left panel, then select **Create new entity**.
2. In the pop-up dialog box, type `Location` in the **Entity name** box, select **Simple** from the **Entity type** list, and then select **Done**.

Once this entity is created, go to all intents that have example utterances that contain the entity. Select the text in the example utterance and mark the text as the entity.

A [phrase list](#) is commonly used to boost the signal of a simple entity.

Add regular expression entities for highly structured concepts

A regular expression entity is used to pull out data from the utterance based on a regular expression you provide.

1. In your app, select **Entities** from the left navigation, and then select **Create new entity**.
2. In the pop-up dialog box, enter `Human resources form name` in the **Entity name** box, select **Regular expression** from the **Entity type** list, enter the regular expression `hrf-[0-9]{6}`, and then select **Done**.

This regular expression matches literal characters `hrf-`, then 6 digits to represent a form number for a Human resources form.

Add composite entities to group into a parent-child relationship

You can define relationships between entities of different types by creating a composite entity. In the following example, the entity contains a regular expression, and a prebuilt entity of name.

In the utterance `Send hrf-123456 to John Smith`, the text `hrf-123456` is matched to a human resources [regular expression](#) and `John Smith` is extracted with the prebuilt entity `personName`. Each entity is part of a larger, parent entity.

1. In your app, select **Entities** from the left navigation of the **Build** section, and then select **Add prebuilt entity**.
2. Add the prebuilt entity **PersonName**. For instructions, see [Add Prebuilt Entities](#).
3. Select **Entities** from the left navigation, and then select **Create new entity**.
4. In the pop-up dialog box, enter `SendHrForm` in the **Entity name** box, then select **Composite** from the **Entity type** list.
5. Select **Add Child** to add a new child.
6. In **Child #1**, select the entity **number** from the list.
7. In **Child #2**, select the entity **Human resources form name** from the list.
8. Select **Done**.

Add Pattern.any entities to capture free-form entities

[Pattern.any](#) entities are only valid in [patterns](#), not intents. This type of entity helps LUIS find the end of entities of varying length and word choice. Because this entity is used in a pattern, LUIS knows where the end of the entity is in the utterance template.

If an app has a `FindHumanResourcesForm` intent, the extracted form title may interfere with the intent prediction. In order to clarify which words are in the form title, use a `Pattern.any` within a pattern. The LUIS prediction begins with the utterance. First, the utterance is checked and matched for entities, when the entities are found, then the pattern is checked and matched.

In the utterance `Where is Request relocation from employee new to the company on the server?`, the form title is tricky because it is not contextually obvious where the title ends and where the rest of the utterance begins. Titles can be any order of words including a single word, complex phrases with punctuation, and nonsensical ordering of words. A pattern allows you to create an entity where the full and exact entity can be extracted. Once the title is found, the `FindHumanResourcesForm` intent is predicted because that is the intent for the pattern.

1. From the **Build** section, select **Entities** in the left panel, and then select **Create new entity**.

2. In the **Add Entity** dialog box, enter `HumanResourcesFormTitle` in the **Entity name** box, and select **Pattern.any** as the **Entity type**.

To use the pattern.any entity, add a pattern on the **Patterns** page, in the **Improve app performance** section, with the correct curly brace syntax, such as `Where is **{HumanResourcesFormTitle}** on the server?`.

If you find that your pattern, when it includes a `Pattern.any`, extracts entities incorrectly, use an [explicit list](#) to correct this problem.

Add a role to distinguish different contexts

A role is a named subtype based on context. It is available in all entities including prebuilt and non-machine-learned entities.

The syntax for a role is `{Entityname:Rolename}` where the entity name is followed by a colon, then the role name. For example, `Move {personName} from {Location:Origin} to {Location:Destination}`.

1. From the **Build** section, select **Entities** in the left panel.

2. Select **Create new entity**. Enter the name of `Location`. Select the type **Simple** and select **Done**.

3. Select **Entities** from the left panel, then select the new entity **Location** created in the previous step.

4. In the **Role name** textbox, enter the name of the role `Origin` and enter. Add a second role name of `Destination`.

The screenshot shows the Microsoft LUIS Studio interface. The top navigation bar includes 'TravieAgent (V 0.1) ▾', 'DASHBOARD', 'BUILD' (which is selected), 'MANAGE', 'Train', 'Test', and 'Publish'. On the left, a sidebar menu is open under 'App Assets' with 'Intents' and 'Entities' listed. Under 'Intents', 'FindHumanResourcesForm' is visible. Under 'Entities', 'Location' is selected, showing its details: 'Entity type: Simple'. Below this, the 'Roles' section lists 'Destination' and 'Origin'. The main area is titled 'Labeled Utterances' and contains a table with columns for 'Edit', 'Reassign intent', 'Add as pattern', 'Delete', 'Search', 'Filter', and 'View options'. A note at the bottom says 'No items created'.

Add list entities for exact matches

List entities represent a fixed, closed set of related words.

For a Human Resources app, you can have a list of all departments along with any synonyms for the departments. You don't have to know all the values when you create the entity. You can add more after reviewing real user utterances with synonyms.

1. From the **Build** section, select **Entities** in the left panel, and then select **Create new entity**.
2. In the **Add Entity** dialog box, type **Department** in the **Entity name** box and select **List** as the **Entity type**. Select **Done**.
3. The list entity page allows you to add normalized names. In the **Values** textbox, enter a department name for the list, such as **HumanResources** then press Enter on the keyboard.
4. To the right of the normalized value, enter synonyms, pressing Enter on the keyboard after each item.
5. If you want more normalized items for the list, select **Recommend** to see options from the [semantic dictionary](#).

The screenshot shows the 'Department' list entity configuration page. At the top, it says 'Entity type: List'. Below that is a 'Recommend' button followed by a list of suggested items: operational +, logistics +, manager +, management +, officer +, operations +, procurement +, personnel +, strategic +, and specialist +. To the right of these suggestions is a 'Delete Entity' button. Below this is a 'Values' section with a table. The table has two columns: 'Normalized Value' and 'Synonyms'. The rows are:

Normalized Value	Synonyms
Accounting	Acct X dept1234 X 1234 X GL-x4123bv X
Information Technology	IT X dept1235 X 1235 X GL-x456ru X
Research and Development	R&D X RnD X 2298 X GL-x831rx X
Operations	OP X 3296 X GL-x108op X dept3296 X

At the bottom of the 'Values' section are buttons for 'Add new sublist', 'Import values', and 'Search ...'.

6. Select an item in the recommended list to add it as a normalized value or select **Add all** to add all the items. You can import values into an existing list entity using the following JSON format:

```
[  
  {  
    "canonicalForm": "Blue",  
    "list": [  
      "navy",  
      "royal",  
      "baby"  
    ]  
  },  
  {  
    "canonicalForm": "Green",  
    "list": [  
      "kelly",  
      "forest",  
      "avocado"  
    ]  
  }]  
]
```

Do not change entity type

Luis does not allow you to change the type of the entity because it doesn't know what to add or remove to construct that entity. In order to change the type, it is better to create a new entity of the correct type with a slightly different name. Once the entity is created, in each utterance, remove the old labeled entity name and add the new entity name. Once all the utterances have been relabeled, delete the old entity.

Create a pattern from an example utterance

See [Add pattern from existing utterance on intent or entity page](#).

Train your app after changing model with entities

After you add, edit, or remove entities, [train](#) and [publish](#) your app for your changes to affect endpoint queries.

Next steps

For more information about prebuilt entities, see the [Recognizers-Text](#) project.

For information about how the entity appears in the JSON endpoint query response, see [Data Extraction](#)

Now that you have added intents, utterances and entities, you have a basic Luis app. Learn how to [train](#), [test](#), and [publish](#) your app.

Add prebuilt domains for common usage scenarios

8/9/2019 • 2 minutes to read • [Edit Online](#)

LUIS includes a set of prebuilt intents from the prebuilt domains for quickly adding common intents and utterances. This is a quick and easy way to add abilities to your conversational client app without having to design the models for those abilities.

Add a prebuilt domain

1. On the **My Apps** page, select your app. This opens your app to the **Build** section of the app.
2. On the **Intents** page, select **Add prebuilt domains** from the bottom, left toolbar.
3. Select the **Calendar** intent then select **Add domain** button.

The screenshot shows the 'Prebuilt domains' section of the LUIS interface. On the left, there's a sidebar with navigation links: 'App Assets' (Intents, Entities), 'Improve app performance' (Review endpoint utterances, Phrase lists, Patterns), and a 'PREVIEW' button under 'Prebuilt Domains'. The main area is titled 'Prebuilt domains' and contains a search bar. Below it are nine cards, each representing a different domain:

- Calendar**: Provides intents and entities related to calendar entries. Includes buttons for 'Add domain' and 'Learn more'.
- Camera**: Provides intents and entities related to using a camera. Includes buttons for 'Add domain' and 'Learn more'.
- Communication**: Provides intents and entities related to email, messages, and phone calls. Includes a 'Learn more' link and an 'Add domain' button.
- Entertainment**: Provides intents and entities related to searching for movies, music, games, and TV shows. Includes buttons for 'Add domain' and 'Learn more'.
- Events**: Provides intents and entities related to booking tickets for events like concerts, festivals, sports games, and comedy shows. Includes buttons for 'Add domain' and 'Learn more'.
- Fitness**: Provides intents and entities related to tracking fitness activities. Includes a 'Learn more' link and an 'Add domain' button.
- Gaming**: Provides intents and entities related to gaming. Includes a 'The Game domain provides...' note and an 'Add domain' button.
- HomeAutomation**: Provides intents and entities related to home automation. Includes a 'The Home Automation domain...' note and an 'Add domain' button.
- MovieTickets**: Provides intents and entities related to movie tickets. Includes a 'The Movie Tickets domain...' note and an 'Add domain' button.

4. Select **Intents** in the left navigation to view the Calendar intents. Each intent from this domain is prefixed with `Calendar.`. Along with utterances, two entities for this domain are added to the app: `Calendar.Location` and `Calendar.Subject`.

Train and publish

1. After the domain is added, train the app by selecting **Train** in the top, right toolbar.
2. In the top toolbar, select **Publish**. Publish to **Production**.
3. When the green success notification appears, select the **Refer to the list of endpoints** link to see the

endpoints.

4. Select an endpoint. A new browser tab opens to that endpoint. Keep the browser tab open and continue to the **Test** section.

Test

Test the new intent at the endpoint by added a value for the **q** parameter:

```
Schedule a meeting with John Smith in Seattle next week .
```

LUIS returns the correct intent and meeting subject:

```
{
  "query": "Schedule a meeting with John Smith in Seattle next week",
  "topScoringIntent": {
    "intent": "Calendar.Add",
    "score": 0.824783146
  },
  "entities": [
    {
      "entity": "a meeting with john smith",
      "type": "Calendar.Subject",
      "startIndex": 9,
      "endIndex": 33,
      "score": 0.484055847
    }
  ]
}
```

Next steps

[Prebuilt domain reference](#)

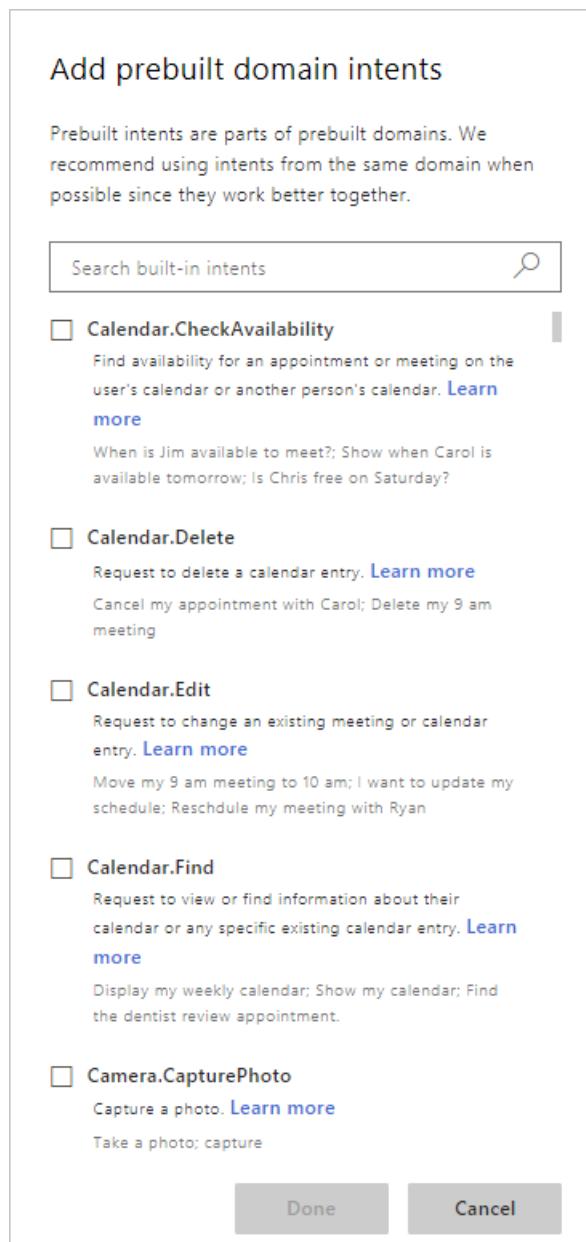
Add prebuilt intents for common intents

8/9/2019 • 2 minutes to read • [Edit Online](#)

LUIS includes a set of prebuilt intents from the prebuilt domains for quickly adding common intents and utterances. This is a quick and easy way to add abilities to your conversational client app without having to design the models for those abilities.

Add a prebuilt intent

1. On the **My Apps** page, select your app. This opens your app to the **Build** section of the app.
2. On the **Intents** page, select **Add prebuilt intent** from the toolbar above the intents list.
3. Select the **Utilities.Cancel** intent from the pop-up dialog.



4. Select the **Done** button.

Train and test

1. After the intent is added, train the app by selecting **Train** in the top, right toolbar.
2. Test the new intent by selecting **Test** in the right toolbar.
3. In the textbox, enter utterances for canceling:

TEST UTTERANCE	PREDICTION SCORE
I want to cancel my flight.	0.67
Cancel the purchase.	0.52
Cancel the meeting.	0.56

The screenshot shows the Microsoft Bot Framework's Test panel. At the top, there are buttons for "Test" (highlighted), "Start over", and "Batch testing panel". Below is a text input field with placeholder "Type a test utterance". Three utterances are listed in a scrollable list:

- "cancel the meeting." (Utilities.Cancel (0.56))
- "cancel the purchase." (Utilities.Cancel (0.52))
- "i want to cancel my flight." (Utilities.Cancel (0.67))

Each entry has an "Inspect" button to its right.

Next steps

[Prebuilt entities](#)

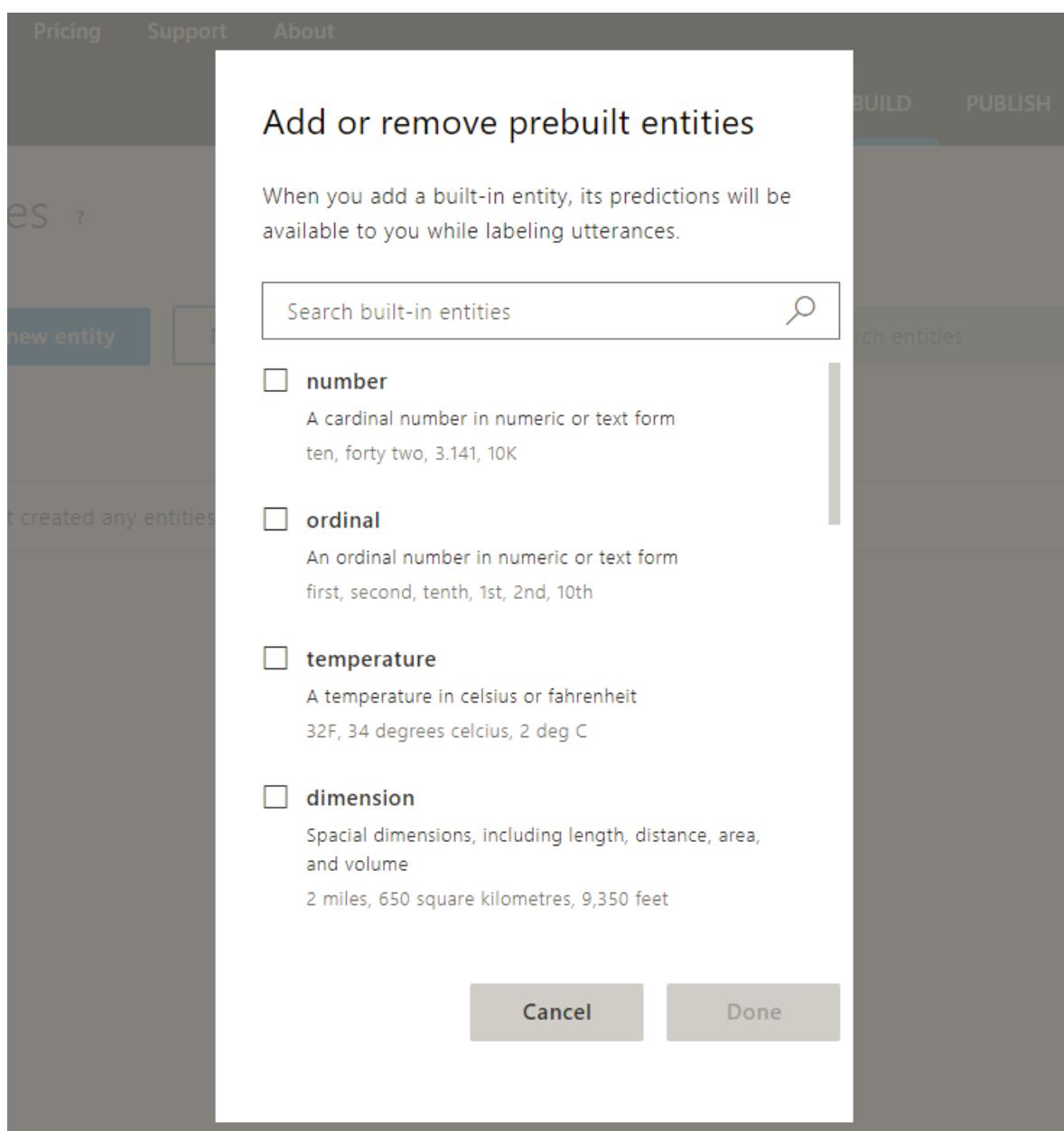
Prebuilt entities to recognize common data types

8/9/2019 • 2 minutes to read • [Edit Online](#)

LUIS includes a set of prebuilt entities for recognizing common types of information, like dates, times, numbers, measurements, and currency.

Add a prebuilt entity

1. Open your app by clicking its name on **My Apps** page, and then click **Entities** in the left side.
2. On the **Entities** page, click **Add prebuilt entity**.
3. In **Add prebuilt entities** dialog box, select the datetimeV2 prebuilt entity.



4. Select **Done**.

Publish the app

The easiest way to view the value of a prebuilt entity is to query from the published endpoint.

1. In the top toolbar, select **Publish**. Publish to **Production**.
2. When the green success notification appears, select the **Refer to the list of endpoints** link to see the endpoints.
3. Select an endpoint. A new browser tab opens to that endpoint. Keep the browser tab open and continue to the **Test** section.

Test

After the entity is added, you do not need to train the app.

Test the new intent at the endpoint by added a value for the **q** parameter. Use the following table for suggested utterances for **q**:

TEST UTTERANCE	ENTITY VALUE
Book a flight tomorrow	2018-10-19
cancel the appointment on March 3	LUIS returned the most recent March 3 in the past (2018-03-03) and March 3 in the future (2019-03-03) because the utterance didn't specify a year.
Schedule a meeting at 10am	10:00:00

Marking entities containing a prebuilt entity token

If you have text, such as `HH-1234`, that you want to mark as a custom entity *and* you have **Prebuilt Number** added to the model, you won't be able to mark the custom entity in the LUIS portal. You can mark it with the API.

In order to mark this type of token, where part of it is already marked with a prebuilt entity, remove the prebuilt entity from the LUIS app. You don't need to train the app. Then mark the token with your own custom entity. Then add the prebuilt entity back to the LUIS app.

For another example, consider the utterance as a list of class preferences:

`I want first year spanish, second year calculus, and fourth year english lit.` If the LUIS app has the Prebuild Ordinal added, `first`, `second`, and `fourth` will already be marked with ordinals. If you want to capture the ordinal and the class, you can create a composite entity and wrap it around the Prebuilt Ordinal and the custom entity for class name.

Next steps

[Prebuilt entity reference](#)

Build a LUIS app programmatically using Node.js

8/9/2019 • 13 minutes to read • [Edit Online](#)

Luis provides a programmatic API that does everything that the [LUIS](#) website does. This can save time when you have pre-existing data and it would be faster to create a LUIS app programmatically than by entering information by hand.

Prerequisites

- Sign in to the [LUIS](#) website and find your [authoring key](#) in Account Settings. You use this key to call the Authoring APIs.
- If you don't have an Azure subscription, create a [free account](#) before you begin.
- This tutorial starts with a CSV for a hypothetical company's log files of user requests. Download it [here](#).
- Install the latest Nodejs with NPM. Download it from [here](#).
- **[Recommended]** Visual Studio Code for IntelliSense and debugging, download it from [here](#) for free.

All of the code in this tutorial is available on the [Azure-Samples Language Understanding GitHub repository](#).

Map preexisting data to intents and entities

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

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

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

You see that the **RequestType** column could be intents, and the **Request** column shows an example utterance. The other fields could be entities if they occur in the utterance. Because there are intents, entities, and example utterances, you have the requirements for a simple, sample app.

Steps to generate a LUIS app from non-LUIS data

To generate a new LUIS app from the CSV file:

- Parse the data from the CSV file:

- Convert to a format that you can upload to LUIS using the Authoring API.
- From the parsed data, gather information about intents and entities.
- Make authoring API calls to:
 - Create the app.
 - Add intents and entities that were gathered from the parsed data.
 - Once you have created the LUIS app, you can add the example utterances from the parsed data.

You can see this program flow in the last part of the `index.js` file. Copy or [download](#) this code and save it in `index.js`.

```

var path = require('path');

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

// Change these values
const LUIS_authoringKey = "YOUR_AUTHORING_KEY";
const LUIS_appName = "Sample App - build from IoT csv file";
const LUIS_appCulture = "en-us";
const LUIS_versionId = "0.1";

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

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

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

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

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

/* add entities parameters */
var configAddEntities = {
  LUIS_subscriptionKey: LUIS_authoringKey,

```

```

    LUIS_appId: LUIS_appId,
    LUIS_versionId: LUIS_versionId,
    entityList: entities,
    uri: "https://westus.api.cognitive.microsoft.com/luis/api/v2.0/apps/{appId}/versions/{versionId}/entities"
};

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

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

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

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

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

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

```

Parse the CSV

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

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

```

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

```

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

```

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

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

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

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

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

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

var utterance = function (rowAsString) {

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

```

```

        if (!rowAsString) return json;

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

const convert = async (config) => {

    try {
        var i = 0;

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

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

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

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

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

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

                return;
            }

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

        }).then(() => {
            console.log("intents: " + JSON.stringify(listOfIntents(utterances)));
            console.log("entities: " + JSON.stringify(listOfEntities(utterances)));
            myOutFile.write(JSON.stringify({ "converted_date": new Date().toLocaleString(), "utterances": utterances }));
        })
    }
}

```

```

    ...
    myOutFile.end();
    console.log("parse done");
    console.log("JSON file should contain utterances. Next step is to create an app with the intents
and entities it found.");
}

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

});

} catch (err) {
    throw err;
}

}

module.exports = convert;

```

Create the LUIS app

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

```

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

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

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

    try {

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

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

        let results = await createAppPromise;

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

```

```

        return appId;

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

}

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

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

    } catch (err) {
        throw err;
    }
}

module.exports = createApp;

```

Add intents

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

```

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

// time delay between requests
const delayMS = 1000;

// retry recount
const maxRetry = 5;

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

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

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

            // JSON for the request body
            var jsonBody = {

```

```

        "name": config.intentName,
    };

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

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

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

},
this);

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

}

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

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

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

module.exports = addIntents;

```

Add entities

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

```

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

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

// time delay between requests
const delayMS = 1000;

// retry recount

```

```

const maxRetry = 5;

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

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

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

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

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

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

}

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

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

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

module.exports = addEntities;

```

Add utterances

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

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

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

// time delay between requests
const delayMS = 500;

// retry recount
const maxRetry = 5;

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

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

    try{

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

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

        var uploadPromises = [];

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

            uploadPromises.push(pagePromise);
        })

        //execute promise array
    })
}
```

```

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

console.log("upload done");

} catch(err){
    throw err;
}

}

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

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

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

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

        var currentStart = currentPage * maxItems;
        var currentEnd = currentStart + maxItems;
        var pagedBatch = batch.slice(currentStart,currentEnd);

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

        pages.push(pagedBatch);

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

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

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

module.exports = upload;

```

Run the code

Install Node.js dependencies

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

```
> npm install
```

Change Configuration Settings

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

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

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

Run the script

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

```
> node index.js
```

or

```
> npm start
```

Application progress

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

```
> node index.js
intents: ["TurnOn", "TurnOff", "Dim", "Other"]
entities: ["Operation", "Device", "Room"]
parse done
JSON file should contain utterances. Next step is to create an app with the intents and entities it found.
Called createApp, created app with ID 314b306c-0033-4e09-92ab-94fe5ed158a2
Called addIntents for intent named TurnOn.
Called addIntents for intent named TurnOff.
Called addIntents for intent named Dim.
Called addIntents for intent named Other.
Results of all calls to addIntent = [{"response": "e7eaf224-8c61-44ed-a6b0-2ab4dc56f1d0"}, {"response": "a8a17efd-f01c-488d-ad44-a31a818cf7d7"}, {"response": "bc7c32fc-14a0-4b72-bad4-d345d807f965"}, {"response": "727a8d73-cd3b-4096-bc8d-d7cfba12eb44"}]
called addEntity for entity named Operation.
called addEntity for entity named Device.
called addEntity for entity named Room.
Results of all calls to addEntity= [{"response": "6a7e914f-911d-4c6c-a5bc-377afdcce4390"}, {"response": "56c35237-593d-47f6-9d01-2912fa488760"}, {"response": "f1dd440c-2ce3-4a20-a817-a57273f169f3"}]
retrying add examples...

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

Open the LUIS app

Once the script completes, you can sign in to [LUIS](#) and see the LUIS app you created under **My Apps**. You should be able to see the utterances you added under the **TurnOn**, **TurnOff**, and **None** intents.

The screenshot shows the LUIS Intents page for the 'HomeAutomation.TurnOn' intent. On the left, there's a sidebar with 'App Assets' sections for 'Intents' (selected), 'Entities', and 'Improve app performance' (Review endpoint utterances, Phrase lists, Patterns). Below that is a 'PREVIEW' section for 'Prebuilt Domains'. The main area has a title 'HomeAutomation.TurnOn' with a pencil icon and a 'Delete Intent' button. A text input field says 'Type about 5 examples of what a user might say and hit Enter'. Below it are 'Entity filters' and a 'Show All' switch. A 'Labeled intent?' link is also present. The list of labeled utterances includes:

- change HomeAutomation.Operation to seventy two degrees
- HomeAutomation.Device HomeAutomation.Operation please
- play HomeAutomation.Device
- turn HomeAutomation.Device on 70 .
- HomeAutomation.Device please
- make some HomeAutomation.Device

Next steps

[Test and train your app in LUIS website](#)

Additional resources

This sample application uses the following LUIS APIs:

- [create app](#)
- [add intents](#)
- [add entities](#)
- [add utterances](#)

Use a list entity to increase entity detection

8/9/2019 • 9 minutes to read • [Edit Online](#)

This tutorial demonstrates the use of a [list entity](#) to increase entity detection. List entities do not need to be labeled as they are an exact match of terms.

In this tutorial, you learn how to:

- Create a list entity
- Add normalized values and synonyms
- Validate improved entity identification

Prerequisites

- Latest [Node.js](#)
- [HomeAutomation LUIS app](#). If you do not have the Home Automation app created, create a new app, and add the Prebuilt Domain **HomeAutomation**. Train and publish the app.
- [AuthoringKey](#), [EndpointKey](#) (if querying many times), app ID, version ID, and [region](#) for the LUIS app.

TIP

If you do not already have a subscription, you can register for a [free account](#).

All of the code in this tutorial is available on the [Azure-Samples GitHub repository](#).

Use HomeAutomation app

The HomeAutomation app gives you control of devices such as lights, entertainment systems, and environment controls such as heating and cooling. These systems have several different names that can include Manufacturer names, nicknames, acronyms, and slang.

One system that has many names across different cultures and demographics is the thermostat. A thermostat can control both cooling and heating systems for a house or building.

Ideally the following utterances should resolve to the Prebuilt entity **HomeAutomation.Device**:

#	UTTERANCE	ENTITY IDENTIFIED	SCORE
1	turn on the ac	HomeAutomation.Device - "ac"	0.8748562
2	turn up the heat	HomeAutomation.Device - "heat"	0.784990132
3	make it colder		

The first two utterances map to different devices. The third utterance, "make it colder", doesn't map to a device but instead requests a result. LUIS doesn't know that the term, "colder", means the thermostat is the requested device. Ideally, LUIS should resolve all of these utterances to the same device.

Use a list entity

The HomeAutomation.Device entity is great for a small number of devices or with few variations of the names. For an office building or campus, the device names grow beyond the usefulness of the HomeAutomation.Device entity.

A **list entity** is a good choice for this scenario because the set of terms for a device in a building or campus is a known set, even if it is a huge set. By using a list entity, LUIS can receive any possible value in the set for the thermostat, and resolve it down to just the single device "thermostat".

This tutorial is going to create an entity list with the thermostat. The alternative names for a thermostat in this tutorial are:

ALTERNATIVE NAMES FOR THERMOSTAT

ac

a/c

a-c

heater

hot

hotter

cold

colder

If LUIS needs to determine a new alternative often, then a [phrase list](#) is a better answer.

Create a list entity

Create a Node.js file and copy the following code into it. Change the authoringKey, appId, versionId, and region values.

```
/*-----  
This template demonstrates how to use list entities.  
. For a complete walkthrough, see the article at  
https://aka.ms/luis-tutorial-list-entity  
-----*/  
  
// NPM Dependencies  
var request = require('request-promise');  
  
// To run this sample, change these constants.  
  
// Authoring/starter key, available in www.luis.ai under Account Settings  
const authoringKey = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";  
  
// ID of your LUIS app to which you want to add an utterance  
const appId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";  
  
// Version number of your LUIS app  
const versionId = "0.1";  
  
// Region of app
```

```

const region = "westus";

// Construct HTTP uri
const uri=
`https://${region}.api.cognitive.microsoft.com/luis/api/v2.0/apps/${appId}/versions/${versionId}/closedlists`;

// create list entity
var addListEntity = async () => {

    try {

        // LUIS model definition for list entity
        var body = {
            "name": "DevicesList",
            "sublists":
            [
                {
                    "canonicalForm": "Thermostat",
                    "list": [ "ac", "a/c", "a-c","heater","hot","hotter","cold","colder" ]
                }
            ]
        }

        // HTTP request
        var myHTTPRequest = request({
            uri: uri,
            method: 'POST',
            headers: {
                'Ocp-Apim-Subscription-Key': authoringKey
            },
            json: true,
            body: body
        });

        return await myHTTPRequest;

    } catch (err) {
        throw err;
    }
}

// MAIN
addListEntity().then( (response) => {
    // return GUID of list entity model
    console.log(response);
}).catch(err => {
    console.log(`Error adding list entity: ${err.message}`);
});

```

Use the following command to install the NPM dependencies and run the code to create the list entity:

```
npm install && node add-entity-list.js
```

The output of the run is the ID of the list entity:

```
026e92b3-4834-484f-8608-6114a83b03a6
```

Train the model

Train LUIS in order for the new list to affect the query results. Training is a two-part process of training, then checking status if the training is done. An app with many models may take a few moments to train. The following code trains the app then waits until the training is successful. The code uses a wait-and-retry strategy to avoid the 429 "Too many requests" error.

Create a Node.js file and copy the following code into it. Change the authoringKey, appId, versionId, and region values.

```
/*
----- This template demonstrates how to use list entities.
.
For a complete walkthrough, see the article at
https://aka.ms/luis-tutorial-list-entity
-----*/



// NPM Dependencies
var request = require('request-promise');
const Promise = require("bluebird");
const promiseRetry = require('promise-retry');
const promiseDelay = require('sleep-promise');

// To run this sample, change these constants.

// Authoring/starter key, available in www.luis.ai under Account Settings
const authoringKey = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";

// ID of your LUIS app to which you want to add an utterance
const appId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

// Version number of your LUIS app
const versionId = "0.1";

// Region of app
const region = "westus";

// Construct HTTP uri
const uri=
`https://${region}.api.cognitive.microsoft.com/luis/api/v2.0/apps/${appId}/versions/${versionId}/train`;

// times to request training status
const retryCount = 10;

// wait time between training status requests
const retryInterval = 2000;

// current count of retries
let count = 0;

// determine if complete model, all entities, is trained
var isTrained = (trainingStatus) => {

    var untrainedModels = trainingStatus.filter(model => {
        return model.details.status == 'Fail' || model.details.status == 'InProgress';
    });
    return (untrainedModels.length==0);
}

// request training (POST)
var train = async (config) => {

    try {

        var body = undefined;

        // Add an utterance
        var myHTTPRequest = request({
            uri: uri,
            method: 'POST',
            headers: {
                'Ocp-Apim-Subscription-Key': authoringKey
            },
-----
```

```

        json: true,
        body: body
    });

    return await myHTTPRequest;

} catch (err) {
    throw err;
}
}

// request training status (GET)
var getTrainStatus = async (config) => {

try {

    var body = undefined;

    var myHTTPRequest = request({
        uri: uri,
        method: 'GET',
        headers: {
            'Ocp-Apim-Subscription-Key': authoringKey
        }
    });

    return await myHTTPRequest;

} catch (err) {
    throw err;
}
}

// retry if not trained
var manageRetries = (client) => {

    count += 1;

    return promiseRetry((retry, number) => {

        return promiseDelay(retryInterval)
            .then( () => {
                return getTrainStatus();
            }).then(response => {

                // response is a string
                // convert it to an array of JSON
                response = JSON.parse(response);

                // 2xx http response
                let trained = isTrained(response);

                console.log(number + " trained = " + trained);

                if (count < retryCount && !trained) retry("not trained");

                return response;
            })
            .catch((err) => {
                throw err;
            });
    });
}

// wait until model is trained -- may take a few moments
var waitUntilTrained = async () => {
    let setTraining = await train();
    let trained = await manageRetries();
    return trained;
}

// MAIN

```

```
waitUntilTrained().then( (response) => {
    console.log(response);
}).catch(err => {
    console.log(`Error adding list entity: ${err.message}`);
});
```

Use the following command to run the code to train the app:

```
node train.js
```

The output of the run is the status of each iteration of the training of the LUIS models. The following execution required only one check of training:

```
1 trained = true
[ { modelId: '2c549f95-867a-4189-9c35-44b95c78b70f',
  details: { statusId: 2, status: 'UpToDate', exampleCount: 45 } },
{ modelId: '5530e900-571d-40ec-9c78-63e66b50c7d4',
  details: { statusId: 2, status: 'UpToDate', exampleCount: 45 } },
{ modelId: '519faa39-ae1a-4d98-965c-abff6f743fe6',
  details: { statusId: 2, status: 'UpToDate', exampleCount: 45 } },
{ modelId: '9671a485-36a9-46d5-aacd-b16d05115415',
  details: { statusId: 2, status: 'UpToDate', exampleCount: 45 } },
{ modelId: '9ef7d891-54ab-48bf-8112-c34dc75d5e2',
  details: { statusId: 2, status: 'UpToDate', exampleCount: 45 } },
{ modelId: '8e16a660-8781-4abf-bf3d-f296ebe1bf2d',
  details: { statusId: 2, status: 'UpToDate', exampleCount: 45 } } ]
```

Publish the model

Publish so the list entity is available from the endpoint.

Create a Node.js file and copy the following code into it. Change the endpointKey, appId, and region values. You can use your authoringKey if you do not plan to call this file beyond your quota limit.

```

/*
-----.
This template demonstrates how to use list entities.
.
For a complete walkthrough, see the article at
https://aka.ms/luis-tutorial-list-entity
-----*/

```

```

// NPM Dependencies
var request = require('request-promise');

// To run this sample, change these constants.

// Authoring/starter key, available in www.luis.ai under Account Settings
const authoringKey = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";

// ID of your LUIS app to which you want to add an utterance
const appId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

// Version number of your LUIS app
const versionId = "0.1";

// Region of app
const region = "westus";

// Construct HTTP uri
const uri= `https://${region}.api.cognitive.microsoft.com/luis/api/v2.0/apps/${appId}/publish`;

// publish
var publish = async () => {

    try {

        // LUIS publish definition
        var body = {
            "versionId": "0.1",
            "isStaging": false,
            "region": "westus"
        };

        // HTTP request
        var myHTTPRequest = request({
            uri: uri,
            method: 'POST',
            headers: {
                'Ocp-Apim-Subscription-Key': authoringKey
            },
            json: true,
            body: body
        });

        return await myHTTPRequest;

    } catch (err) {
        throw err;
    }
}

// MAIN
publish().then( (response) => {
    // return GUID of list entity model
    console.log(response);
}).catch(err => {
    console.log(`Error adding list entity: ${err.message}`);
});

```

Use the following command to run the code to query the app:

```
node publish.js
```

The following output includes the endpoint url for any queries. Real JSON results would include the real appId.

```
{  
    versionId: null,  
    isStaging: false,  
    endpointUrl: 'https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>',  
    region: null,  
    assignedEndpointKey: null,  
    endpointRegion: 'westus',  
    publishedDateTime: '2018-01-29T22:17:38Z' }  
}
```

Query the app

Query the app from the endpoint to prove that the list entity helps LUIS determine the device type.

Create a Node.js file and copy the following code into it. Change the endpointKey, appId, and region values. You can use your authoringKey if you do not plan to call this file beyond your quota limit.

```

/*
----- This template demonstrates how to use list entities.
.
For a complete walkthrough, see the article at
https://aka.ms/luis-tutorial-list-entity
-----*/

```

```

// NPM Dependencies
var argv = require('yargs').argv;
var request = require('request-promise');

// To run this sample, change these constants.

// endpointKey key - if using a few times - use authoring key, otherwise use endpoint key
const endpointKey = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" || argv.endpointKey;

// ID of your LUIS app to which you want to add an utterance
const appId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx" || argv.appId;

// Region of app
const region = "westus" || argv.region;

// Get query from command line
// you do NOT need to add quotes around query phrase
// example: node query.js turn on the lights
// q: "turn on the lights"
// verbose: true means results return all intents, not just top intent
const q = argv._.join(" ");
const uri= `https://${region}.api.cognitive.microsoft.com/luis/v2.0/apps/${appId}?q=${q}&verbose=true`;

// query endpoint with endpoint key
var query = async () => {

    try {

        var myHTTPRequest = request({
            uri: uri,
            method: 'GET',
            headers: {
                'Ocp-Apim-Subscription-Key': endpointKey
            }
        });

        return await myHTTPRequest;

    } catch (err) {
        throw err;
    }
}

// MAIN
query().then( (response) => {
    // return verbose results (all intents, all entities)
    console.log(response);
}).catch(err => {
    console.log(`Error querying: ${err.message}`);
});

```

Use the following command to run the code and query the app:

```
node train.js
```

The output is the query results. Because the code added the **verbose** name/value pair to the query string, the output includes all intents and their scores:

```
{
  "query": "turn up the heat",
  "topScoringIntent": {
    "intent": "HomeAutomation.TurnOn",
    "score": 0.139018849
  },
  "intents": [
    {
      "intent": "HomeAutomation.TurnOn",
      "score": 0.139018849
    },
    {
      "intent": "None",
      "score": 0.120624863
    },
    {
      "intent": "HomeAutomation.TurnOff",
      "score": 0.06746891
    }
  ],
  "entities": [
    {
      "entity": "heat",
      "type": "HomeAutomation.Device",
      "startIndex": 12,
      "endIndex": 15,
      "score": 0.784990132
    },
    {
      "entity": "heat",
      "type": "DevicesList",
      "startIndex": 12,
      "endIndex": 15,
      "resolution": {
        "values": [
          "Thermostat"
        ]
      }
    }
  ]
}
```

The specific device of **Thermostat** is identified with a result-oriented query of "turn up the heat". Since the original HomeAutomation.Device entity is still in the app, you can see its results as well.

Try the other two utterances to see that they are also returned as a thermostat.

#	UTTERANCE	ENTITY	TYPE	VALUE
1	turn on the ac	ac	DevicesList	Thermostat
2	turn up the heat	heat	DevicesList	Thermostat
3	make it colder	colder	DevicesList	Thermostat

Next steps

You can create another List entity to expand device locations to rooms, floors, or buildings.

Correct misspelled words with Bing Spell Check

8/9/2019 • 2 minutes to read • [Edit Online](#)

You can integrate your LUIS app with [Bing Spell Check API V7](#) to correct misspelled words in utterances before LUIS predicts the score and entities of the utterance.

Create first key for Bing Spell Check V7

Your [first Bing Spell Check API v7 key](#) is free.

 [Bing Spell Check API v7](#)

Help users correct spelling errors, recognize the difference among names, brand names, and slang, as well as understand homophones as they're typing.

1,000 transactions per month, up to 1 per second. Trial keys expire after a 30 day period.

Please create a free Azure Account to get free access to Cognitive Services or explore purchase options for production use.

[Start your free Azure account now >](#)

[Quick-start Guide >](#)

Create Endpoint key

If your free key expired, create an endpoint key.

1. Log in to the [Azure portal](#).
2. Select **Create a resource** in the top left corner.
3. In the search box, enter .

NAME	PUBLISHER	CATEGORY
 Bing Spell Check v7 API	Microsoft	AI + Cognitive Services

4. Select the service.
5. An information panel appears to the right containing information including the Legal Notice. Select **Create** to begin the subscription creation process.
6. In the next panel, enter your service settings. Wait for service creation process to finish.

* Name
LUIS-Spell-Check ✓

* Subscription
Pay-As-You-Go

* Pricing tier ([View full pricing details](#))
S1 (100 Calls per second)

* Resource group
 Create new Use existing
luis-spell-check ✓

* Resource group location [?](#)
West US

* I confirm I have read and understood the notice below.
Microsoft will use data you send to Bing Search Services to improve Microsoft products and services. Where you send personal data to this service, 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 this Service.
Please refer to the [Online Services Terms](#) for details. Microsoft offers [policy controls](#) that may be used to disable new deployments.

Pin to dashboard

Create [Automation options](#)

7. Select **All resources** under the **Favorites** title on the left side navigation.

8. Select the new service. Its type is **Cognitive Services** and the location is **global**.

9. In the main panel, select **Keys** to see your new keys.

Home > All resources > LUIS-Spell-Check - Quick start

LUIS-Spell-Check - Quick start Cognitive Services

Search (Ctrl+ /)

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems

RESOURCE MANAGEMENT

- Keys
- Quick start
- Pricing tier
- Billing By Subscription
- Properties
- Locks
- Automation script

Congratulations! Your keys are ready.
Now explore the Quickstart guidance to get up and running with Bing Spell Check API.

1 Grab your keys
Every call to the Bing Spell Check API requires a subscription key. This key needs to be either passed through a query string parameter or specified in the request header. You can find your keys in the API resource 'Overview' or 'Keys' from the left menu.
Keys

2 Make an API call to endpoint <https://api.cognitive.microsoft.com/bing/v7.0/spellcheck>
Get in-depth information about each properties and methods of the API. Test your keys with the built-in testing console without writing a single line of code. Once you have the API running, you can check your consumption and the API health on Azure portal in your API 'Overview'.
[Bing Spell Check API reference](#)
[Realtime API usages](#)
[API metrics alert](#)
[Billing by subscription](#)
[Resource health status](#)

3 Enjoy coding
Learn more about the features, tutorials, developer tools, examples and how-to guidance to speed up.
[Documentation](#)

10. Copy the first key. You only need one of the two keys.

Using the key in LUIS test panel

There are two places in LUIS to use the key. The first is in the [test panel](#). The key isn't saved into LUIS but instead is a session variable. You need to set the key every time you want the test panel to apply the Bing Spell Check API v7 service to the utterance. See [instructions](#) in the test panel for setting the key.

Adding the key to the endpoint URL

The endpoint query needs the key passed in the query string parameters for each query you want to apply spelling correction. You may have a chatbot that calls LUIS or you may call the LUIS endpoint API directly. Regardless of how the endpoint is called, each and every call must include the required information for spelling corrections to work properly.

The endpoint URL has several values that need to be passed correctly. The Bing Spell Check API v7 key is just another one of these. You must set the **spellCheck** parameter to true and you must set the value of **bing-spell-check-subscription-key** to the key value:

```
https://{{region}}.api.cognitive.microsoft.com/luis/v2.0/apps/{{appID}}?subscription-key={{luisKey}}&spellCheck=**true**&bing-spell-check-subscription-key=**{{bingKey}}**&verbose=true&timezoneOffset=0&q={{utterance}}
```

Send misspelled utterance to LUIS

1. In a web browser, copy the preceding string and replace the `region`, `appId`, `luisKey`, and `bingKey` with your own values. Make sure to use the endpoint region, if it is different from your publishing [region](#).
2. Add a misspelled utterance such as "How far is the mountainn?". In English, `mountain`, with one `n`, is the correct spelling.
3. Select enter to send the query to LUIS.
4. LUIS responds with a JSON result for `How far is the mountain?`. If Bing Spell Check API v7 detects a misspelling, the `query` field in the LUIS app's JSON response contains the original query, and the `alteredQuery` field contains the corrected query sent to LUIS.

```
{  
  "query": "How far is the mountainn?",  
  "alteredQuery": "How far is the mountain?",  
  "topScoringIntent": {  
    "intent": "Concierge",  
    "score": 0.183866  
  },  
  "entities": []  
}
```

Ignore spelling mistakes

If you don't want to use the Bing Spell Check API v7 service, you can label utterances that have spelling mistakes so that LUIS can learn proper spelling as well as typos. This option requires more labeling effort than using a spell checker.

Publishing page

The [publishing](#) page has an **Enable Bing spell checker** checkbox. This is a convenience to create the key and understand how the endpoint URL changes. You still have to use the correct endpoint parameters in order to have spelling corrected for each utterance.

[Learn more about example utterances](#)

Tutorial: Recognize intents from speech using the Speech SDK for C#

7/5/2019 • 11 minutes to read • [Edit Online](#)

The Cognitive Services [Speech SDK](#) integrates with the [Language Understanding service \(LUIS\)](#) to provide **intent recognition**. An intent is something the user wants to do: book a flight, check the weather, or make a call. The user can use whatever terms feel natural. Using machine learning, LUIS maps user requests to the intents you have defined.

NOTE

A LUIS application defines the intents and entities you want to recognize. It's separate from the C# application that uses the Speech service. In this article, "app" means the LUIS app, while "application" means the C# code.

In this tutorial, you use the Speech SDK to develop a C# console application that derives intents from user utterances through your device's microphone. You'll learn how to:

- Create a Visual Studio project referencing the Speech SDK NuGet package
- Create a speech config and get an intent recognizer
- Get the model for your LUIS app and add the intents you need
- Specify the language for speech recognition
- Recognize speech from a file
- Use asynchronous, event-driven continuous recognition

Prerequisites

Be sure you have the following before you begin this tutorial.

- A LUIS account. You can get one for free through the [LUIS portal](#).
- Visual Studio 2017 (any edition).

LUIS and speech

LUIS integrates with the Speech Services to recognize intents from speech. You don't need a Speech Services subscription, just LUIS.

LUIS uses two kinds of keys:

KEY TYPE	PURPOSE
authoring	lets you create and modify LUIS apps programmatically
endpoint	authorizes access to a particular LUIS app

The endpoint key is the LUIS key needed for this tutorial. This tutorial uses the example Home Automation LUIS app, which you can create by following [Use prebuilt Home automation app](#). If you have created a LUIS app of your own, you can use it instead.

When you create a LUIS app, a starter key is automatically generated so you can test the app using text queries.

This key does not enable the Speech Services integration and won't work with this tutorial. You must create a LUIS resource in the Azure dashboard and assign it to the LUIS app. You can use the free subscription tier for this tutorial.

After creating the LUIS resource in the Azure dashboard, log into the [LUIS portal](#), choose your application on the My Apps page, then switch to the app's Manage page. Finally, click **Keys and Endpoints** in the sidebar.

The screenshot shows the LUIS portal interface. At the top, there is a navigation bar with links: Language Understanding, My apps, Docs, Pricing, Support, About, HomeAutomation (V 0.1)~, DASHBOARD, BUILD, MANAGE, and Train. The MANAGE tab is currently selected. On the left, a sidebar menu has 'Keys and Endpoints' selected and highlighted in blue. The main content area is titled 'Keys and Endpoint settings' with a sub-section titled 'Authoring Key'. Below this, there is a file icon followed by a progress bar.

On the Keys and Endpoint settings page:

1. Scroll down to the Resources and Keys section and click **Assign resource**.
2. In the **Assign a key to your app** dialog, choose the following:
 - Choose Microsoft as the Tenant.
 - Under Subscription Name, choose the Azure subscription that contains the LUIS resource you want to use.
 - Under Key, choose the LUIS resource that you want to use with the app.

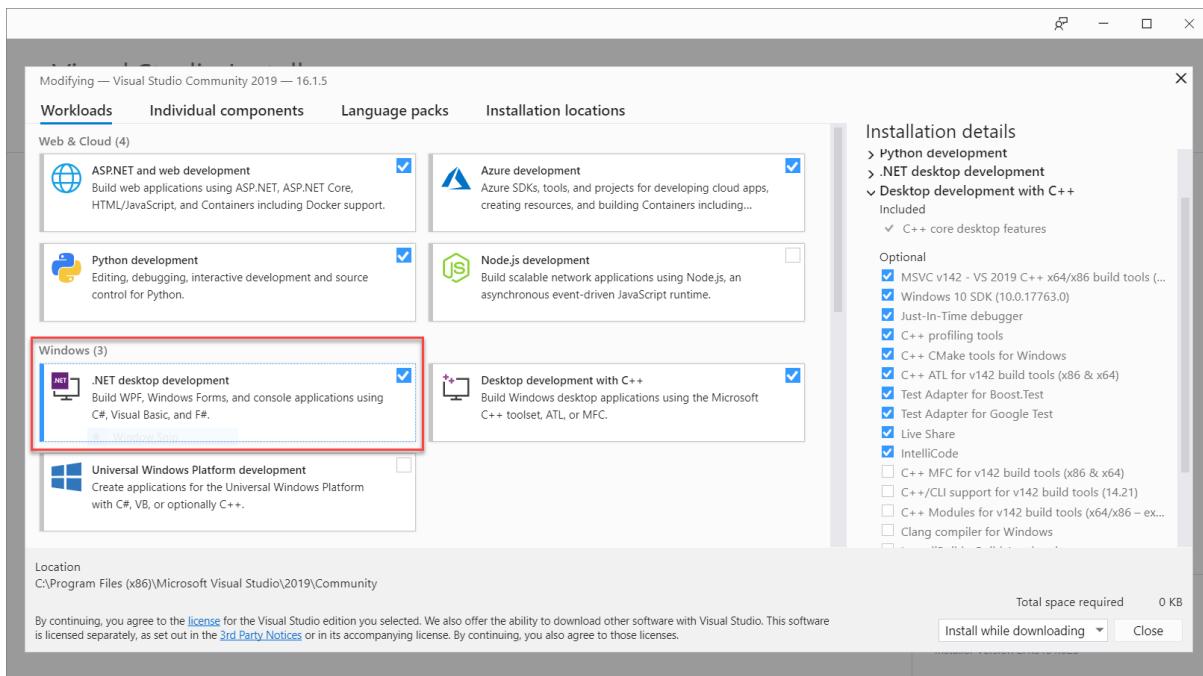
In a moment, the new subscription appears in the table at the bottom of the page. Click the icon next to a key to copy it to the clipboard. (You may use either key.)

Assign resource				
Resource Name	Region	Time zone	Key 1	Key 2
Starter_Key	westus	GMT -6:00	a38608 ...	
luis	westus	GMT -6:00	ad544d ...	0e6269 ...

Create a speech project in Visual Studio

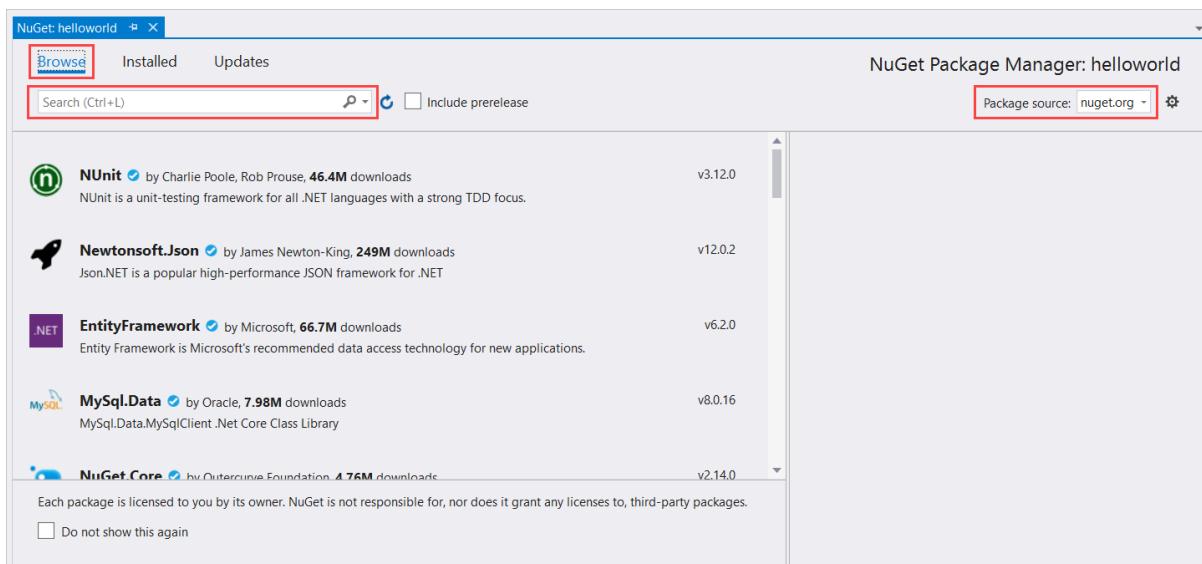
1. Open Visual Studio 2019.
2. In the Start window, select **Create a new project**.
3. Select **Console App (.NET Framework)**, and then select **Next**.
4. In **Project name**, enter `helloworld`, and then select **Create**.
5. From the menu bar in Visual Studio, select **Tools > Get Tools and Features**, and check whether the **.NET desktop development** workload is available. If the workload hasn't been installed, mark the checkbox, then select **Modify** to start the installation. It may take a few minutes to download and install.

If the checkbox next to **.NET desktop development** is selected, you can close the dialog box now.

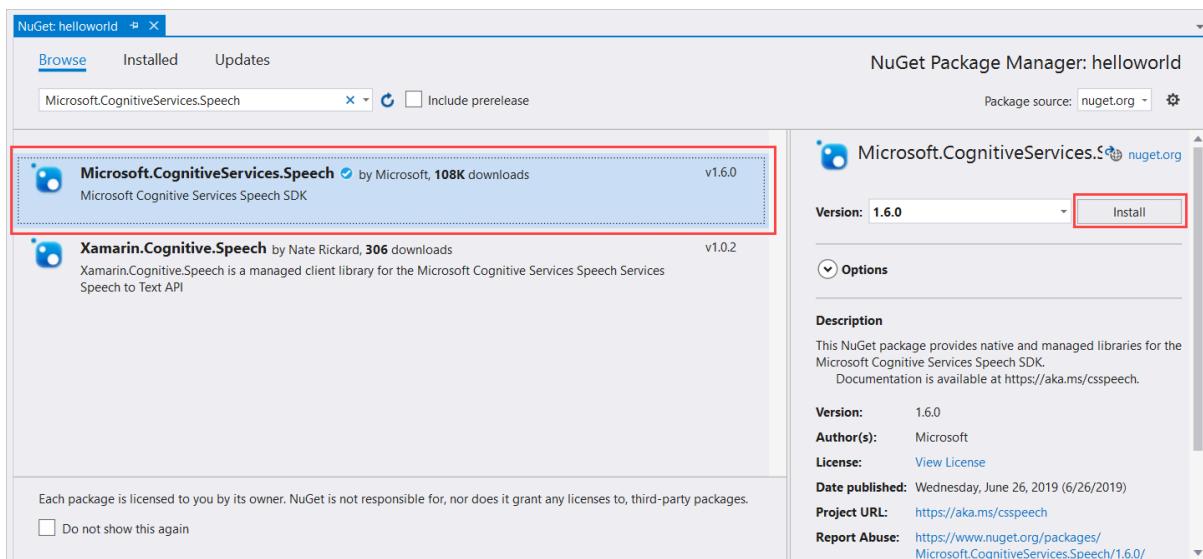


The next step is to install the [Speech SDK NuGet package](#), so you can reference it in the code.

1. In the Solution Explorer, right-click `helloworld`, and then select **Manage NuGet Packages** to show the NuGet Package Manager.



2. In the upper-right corner, find the **Package Source** drop-down box, and make sure that **nuget.org** is selected.
3. In the upper-left corner, select **Browse**.
4. In the search box, type `Microsoft.CognitiveServices.Speech` package and press Enter.
5. Select `Microsoft.CognitiveServices.Speech`, and then select **Install** to install the latest stable version.

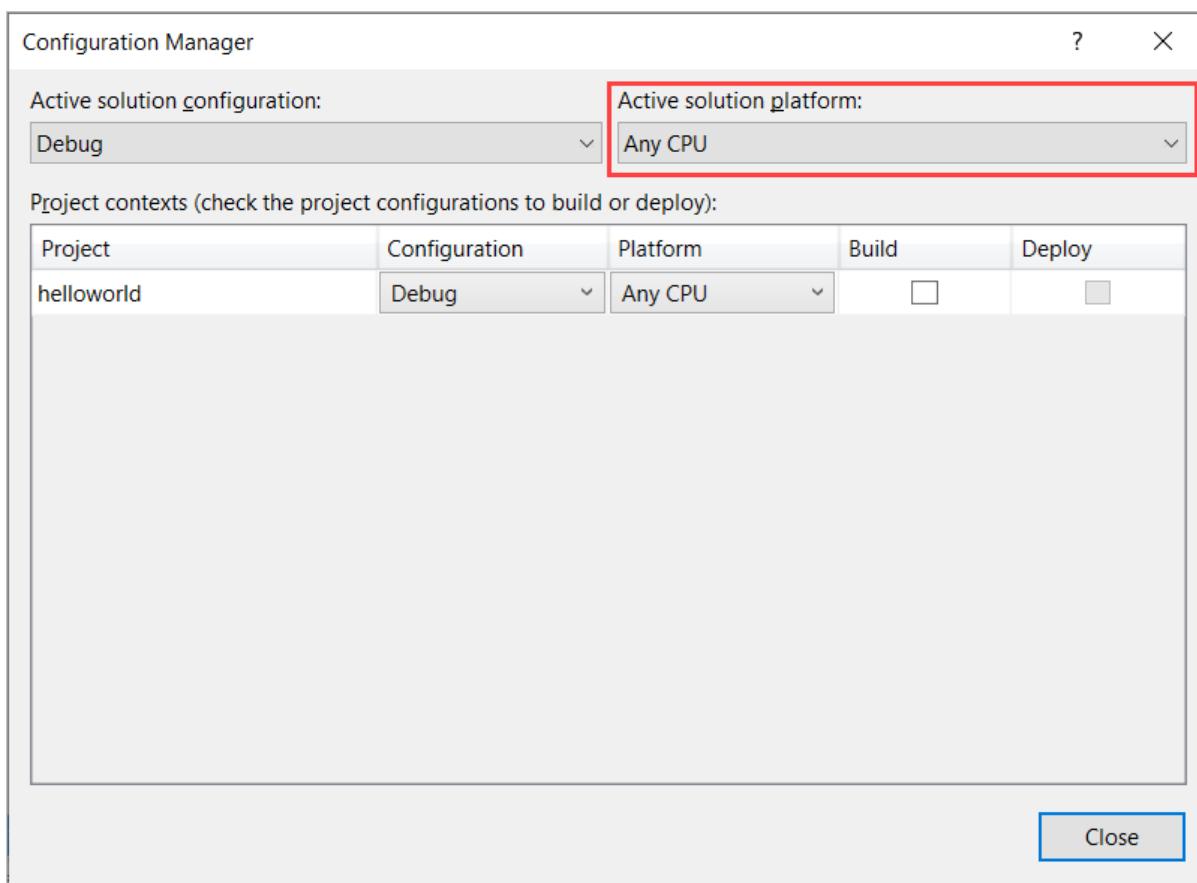


6. Accept all agreements and licenses to start the installation.

After the package is installed, a confirmation appears in the **Package Manager Console** window.

Now, to build and run the console application, create a platform configuration matching your computer's architecture.

1. From the menu bar, select **Build > Configuration Manager**. The **Configuration Manager** dialog box appears.



2. In the **Active solution platform** drop-down box, select **New**. The **New Solution Platform** dialog box appears.
3. In the **Type or select the new platform** drop-down box:
 - If you're running 64-bit Windows, select **x64**.
 - If you're running 32-bit Windows, select **x86**.

4. Select **OK** and then **Close**.

Add the code

Open the file `Program.cs` in the Visual Studio project and replace the block of `using` statements at the beginning of the file with the following declarations.

```
using System;
using System.Threading.Tasks;
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.Speech.Intent;
```

Inside the provided `Main()` method, add the following code.

```
RecognizeIntentAsync().Wait();
Console.WriteLine("Please press Enter to continue.");
Console.ReadLine();
```

Create an empty asynchronous method `RecognizeIntentAsync()`, as shown here.

```
static async Task RecognizeIntentAsync()
{
}
```

In the body of this new method, add this code.

```

// Creates an instance of a speech config with specified subscription key
// and service region. Note that in contrast to other services supported by
// the Cognitive Services Speech SDK, the Language Understanding service
// requires a specific subscription key from https://www.luis.ai/.
// The Language Understanding service calls the required key 'endpoint key'.
// Once you've obtained it, replace with below with your own Language Understanding subscription key
// and service region (e.g., "westus").
// The default language is "en-us".
var config = SpeechConfig.FromSubscription("YourLanguageUnderstandingSubscriptionKey",
"YourLanguageUnderstandingServiceRegion");

// Creates an intent recognizer using microphone as audio input.
using (var recognizer = new IntentRecognizer(config))
{
    // Creates a Language Understanding model using the app id, and adds specific intents from your model
    var model = LanguageUnderstandingModel.FromAppId("YourLanguageUnderstandingAppId");
    recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
    recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
    recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");

    // Starts recognizing.
    Console.WriteLine("Say something...");

    // Starts intent recognition, and returns after a single utterance is recognized. The end of a
    // single utterance is determined by listening for silence at the end or until a maximum of 15
    // seconds of audio is processed. The task returns the recognition text as result.
    // Note: Since RecognizeOnceAsync() returns only a single utterance, it is suitable only for single
    // shot recognition like command or query.
    // For long-running multi-utterance recognition, use StartContinuousRecognitionAsync() instead.
    var result = await recognizer.RecognizeOnceAsync().ConfigureAwait(false);

    // Checks result.
    if (result.Reason == ResultReason.RecognizedIntent)
    {
        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
        Console.WriteLine($"    Intent Id: {result.IntentId}.");
        Console.WriteLine($"    Language Understanding JSON:");
        {result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult).};
    }
    else if (result.Reason == ResultReason.RecognizedSpeech)
    {
        Console.WriteLine($"RECOGNIZED: Text={result.Text}");
        Console.WriteLine($"    Intent not recognized.");
    }
    else if (result.Reason == ResultReason.NoMatch)
    {
        Console.WriteLine($"NOMATCH: Speech could not be recognized.");
    }
    else if (result.Reason == ResultReason.Canceled)
    {
        var cancellation = CancellationDetails.FromResult(result);
        Console.WriteLine($"CANCELED: Reason={cancellation.Reason}");

        if (cancellation.Reason == CancellationReason.Error)
        {
            Console.WriteLine($"CANCELED: ErrorCode={cancellation.ErrorCode}");
            Console.WriteLine($"CANCELED: ErrorDetails={cancellation.ErrorDetails}");
            Console.WriteLine($"CANCELED: Did you update the subscription info?");
        }
    }
}

```

Replace the placeholders in this method with your LUIS subscription key, region, and app ID as follows.

PLACEHOLDER	REPLACE WITH
YourLanguageUnderstandingSubscriptionKey	Your LUIS endpoint key. As previously noted, this must be a key obtained from your Azure dashboard, not a "starter key." You can find it on your app's Keys and Endpoints page (under Manage) in the LUIS portal .
YourLanguageUnderstandingServiceRegion	The short identifier for the region your LUIS subscription is in, such as <code>westus</code> for West US. See Regions .
YourLanguageUnderstandingAppId	The LUIS app ID. You can find it on your app's Settings page of the LUIS portal .

With these changes made, you can build (Control-Shift-B) and run (F5) the tutorial application. When prompted, try saying "Turn off the lights" into your PC's microphone. The result is displayed in the console window.

The following sections include a discussion of the code.

Create an intent recognizer

The first step in recognizing intents in speech is to create a speech config from your LUIS endpoint key and region. Speech configs can be used to create recognizers for the various capabilities of the Speech SDK. The speech config has multiple ways to specify the subscription you want to use; here, we use `FromSubscription`, which takes the subscription key and region.

NOTE

Use the key and region of your LUIS subscription, not of a Speech Services subscription.

Next, create an intent recognizer using `new IntentRecognizer(config)`. Since the configuration already knows which subscription to use, there's no need to specify the subscription key and endpoint again when creating the recognizer.

Import a LUIS model and add intents

Now import the model from the LUIS app using `LanguageUnderstandingModel.FromAppId()` and add the LUIS intents that you wish to recognize via the recognizer's `AddIntent()` method. These two steps improve the accuracy of speech recognition by indicating words that the user is likely to use in their requests. It is not necessary to add all the app's intents if you do not need to recognize them all in your application.

Adding intents requires three arguments: the LUIS model (which has been created and is named `model`), the intent name, and an intent ID. The difference between the ID and the name is as follows.

ADDINTENT() ARGUMENT	PURPOSE
intentName	The name of the intent as defined in the LUIS app. Must match the LUIS intent name exactly.
intentID	An ID assigned to a recognized intent by the Speech SDK. Can be whatever you like; does not need to correspond to the intent name as defined in the LUIS app. If multiple intents are handled by the same code, for instance, you could use the same ID for them.

The Home Automation LUIS app has two intents: one for turning on a device, and another for turning a device off.

The lines below add these intents to the recognizer; replace the three `AddIntent` lines in the `RecognizeIntentAsync()` method with this code.

```
recognizer.AddIntent(model, "HomeAutomation.TurnOff", "off");
recognizer.AddIntent(model, "HomeAutomation.TurnOn", "on");
```

Instead of adding individual intents, you can also use the `AddAllIntents` method to add all the intents in a model to the recognizer.

Start recognition

With the recognizer created and the intents added, recognition can begin. The Speech SDK supports both single-shot and continuous recognition.

RECOGNITION MODE	METHODS TO CALL	RESULT
Single-shot	<code>RecognizeOnceAsync()</code>	Returns the recognized intent, if any, after one utterance.
Continuous	<code>StartContinuousRecognitionAsync()</code> <code>StopContinuousRecognitionAsync()</code>	Recognizes multiple utterances. Emits events (e.g. <code>IntermediateResultReceived</code>) when results are available.

The tutorial application uses single-shot mode and so calls `RecognizeOnceAsync()` to begin recognition. The result is an `IntentRecognitionResult` object containing information about the intent recognized. The LUIS JSON response is extracted by the following expression:

```
result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult)
```

The tutorial application doesn't parse the JSON result, only displaying it in the console window.

```
Say something...
We recognized: Hey turn off the lights..

Intent Id: TurnOff.

Language Understanding JSON: {
  "query": "Hey turn off the lights",
  "topScoringIntent": {
    "intent": "HomeAutomation.TurnOff",
    "score": 0.984684
  },
  "entities": [
    {
      "entity": "lights",
      "type": "HomeAutomation.Device",
      "startIndex": 17,
      "endIndex": 22,
      "score": 0.9835096
    }
  ]
}.
Please press Enter to continue.
```

Specify recognition language

By default, LUIS recognizes intents in US English (`en-us`). By assigning a locale code to the `SpeechRecognitionLanguage` property of the speech configuration, you can recognize intents in other languages. For example, add `config.SpeechRecognitionLanguage = "de-de";` in our tutorial application before creating the recognizer to recognize intents in German. See [Supported Languages](#).

Continuous recognition from a file

The following code illustrates two additional capabilities of intent recognition using the Speech SDK. The first, previously mentioned, is continuous recognition, where the recognizer emits events when results are available. These events can then be processed by event handlers that you provide. With continuous recognition, you call the recognizer's `StartContinuousRecognitionAsync()` to start recognition instead of `RecognizeOnceAsync()`.

The other capability is reading the audio containing the speech to be processed from a WAV file. This involves creating an audio configuration that can be used when creating the intent recognizer. The file must be single-channel (mono) with a sampling rate of 16 kHz.

To try out these features, replace the body of the `RecognizeIntentAsync()` method with the following code.

```
// Creates an instance of a speech config with specified subscription key
// and service region. Note that in contrast to other services supported by
// the Cognitive Services Speech SDK, the Language Understanding service
// requires a specific subscription key from https://www.luis.ai/.
// The Language Understanding service calls the required key 'endpoint key'.
// Once you've obtained it, replace with below with your own Language Understanding subscription key
// and service region (e.g., "westus").
var config = SpeechConfig.FromSubscription("YourLanguageUnderstandingSubscriptionKey",
"YourLanguageUnderstandingServiceRegion");

// Creates an intent recognizer using file as audio input.
// Replace with your own audio file name.
using (var audioInput = AudioConfig.FromWavFileInput("whatstheweatherlike.wav"))
{
    using (var recognizer = new IntentRecognizer(config, audioInput))
    {
        // The TaskCompletionSource to stop recognition.
        var stopRecognition = new TaskCompletionSource<int>();

        // Creates a Language Understanding model using the app id, and adds specific intents from your model
        var model = LanguageUnderstandingModel.FromAppId("YourLanguageUnderstandingAppId");
        recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName1", "id1");
        recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName2", "id2");
        recognizer.AddIntent(model, "YourLanguageUnderstandingIntentName3", "any-IntentId-here");

        // Subscribes to events.
        recognizer.Recognizing += (s, e) => {
            Console.WriteLine($"RECOGNIZING: Text={e.Result.Text}");
        };

        recognizer.Recognized += (s, e) => {
            if (e.Result.Reason == ResultReason.RecognizedIntent)
            {
                Console.WriteLine($"RECOGNIZED: Text={e.Result.Text}");
                Console.WriteLine($"    Intent Id: {e.Result.IntentId}.");
                Console.WriteLine($"    Language Understanding JSON:");
                {e.Result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse_JsonResult)."};
            }
            else if (e.Result.Reason == ResultReason.RecognizedSpeech)
            {
                Console.WriteLine($"RECOGNIZED: Text={e.Result.Text}");
                Console.WriteLine($"    Intent not recognized.");
            }
            else if (e.Result.Reason == ResultReason.NoMatch)
            {
                Console.WriteLine($"NOMATCH: Speech could not be recognized.");
            }
        };
    };

    recognizer.Canceled += (s, e) => {
        Console.WriteLine($"CANCELED: Reason={e.Reason}");

        if (e.Reason == CancellationReason.Error)

```

```

        if (e.Reason == CancellationTokenReason.Error)
    {
        Console.WriteLine($"CANCELED: ErrorCode={e.ErrorCode}");
        Console.WriteLine($"CANCELED: ErrorDetails={e.ErrorDetails}");
        Console.WriteLine($"CANCELED: Did you update the subscription info?");

    }

    stopRecognition.TrySetResult(0);
};

recognizer.SessionStarted += (s, e) => {
    Console.WriteLine("\n    Session started event.");
};

recognizer.SessionStopped += (s, e) => {
    Console.WriteLine("\n    Session stopped event.");
    Console.WriteLine("\nStop recognition.");
    stopRecognition.TrySetResult(0);
};

// Starts continuous recognition. Uses StopContinuousRecognitionAsync() to stop recognition.
Console.WriteLine("Say something...");
await recognizer.StartContinuousRecognitionAsync().ConfigureAwait(false);

// Waits for completion.
// Use Task.WaitAny to keep the task rooted.
Task.WaitAny(new[] { stopRecognition.Task });

// Stops recognition.
await recognizer.StopContinuousRecognitionAsync().ConfigureAwait(false);
}
}
}

```

Revise the code to include your LUIS endpoint key, region, and app ID and to add the Home Automation intents, as before. Change `whatsttheweatherlike.wav` to the name of your audio file. Then build and run.

Get the samples

For the latest samples, see the [Cognitive Services Speech SDK sample code repository](#) on GitHub.

Look for the code from this article in the samples/csharp/sharedcontent/console folder.

Next steps

[How to recognize speech](#)

How to use the Dashboard to improve your app

8/9/2019 • 6 minutes to read • [Edit Online](#)

Find and fix problems with your trained app's intents when you are using example utterances. The dashboard displays overall app information, with highlights of intents that should be fixed.

Review Dashboard analysis is an iterative process, repeat as you change and improve your model.

This page will not have relevant analysis for apps that do not have any example utterances in the intents, known as *pattern-only* apps.

What issues can be fixed from dashboard?

The three problems addressed in the dashboard are:

ISSUE	CHART COLOR	EXPLANATION
Data imbalance	-	<p>This occurs when the quantity of example utterances varies significantly. All intents need to have <i>roughly</i> the same number of example utterances - except the None intent. It should only have 10%-15% of the total quantity of utterances in the app.</p> <p>If the data is imbalanced but the intent accuracy is above certain threshold, this imbalance is not reported as an issue.</p> <p>Start with this issue - it may be the root cause of the other issues.</p>
Unclear predictions	Orange	<p>This occurs when the top intent and the next intent's scores are close enough that they may flip on the next training, due to negative sampling or more example utterances added to intent.</p>
Incorrect predictions	Red	<p>This occurs when an example utterance is not predicted for the labeled intent (the intent it is in).</p>

Correct predictions are represented with the color blue.

The dashboard shows these issues and tells you which intents are affected and suggests what you should do to improve the app.

Before app is trained

Before you train the app, the dashboard does not contain any suggestions for fixes. Train your app to see these suggestions.

Check your publishing status

The **Publishing status** card contains information about the active version's last publish.

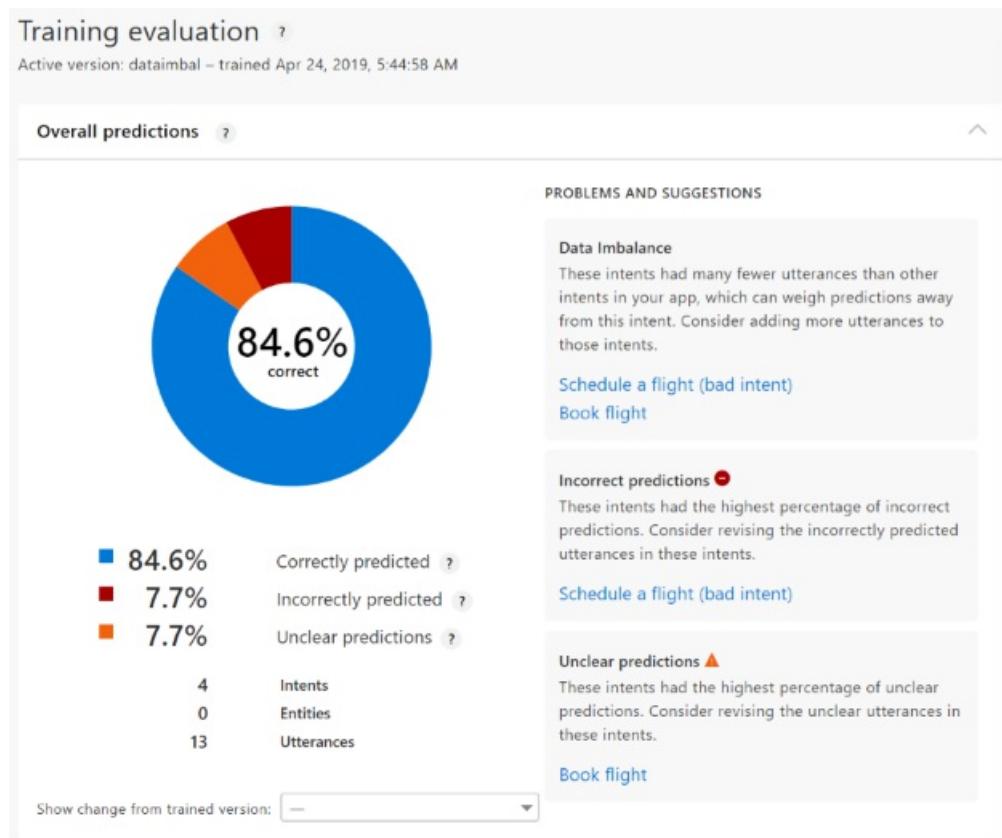
Check that the active version is the version you want to fix.

Publishing status	Endpoint hits per day
Last published:	
Not published yet	
Version: entities	
Slot: —	
External services	No endpoint hits.
No services	
Regions	
Not published yet	
Show results from <input type="button" value="Select an option"/>	

This also shows any external services, published regions, and aggregated endpoint hits.

Review training evaluation

The **Training evaluation** card contains the aggregated summary of your app's overall accuracy by area. The score indicates intent quality.



The chart indicates the correctly predicted intents and the problem areas with different colors. As you improve the app with the suggestions, this score increases.

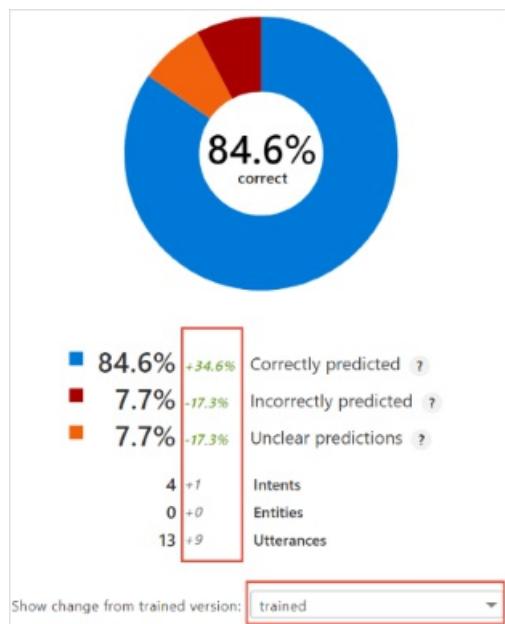
The suggested fixes are separated out by problem type and are the most significant for your app. If you would prefer to review and fix issues per intent, use the **Intents with errors** card at the bottom of the page.

Each problem area has intents that need to be fixed. When you select the intent name, the **Intent** page opens with a filter applied to the utterances. This filter allows you to focus on the utterances that are causing the problem.

Compare changes across versions

Create a new version before making changes to the app. In the new version, make the suggested changes to the

intent's example utterances, then train again. On the Dashboard page's **Training evaluation** card, use the **Show change from trained version** to compare the changes.



Fix version by adding or editing example utterances and retraining

The primary method of fixing your app will be to add or edit example utterances and retrain. The new or changed utterances need to follow guidelines for [varied utterances](#).

Adding example utterances should be done by someone who:

- has a high degree of understanding of what utterances are in the different intents
- knows how utterances in one intent may be confused with another intent
- is able to decide if two intents, which are frequently confused with each other, should be collapsed into a single intent, and the different data pulled out with entities

Patterns and phrase lists

The analytics page doesn't indicate when to use [patterns](#) or [phrase lists](#). If you do add them, it can help with incorrect or unclear predictions but won't help with data imbalance.

Review data imbalance

Start with this issue - it may be the root cause of the other issues.

The **data imbalance** intent list shows intents that need more utterances in order to correct the data imbalance.

To fix this issue:

- Add more utterances to the intent then train again.

Do not add utterances to the None intent unless that is suggested on the dashboard.

TIP

Use the third section on the page, **Utterances per intent** with the **Utterances (number)** setting, as a quick visual guide of which intents need more utterances.

A screenshot of a card titled "Predictions per intent". Below the title is a dropdown menu labeled "Utterances (number)" with the value "9" selected. The entire dropdown menu is highlighted with a red border.

Review incorrect predictions

The **incorrect prediction** intent list shows intents that have utterances, which are used as examples for a specific intent, but are predicted for different intents.

To fix this issue:

- Edit utterances to be more specific to the intent and train again.
- Combine intents if utterances are too closely aligned and train again.

Review unclear predictions

The **unclear prediction** intent list shows intents with utterances with prediction scores that are not far enough way from their nearest rival, that the top intent for the utterance may change on the next training, due to [negative sampling](#).

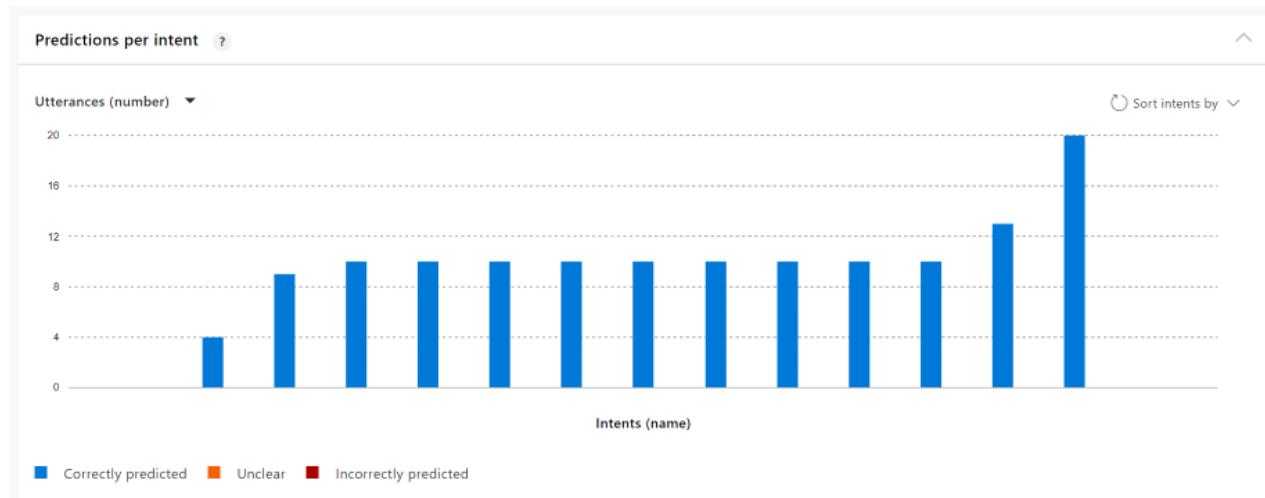
To fix this issue:

- Edit utterances to be more specific to the intent and train again.
- Combine intents if utterances are too closely aligned and train again.

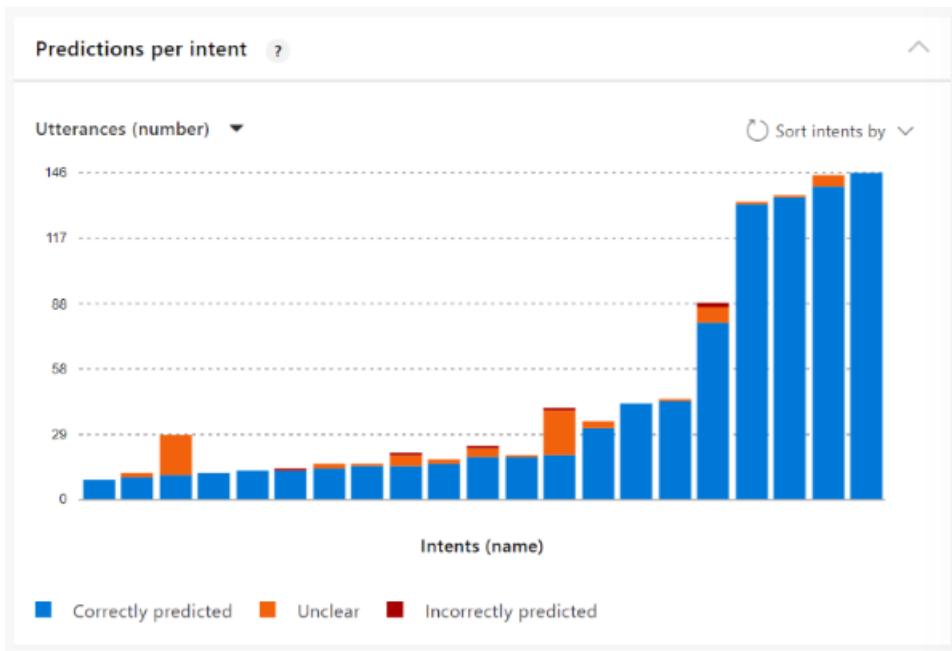
Utterances per intent

This card shows the overall app health across the intents. As you fix intents and retrain, continue to glance at this card for issues.

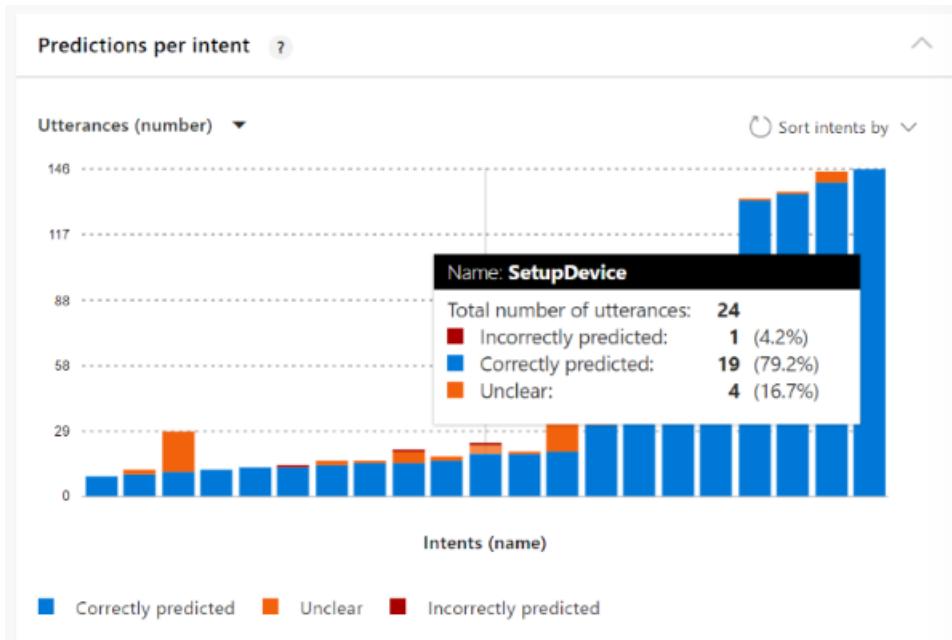
The following chart shows a well-balanced app with almost no issues to fix.



The following chart shows a poorly balanced app with many issues to fix.



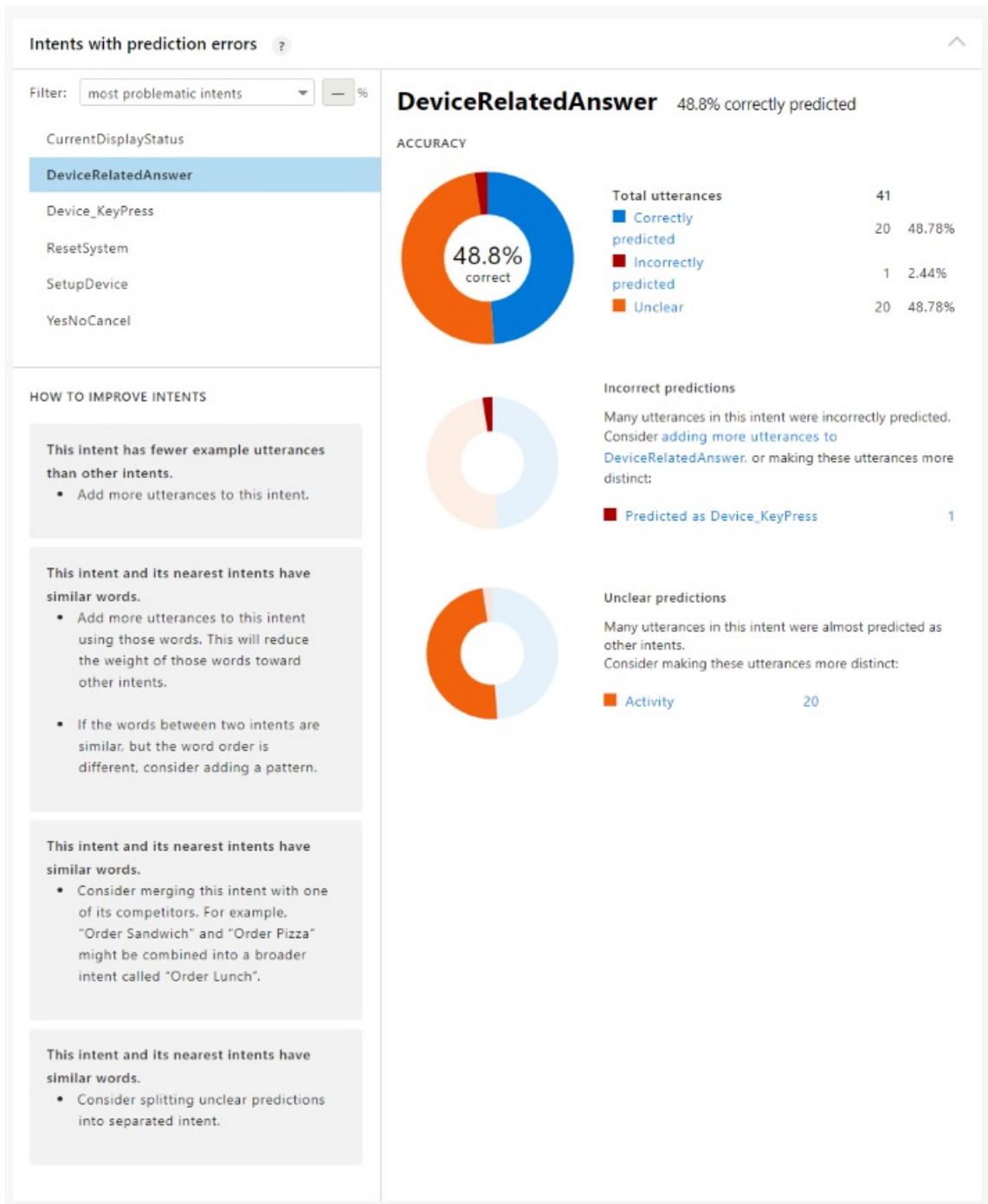
Hover over each intent's bar to get information about the intent.



Use the **Sort by** feature to arrange the intents by issue type so you can focus on the most problematic intents with that issue.

Intents with errors

This card allows you to review issues for a specific intent. The default view of this card is the most problematic intents so you know where to focus your efforts.



The top donut chart shows the issues with the intent across the three problem types. If there are issues in the three problem types, each type has its own chart below, along with any rival intents.

Filter intents by issue and percentage

This section of the card allows you to find example utterances that are falling outside your error threshold. Ideally you want correct predictions to be significant. That percentage is business and customer driven.

Determine the threshold percentages that you are comfortable with for your business.

The filter allows you to find intents with specific issue:

FILTER	SUGGESTED PERCENTAGE	PURPOSE
Most problematic intents	-	Start here - Fixing the utterances in this intent will improve the app more than other fixes.

FILTER	SUGGESTED PERCENTAGE	PURPOSE
Correct predictions below	60%	This is the percentage of utterances in the selected intent that are correct but have a confidence score below the threshold.
Unclear predictions above	15%	This is the percentage of utterances in the selected intent that are confused with the nearest rival intent.
Incorrect predictions above	15%	This is the percentage of utterances in the selected intent that are incorrectly predicted.

Correct prediction threshold

What is a confident prediction confidence score to you? At the beginning of app development, 60% may be your target. Use the **Correct predictions below** with the percentage of 60% to find any utterances in the selected intent that need to be fixed.

Unclear or incorrect prediction threshold

These two filters allow you to find utterances in the selected intent beyond your threshold. You can think of these two percentages as error percentages. If you are comfortable with a 10-15% error rate for predictions, set the filter threshold to 15% to find all utterances above this value.

Next steps

- [Manage your Azure resources](#)

Use phrase lists to boost signal of word list

8/9/2019 • 2 minutes to read • [Edit Online](#)

You can add features to your LUIS app to improve its accuracy. Features help LUIS by providing hints that certain words and phrases are part of an app domain vocabulary.

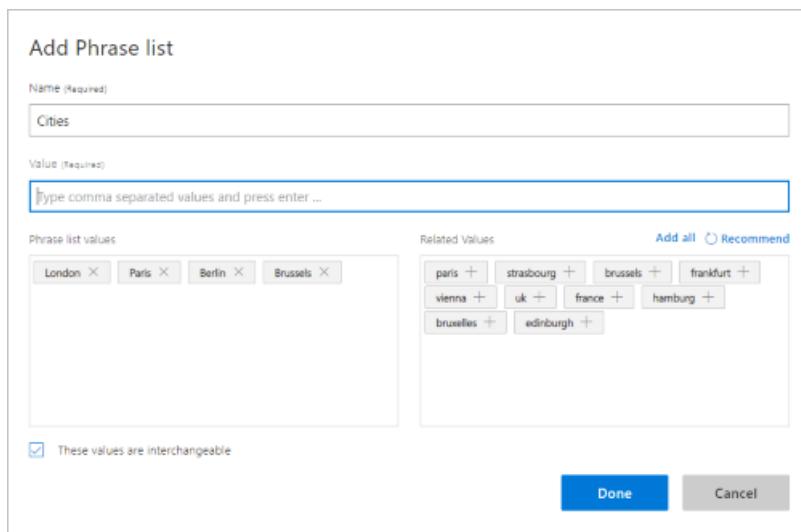
A **phrase list** includes a group of values (words or phrases) that belong to the same class and must be treated similarly (for example, names of cities or products). What LUIS learns about one of them is automatically applied to the others as well. This list is not the same thing as a **list entity** (exact text matches) of matched words.

A phrase list adds to the vocabulary of the app domain as a second signal to LUIS about those words.

Add phrase list

LUIS allows up to 10 phrase lists per app.

1. Open your app by clicking its name on **My Apps** page, and then click **Build**, then click **Phrase lists** in your app's left panel.
2. On the **Phrase lists** page, click **Create new phrase list**.
3. In the **Add Phrase List** dialog box, type **Cities** as the name of the phrase list. 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, and then press **Enter**.



4. LUIS can propose 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, or click **Add All** to add them all.

Add Phrase list

Name (Required)
Cities

Value (Required)
Type comma separated values and press enter ...

Phrase list values
London X Paris X Berlin X Brussels X

Related Values
paris + strasbourg + brussels + frankfurt +
vienna + uk + france + hamburg +
brussels + edinburgh +

Add all Recommend

These values are interchangeable

Done Cancel

5. Click **These values are interchangeable** if the added phrase list values are alternatives that can be used interchangeably.

Add Phrase list

Name (Required)
Cities

Value (Required)
Type comma separated values and press enter ...

Phrase list values
London X Paris X Berlin X Brussels X

Related Values
paris + strasbourg + brussels + frankfurt +
vienna + uk + france + hamburg +
brussels + edinburgh +

Add all Recommend

These values are interchangeable

Done Cancel

6. Click **Done**. The "Cities" phrase list is added to the **Phrase lists** page.

NOTE

You can delete, or deactivate a phrase list from the contextual toolbar on the **Phrase lists** page.

Next steps

After adding, editing, deleting, or deactivating a phrase list, [train and test the app](#) again to see if performance improves.

How to review endpoint utterances in LUIS portal for active learning

8/9/2019 • 2 minutes to read • [Edit Online](#)

Active learning captures endpoint queries and selects user's endpoint utterances that it is unsure of. You review these utterances to select the intent and mark entities for these real-world utterances. Accept these changes into your example utterances then train and publish. LUIS then identifies utterances more accurately.

Enable active learning

To enable active learning, log user queries. This is accomplished by setting the [endpoint query](#) with the `log=true` querystring parameter and value.

Disable active learning

To disable active learning, don't log user queries. This is accomplished by setting the [endpoint query](#) with the `log=false` querystring parameter and value.

Filter utterances

1. Open your app (for example, TravelAgent) by selecting its name on **My Apps** page, then select **Build** in the top bar.
2. Under the **Improve app performance**, select **Review endpoint utterances**.
3. On the **Review endpoint utterances** page, select in the **Filter list by intent or entity** text box. This drop-down list includes all intents under **INTENTS** and all entities under **ENTITIES**.

The screenshot shows the LUIS portal interface for reviewing endpoint utterances. The left sidebar has a tree structure with 'App Assets' expanded, showing 'Intents' and 'Entities'. Under 'Improve app performance', 'Review endpoint utterances' is selected. The main content area has a title 'Review endpoint utterances' with a question mark icon. A red box highlights the 'Filter list by intent or entity' dropdown. This dropdown has a search bar 'Type intent or entity name' and two sections: 'INTENTS' (containing BookFlight, Concierge, None) and 'ENTITIES' (empty). To the right are buttons for adding selected utterances to aligned intents, switching to 'Entities view', and managing aligned intents. Below these are buttons for 'Aligned intent', 'Add to aligned intent', and 'Delete'. At the bottom left is a 'PREVIEW' button, and at the bottom right is a 'Prebuilt Domains' link.

4. Select a category (intents or entities) in the drop-down list and review the utterances.

The screenshot shows the LUIS interface for reviewing endpoint utterances. On the left, there's a sidebar with sections like App Assets (Intents, Entities), Improve app performance (Review endpoint utterances, Phrase lists), and Prebuilt Domains. The main area is titled "Review endpoint utterances". It has a search bar for "BookFlight", a switch for "Entities view", and buttons for "Add all selected utterances to their aligned intents", "Add selected", and "Delete". There's a table with columns for Utterance, Aligned intent, Add to aligned intent, and Delete. Two utterances are listed:

- "buy number Seat Category seats to london" is aligned to "BookFlight 0.84" and has a checked checkbox under "Add to aligned intent".
- "where is the travel guide about london ?" is aligned to "Concierge 0.22" and has a checked checkbox under "Add to aligned intent".

Label entities

Luis replaces entity tokens (words) with entity names highlighted in blue. If an utterance has unlabeled entities, label them in the utterance.

1. Select on the word(s) in the utterance.
2. Select an entity from the list.

This screenshot shows the LUIS interface with a red box highlighting the entity selection dropdown menu. The menu is open over an utterance containing several entity tokens. The menu options include "Search or create", "Actions", "Wrap in composite entity", and "Browse prebuilt entities". Below the menu, a list of entities is shown with checkboxes next to them. The entities listed are Airline, Category, Cities, Location, and Menu. The "Category" entity is currently selected, indicated by a checked checkbox.

Align single utterance

Each utterance has a suggested intent displayed in the **Aligned intent** column.

1. If you agree with that suggestion, select on the check mark.

This screenshot shows the LUIS interface with a red box highlighting the checkmark in the "Add to aligned intent" column for the first utterance. The utterance "Cities to orlando in Seat datetimeV2" is aligned to "BookFlight 1". The checkmark is located in the "Add to aligned intent" column for this row.

2. If you disagree with the suggestion, select the correct intent from the aligned intent drop-down list, then select on the check mark to the right of the aligned intent.

The screenshot shows the 'Review endpoint utterances' page. On the left, there's a sidebar with sections like 'App Assets' (Intents, Entities), 'Improve app performance' (Review endpoint utterances, Phrase lists), and 'Prebuilt Domains'. The main area has a header 'Review endpoint utterances' with a filter input 'BookFlight' and an 'Entities view' toggle. A button 'Add all selected utterances to their aligned intents' is at the top right. Below is a table with columns 'Utterance', 'Aligned intent', and 'Actions'. The first row shows an utterance 'Cities to orlando in Seat datetimeV2' aligned to 'BookFlight 1' (with a checkmark). The second row shows 'book a Category seat to new york' aligned to 'None 0.07'. The third row shows 'new york to Location::ToLocation in first class datetimeV2' aligned to 'Concierge 0.04'. The fourth row shows 'where is the travel guide about london ?' aligned to 'Concierge 0.22'. The fifth row shows 'buy number Seat Category seats to london' aligned to 'BookFlight 0.84'. The sixth row shows 'reserve a seat on the redeye to Location::ToLocation' aligned to 'BookFlight 1'. Each row has a trash bin icon in the 'Delete' column.

3. After you select on the check mark, the utterance is removed from the list.

Align several utterances

To align several utterances, check the box to the left of the utterances, then select on the **Add selected** button.

This screenshot is similar to the previous one but highlights the selection process. The first three utterances ('Cities to orlando in Seat datetimeV2', 'book a Category seat to new york', and 'new york to Location::ToLocation in first class datetimeV2') have their checkboxes checked, indicated by a red box around them. The 'Add selected' button is also highlighted with a red box. The rest of the interface and data are identical to the first screenshot.

Verify aligned intent

You can verify the utterance was aligned with the correct intent by going to the **Intents** page, select the intent name, and reviewing the utterances. The utterance from **Review endpoint utterances** is in the list.

Delete utterance

Each utterance can be deleted from the review list. Once deleted, it will not appear in the list again. This is true even if the user enters the same utterance from the endpoint.

If you are unsure if you should delete the utterance, either move it to the None intent, or create a new intent such as "miscellaneous" and move the utterance to that intent.

Delete several utterances

To delete several utterances, select each item and select on the trash bin to the right of the **Add selected** button.

App Assets

- Intents
- Entities

Improve app performance

- Review endpoint utterances
- Phrase lists

PREVIEW Prebuilt Domains

Review endpoint utterances ?

Filter list by intent or entity BookFlight Entities view

Add all selected utterances to their aligned intents Add selected Delete

Utterance	Aligned intent ?	Add to aligned intent	Delete
<input checked="" type="checkbox"/> Cities to orlando in Seat datetimeV2	BookFlight 1 ▾	<input checked="" type="checkbox"/>	
<input checked="" type="checkbox"/> book a Category seat to new york	BookFlight 0.99 ▾	<input checked="" type="checkbox"/>	
<input checked="" type="checkbox"/> new york to Location::ToLocation in first class datetimeV2	BookFlight 1 ▾	<input checked="" type="checkbox"/>	
<input type="checkbox"/> where is the travel guide about london ?	Concierge 0.22 ▾	<input checked="" type="checkbox"/>	
<input type="checkbox"/> buy number Seat Category seats to london	BookFlight 0.84 ▾	<input checked="" type="checkbox"/>	
<input type="checkbox"/> reserve a seat on the redeye to Location::ToLocation	BookFlight 1 ▾	<input checked="" type="checkbox"/>	

Next steps

To test how performance improves after you label suggested utterances, you can access the test console by selecting **Test** in the top panel. For instructions on how to test your app using the test console, see [Train and test your app](#).

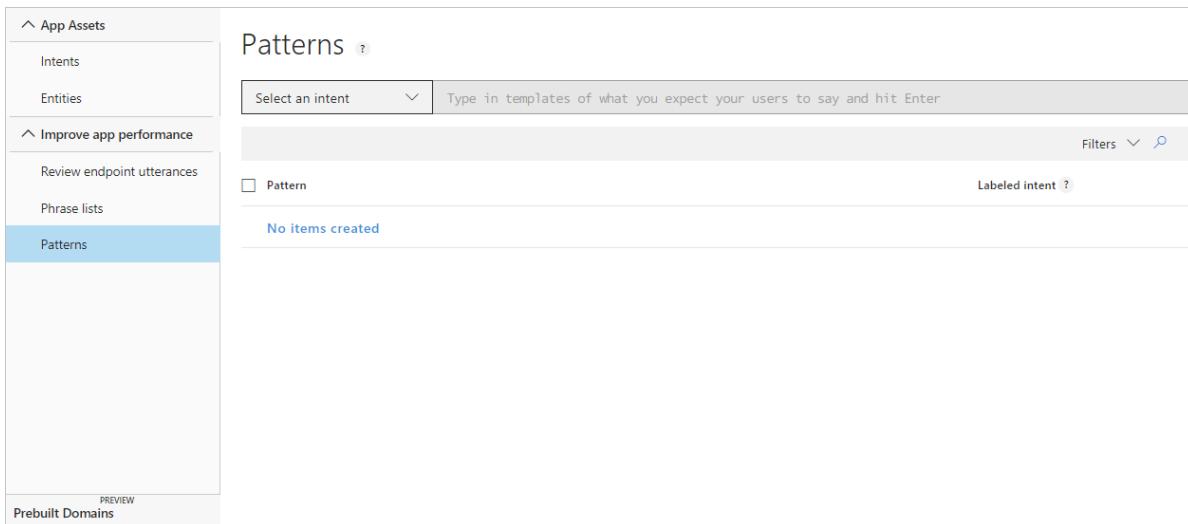
How to add patterns to improve prediction accuracy

7/30/2019 • 2 minutes to read • [Edit Online](#)

After a LUIS app receives endpoint utterances, use a [pattern](#) to improve prediction accuracy for utterances that reveal a pattern in word order and word choice. Patterns use specific [syntax](#) to indicate the location of: [entities](#), entity [roles](#), and optional text.

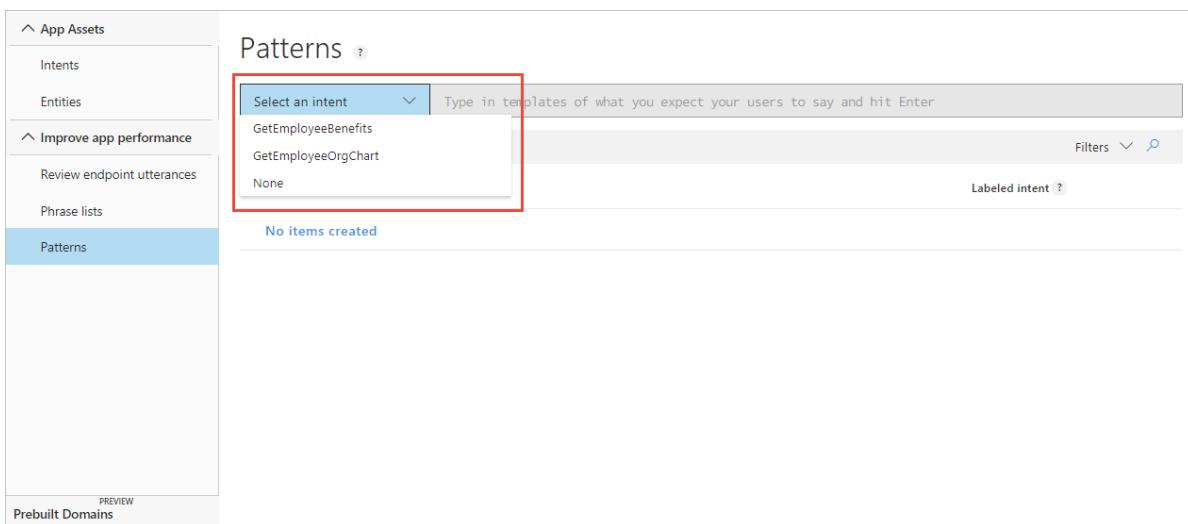
Add template utterance to create pattern

1. Open your app by selecting its name on **My Apps** page, and then select **Patterns** in the left panel, under **Improve app performance**.



The screenshot shows the LUIS Patterns page. On the left, there's a navigation sidebar with sections like App Assets, Intents, Entities, Improve app performance (with sub-options like Review endpoint utterances, Phrase lists, and Patterns), and Prebuilt Domains. The Patterns section is selected. The main area is titled "Patterns" and contains a search bar with "Select an intent" and "Type in templates of what you expect your users to say and hit Enter". Below the search bar is a "Filters" button. A table header row includes columns for "Pattern" and "Labeled intent ?". The body of the table says "No items created". At the bottom left, it says "PREVIEW".

2. Select the correct intent for the pattern.



This screenshot is similar to the previous one, but the "Select an intent" dropdown menu is open, showing three options: "GetEmployeeBenefits", "GetEmployeeOrgChart", and "None". The "None" option is at the bottom of the list. The rest of the interface is identical to the first screenshot.

3. In the template textbox, type the template utterance and select Enter. When you want to enter the entity name, use the correct pattern entity syntax. Begin the entity syntax with `{`. The list of entities displays. Select the correct entity, and then select Enter.

The screenshot shows the Microsoft Bot Framework's Patterns editor. On the left, a sidebar lists 'App Assets' (Intents, Entities), 'Improve app performance' (Review endpoint utterances, Phrase lists, Patterns), and 'Prebuilt Domains'. The 'Patterns' section is selected. In the main area, under the 'Patterns' tab, there is a search bar with 'GetEmployeeOrgChart' and 'Who manages' followed by a placeholder '(entity)'. Below the search bar, a dropdown menu shows 'number' and 'Employee'. A red box highlights the 'Employee' option. To the right of the dropdown are 'Entity filters' and 'Intent filters' with a search icon. A status message 'Missing entity' is displayed above the search bar. The bottom of the screen shows a message 'No items created'.

If your entity includes a [role](#), indicate the role with a single colon, `:`, after the entity name, such as `{Location:Origin}`. The list of roles for the entities displays in a list. Select the role, and then select Enter.

This screenshot shows the same Patterns editor interface. The search bar now contains 'MoveAsset' and 'Move {Employee} from {Location}'. A dropdown menu is open over the placeholder '{Location}', showing 'Origin' and 'Destination' with 'Origin' highlighted. A red box surrounds this dropdown menu. The rest of the interface is identical to the first screenshot, including the 'Entity filters' and 'Intent filters' sections and the 'No items created' message.

After you select the correct entity, finish entering the pattern, and then select Enter. When you are done entering patterns, [train](#) your app.

The screenshot shows the Patterns editor with the search bar containing 'MoveAsset' and 'move Employee from Location:Origin to Location:Destination'. A red box highlights the entire pattern entry. The rest of the interface is consistent with the previous screenshots, including the sidebar and the 'No items created' message at the bottom.

Train your app after changing model with patterns

After you add, edit, remove, or reassign a pattern, [train](#) and [publish](#) your app for your changes to affect endpoint queries.

Use contextual toolbar

The contextual toolbar above the patterns list allows you to:

- Search for patterns
- Edit a pattern
- Reassign individual pattern to different intent
- Reassign several patterns to different intent
- Delete-a-single-pattern
- Delete several patterns
- Filter pattern list by entity
- Filter-pattern-list-by-intent
- Remove entity or intent filter
- Add pattern from existing utterance on intent or entity page

Next steps

- Learn how to [build a pattern](#) with a pattern.any and roles with a tutorial.
- Learn how to [train](#) your app.

Train your active version of the LUIS app

8/9/2019 • 2 minutes to read • [Edit Online](#)

Training is the process of teaching your Language Understanding (LUIS) app to improve its natural language understanding. Train your LUIS app after updates to the model such as adding, editing, labeling, or deleting entities, intents, or utterances.

Training and [testing](#) an app 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 they're not, make updates to the LUIS app, train, and test again.

Training is applied to the active version in the LUIS portal.

How to train interactively

To start the iterative process in the [LUIS portal](#), you first need to train your LUIS app at least once. Make sure every intent has at least one utterance before training.

1. Access your app by selecting its name on the **My Apps** page.
2. In your app, select **Train** in the top panel.
3. When training is complete, a green notification bar appears at the top of the browser.

NOTE

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

Training date and time

Training date and time is GMT + 2.

Train with all data

Training uses a small percentage of negative sampling. If you want to use all data instead of the small negative sampling, use the [Version settings API](#) with the `UseAllTrainingData` set to true to turn off this feature.

Unnecessary training

You do not need to train after every single change. Training should be done after a group of changes are applied to the model, and the next step you want to do is to test or publish. If you do not need to test or publish, training isn't necessary.

Training with the REST APIs

Training in the LUIS portal is a single step of pressing the **Train** button. Training with the REST APIs is a two-step process. The first is to [request training](#) with HTTP POST. Then request the [training status](#) with HTTP Get.

In order to know when training is complete, you have to poll the status until all models are successfully trained.

Next steps

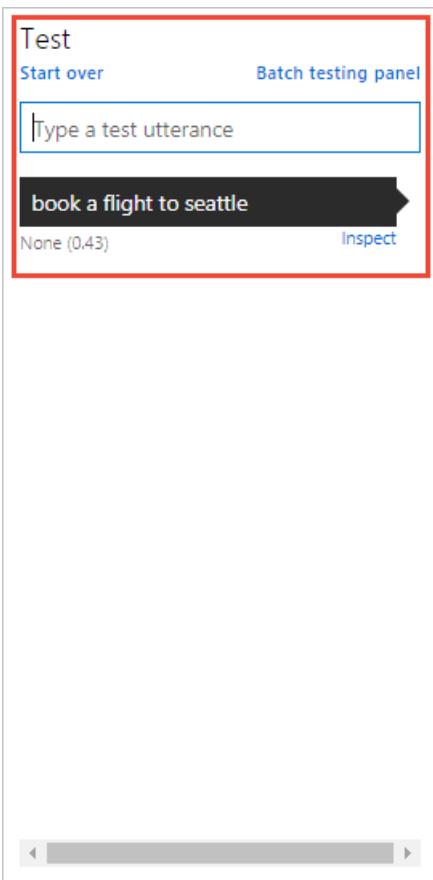
- [Label suggested utterances with LUIS](#)
- [Use features to improve your LUIS app's performance](#)

Test your LUIS app in the LUIS portal

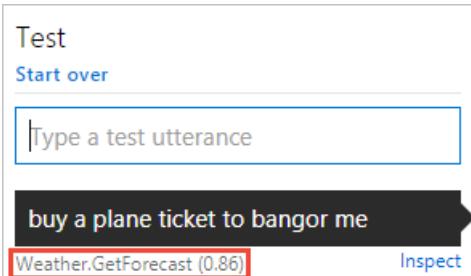
Testing an app is an iterative process. After training your LUIS app, test it with sample utterances to see if the intents and entities are recognized correctly. If they're not, make updates to the LUIS app, train, and test again.

Test an utterance

1. Access your app by selecting its name on the **My Apps** page.
2. To access the **Test** slide-out panel, select **Test** in your application's top panel.



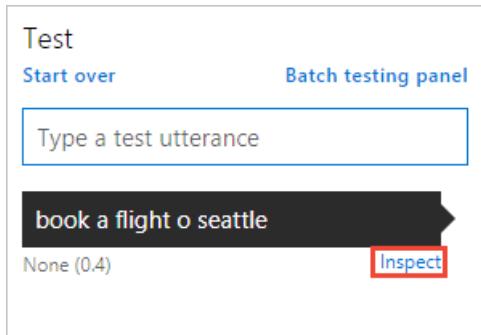
3. Enter an utterance in the text box and select Enter. You can type as many test utterances as you want in the **Test**, but only one utterance at a time.
4. The utterance, its top intent, and score are added to the list of utterances under the text box.



Inspect score

You inspect details of the test result in the **Inspect** panel.

1. With the **Test** slide-out panel open, select **Inspect** for an utterance you want to compare.



2. The **Inspection** panel appears. The panel includes the top scoring intent as well as any identified entities. The panel shows the result of the selected utterance.

The screenshot shows the 'Inspection' panel. At the top, it says 'Current: (V 0.1)' and has a 'Compare with published' button. Below that is a list of utterances: 'book a flight to seattle'. Underneath is a section for the 'Top scoring intent': 'BookFlight (0.667)' with an 'Edit' button. A table titled 'Entities' lists one entity: "'seattle'" under 'Text' and 'Location::ToLocation' under 'Entity'. Below that is a section for 'Top matched pattern(s)': 'No matched patterns'. At the bottom is a 'Sentiment' section with the text 'Enable sentiment analysis to get sentiment score'.

Correct top scoring intent

1. If the top scoring intent is incorrect, select the **Edit** button.
2. In the drop-down list, select the correct intent for the utterance.

Test

[Start over](#) [Batch testing panel](#)

Type a test utterance

book a flight to seattle

BookFlight (0.667) [Inspect](#)

Current: (v 0.1)

[Compare with published](#)

book a flight to seattle

Top scoring intent

BookFlight (0.667)

Select an intent

BookFlight (0.667)
None (0.108)
Concierge (0.027)

Top matched pattern(s)

No matched patterns

Sentiment

Enable sentiment analysis to get sentiment score

View sentiment results

If **Sentiment analysis** is configured on the [Publish](#) page, the test results include the sentiment found in the utterance.

Test

[Start over](#) [Batch testing panel](#)

Type a test utterance

that was a terrible flight. please don't shake the plane so much.

BookFlight (0.911) [Inspect](#)

Current: (v 0.1)

[Compare with published](#)

that was a terrible flight. please don't shake the plane so much.

Top scoring intent

BookFlight (0.911) [Edit](#)

Entities

No predicted entities

Top matched pattern(s)

No matched patterns

Sentiment

Label	Score
negative	0.0576811433

Correct matched pattern's intent

If you are using [Patterns](#) and the utterance matched a pattern, but the wrong intent was predicted, select the [Edit](#) link by the pattern, then select the correct intent.

Compare with published version

You can test the active version of your app with the published endpoint version. In the **Inspect** panel, select **Compare with published**. Any testing against the published model is deducted from your Azure subscription quota balance.

The screenshot shows two side-by-side panels. The left panel is titled 'Test' and shows an input field with the text 'book 5 seats from seattle to cairo next monday at 5pm'. Below it, the score 'BookFlight (0.728)' and the 'Inspect' button are visible. The right panel is titled 'Published: Production (V 0.1)' and also shows the same input text. It includes sections for 'Top scoring intent' (BookFlight (0.728)), 'Entities' (listing '5' as a number, 'seattle' and 'cairo' as geographyV2 entities, and 'next monday at 5pm' as a datetimeV2 entity), 'Top matched pattern(s)' (No matched patterns), and 'Sentiment' (neutral score 0.5). Both panels have a 'Batch testing panel' tab at the top.

View endpoint JSON in test panel

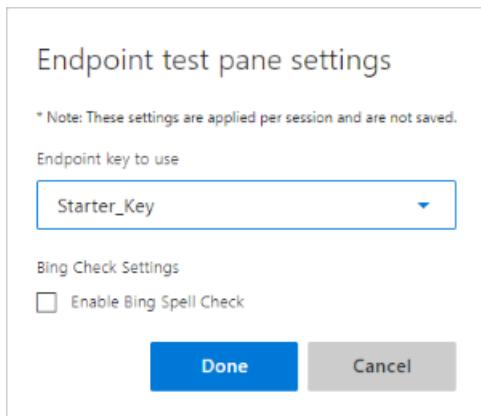
You can view the endpoint JSON returned for the comparison by selecting the **Show JSON view**.

This screenshot is similar to the previous one but with the 'Show JSON view' option selected in both the 'Test' and 'Published' panels. The right panel now displays the raw JSON response for the utterance 'book 5 seats from seattle to cairo next monday at 5pm'. The JSON output includes the query, prediction details, entities (number 5, datetimeV2 for 'next monday at 5pm' with value '2019-08-05T17'), and an instance object for the number entity.

Additional settings in test panel

LUIS endpoint

If you have several LUIS endpoints, use the **Additional Settings** link on the Test's Published pane to change the endpoint used for testing. If you are not sure which endpoint to use, select the default **Starter_Key**.



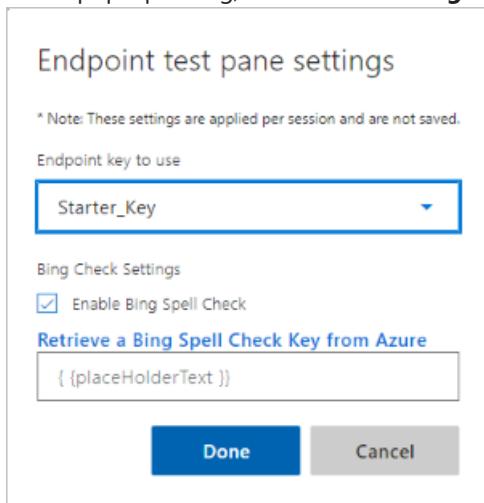
View Bing Spell Check corrections in test panel

Requirements to view the spelling corrections:

- Published app
- Bing Spell Check [service key](#). The service key is not stored and needs to be reset for each browser session.

Use the following procedure to include the [Bing Spell Check v7](#) service in the Test pane results.

1. In the **Test** pane, enter an utterance. When the utterance is predicted, select **Inspect** underneath the utterance you entered.
2. When the **Inspect** panel opens, select **Compare with Published**.
3. When the **Published** panel opens, select **Additional Settings**.
4. In the pop-up dialog, check **Enable Bing Spell Check** and enter the key, then select **Done**.



5. Enter a query with an incorrect spelling such as `book flite to seattle` and select enter. The incorrect spelling of the word `flite` is replaced in the query sent to LUIS and the resulting JSON shows both the original query, as `query`, and the corrected spelling in the query, as `alteredQuery`.

Batch testing

See batch testing [concepts](#) and learn [how to](#) test a batch of utterances.

Next steps

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

- [Label suggested utterances with LUIS](#)

- Use features to improve your LUIS app's performance

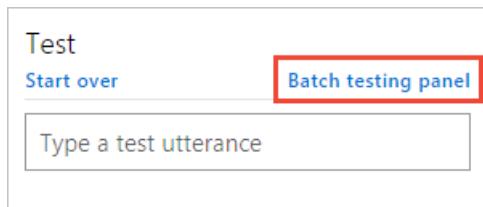
Batch testing with a set of example utterances

8/9/2019 • 2 minutes to read • [Edit Online](#)

Batch testing is a comprehensive test on your current trained model to measure its performance in LUIS. The data sets used for batch testing should not include example utterances in the intents or utterances received from the prediction runtime endpoint.

Import a dataset file for batch testing

1. Select **Test** in the top bar, and then select **Batch testing panel**.



2. Select **Import dataset**. The **Import new dataset** dialog box appears. Select **Choose File** and locate a JSON file with the correct **JSON format** that contains *no more than 1,000* utterances to test.

Import errors are reported in a red notification bar at the top of the browser. When an import has errors, no dataset is created. For more information, see [Common errors](#).

3. In the **Dataset Name** field, enter a name for your dataset file. The dataset file includes an **array of utterances** including the *labeled intent* and *entities*. Review the [example batch file](#) for syntax.
4. Select **Done**. The dataset file is added.

Run, rename, export, or delete dataset

To run, rename, export, or delete the dataset, use the ellipsis (...) button at the end of the dataset row.

A screenshot of the LUIS Dataset list page. At the top left, it says 'Batch testing' and 'Single testing panel'. Below that is a blue button labeled 'Import dataset'. The main area shows a table with three rows of datasets. The columns are 'State', 'Name', 'Size', 'Last Run', and 'Last Result'. The first row has a 'Run' button next to it. The second row has a 'Run' button next to it. The third row has a 'Run' button next to it. To the right of the table is an ellipsis (...) button, which has a red box around it. A dropdown menu is shown, containing four options: 'Run' (highlighted), 'Rename', 'Export', and 'Delete'.

State	Name	Size	Last Run	Last Result
▷ Run	set 1 fix	6	3/19/2018	1
▷ Run	ds1 - fix 2	6	3/19/2018	1
▷ Run	set 1	6	3/28/2018	1

Run a batch test on your trained app

To run the test, select the dataset name. When the test completes, this row displays the test result of the dataset.

Batch testing

Single testing panel

Import dataset

State	Name	Size	Last Run	Last Result	...
▷ Run	DataSet1	4	Not run yet	Not run yet	...

The downloadable dataset is the same file that was uploaded for batch testing.

STATE	MEANING
✓	All utterances are successful.
✗	At least one utterance intent did not match the prediction.
▷	Test is ready to run.

View batch test results

To review the batch test results, select **See results**.

Batch testing

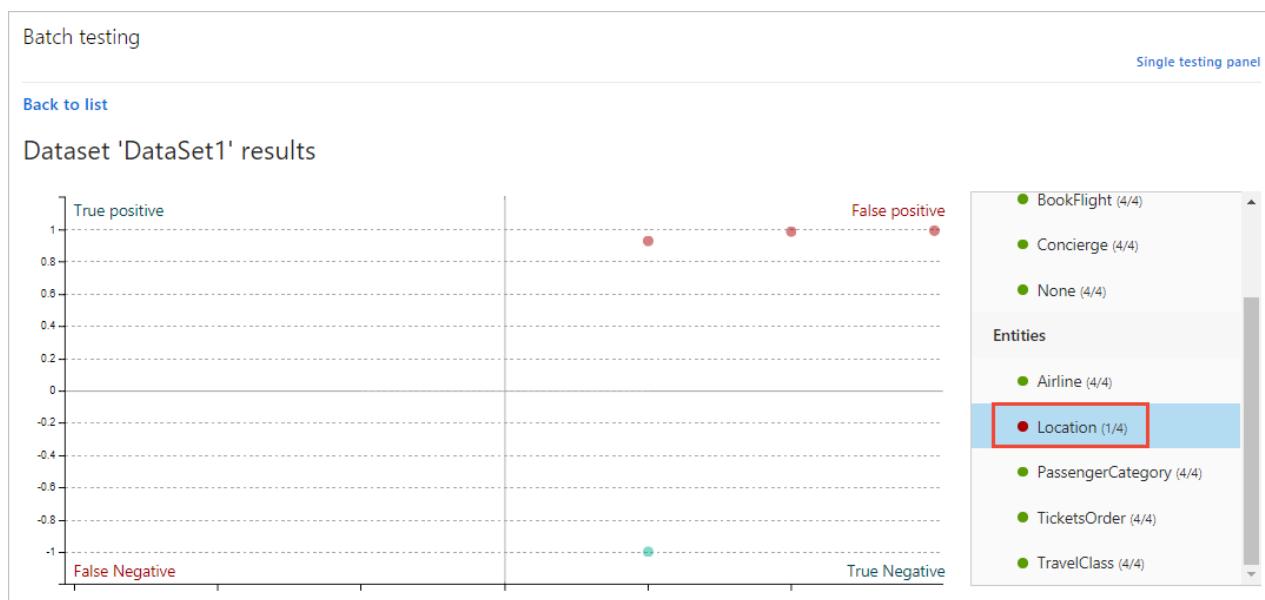
Single testing panel

Import dataset

State	Name	Size	Last Run	Last Result	...
✗ See results	DataSet1	6	3/14/2018	2	...

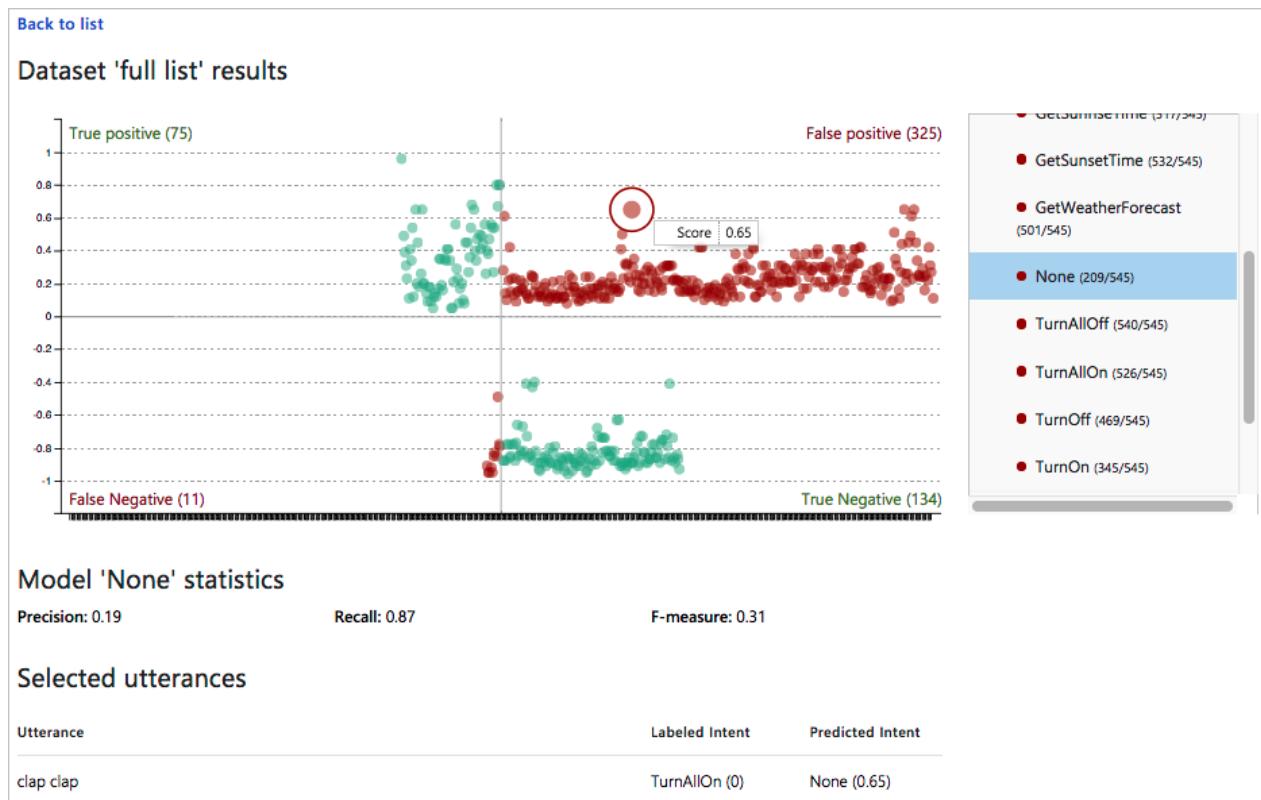
Filter chart results

To filter the chart by a specific intent or entity, select the intent or entity in the right-side filtering panel. The data points and their distribution update in the graph according to your selection.



View single-point utterance data

In the chart, hover over a data point to see the certainty score of its prediction. Select a data point to retrieve its corresponding utterance in the utterances list at the bottom of the page.



View section data

In the four-section chart, select the section name, such as **False Positive** at the top-right of the chart. Below the chart, all utterances in that section display below the chart in a list.

[Back to list](#)

Dataset 'full list' results



Model 'None' statistics

Precision: 0.19

Recall: 0.87

F-measure: 0.31

Selected utterances

Utterance	Labeled Intent	Predicted Intent
switch on the light	TurnAllOn (0)	None (0.28)
switch on	TurnAllOn (0.09)	None (0.61)
give me the weather in munich	GetCurrentWeather (0)	None (0.14)

In this preceding image, the utterance `switch on` is labeled with the TurnAllOn intent, but received the prediction of None intent. This is an indication that the TurnAllOn intent needs more example utterances in order to make the expected prediction.

The two sections of the chart in red indicate utterances that did not match the expected prediction. These indicate utterances which LUIS needs more training.

The two sections of the chart in green did match the expected prediction.

Roles in batch testing

Caution

Entity roles are not supported in batch testing.

Next steps

If testing indicates that your LUIS app doesn't recognize the correct intents and entities, you can work 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](#)
- [Understand batch testing with this tutorial](#)
- [Learn batch testing concepts.](#)

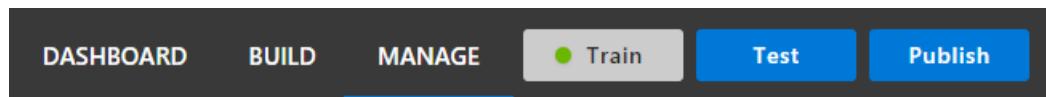
Publish your active, trained app to a staging or production endpoint

8/9/2019 • 2 minutes to read • [Edit Online](#)

When you finish building and testing your active LUIS app, make it available to your client application by publishing it to the endpoint.

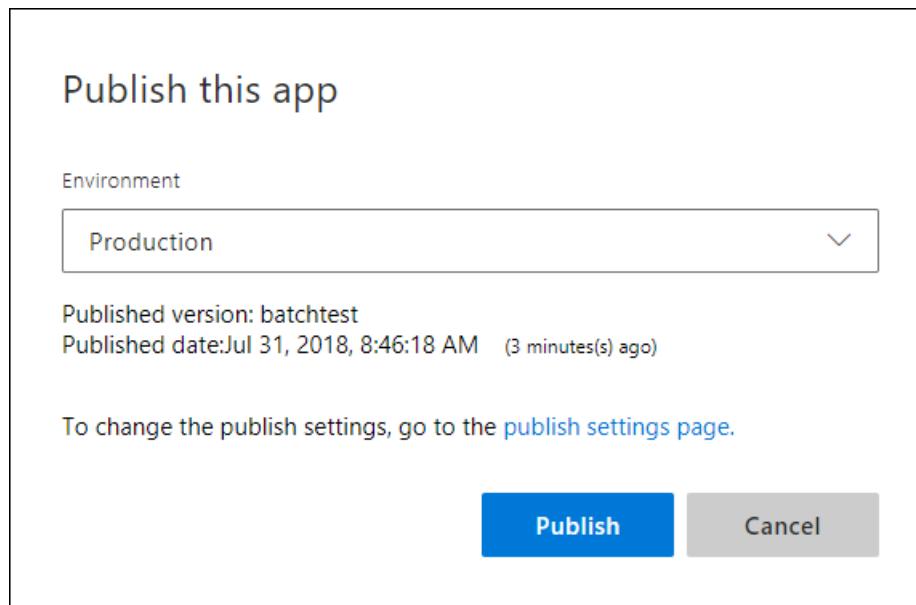
Publishing

To publish to the endpoint, select **Publish** in the top, right panel.



Select the correct slot when the pop-up window displays: staging or production. By using two publishing slots, this allows you to have two different versions with published endpoints or the same version on two different endpoints.

The app is published to all regions associated with the LUIS resources added in the LUIS portal. For example, for an app created on www.luis.ai, if you create a LUIS resource in **westus** and add it to the app as a resource, the app is published in that region. For more information about LUIS regions, see [Regions](#).



When your app is successfully published, a green success notification appears at the top of the browser. The green notification bar also includes a link to the endpoints.



Publishing complete. Refer to the list of endpoints to access your endpoint URL

If you need the endpoint URL, select the link. You can also get to the endpoint URLs by selecting **Manage** in the top menu, then select **Keys and Endpoints** in the left menu.

Configuring publish settings

Configure publish settings by selecting **Manage** in the top, right navigation, then selecting **Publish Settings**.

The screenshot shows the 'Publish settings' section of the LUIS application configuration interface. On the left, a sidebar lists several tabs: Application Information, Keys and Endpoints, Publish Settings (which is selected and highlighted in blue), Versions, and Collaborators. The main content area is titled 'Publish settings' with a help icon. It contains a note: 'These publish settings will apply the next time you publish your app.' Below this, there's a section titled 'External services' with two options: 'Use sentiment analysis to determine if a user's utterance is positive, negative, or neutral' and 'Enable speech priming to allow a single request to receive audio and return LUIS prediction JSON objects'. Both options have checkboxes next to them.

Publish after enabling sentiment analysis

Sentiment analysis allows LUIS to integrate with [Text Analytics](#) to provide sentiment and key phrase analysis.

You do not have to provide a Text Analytics key and there is no billing charge for this service to your Azure account. Once you check this setting, it is persistent.

Sentiment data is a score between 1 and 0 indicating the positive (closer to 1) or negative (closer to 0) sentiment of the data. The sentiment label of `positive`, `neutral`, and `negative` is per supported culture. Currently, only English supports sentiment labels.

For more information about the JSON endpoint response with sentiment analysis, see [Sentiment analysis](#)

Next steps

- See [Manage keys](#) to add keys to Azure subscription key to LUIS and how to set the Bing Spell Check key and include all intents in results.
- See [Train and test your app](#) for instructions on how to test your published app in the test console.

Using subscription keys with your LUIS app

8/9/2019 • 6 minutes to read • [Edit Online](#)

When you first use Language Understanding (LUIS), you do not need to create subscription keys. You are given 1000 endpoint queries to begin with.

For testing and prototype only, use the free (F0) tier. For production systems, use a [paid](#) tier. Do not use the [authoring key](#) for endpoint queries in production.

Create prediction endpoint runtime resource in the Azure portal

You create the [prediction endpoint resource](#) in the Azure portal. This resource should only be used for endpoint prediction queries. Do not use this resource for authoring changes to the app.

You can create a Language Understanding resource or a Cognitive Services resource. If you are creating a Language Understanding resource, a good practice is to postpend the resource type to the resource name.

Using resource from LUIS portal

If you are using the resource from the LUIS portal, you do not need to know your key and location. Instead you need to know your resource tenant, subscription, and resource name.

Once you [assign](#) your resource to your LUIS app in the LUIS portal, the key and location are provided as part of the query prediction endpoint URL in the Manage section's **Keys and Endpoint settings** page.

Using resource from REST API or SDK

If you are using the resource from the REST API(s) or SDK, you need to know your key and location. This information is provided as part of the query prediction endpoint URL in the Manage section's **Keys and Endpoint settings** page as well as in the Azure portal, on the resource's Overview and Keys pages.

Assign resource key to LUIS app in LUIS Portal

Every time you create a new resource for LUIS, you need to [assign the resource to the LUIS app](#). After it's assigned, you won't need to do this step again unless you create a new resource. You might create a new resource to expand the regions of your app or to support a higher number of prediction queries.

Unassign resource

When you unassign the endpoint key, it is not deleted from Azure. It is only unlinked from LUIS.

When an endpoint key is unassigned, or not assigned to the app, any request to the endpoint URL returns an error: `401 This application cannot be accessed with the current subscription`.

Include all predicted intent scores

The **Include all predicted intent scores** checkbox allows the endpoint query response to include the prediction score for each intent.

This setting allows your chatbot or LUIS-calling application to make a programmatic decision based on the scores of the returned intents. Generally the top two intents are the most interesting. If the top score is the None intent, your chatbot can choose to ask a follow-up question that makes a definitive choice between the None intent and the other high-scoring intent.

The intents and their scores are also included the endpoint logs. You can [export](#) those logs and analyze the scores.

```
{
  "query": "book a flight to Cairo",
  "topScoringIntent": {
    "intent": "None",
    "score": 0.5223427
  },
  "intents": [
    {
      "intent": "None",
      "score": 0.5223427
    },
    {
      "intent": "BookFlight",
      "score": 0.372391433
    }
  ],
  "entities": []
}
```

Enable Bing spell checker

In the **Endpoint url settings**, the **Bing spell checker** toggle allows LUIS to correct misspelled words before prediction. Create a [Bing Spell Check key](#).

Add the **spellCheck=true** querystring parameter and the **bing-spell-check-subscription-key={YOUR_BING_KEY_HERE}**. Replace the `{YOUR_BING_KEY_HERE}` with your Bing spell checker key.

```
{
  "query": "Book a flite to London?",
  "alteredQuery": "Book a flight to London?",
  "topScoringIntent": {
    "intent": "BookFlight",
    "score": 0.780123
  },
  "entities": []
}
```

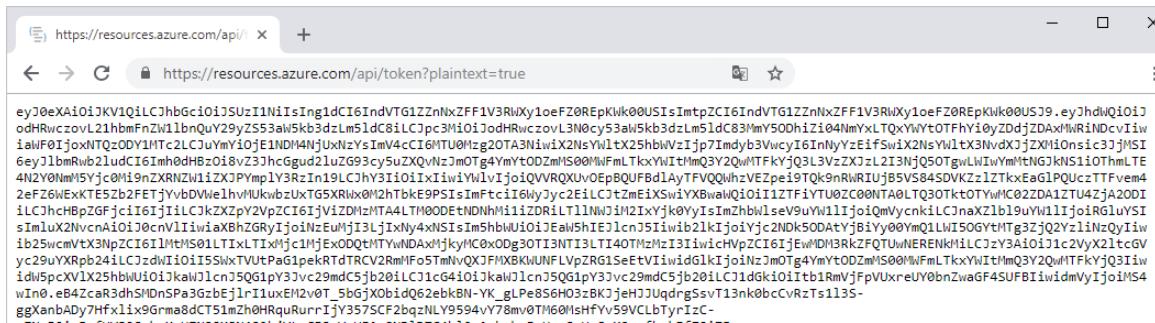
Publishing regions

Learn more about publishing [regions](#) including publishing in [Europe](#), and [Australia](#). Publishing regions are different from authoring regions. Create an app in the authoring region corresponding to the publishing region you want for the query endpoint.

Assign resource without LUIS portal

For automation purposes such as a CI/CD pipeline, you may want to automate the assignment of a LUIS resource to a LUIS app. In order to do that, you need to perform the following steps:

1. Get an Azure Resource Manager token from this [website](#). This token does expire so use it immediately.
The request returns an Azure Resource Manager token.



2. Use the token to request the LUIS resources across subscriptions, from the [Get LUIS azure accounts API](#), your user account has access to.

This POST API requires the following settings:

HEADER	VALUE
Authorization	The value of Authorization is Bearer {token}. Notice that the token value must be preceded by the word Bearer and a space.
Ocp-Apim-Subscription-Key	Your authoring key.

This API returns an array of JSON objects of your LUIS subscriptions including subscription ID, resource group, and resource name, returned as account name. Find the one item in the array that is the LUIS resource to assign to the LUIS app.

3. Assign the token to the LUIS resource with the [Assign a LUIS azure accounts to an application API](#).

This POST API requires the following settings:

TYPE	SETTING	VALUE
Header	Authorization	The value of Authorization is Bearer {token}. Notice that the token value must be preceded by the word Bearer and a space.
Header	Ocp-Apim-Subscription-Key	Your authoring key.
Header	Content-type	application/json
Querystring	appid	The LUIS app ID.
Body		{"AzureSubscriptionId": "ddda2925-af7f-4b05-9ba1-2155c5fe8a8e", "ResourceGroup": "resourcegroup-2", "AccountName": "luis-uswest-S0-2"}

When this API is successful, it returns a 201 - created status.

Change pricing tier

1. In [Azure](#), find your LUIS subscription. Select the LUIS subscription.

Microsoft Azure All resources

All resources (Default Directory)

+ Add Assign Tags Columns Refresh Delete

Subscriptions: Pay-As-You-Go

Filter by name... All resource groups Cognitive Services All locations No grouping

1 items

NAME	TYPE	RESOURCE GROUP	LOCATION	SUBSCRIPTION
LUIS	Cognitive Services	LUIS	West US	Pay-As-You-Go

New

- Dashboard
- All resources
- Resource groups
- App Services
- SQL databases
- SQL data warehouses
- Azure Cosmos DB
- Virtual machines
- Load balancers
- Storage accounts
- Virtual networks
- Azure Active Directory
- More services >

2. Select **Pricing tier** in order to see the available pricing tiers.

Microsoft Azure LUIS

LUIS Cognitive Services

Search (Ctrl+ /)

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Resource management Keys Quick start Pricing tier Properties Locks Automation script

Monitoring Total Calls and Total Errors

100
80
60
40
20
0

Delete

Essentials

Resource group (change)
LUIS

Status
Active

Location
West US

Subscription name (change)
Pay-As-You-Go

API type
Language Understanding Intelligent Service (I)

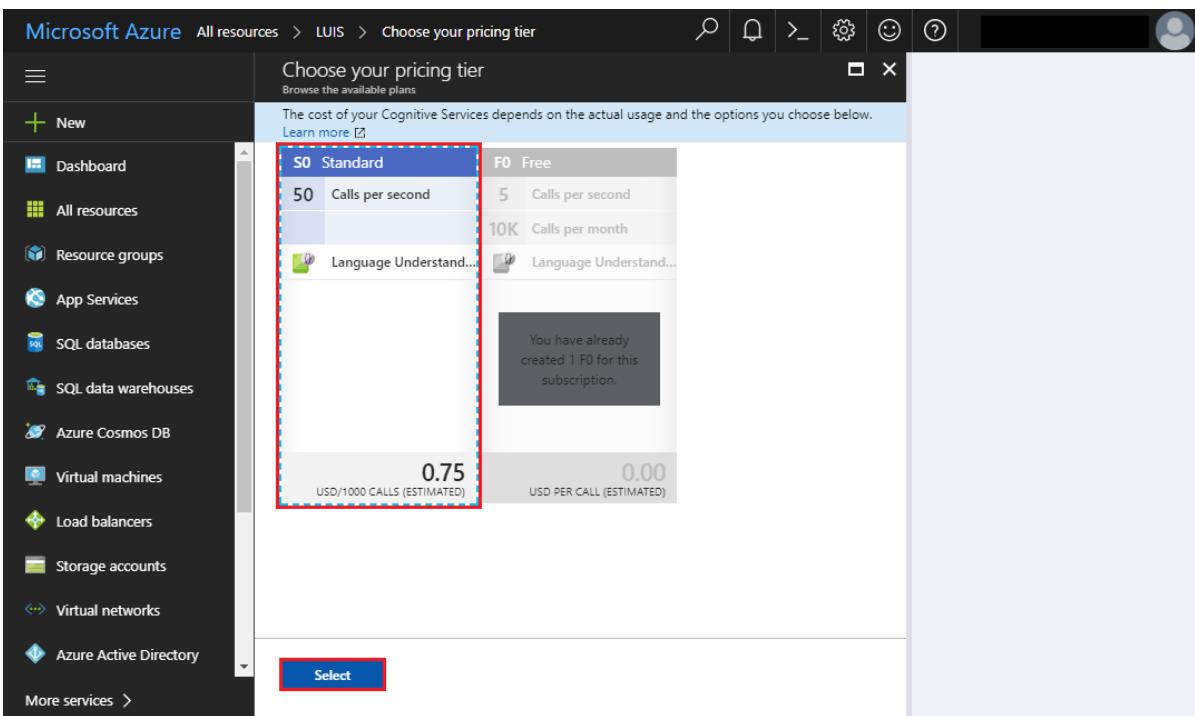
Pricing tier
Free

Endpoint
https://westus.api.cognitive.microsoft.com/luis

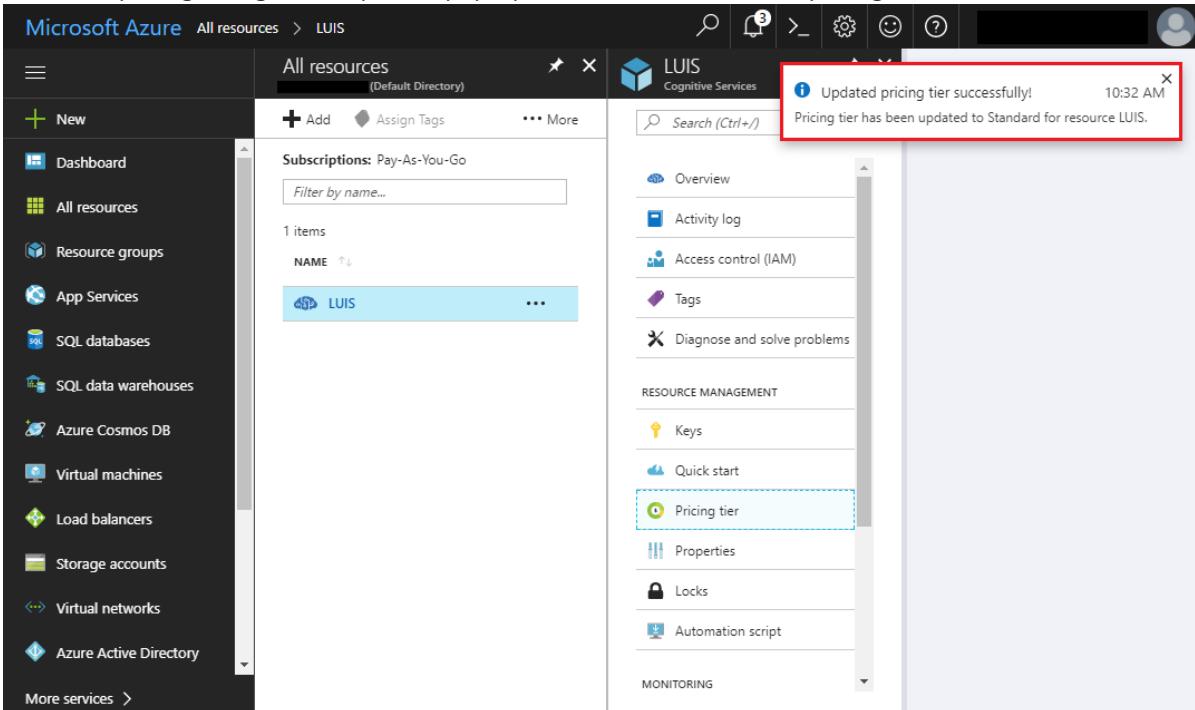
Manage keys
Show access keys ...

Subscription ID

3. Select the pricing tier and select **Select** to save your change.



- When the pricing change is complete, a pop-up window verifies the new pricing tier.



- Remember to [assign this endpoint key](#) on the **Publish** page and use it in all endpoint queries.

Fix HTTP status code 403 and 429

You get 403 and 429 error status codes when you exceed the transactions per second or transactions per month for your pricing tier.

When you receive an HTTP 403 error status code

When you use all those free 1000 endpoint queries or you exceed your pricing tier's monthly transactions quota, you receive an HTTP 403 error status code.

To fix this error, you need to either [change your pricing tier](#) to a higher tier or [create a new resource](#) and assign it to your app.

Solutions for this error include:

- In the [Azure portal](#), on your Language Understanding resource, on the **Resource Management -> Pricing tier**, change your pricing tier to a higher TPS tier. You don't need to do anything in the Language Understanding portal if your resource is already assigned to your Language Understanding app.
- If your usage exceeds the highest pricing tier, add more Language Understanding resources with a load balancer in front of them. The [Language Understanding container](#) with Kubernetes or Docker Compose can help with this.

When you receive an HTTP 429 error status code

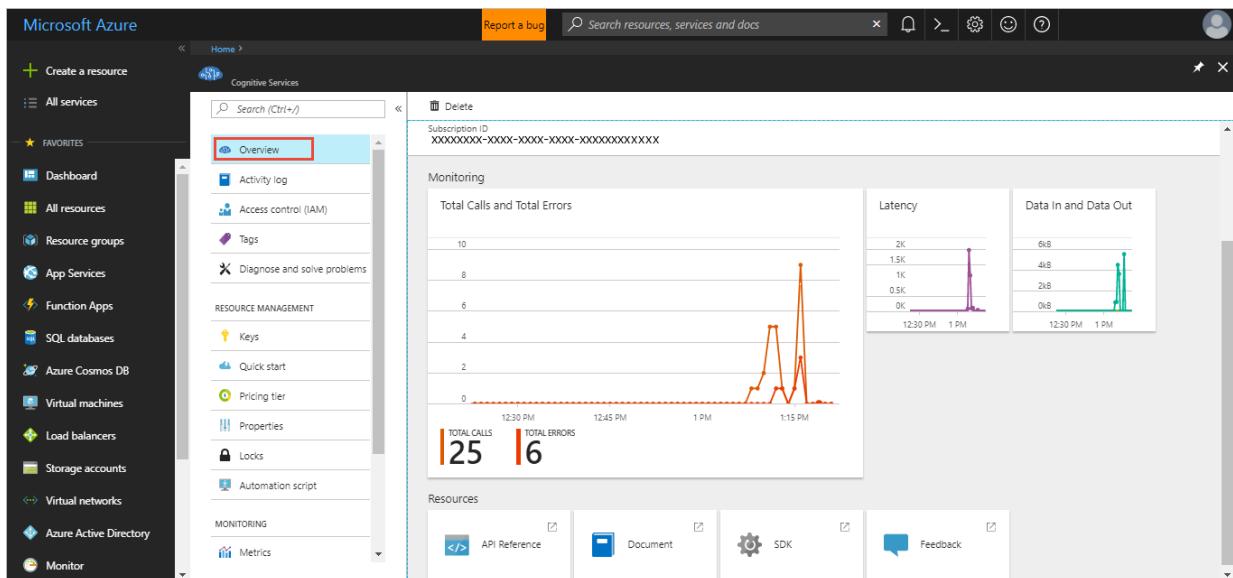
This status code is returned when your transactions per second exceeds your pricing tier.

Solutions include:

- You can [increase your pricing tier](#), if you are not at the highest tier.
- If your usage exceeds the highest pricing tier, add more Language Understanding resources with a load balancer in front of them. The [Language Understanding container](#) with Kubernetes or Docker Compose can help with this.
- You can gate your client application requests with a [retry policy](#) you implement yourself when you get this status code.

Viewing summary usage

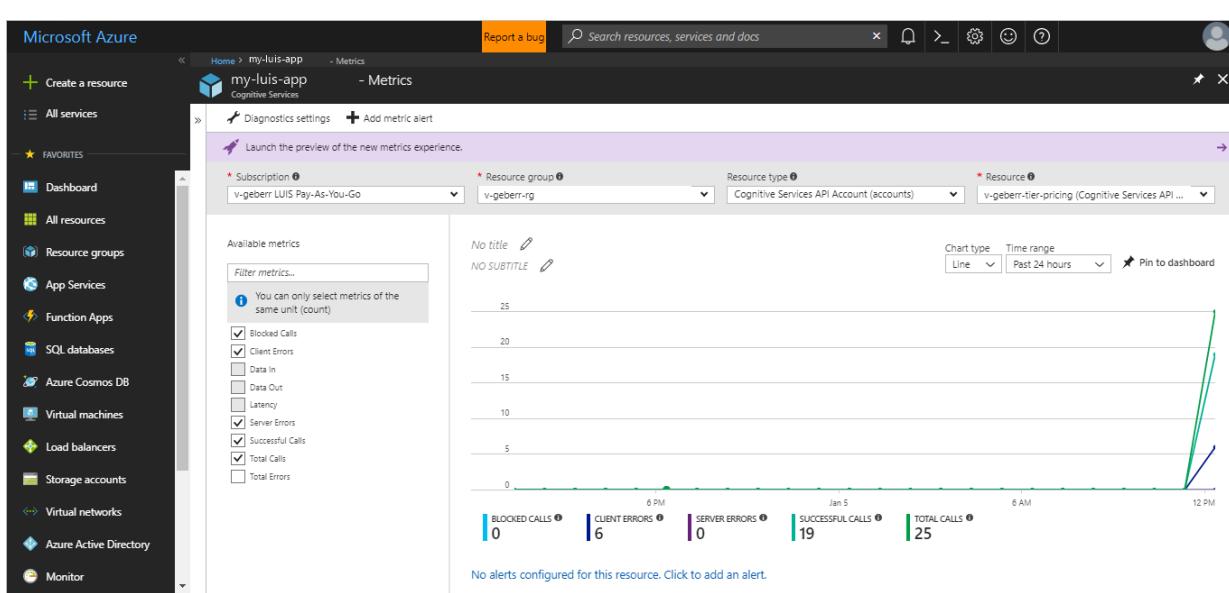
You can view LUIS usage information in Azure. The **Overview** page shows recent summary information including calls and errors. If you make a LUIS endpoint request, then immediately watch the **Overview page**, allow up to five minutes for the usage to show up.



Customizing usage charts

Metrics provides a more detailed view into the data.

You can configure your metrics charts for time period and metric type.



Total transactions threshold alert

If you would like to know when you have reached a certain transaction threshold, for example 10,000 transactions, you can create an alert.

Add a metric alert for the **total calls** metric for a certain time period. Add email addresses of all people that should receive the alert. Add webhooks for all systems that should receive the alert. You can also run a logic app when the alert is triggered.

Next steps

Learn how to use [versions](#) to manage changes to your LUIS app.

Use versions to edit and test without impacting staging or production apps

8/9/2019 • 2 minutes to read • [Edit Online](#)

Versions allow you to build and publish different models. A good practice is to clone the current active model to a different [version](#) of the app before making changes to the model.

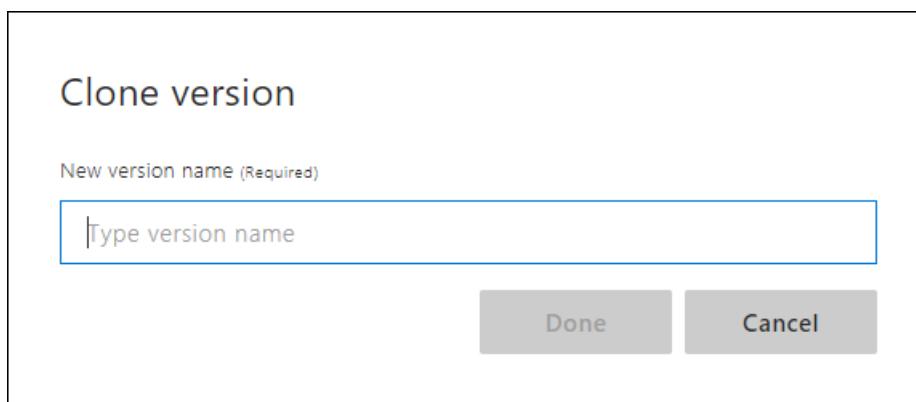
To work with versions, open your app by selecting its name on **My Apps** page, and then select **Manage** in the top bar, then select **Versions** in the left navigation.

The list of versions shows which versions are published, where they are published, and which version is currently active.

Version name	Published date	Created	Last modified
0.4 (Active)	Not published yet	8/1/18	8/1/18
0.3 (Production)	8/1/18	8/1/18	8/1/18
0.2 (Staging)	8/1/18	8/1/18	8/1/18
0.1	8/1/18	8/1/18	8/1/18

Clone a version

1. Select the version you want to clone then select **Clone** from the toolbar.
2. In the **Clone version** dialog box, type a name for the new version such as "0.2".



NOTE

Version ID can consist only of characters, digits or '.' and cannot be longer than 10 characters.

A new version with the specified name is created and set as the active version.

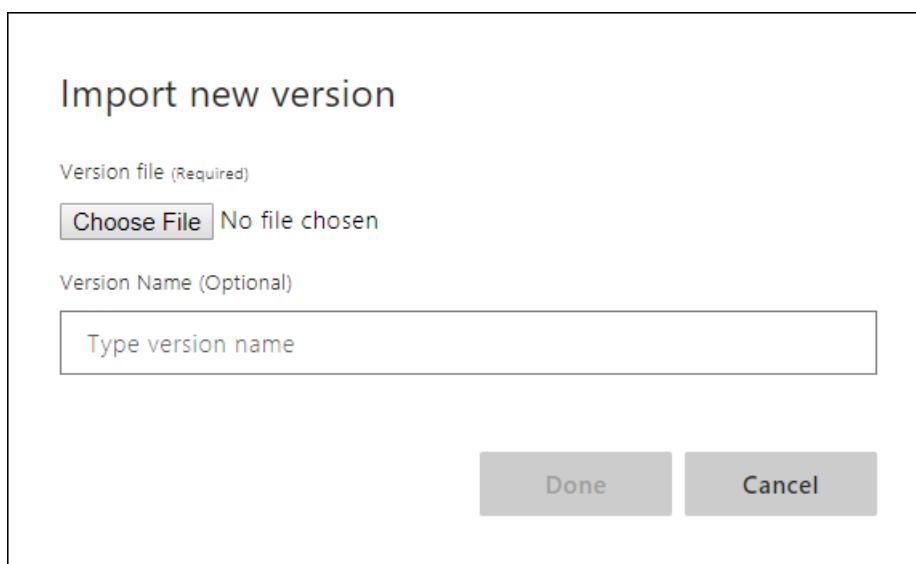
Set active version

Select a version from the list, then select **Make Active** from the toolbar.

<div style="border: 1px solid #ccc; padding: 5px;"> Application Information Keys and Endpoints Publish Settings Versions Collaborators </div>	<h2>Versions <small>?</small></h2> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;"> <input checked="" type="checkbox"/> Make active <input type="checkbox"/> Rename <input type="checkbox"/> Clone <input type="checkbox"/> Export app <input type="checkbox"/> Delete All <small>▼</small> <input style="width: 150px; height: 15px; border: 1px solid #ccc; margin-left: 10px;" type="text" value="Search for version(s)"/> </div> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 20%;">Version name</th><th style="width: 20%;">Published date</th><th style="width: 20%;">Created</th><th style="width: 20%;">Last modified</th></tr> </thead> <tbody> <tr> <td>0.4 (Active)</td><td>Not published yet</td><td>8/1/18</td><td>8/1/18</td></tr> <tr> <td>0.3 (Production)</td><td>8/1/18</td><td>8/1/18</td><td>8/1/18</td></tr> <tr> <td>0.2 (Staging)</td><td>8/1/18</td><td>8/1/18</td><td>8/1/18</td></tr> <tr> <td>0.1</td><td>8/1/18</td><td>8/1/18</td><td>8/1/18</td></tr> </tbody> </table>	Version name	Published date	Created	Last modified	0.4 (Active)	Not published yet	8/1/18	8/1/18	0.3 (Production)	8/1/18	8/1/18	8/1/18	0.2 (Staging)	8/1/18	8/1/18	8/1/18	0.1	8/1/18	8/1/18	8/1/18
Version name	Published date	Created	Last modified																		
0.4 (Active)	Not published yet	8/1/18	8/1/18																		
0.3 (Production)	8/1/18	8/1/18	8/1/18																		
0.2 (Staging)	8/1/18	8/1/18	8/1/18																		
0.1	8/1/18	8/1/18	8/1/18																		

Import version

1. Select **Import version** from the toolbar.
2. In the **Import new version** pop-up window, enter the new ten character version name. You only need to set a version ID if the version in the JSON file already exists in the app.



Once you import a version, the new version becomes the active version.

Import errors

- Tokenizer errors: If you get a **tokenizer error** when importing, you are trying to import a version that uses a different **tokenizer** than the app currently uses. To fix this, see [Migrating between tokenizer versions](#).

Other actions

- To **delete** a version, select a version from the list, then select **Delete** from the toolbar. Select **Ok**.
- To **rename** a version, select a version from the list, then select **Rename** from the toolbar. Enter new name and select **Done**.
- To **export** a version, select a version from the list, then select **Export app** from the toolbar. Choose JSON to export for backup, choose **Export for container** to use this app in a LUIS container.

How to manage authors and collaborators

8/9/2019 • 2 minutes to read • [Edit Online](#)

An app owner can add collaborators to the app. These collaborators can modify the model, train, and publish the app.

Add collaborator

An app has a single author, the owner, but can have many collaborators. To allow collaborators to edit your LUIS app, you must add the email they use to access the LUIS portal to the collaborators list. Once they are added, the app shows in their LUIS portal.

1. Select **Manage** from the top right menu, then select **Collaborators** in the left menu.
2. Select **Add Collaborator** from the toolbar.

The screenshot shows the 'Collaborators' section of the LUIS application management interface. On the left, a sidebar lists 'Application Information', 'Keys and Endpoints', 'Publish Settings', 'Versions', and 'Collaborators'. The 'Collaborators' item is highlighted with a blue background. The main area is titled 'Users' with a 'Owner' section containing the email 'Hugo.Linville@mycompany.outlook.com'. Below it is a 'Collaborators' section with a button '+ Add Collaborator' and a search bar 'Search for collaborator(s)'. A note at the bottom says 'You have no collaborators on this app.'

3. Enter the email address the collaborator uses to sign in to the LUIS portal.

The screenshot shows the 'Add Collaborator' dialog box. It has a title 'Add Collaborator' and a field labeled 'Email (Required)' containing the email 'Morris.Tolley@mycompany.outlook.com'. At the bottom are two buttons: a blue 'Add' button and a grey 'Cancel' button.

Transfer of ownership

While LUIS doesn't currently support transfer of ownership, you can export your app, and another LUIS user can import the app. There may be minor differences in LUIS scores between the two applications.

Azure Active Directory resources

If you use [Azure Active Directory](#) (Azure AD) in your organization, Language Understanding (LUIS) needs

permission to the information about your users' access when they want to use LUIS. The resources that LUIS requires are minimal.

You see the detailed description when you attempt to sign up with an account that has admin consent or does not require admin consent, such as administrator consent:

- Allows you to sign in to the app with your organizational account and let the app read your profile. It also allows the app to read basic company information. This gives LUIS permission to read basic profile data, such as user ID, email, name
- Allows the app to see and update your data, even when you are not currently using the app. The permission is required to refresh the access token of the user.

Azure Active Directory tenant user

LUIS uses standard Azure Active Directory (Azure AD) consent flow.

The tenant admin should work directly with the user who needs access granted to use LUIS in the Azure AD.

- First, the user signs into LUIS, and sees the pop-up dialog needing admin approval. The user contacts the tenant admin before continuing.
- Second, the tenant admin signs into LUIS, and sees a consent flow pop-up dialog. This is the dialog the admin needs to give permission for the user. Once the admin accepts the permission, the user is able to continue with LUIS. If the tenant admin will not sign in to LUIS, the admin can access [consent](#) for LUIS, shown in the following screenshot. Notice the list is filtered to items that include the name **Luis**.

The screenshot shows the Microsoft Azure Active Directory Applications page. At the top, there's a search bar with the text "luis". Below the search bar, there are two application cards highlighted with red boxes:

- Europe.Luis.ai**: This card is located under the "Groups" section and has a blue icon with a white cube.
- Australia.Luis.ai** and **luis.ai.live**: These cards are located under the "LUIS-admin" section and have blue icons with white cubes.

At the bottom of the page, there's a link "Didn't find what you were looking for? [Add an app.](#)".

If the tenant admin only wants certain users to use LUIS, there are a couple of possible solutions:

- Giving the "admin consent" (consent to all users of the Azure AD), but then set to "Yes" the "User assignment required" under Enterprise Application Properties, and finally assign/add only the wanted users to the Application. With this method, the Administrator is still providing "admin consent" to the App, however, it's possible to control the users that can access it.
- A second solution, is by using [Azure AD Graph API](#) to provide consent to each specific user.

Learn more about Azure active directory users and consent:

- [Restrict your app](#) to a set of users

User accounts with multiple emails for collaborators

If you add collaborators to a LUIS app, you are specifying the exact email address a collaborator needs to use LUIS as a collaborator. While Azure Active Directory (Azure AD) allows a single user to have more than one email account used interchangeably, LUIS requires the user to sign in with the email address specified in the collaborator's list.

Manage account and authoring key

8/8/2019 • 2 minutes to read • [Edit Online](#)

The two key pieces of information for a LUIS account are the user account and the authoring key. Your login information is managed at account.microsoft.com. Your authoring key is managed from the [LUIS](#) portal **Settings** page.

Authoring key

This single, region-specific authoring key, on the **Settings** page, allows you to author all your apps from the [LUIS](#) portal as well as the [authoring APIs](#). As a convenience, the authoring key is allowed to make a [limited](#) number of endpoint queries each month.

The screenshot shows the LUIS User Settings Page. At the top, there's a navigation bar with links for Language Understanding, My apps, Docs, Pricing, Support, and About. On the right, it shows the user's name, Patti Owens. Below the navigation bar, the page title is "User settings".

Authoring Key

A large text input field contains the placeholder "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" with a "(Reset)" link and a help icon. Below this is a section titled "Other settings".

Country/Region (Required)

An empty text input field.

Company

An empty text input field with placeholder text "Enter company name".

Where did you hear about us?

An empty text input field with placeholder text "Other".

Contact me with promotional offers and updates about Cognitive Services.

Save changes | **Export user data**

Danger Zone

Delete your account

The authoring key is used for any apps you own as well as any apps you are listed as a collaborator.

Authoring key regions

The authoring key is specific to the [authoring region](#). The key does not work in a different region.

Reset authoring key

If your authoring key is compromised, reset the key. The key is reset on all your apps in the [LUIS](#) portal. If you author your apps via the authoring APIs, you need to change the value of `Ocp-Apim-Subscription-Key` to the new key.

Delete account

See [Data storage and removal](#) for information about what data is deleted when you delete your account.

Next steps

Learn more about your [authoring key](#).

Use Microsoft Azure Traffic Manager to manage endpoint quota across keys

8/20/2019 • 12 minutes to read • [Edit Online](#)

Language Understanding (LUIS) offers the ability to increase the endpoint request quota beyond a single key's quota. This is done by creating more keys for LUIS and adding them to the LUIS application on the **Publish** page in the **Resources and Keys** section.

The client-application has to manage the traffic across the keys. LUIS doesn't do that.

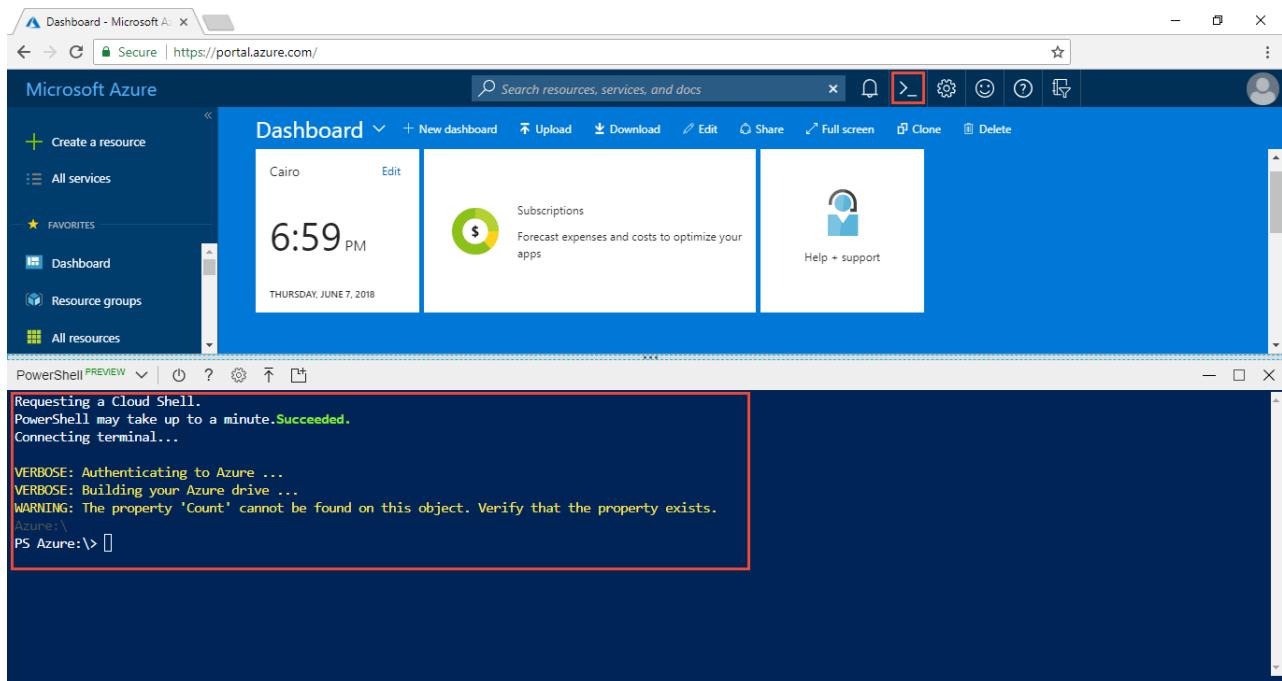
This article explains how to manage the traffic across keys with Azure [Traffic Manager](#). You must already have a trained and published LUIS app. If you do not have one, follow the Prebuilt domain [quickstart](#).

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

Connect to PowerShell in the Azure portal

In the [Azure](#) portal, open the PowerShell window. The icon for the PowerShell window is the >_ in the top navigation bar. By using PowerShell from the portal, you get the latest PowerShell version and you are authenticated. PowerShell in the portal requires an [Azure Storage](#) account.



The following sections use [Traffic Manager PowerShell cmdlets](#).

Create Azure resource group with PowerShell

Before creating the Azure resources, create a resource group to contain all the resources. Name the resource group

`luis-traffic-manager` and use the region is `West US`. The region of the resource group stores metadata about the group. It won't slow down your resources if they are in another region.

Create resource group with **New-AzResourceGroup** cmdlet:

```
New-AzResourceGroup -Name luis-traffic-manager -Location "West US"
```

Create LUIS keys to increase total endpoint quota

1. In the Azure portal, create two **Language Understanding** keys, one in the `West US` and one in the `East US`. Use the existing resource group, created in the previous section, named `luis-traffic-manager`.

NAME	TYPE	RESOURCE GROUP	LOCATION	SUBSCRIPTION
luis-tm-east	Cognitive Services	luis-traffic-manager	East US	Pay-As-You-Go
luis-tm-west	Cognitive Services	luis-traffic-manager	West US	Pay-As-You-Go

2. In the [LUIS](#) website, in the **Manage** section, on the **Keys and endpoints** page, assign keys to the app, and republish the app by selecting the **Publish** button in the top right menu.

The example URL in the **endpoint** column uses a GET request with the endpoint key as a query parameter. Copy the two new keys' endpoint URLs. They are used as part of the Traffic Manager configuration later in this article.

Manage LUIS endpoint requests across keys with Traffic Manager

Traffic Manager creates a new DNS access point for your endpoints. It does not act as a gateway or proxy but strictly at the DNS level. This example doesn't change any DNS records. It uses a DNS library to communicate with Traffic Manager to get the correct endpoint for that specific request. *Each* request intended for LUIS first requires a Traffic Manager request to determine which LUIS endpoint to use.

Polling uses LUIS endpoint

Traffic Manager polls the endpoints periodically to make sure the endpoint is still available. The Traffic Manager URL polled needs to be accessible with a GET request and return a 200. The endpoint URL on the **Publish** page does this. Since each endpoint key has a different route and query string parameters, each endpoint key needs a different polling path. Each time Traffic Manager polls, it does cost a quota request. The query string parameter `q` of the LUIS endpoint is the utterance sent to LUIS. This parameter, instead of sending an utterance, is used to add Traffic Manager polling to the LUIS endpoint log as a debugging technique while getting Traffic Manager configured.

Because each LUIS endpoint needs its own path, it needs its own Traffic Manager profile. In order to manage across profiles, create a [nested Traffic Manager](#) architecture. One parent profile points to the children profiles and manage traffic across them.

Once the Traffic Manager is configured, remember to change the path to use the `logging=false` query string parameter so your log is not filling up with polling.

Configure Traffic Manager with nested profiles

The following sections create two child profiles, one for the East LUIS key and one for the West LUIS key. Then a parent profile is created and the two child profiles are added to the parent profile.

Create the East US Traffic Manager profile with PowerShell

To create the East US Traffic Manager profile, there are several steps: create profile, add endpoint, and set endpoint. A Traffic Manager profile can have many endpoints but each endpoint has the same validation path. Because the LUIS endpoint URLs for the east and west subscriptions are different due to region and endpoint key, each LUIS endpoint has to be a single endpoint in the profile.

1. Create profile with [New-AzTrafficManagerProfile](#) cmdlet

Use the following cmdlet to create the profile. Make sure to change the `<appIdLuis>` and `<subscriptionKeyLuis>`. The subscriptionKey is for the East US LUIS key. If the path is not correct, including the LUIS app ID and endpoint key, the Traffic Manager polling is a status of `degraded` because Traffic Manager can't successfully request the LUIS endpoint. Make sure the value of `<q>` is `traffic-manager-east` so you can see this value in the LUIS endpoint logs.

```
$eastprofile = New-AzTrafficManagerProfile -Name luis-profile-eastus -ResourceGroupName luis-traffic-manager -TrafficRoutingMethod Performance -RelativeDnsName luis-dns-eastus -Ttl 30 -MonitorProtocol HTTPS -MonitorPort 443 -MonitorPath "/luis/v2.0/apps/<appID>?subscription-key=<subscriptionKey>&q=traffic-manager-east"
```

This table explains each variable in the cmdlet:

CONFIGURATION PARAMETER	VARIABLE NAME OR VALUE	PURPOSE
-Name	luis-profile-eastus	Traffic Manager name in Azure portal
-ResourceGroupName	luis-traffic-manager	Created in previous section
-TrafficRoutingMethod	Performance	For more information, see Traffic Manager routing methods . If using performance, the URL request to the Traffic Manager must come from the region of the user. If going through a chatbot or other application, it is the chatbot's responsibility to mimic the region in the call to the Traffic Manager.
-RelativeDnsName	luis-dns-eastus	This is the subdomain for the service: luis-dns-eastus.trafficmanager.net
-Ttl	30	Polling interval, 30 seconds
-MonitorProtocol -MonitorPort	HTTPS 443	Port and protocol for LUIS is HTTPS/443
-MonitorPath	/luis/v2.0/apps/<appIDLuis>?subscription-key=<subscriptionKeyLuis>&q=traffic-manager-east	Replace <code><appIdLuis></code> and <code><subscriptionKeyLuis></code> with your own values.

A successful request has no response.

2. Add East US endpoint with [Add-AzTrafficManagerEndpointConfig](#) cmdlet

```
Add-AzTrafficManagerEndpointConfig -EndpointName luis-east-endpoint -TrafficManagerProfile $eastprofile
-Type ExternalEndpoints -Target eastus.api.cognitive.microsoft.com -EndpointLocation "eastus" -
EndpointStatus Enabled
```

This table explains each variable in the cmdlet:

CONFIGURATION PARAMETER	VARIABLE NAME OR VALUE	PURPOSE
-EndpointName	luis-east-endpoint	Endpoint name displayed under the profile
-TrafficManagerProfile	\$eastprofile	Use profile object created in Step 1
-Type	ExternalEndpoints	For more information, see Traffic Manager endpoint
-Target	eastus.api.cognitive.microsoft.com	This is the domain for the LUIS endpoint.
-EndpointLocation	"eastus"	Region of the endpoint
-EndpointStatus	Enabled	Enable endpoint when it is created

The successful response looks like:

```
Id : /subscriptions/<azure-subscription-id>/resourceGroups/luis-traffic-
manager/providers/Microsoft.Network/trafficManagerProfiles/luis-profile-eastus
Name : luis-profile-eastus
ResourceGroupName : luis-traffic-manager
RelativeDnsName : luis-dns-eastus
Ttl : 30
ProfileStatus : Enabled
TrafficRoutingMethod : Performance
MonitorProtocol : HTTPS
MonitorPort : 443
MonitorPath : /luis/v2.0/apps/<luis-app-id>?subscription-
key=f0517d185bcf467cba5147d6260bb868&q=traffic-manager-east
MonitorIntervalInSeconds : 30
MonitorTimeoutInSeconds : 10
MonitorToleratedNumberOfFailures : 3
Endpoints : {luis-east-endpoint}
```

3. Set East US endpoint with [Set-AzTrafficManagerProfile](#) cmdlet

```
Set-AzTrafficManagerProfile -TrafficManagerProfile $eastprofile
```

A successful response will be the same response as step 2.

Create the West US Traffic Manager profile with PowerShell

To create the West US Traffic Manager profile, follow the same steps: create profile, add endpoint, and set endpoint.

1. Create profile with [New-AzTrafficManagerProfile](#) cmdlet

Use the following cmdlet to create the profile. Make sure to change the `appIdLuis` and `subscriptionKeyLuis`. The subscriptionKey is for the East US LUIS key. If the path is not correct including the LUIS app ID and endpoint key, the Traffic Manager polling is a status of `degraded` because Traffic Manager can't successfully

request the LUIS endpoint. Make sure the value of `q` is `traffic-manager-west` so you can see this value in the LUIS endpoint logs.

```
$westprofile = New-AzTrafficManagerProfile -Name luis-profile-westus -ResourceGroupName luis-traffic-
manager -TrafficRoutingMethod Performance -RelativeDnsName luis-dns-westus -Ttl 30 -MonitorProtocol
HTTPS -MonitorPort 443 -MonitorPath "/luis/v2.0/apps/<appIdLuis>?subscription-key=
<subscriptionKeyLuis>&q=traffic-manager-west"
```

This table explains each variable in the cmdlet:

CONFIGURATION PARAMETER	VARIABLE NAME OR VALUE	PURPOSE
-Name	luis-profile-westus	Traffic Manager name in Azure portal
-ResourceGroupName	luis-traffic-manager	Created in previous section
-TrafficRoutingMethod	Performance	For more information, see Traffic Manager routing methods . If using performance, the URL request to the Traffic Manager must come from the region of the user. If going through a chatbot or other application, it is the chatbot's responsibility to mimic the region in the call to the Traffic Manager.
-RelativeDnsName	luis-dns-westus	This is the subdomain for the service: luis-dns-westus.trafficmanager.net
-Ttl	30	Polling interval, 30 seconds
-MonitorProtocol -MonitorPort	HTTPS 443	Port and protocol for LUIS is HTTPS/443
-MonitorPath	/luis/v2.0/apps/<appIdLuis>?subscription-key=<subscriptionKeyLuis>&q=traffic- manager-west	Replace <code><appId></code> and <code><subscriptionKey></code> with your own values. Remember this endpoint key is different than the east endpoint key

A successful request has no response.

2. Add West US endpoint with [Add-AzTrafficManagerEndpointConfig](#) cmdlet

```
Add-AzTrafficManagerEndpointConfig -EndpointName luis-west-endpoint -TrafficManagerProfile $westprofile
-Type ExternalEndpoints -Target westus.api.cognitive.microsoft.com -EndpointLocation "westus" -
EndpointStatus Enabled
```

This table explains each variable in the cmdlet:

CONFIGURATION PARAMETER	VARIABLE NAME OR VALUE	PURPOSE
-EndpointName	luis-west-endpoint	Endpoint name displayed under the profile
-TrafficManagerProfile	\$westprofile	Use profile object created in Step 1

CONFIGURATION PARAMETER	VARIABLE NAME OR VALUE	PURPOSE
-Type	ExternalEndpoints	For more information, see Traffic Manager endpoint
-Target	westus.api.cognitive.microsoft.com	This is the domain for the LUIS endpoint.
-EndpointLocation	"westus"	Region of the endpoint
-EndpointStatus	Enabled	Enable endpoint when it is created

The successful response looks like:

```

Id : /subscriptions/<azure-subscription-id>/resourceGroups/luis-traffic-
manager/providers/Microsoft.Network/trafficManagerProfiles/luis-profile-westus
Name : luis-profile-westus
ResourceGroupName : luis-traffic-manager
RelativeDnsName : luis-dns-westus
Ttl : 30
ProfileStatus : Enabled
TrafficRoutingMethod : Performance
MonitorProtocol : HTTPS
MonitorPort : 443
MonitorPath : /luis/v2.0/apps/c3fc5d1e-5187-40cc-af0f-fbde328aa16b?subscription-
key=e3605f07e3cc4bedb7e02698a54c19cc&q=traffic-manager-west
MonitorIntervalInSeconds : 30
MonitorTimeoutInSeconds : 10
MonitorToleratedNumberOfFailures : 3
Endpoints : {luis-west-endpoint}

```

3. Set West US endpoint with [Set-AzTrafficManagerProfile](#) cmdlet

```
Set-AzTrafficManagerProfile -TrafficManagerProfile $westprofile
```

A successful response is the same response as step 2.

Create parent Traffic Manager profile

Create the parent Traffic Manager profile and link two child Traffic Manager profiles to the parent.

1. Create parent profile with [New-AzTrafficManagerProfile](#) cmdlet

```
$parentprofile = New-AzTrafficManagerProfile -Name luis-profile-parent -ResourceGroupName luis-traffic-
manager -TrafficRoutingMethod Performance -RelativeDnsName luis-dns-parent -Ttl 30 -MonitorProtocol
HTTPS -MonitorPort 443 -MonitorPath "/"
```

This table explains each variable in the cmdlet:

CONFIGURATION PARAMETER	VARIABLE NAME OR VALUE	PURPOSE
-Name	luis-profile-parent	Traffic Manager name in Azure portal
-ResourceGroupName	luis-traffic-manager	Created in previous section

CONFIGURATION PARAMETER	VARIABLE NAME OR VALUE	PURPOSE
-TrafficRoutingMethod	Performance	For more information, see Traffic Manager routing methods . If using performance, the URL request to the Traffic Manager must come from the region of the user. If going through a chatbot or other application, it is the chatbot's responsibility to mimic the region in the call to the Traffic Manager.
-RelativeDnsName	luis-dns-parent	This is the subdomain for the service: luis-dns-parent.trafficmanager.net
-Ttl	30	Polling interval, 30 seconds
-MonitorProtocol -MonitorPort	HTTPS 443	Port and protocol for LUIS is HTTPS/443
-MonitorPath	/	This path doesn't matter because the child endpoint paths are used instead.

A successful request has no response.

- Add East US child profile to parent with [Add-AzTrafficManagerEndpointConfig](#) and **NestedEndpoints** type

```
Add-AzTrafficManagerEndpointConfig -EndpointName child-endpoint-useast -TrafficManagerProfile $parentprofile -Type NestedEndpoints -TargetResourceId $eastprofile.Id -EndpointStatus Enabled -EndpointLocation "eastus" -MinChildEndpoints 1
```

This table explains each variable in the cmdlet:

CONFIGURATION PARAMETER	VARIABLE NAME OR VALUE	PURPOSE
-EndpointName	child-endpoint-useast	East profile
-TrafficManagerProfile	\$parentprofile	Profile to assign this endpoint to
-Type	NestedEndpoints	For more information, see Add-AzTrafficManagerEndpointConfig .
-TargetResourceId	\$eastprofile.Id	ID of the child profile
-EndpointStatus	Enabled	Endpoint status after adding to parent
-EndpointLocation	"eastus"	Azure region name of resource
-MinChildEndpoints	1	Minimum number to child endpoints

The successful response look like the following and includes the new `child-endpoint-useast` endpoint:

```

Id : /subscriptions/<azure-subscription-id>/resourceGroups/luis-traffic-
manager/providers/Microsoft.Network/trafficManagerProfiles/luis-profile-parent
Name : luis-profile-parent
ResourceGroupName : luis-traffic-manager
RelativeDnsName : luis-dns-parent
Ttl : 30
ProfileStatus : Enabled
TrafficRoutingMethod : Performance
MonitorProtocol : HTTPS
MonitorPort : 443
MonitorPath : /
MonitorIntervalInSeconds : 30
MonitorTimeoutInSeconds : 10
MonitorToleratedNumberOfFailures : 3
Endpoints : {child-endpoint-useast}

```

3. Add West US child profile to parent with [Add-AzTrafficManagerEndpointConfig](#) cmdlet and **NestedEndpoints** type

```

Add-AzTrafficManagerEndpointConfig -EndpointName child-endpoint-uswest -TrafficManagerProfile
$parentprofile -Type NestedEndpoints -TargetResourceId $westprofile.Id -EndpointStatus Enabled -
EndpointLocation "westus" -MinChildEndpoints 1

```

This table explains each variable in the cmdlet:

CONFIGURATION PARAMETER	VARIABLE NAME OR VALUE	PURPOSE
-EndpointName	child-endpoint-uswest	West profile
-TrafficManagerProfile	\$parentprofile	Profile to assign this endpoint to
-Type	NestedEndpoints	For more information, see Add-AzTrafficManagerEndpointConfig .
-TargetResourceId	\$westprofile.Id	ID of the child profile
-EndpointStatus	Enabled	Endpoint status after adding to parent
-EndpointLocation	"westus"	Azure region name of resource
-MinChildEndpoints	1	Minimum number to child endpoints

The successful response look like and includes both the previous `child-endpoint-useast` endpoint and the new `child-endpoint-uswest` endpoint:

```

Id : /subscriptions/<azure-subscription-id>/resourceGroups/luis-traffic-
manager/providers/Microsoft.Network/trafficManagerProfiles/luis-profile-parent
Name : luis-profile-parent
ResourceGroupName : luis-traffic-manager
RelativeDnsName : luis-dns-parent
Ttl : 30
ProfileStatus : Enabled
TrafficRoutingMethod : Performance
MonitorProtocol : HTTPS
MonitorPort : 443
MonitorPath : /
MonitorIntervalInSeconds : 30
MonitorTimeoutInSeconds : 10
MonitorToleratedNumberOfFailures : 3
Endpoints : {child-endpoint-useast, child-endpoint-uswest}

```

4. Set endpoints with [Set-AzTrafficManagerProfile](#) cmdlet

```
Set-AzTrafficManagerProfile -TrafficManagerProfile $parentprofile
```

A successful response is the same response as step 3.

PowerShell variables

In the previous sections, three PowerShell variables were created: `$eastprofile`, `$westprofile`, `$parentprofile`. These variables are used toward the end of the Traffic Manager configuration. If you chose not to create the variables, or forgot to, or your PowerShell window times out, you can use the PowerShell cmdlet, [Get-AzTrafficManagerProfile](#), to get the profile again and assign it to a variable.

Replace the items in angle brackets, `<>`, with the correct values for each of the three profiles you need.

```
$<variable-name> = Get-AzTrafficManagerProfile -Name <profile-name> -ResourceGroupName luis-traffic-manager
```

Verify Traffic Manager works

To verify that the Traffic Manager profiles work, the profiles need to have the status of `Online`. This status is based on the polling path of the endpoint.

View new profiles in the Azure portal

You can verify that all three profiles are created by looking at the resources in the `luis-traffic-manager` resource group.

NAME	TYPE	RESOURCE GROUP	LOCATION	SUBSCRIPTION
luis-profile-eastus	Traffic Manager profile	luis-traffic-manager	global	Pay-As-You-Go
luis-profile-parent	Traffic Manager profile	luis-traffic-manager	global	Pay-As-You-Go
luis-profile-westus	Traffic Manager profile	luis-traffic-manager	global	Pay-As-You-Go
luis-tm-east	Cognitive Services	luis-traffic-manager	East US	Pay-As-You-Go
luis-tm-west	Cognitive Services	luis-traffic-manager	West US	Pay-As-You-Go

Verify the profile status is Online

The Traffic Manager polls the path of each endpoint to make sure it is online. If it is online, the status of the child profiles are **Online**. This is displayed on the **Overview** of each profile.

NAME	STATUS	MONITOR STATUS	TYPE	LOCATION
luis-east-endpoint	Enabled	Online	External endpoint	East US

Validate Traffic Manager polling works

Another way to validate the traffic manager polling works is with the LUIS endpoint logs. On the [LUIS](#) website apps list page, export the endpoint log for the application. Because Traffic Manager polls often for the two endpoints, there are entries in the logs even if they have only been on a few minutes. Remember to look for entries where the query begins with `traffic-manager-`.

```
traffic-manager-west 6/7/2018 19:19 {"query":"traffic-manager-west","intents": [{"intent":"None","score":0.944767}],"entities":[]}
traffic-manager-east 6/7/2018 19:20 {"query":"traffic-manager-east","intents": [{"intent":"None","score":0.944767}],"entities":[]}
```

Validate DNS response from Traffic Manager works

To validate that the DNS response returns a LUIS endpoint, request the Traffic Manager parent profile DNS using a DNS client library. The DNS name for the parent profile is `luis-dns-parent.trafficmanager.net`.

The following Node.js code makes a request for the parent profile and returns a LUIS endpoint:

```
const dns = require('dns');

dns.resolveAny('luis-dns-parent.trafficmanager.net', (err, ret) => {
  console.log('ret', ret);
});
```

The successful response with the LUIS endpoint is:

```
[{
  value: 'westus.api.cognitive.microsoft.com',
  type: 'CNAME'
}]
```

Use the Traffic Manager parent profile

In order to manage traffic across endpoints, you need to insert a call to the Traffic Manager DNS to find the LUIS endpoint. This call is made for every LUIS endpoint request and needs to simulate the geographic location of the user of the LUIS client application. Add the DNS response code in between your LUIS client application and the request to LUIS for the endpoint prediction.

Resolving a degraded state

Enable [diagnostic logs](#) for Traffic Manager to see why endpoint status is degraded.

Clean up

Remove the two LUIS endpoint keys, the three Traffic Manager profiles, and the resource group that contained these five resources. This is done from the Azure portal. You delete the five resources from the resources list. Then delete the resource group.

Next steps

Review [middleware](#) options in BotFramework v4 to understand how this traffic management code can be added to a BotFramework bot.

API v1 to v2 Migration guide for LUIS apps

8/9/2019 • 2 minutes to read • [Edit Online](#)

The version 1 [endpoint](#) and [authoring](#) APIs are deprecated. Use this guide to understand how to migrate to version 2 [endpoint](#) and [authoring](#) APIs.

New Azure regions

Luis has new [regions](#) provided for the Luis APIs. Luis provides a different portal for region groups. The application must be authored in the same region you expect to query. Applications do not automatically migrate regions. You export the app from one region then import into another for it to be available in a new region.

Authoring route changes

The authoring API route changed from using the **prog** route to using the **api** route.

VERSION	ROUTE
1	/luis/v1.0/ prog /apps
2	/luis/ api /v2.0/apps

Endpoint route changes

The endpoint API has new query string parameters as well as a different response. If the verbose flag is true, all intents, regardless of score, are returned in an array named intents, in addition to the topScoringIntent.

VERSION	GET ROUTE
1	/luis/v1/application?ID={appId}&q={q}
2	/luis/v2.0/apps/{appId}?q={q}[&timezoneOffset][&verbose][&spellCheck][&staging][&bing-spell-check-subscription-key][&log]

v1 endpoint success response:

```
{  
  "odata.metadata": "https://dialogice.cloudapp.net/odata/$metadata#domain", "value": [  
    {  
      "id": "bccb84ee-4bd6-4460-a340-0595b12db294", "q": "turn on the camera", "response": "  
[{\\"intent\\":\\"OpenCamera\\", \\"score\\":0.976928055}, {\\"intent\\":\\"None\\", \\"score\\":0.0230718572}]"  
    }  
  ]  
}
```

v2 endpoint success response:

```
{
  "query": "forward to frank 30 dollars through HSBC",
  "topScoringIntent": {
    "intent": "give",
    "score": 0.3964121
  },
  "entities": [
    {
      "entity": "30",
      "type": "builtin.number",
      "startIndex": 17,
      "endIndex": 18,
      "resolution": {
        "value": "30"
      }
    },
    {
      "entity": "frank",
      "type": "frank",
      "startIndex": 11,
      "endIndex": 15,
      "score": 0.935219169
    },
    {
      "entity": "30 dollars",
      "type": "builtin.currency",
      "startIndex": 17,
      "endIndex": 26,
      "resolution": {
        "unit": "Dollar",
        "value": "30"
      }
    },
    {
      "entity": "hsbc",
      "type": "Bank",
      "startIndex": 36,
      "endIndex": 39,
      "resolution": {
        "values": [
          "BankeName"
        ]
      }
    }
  ]
}
```

Key management no longer in API

The subscription endpoint key APIs are deprecated, returning 410 GONE.

VERSION	ROUTE
1	/luis/v1.0/prog/subscriptions
1	/luis/v1.0/prog/subscriptions/{subscriptionKey}

Azure [endpoint keys](#) are generated in the Azure portal. You assign the key to a LUIS app on the [Publish](#) page. You do not need to know the actual key value. LUIS uses the subscription name to make the assignment.

New versioning route

The v2 model is now contained in a [version](#). A version name is 10 characters in the route. The default version is "0.1".

VERSION	ROUTE
1	/luis/v1.0/ prog /apps/{appId}/entities
2	/luis/ api /v2.0/apps/{appId}/ versions / {versionId} /entities

Metadata renamed

Several APIs that return LUIS metadata have new names.

V1 ROUTE NAME	V2 ROUTE NAME
PersonalAssistantApps	assistants
applicationcultures	cultures
applicationdomains	domains
applicationusagescenarios	usagescenarios

"Sample" renamed to "suggest"

Luis suggests utterances from existing [endpoint utterances](#) that may enhance the model. In the previous version, this was named **sample**. In the new version, the name is changed from sample to **suggest**. This is called [Review endpoint utterances](#) in the LUIS website.

VERSION	ROUTE
1	/luis/v1.0/ prog /apps/{appId}/entities/{entityId}/ sample
1	/luis/v1.0/ prog /apps/{appId}/intents/{intentId}/ sample
2	/luis/ api /v2.0/apps/{appId}/ versions / {versionId} /entities/{entityId}/ suggest
2	/luis/ api /v2.0/apps/{appId}/ versions / {versionId} /intents/{intentId}/ suggest

Create app from prebuilt domains

[Prebuilt domains](#) provide a predefined domain model. Prebuilt domains allow you to quickly develop your LUIS application for common domains. This API allows you to create a new app based on a prebuilt domain. The response is the new appId.

V2 ROUTE	VERB
/luis/api/v2.0/apps/customprebuiltdomains	get, post
/luis/api/v2.0/apps/customprebuiltdomains/{culture}	get

Importing 1.x app into 2.x

The exported 1.x app's JSON has some areas that you need to change before importing into [LUIS](#) 2.0.

Prebuilt entities

The [prebuilt entities](#) have changed. Make sure you are using the V2 prebuilt entities. This includes using [datetimeV2](#), instead of [datetime](#).

Actions

The actions property is no longer valid. It should be an empty

Labeled utterances

V1 allowed labeled utterances to include spaces at the beginning or end of the word or phrase. Removed the spaces.

Common reasons for HTTP response status codes

See [LUIS API response codes](#).

Next steps

Use the v2 API documentation to update existing REST calls to LUIS endpoint and [authoring](#) APIs.

Preview: Migrate to API version 3.x for LUIS apps

8/9/2019 • 10 minutes to read • [Edit Online](#)

The query prediction endpoint APIs have changed. Use this guide to understand how to migrate to version 3 endpoint APIs.

This V3 API provides the following new features, which include significant JSON request and/or response changes:

- [External entities](#)
- [Dynamic lists](#)
- [Prebuilt entity JSON changes](#)

The query prediction endpoint [request](#) and [response](#) have significant changes to support the new features listed above, including the following:

- [Response object changes](#)
- [Entity role name references instead of entity name](#)
- [Properties to mark entities in utterances](#)

The following LUIS features are **not supported** in the V3 API:

- Bing Spell Check V7

[Reference documentation](#) is available for V3.

Endpoint URL changes by slot name

The format of the V3 endpoint HTTP call has changed.

METHOD	URL
GET	<code>https://{REGION}.api.cognitive.microsoft.com/luis/v3.0-preview/apps/{APP-ID}/slots/{SLOT-NAME}/predict?query={QUERY}</code>
POST	<code>https://{REGION}.api.cognitive.microsoft.com/luis/v3.0-preview/apps/{APP-ID}/slots/{SLOT-NAME}/predict</code>

Valid values for slots:

- `production`
- `staging`

Endpoint URL changes by version ID

If you want to query by version, you first need to [publish via API](#) with the `"directVersionPublish":true`. Query the endpoint referencing the version ID instead of the slot name.

METHOD	URL
GET	https://{{REGION}}.api.cognitive.microsoft.com/luis/v3.0-preview/apps/{{APP-ID}}/versions/{{VERSION-ID}}/predict?query={{QUERY}}
POST	https://{{REGION}}.api.cognitive.microsoft.com/luis/v3.0-preview/apps/{{APP-ID}}/versions/{{VERSION-ID}}/predict

Prebuilt entities with new JSON

The V3 response object changes include [prebuilt entities](#).

Request changes

Query string parameters

The V3 API has different query string parameters.

PARAM NAME	TYPE	VERSION	DEFAULT	PURPOSE
<code>log</code>	boolean	V2 & V3	false	Store query in log file.
<code>query</code>	string	V3 only	No default - it is required in the GET request	<p>In V2, the utterance to be predicted is in the <code>q</code> parameter.</p> <p>In V3, the functionality is passed in the <code>query</code> parameter.</p>
<code>show-all-intents</code>	boolean	V3 only	false	<p>Return all intents with the corresponding score in the <code>prediction.intents</code> object. Intents are returned as objects in a parent <code>intents</code> object. This allows programmatic access without needing to find the intent in an array:</p> <pre><code>prediction.intents.give</code></pre> <p>. In V2, these were returned in an array.</p>

PARAM NAME	TYPE	VERSION	DEFAULT	PURPOSE
<code>verbose</code>	boolean	V2 & V3	false	<p>In V2, when set to true, all predicted intents were returned. If you need all predicted intents, use the V3 param of <code>show-all-intents</code>.</p> <p>In V3, this parameter only provides entity metadata details of entity prediction.</p>

The query prediction JSON body for the `POST` request

```
{
  "query": "your utterance here",
  "options": {
    "datetimeReference": "2019-05-05T12:00:00",
    "overridePredictions": true
  },
  "externalEntities": [],
  "dynamicLists": []
}
```

PROPERTY	TYPE	VERSION	DEFAULT	PURPOSE
<code>dynamicLists</code>	array	V3 only	Not required.	Dynamic lists allow you to extend an existing trained and published list entity, already in the LUIS app.
<code>externalEntities</code>	array	V3 only	Not required.	External entities give your LUIS app the ability to identify and label entities during runtime, which can be used as features to existing entities.
<code>options.datetimeReference</code>	string	V3 only	No default	Used to determine datetimeV2 offset . The format for the <code>datetimeReference</code> is ISO 8601 .
<code>options.overridePredictions</code>	boolean	V3 only	false	Specifies if user's external entity (with same name as existing entity) is used or the existing entity in the model is used for prediction.

PROPERTY	TYPE	VERSION	DEFAULT	PURPOSE
<code>query</code>	string	V3 only	Required.	<p>In V2, the utterance to be predicted is in the <code>q</code> parameter.</p> <p>In V3, the functionality is passed in the <code>query</code> parameter.</p>

Response changes

The query response JSON changed to allow greater programmatic access to the data used most frequently.

Top level JSON changes

The top JSON properties for V2 are, when `verbose` is set to true, which returns all intents and their scores in the `intents` property:

```
{
  "query": "this is your utterance you want predicted",
  "topScoringIntent": {},
  "intents": [],
  "entities": [],
  "compositeEntities": []
}
```

The top JSON properties for V3 are:

```
{
  "query": "this is your utterance you want predicted",
  "prediction": {
    "normalizedQuery": "this is your utterance you want predicted - after normalization",
    "topIntent": "intent-name-1",
    "intents": {},
    "entities": {}
  }
}
```

The `intents` object is an unordered list. Do not assume the first child in the `intents` corresponds to the `topIntent`. Instead, use the `topIntent` value to find the score:

```
const topIntentName = response.prediction.topIntent;
const score = intents[topIntentName];
```

The response JSON schema changes allow for:

- Clear distinction between original utterance, `query`, and returned prediction, `prediction`.
- Easier programmatic access to predicted data. Instead of enumerating through an array in V2, you can access values by **named** for both intents and entities. For predicted entity roles, the role name is returned because it is unique across the entire app.
- Data types, if determined, are respected. Numerics are no longer returned as strings.
- Distinction between first priority prediction information and additional metadata, returned in the `$instance` object.

Access `$instance` for entity metadata

If you need entity metadata, the query string needs to use the `verbose=true` flag and the response contains the metadata in the `$instance` object. Examples are shown in the JSON responses in the following sections.

Each predicted entity is represented as an array

The `prediction.entities.<entity-name>` object contains an array because each entity can be predicted more than once in the utterance.

List entity prediction changes

The JSON for a list entity prediction has changed to be an array of arrays:

```
"entities":{  
  "my_list_entity": [  
    ["canonical-form-1", "canonical-form-2"],  
    ["canonical-form-2"]  
  ]  
}
```

Each interior array corresponds to text inside the utterance. The interior object is an array because the same text can appear in more than one sublist of a list entity.

When mapping between the `entities` object to the `$instance` object, the order of objects is preserved for the list entity predictions.

```
const item = 0; // order preserved, use same enumeration for both  
const predictedCanonicalForm = entities.my_list_entity[item];  
const associatedMetadata = entities.$instance.my_list_entity[item];
```

Entity role name instead of entity name

In V2, the `entities` array returned all the predicted entities with the entity name being the unique identifier. In V3, if the entity uses roles and the prediction is for an entity role, the primary identifier is the role name. This is possible because entity role names must be unique across the entire app including other model (intent, entity) names.

In the following example: consider an utterance that includes the text, `Yellow Bird Lane`. This text is predicted as a custom `Location` entity's role of `Destination`.

UTTERANCE TEXT	ENTITY NAME	ROLE NAME
<code>Yellow Bird Lane</code>	<code>Location</code>	<code>Destination</code>

In V2, the entity is identified by the *entity name* with the role as a property of the object:

```
"entities": [  
  {  
    "entity": "Yellow Bird Lane",  
    "type": "Location",  
    "startIndex": 13,  
    "endIndex": 20,  
    "score": 0.786378264,  
    "role": "Destination"  
  }  
]
```

In V3, the entity is referenced by the *entity role*, if the prediction is for the role:

```
"entities":{  
    "Destination": [  
        "Yellow Bird Lane"  
    ]  
}
```

In V3, the same result with the `verbose` flag to return entity metadata:

```
"entities":{  
    "Destination": [  
        "Yellow Bird Lane"  
    ],  
    "$instance":{  
        "Destination": [  
            {  
                "role": "Destination",  
                "type": "Location",  
                "text": "Yellow Bird Lane",  
                "startIndex": 25,  
                "length": 16,  
                "score": 0.9837309,  
                "modelTypeId": 1,  
                "modelType": "Entity Extractor"  
            }  
        ]  
    }  
}
```

External entities passed in at prediction time

External entities give your LUIS app the ability to identify and label entities during runtime, which can be used as features to existing entities. This allows you to use your own separate and custom entity extractors before sending queries to your prediction endpoint. Because this is done at the query prediction endpoint, you don't need to retrain and publish your model.

The client-application is providing its own entity extractor by managing entity matching and determining the location within the utterance of that matched entity and then sending that information with the request.

External entities are the mechanism for extending any entity type while still being used as signals to other models like roles, composite and others.

This is useful for an entity that has data available only at query prediction runtime. Examples of this type of data are constantly changing data or specific per user. You can extend a LUIS contact entity with external information from a user's contact list.

Entity already exists in app

The value of `entityName` for the external entity, passed in the endpoint request POST body, must already exist in the trained and published app at the time the request is made. The type of entity doesn't matter, all types are supported.

First turn in conversation

Consider a first utterance in a chat bot conversation where a user enters the following incomplete information:

```
Send Hazem a new message
```

The request from the chat bot to LUIS can pass in information in the POST body about `Hazem` so it is directly matched as one of the user's contacts.

```

"externalEntities": [
  {
    "entityName": "contacts",
    "startIndex": 5,
    "entityLength": 5,
    "resolution": {
      "employeeID": "05013",
      "preferredContactType": "TeamsChat"
    }
  }
]

```

The prediction response includes that external entity, with all the other predicted entities, because it is defined in the request.

Second turn in conversation

The next user utterance into the chat bot uses a more vague term:

Send him a calendar reminder for the party.

In the previous utterance, the utterance uses `him` as a reference to `Hazem`. The conversational chat bot, in the POST body, can map `him` to the entity value extracted from the first utterance, `Hazem`.

```

"externalEntities": [
  {
    "entityName": "contacts",
    "startIndex": 5,
    "entityLength": 3,
    "resolution": {
      "employeeID": "05013",
      "preferredContactType": "TeamsChat"
    }
  }
]

```

The prediction response includes that external entity, with all the other predicted entities, because it is defined in the request.

Override existing model predictions

The `overridePredictions` options property specifies that if the user sends an external entity that overlaps with a predicted entity with the same name, LUIS chooses the entity passed in or the entity existing in the model.

For example, consider the query `today I'm free`. LUIS detects `today` as a datetimeV2 with the following response:

```

"datetimeV2": [
  {
    "type": "date",
    "values": [
      {
        "timex": "2019-06-21",
        "value": "2019-06-21"
      }
    ]
  }
]

```

If the user sends the external entity:

```
{  
    "entityName": "datetimeV2",  
    "startIndex": 0,  
    "entityLength": 5,  
    "resolution": {  
        "date": "2019-06-21"  
    }  
}
```

If the `overridePredictions` is set to `false`, LUIS returns a response as if the external entity were not sent.

```
"datetimeV2": [  
    {  
        "type": "date",  
        "values": [  
            {  
                "timex": "2019-06-21",  
                "value": "2019-06-21"  
            }  
        ]  
    }  
]
```

If the `overridePredictions` is set to `true`, LUIS returns a response including:

```
"datetimeV2": [  
    {  
        "date": "2019-06-21"  
    }  
]
```

Resolution

The *optional* `resolution` property returns in the prediction response, allowing you to pass in the metadata associated with the external entity, then receive it back out in the response.

The primary purpose is to extend prebuilt entities but it is not limited to that entity type.

The `resolution` property can be a number, a string, an object, or an array:

- "Dallas"
- {"text": "value"}
- 12345
- ["a", "b", "c"]

Dynamic lists passed in at prediction time

Dynamic lists allow you to extend an existing trained and published list entity, already in the LUIS app.

Use this feature when your list entity values need to change periodically. This feature allows you to extend an already trained and published list entity:

- At the time of the query prediction endpoint request.
- For a single request.

The list entity can be empty in the LUIS app but it has to exist. The list entity in the LUIS app isn't changed, but the prediction ability at the endpoint is extended to include up to 2 lists with about 1,000 items.

Dynamic list JSON request body

Send in the following JSON body to add a new sublist with synonyms to the list, and predict the list entity for the text, LUIS, with the POST query prediction request:

```
{  
    "query": "Send Hazem a message to add an item to the meeting agenda about LUIS.",  
    "options": {  
        "timezoneOffset": "-8:00"  
    },  
    "dynamicLists": [  
        {  
            "listEntityName": "ProductList",  
            "requestLists": [  
                {  
                    "name": "Azure Cognitive Services",  
                    "canonicalForm": "Azure-Cognitive-Services",  
                    "synonyms": [  
                        "language understanding",  
                        "luis",  
                        "qna maker"  
                    ]  
                }  
            ]  
        }  
    ]  
}
```

The prediction response includes that list entity, with all the other predicted entities, because it is defined in the request.

TimezoneOffset renamed to datetimeReference

In V2, the `timezoneOffset` parameter is sent in the prediction request as a query string parameter, regardless if the request is sent as a GET or POST request.

In V3, the same functionality is provided with the POST body parameter, `datetimeReference`.

Marking placement of entities in utterances

In V2, an entity was marked in an utterance with the `startIndex` and `endIndex`.

In V3, the entity is marked with `startIndex` and `entityLength`.

Deprecation

The V2 API will not be deprecated for at least 9 months after the V3 preview.

Next steps

Use the V3 API documentation to update existing REST calls to LUIS endpoint APIs.

Install and run LUIS docker containers

7/26/2019 • 15 minutes to read • [Edit Online](#)

The Language Understanding (LUIS) container loads your trained or published Language Understanding model, also known as a [LUIS app](#), into a docker container and provides access to the query predictions from the container's API endpoints. You can collect query logs from the container and upload these back to the Language Understanding app to improve the app's prediction accuracy.

The following video demonstrates using this container.



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

Prerequisites

In order to run the LUIS container, you must have the following:

REQUIRED	PURPOSE
Docker Engine	<p>You need the Docker Engine installed on a host computer. Docker provides packages that configure the Docker environment on macOS, Windows, and Linux. For a primer on Docker and container basics, see the Docker overview.</p> <p>Docker must be configured to allow the containers to connect with and send billing data to Azure.</p> <p>On Windows, Docker must also be configured to support Linux containers.</p>
Familiarity with Docker	<p>You should have a basic understanding of Docker concepts, like registries, repositories, containers, and container images, as well as knowledge of basic <code>docker</code> commands.</p>

REQUIRED	PURPOSE
Azure <code>Cognitive Services</code> resource and LUIS packaged app file	<p>In order to use the container, you must have:</p> <ul style="list-style-type: none"> * A <i>Cognitive Services</i> Azure resource and the associated billing key the billing endpoint URI. Both values are available on the Overview and Keys pages for the resource and are required to start the container. You need to add the <code>luis/v2.0</code> routing to the endpoint URI as shown in the following <code>BILLING_ENDPOINT_URI</code> example. * A trained or published app packaged as a mounted input to the container with its associated App ID. You can get the packaged file from the LUIS portal or the Authoring APIs. If you are getting LUIS packaged app from the authoring APIs, you will also need your <i>Authoring Key</i>. <p>These requirements are used to pass command-line arguments to the following variables:</p> <p>{AUTHORING_KEY}: This key is used to get the packaged app from the LUIS service in the cloud and upload the query logs back to the cloud. The format is <code>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</code>.</p> <p>{APPLICATION_ID}: This ID is used to select the App. The format is <code>xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx</code>.</p> <p>{API_KEY}: This key is used to start the container. You can find the endpoint key in two places. The first is the Azure portal within the <i>Cognitive Services</i> resource's keys list. The endpoint key is also available in the LUIS portal on the Keys and Endpoint settings page. Do not use the starter key.</p> <p>{ENDPOINT_URI}: The endpoint as provided on the Overview page.</p> <p>The authoring key and endpoint key have different purposes. Do not use them interchangeably.</p>

Authoring APIs for package file

Authoring APIs for packaged apps:

- [Published package API](#)
- [Not-published, trained-only package API](#)

The host computer

The host is a x64-based computer that runs the Docker container. It can be a computer on your premises or a Docker hosting service in Azure, such as:

- [Azure Kubernetes Service](#).
- [Azure Container Instances](#).
- A *Kubernetes* cluster deployed to [Azure Stack](#). For more information, see [Deploy Kubernetes to Azure Stack](#).

Container requirements and recommendations

This container supports minimum and recommended values for the settings:

CONTAINER	MINIMUM	RECOMMENDED	TPS (MINIMUM, MAXIMUM)
LUIS	1 core, 2-GB memory	1 core, 4-GB memory	20,40

- Each core must be at least 2.6 gigahertz (GHz) or faster.
- TPS - transactions per second

Core and memory correspond to the `--cpus` and `--memory` settings, which are used as part of the `docker run` command.

Get the container image with `docker pull`

Use the `docker pull` command to download a container image from the `mcr.microsoft.com/azure-cognitive-services/luis` repository:

```
docker pull mcr.microsoft.com/azure-cognitive-services/luis:latest
```

Use the `docker pull` command to download a container image.

For a full description of available tags, such as `latest` used in the preceding command, see [LUIS](#) on Docker Hub.

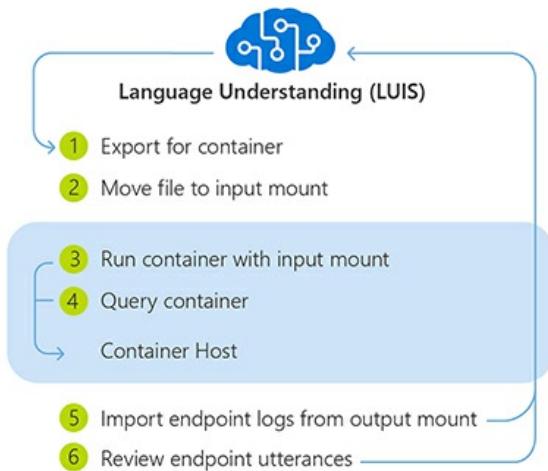
TIP

You can use the `docker images` command to list your downloaded container images. For example, the following command lists the ID, repository, and tag of each downloaded container image, formatted as a table:

```
docker images --format "table {{.ID}}\t{{.Repository}}\t{{.Tag}}"  
IMAGE ID      REPOSITORY          TAG  
<image-id>    <repository-path/name>  <tag-name>
```

How to use the container

Once the container is on the [host computer](#), use the following process to work with the container.



1. [Export package](#) for container from LUIS portal or LUIS APIs.
2. Move package file into the required **input** directory on the [host computer](#). Do not rename, alter, overwrite, or decompress LUIS package file.
3. [Run the container](#), with the required *input mount* and billing settings. More [examples](#) of the `docker run` command are available.
4. [Querying the container's prediction endpoint](#).
5. When you are done with the container, [import the endpoint logs](#) from the output mount in the LUIS portal and [stop](#) the container.
6. Use LUIS portal's [active learning](#) on the **Review endpoint utterances** page to improve the app.

The app running in the container can't be altered. In order to change the app in the container, you need to change the app in the LUIS service using the [LUIS](#) portal or use the LUIS [authoring APIs](#). Then train and/or publish, then download a new package and run the container again.

The LUIS app inside the container can't be exported back to the LUIS service. Only the query logs can be uploaded.

Export packaged app from LUIS

The LUIS container requires a trained or published LUIS app to answer prediction queries of user utterances. In order to get the LUIS app, use either the trained or published package API.

The default location is the `input` subdirectory in relation to where you run the `docker run` command.

Place the package file in a directory and reference this directory as the input mount when you run the docker container.

Package types

The input mount directory can contain the **Production**, **Staging**, and **Trained** versions of the app simultaneously. All the packages are mounted.

PACKAGE TYPE	QUERY ENDPOINT API	QUERY AVAILABILITY	PACKAGE FILENAME FORMAT
Trained	Get, Post	Container only	{APPLICATION_ID}_v{APPLICATION_VERSION}.gz
Staging	Get, Post	Azure and container	{APPLICATION_ID}_STAGING.gz
Production	Get, Post	Azure and container	{APPLICATION_ID}_PRODUCTION.gz

IMPORTANT

Do not rename, alter, overwrite, or decompress the LUIS package files.

Packaging prerequisites

Before packaging a LUIS application, you must have the following:

PACKAGING REQUIREMENTS	DETAILS
Azure Cognitive Services resource instance	Supported regions include West US (<code>westus</code>) West Europe (<code>westeurope</code>) Australia East (<code>australiaeast</code>)
Trained or published LUIS app	With no unsupported dependencies .
Access to the host computer's file system	The host computer must allow an input mount .

Export app package from LUIS portal

The LUIS [portal](#) provides the ability to export the trained or published app's package.

Export published app's package from LUIS portal

The published app's package is available from the **My Apps** list page.

1. Sign on to the LUIS [portal](#).
2. Select the checkbox to the left of the app name in the list.
3. Select the **Export** item from the contextual toolbar above the list.
4. Select **Export for container (GZIP)**.
5. Select the environment of **Production slot** or **Staging slot**.
6. The package is downloaded from the browser.

The screenshot shows the LUIS portal's 'My Apps' section. A specific app named 'Human Resources with Key Phrase entity' is selected. In the top navigation bar, the 'Export' button is highlighted. A contextual menu has appeared, listing options: 'Culture', 'Production slot' (which is currently selected), and 'Staging slot'. Other visible buttons in the top bar include 'Rename', 'Delete', 'Export as JSON', and 'Export for container (GZIP)'. The main table lists the app's details: 'Created date' is 5/3/18 and 'Endpoint hits' is 0.

Export trained app's package from LUIS portal

The trained app's package is available from the **Versions** list page.

1. Sign on to the LUIS [portal](#).
2. Select the app in the list.
3. Select **Manage** in the app's navigation bar.
4. Select **Versions** in the left navigation bar.
5. Select the checkbox to the left of the version name in the list.
6. Select the **Export** item from the contextual toolbar above the list.
7. Select **Export for container (GZIP)**.
8. The package is downloaded from the browser.

The screenshot shows the LUIS portal's 'Versions' list page. A specific version named '0.1 (Active & Production)' is selected. In the top navigation bar, the 'Export' button is highlighted. A contextual menu has appeared, listing options: 'Export as JSON' and 'Export for container (GZIP)' (which is currently selected). Other visible buttons in the top bar include 'Rename', 'Clone', and 'Search for version(s)'. The main table lists the version details: 'Created' is 5/3/18 and 'Last modified' is 9/6/18.

Export published app's package from API

Use the following REST API method, to package a LUIS app that you've already [published](#). Substituting your own appropriate values for the placeholders in the API call, using the table below the HTTP specification.

```
GET /luis/api/v2.0/package/{APPLICATION_ID}/slot/{APPLICATION_ENVIRONMENT}/gzip HTTP/1.1
Host: {AZURE_REGION}.api.cognitive.microsoft.com
Ocp-Apim-Subscription-Key: {AUTHORING_KEY}
```

PLACEHOLDER	VALUE
{APPLICATION_ID}	The application ID of the published LUIS app.
{APPLICATION_ENVIRONMENT}	The environment of the published LUIS app. Use one of the following values: PRODUCTION STAGING
{AUTHORING_KEY}	The authoring key of the LUIS account for the published LUIS app. You can get your authoring key from the User Settings page on the LUIS portal.

PLACEHOLDER	VALUE
{AZURE_REGION}	The appropriate Azure region: <div style="display: flex; justify-content: space-around;"> westus - West US westeurope - West Europe australiaeast - Australia East </div>

To download the published package, please refer to the [API documentation here](#). If successfully downloaded, the response is a LUIS package file. Save the file in the storage location specified for the input mount of the container.

Export trained app's package from API

Use the following REST API method, to package a LUIS application that you've already [trained](#). Substituting your own appropriate values for the placeholders in the API call, using the table below the HTTP specification.

<pre>GET /luis/api/v2.0/package/{APPLICATION_ID}/versions/{APPLICATION_VERSION}/gzip HTTP/1.1 Host: {AZURE_REGION}.api.cognitive.microsoft.com Ocp-Apim-Subscription-Key: {AUTHORING_KEY}</pre>

PLACEHOLDER	VALUE
{APPLICATION_ID}	The application ID of the trained LUIS application.
{APPLICATION_VERSION}	The application version of the trained LUIS application.
{AUTHORING_KEY}	The authoring key of the LUIS account for the published LUIS app. You can get your authoring key from the User Settings page on the LUIS portal.
{AZURE_REGION}	The appropriate Azure region: <div style="display: flex; justify-content: space-around;"> westus - West US westeurope - West Europe australiaeast - Australia East </div>

To download the trained package, please refer to the [API documentation here](#). If successfully downloaded, the response is a LUIS package file. Save the file in the storage location specified for the input mount of the container.

Run the container with `docker run`

Use the [docker run](#) command to run the container. The command uses the following parameters:

PLACEHOLDER	VALUE
{API_KEY}	This key is used to start the container. Do not use the starter key.
{ENDPOINT_URI}	The endpoint value is available on the Azure portal's Cognitive Services Overview page.

Replace these parameters with your own values in the following example `docker run` command. Run the command in the Windows console.

```

docker run --rm -it -p 5000:5000 ^
--memory 4g ^
--cpus 2 ^
--mount type=bind,src=c:\input,target=/input ^
--mount type=bind,src=c:\output,target=/output ^
mcr.microsoft.com/azure-cognitive-services/luis ^
Eula=accept ^
Billing={ENDPOINT_URI} ^
ApiKey={API_KEY}

```

- This example uses the directory off the `c:` drive to avoid any permission conflicts on Windows. If you need to use a specific directory as the input directory, you may need to grant the docker service permission.
- Do not change the order of the arguments unless you are familiar with docker containers.
- If you are using a different operating system, use the correct console/terminal, folder syntax for mounts, and line continuation character for your system. These examples assume a Windows console with a line continuation character `^`. Because the container is a Linux operating system, the target mount uses a Linux-style folder syntax.

This command:

- Runs a container from the LUIS container image
- Loads LUIS app from input mount at `c:\input`, located on container host
- Allocates two CPU cores and 4 gigabytes (GB) of memory
- Exposes TCP port 5000 and allocates a pseudo-TTY for the container
- Saves container and LUIS logs to output mount at `c:\output`, located on container host
- Automatically removes the container after it exits. The container image is still available on the host computer.

More [examples](#) of the `docker run` command are available.

IMPORTANT

The `Eula`, `Billing`, and `ApiKey` options must be specified to run the container; otherwise, the container won't start. For more information, see [Billing](#). The `ApiKey` value is the `Key` from the Keys and Endpoints page in the LUIS portal and is also available on the Azure [Cognitive Services](#) resource keys page.

Run multiple containers on the same host

If you intend to run multiple containers with exposed ports, make sure to run each container with a different exposed port. For example, run the first container on port 5000 and the second container on port 5001.

You can have this container and a different Azure Cognitive Services container running on the HOST together. You also can have multiple containers of the same Cognitive Services container running.

Endpoint APIs supported by the container

Both V2 and [V3 \(Preview\)](#) versions of the API are available with the container.

Query the container's prediction endpoint

The container provides REST-based query prediction endpoint APIs. Endpoints for published (staging or production) apps have a *different* route than endpoints for trained apps.

Use the host, `https://localhost:5000`, for container APIs.

PACKAGE TYPE	METHOD	ROUTE	QUERY PARAMETERS
Published	Get , Post	/luis/v2.0/apps/{appId}?	q={q} &staging [&timezoneOffset] [&verbose] [&log]

PACKAGE TYPE	METHOD	ROUTE	QUERY PARAMETERS
Trained	Get, Post	/luis/v2.0/apps/{appId}/version{s}/{versionId}?	q={q} [&timezoneOffset] [&verbose] [&log]

The query parameters configure how and what is returned in the query response:

QUERY PARAMETER	TYPE	PURPOSE
<code>q</code>	string	The user's utterance.
<code>timezoneOffset</code>	number	The timezoneOffset allows you to change the timezone used by the prebuilt entity datetimeV2.
<code>verbose</code>	boolean	Returns all intents and their scores when set to true. Default is false, which returns only the top intent.
<code>staging</code>	boolean	Returns query from staging environment results if set to true.
<code>log</code>	boolean	Logs queries, which can be used later for active learning . Default is true.

Query published app

An example CURL command for querying the container for a published app is:

```
curl -X GET \
"http://localhost:5000/luis/v2.0/apps/{APPLICATION_ID}?
q=turn%20on%20the%20lights&staging=false&timezoneOffset=0&verbose=false&log=true" \
-H "accept: application/json"
```

To make queries to the **Staging** environment, change the **staging** query string parameter value to true:

```
staging=true
```

Query trained app

An example CURL command for querying the container for a trained app is:

```
curl -X GET \
"http://localhost:5000/luis/v2.0/apps/{APPLICATION_ID}/versions/{APPLICATION_VERSION}?
q=turn%20on%20the%20lights&timezoneOffset=0&verbose=false&log=true" \
-H "accept: application/json"
```

The version name has a maximum of 10 characters and contains only characters allowed in a URL.

Import the endpoint logs for active learning

If an output mount is specified for the LUIS container, app query log files are saved in the output directory, where `{INSTANCE_ID}` is the container ID. The app query log contains the query, response, and timestamps for each prediction query submitted to the LUIS container.

The following location shows the nested directory structure for the container's log files.

```
/output/luis/{INSTANCE_ID}/
```

From the LUIS portal, select your app, then select **Import endpoint logs** to upload these logs.

My Apps

<input type="checkbox"/> Name	Culture	Created date
<input checked="" type="checkbox"/> new-prebuilt (v 0.1)	en-us	9/20/18

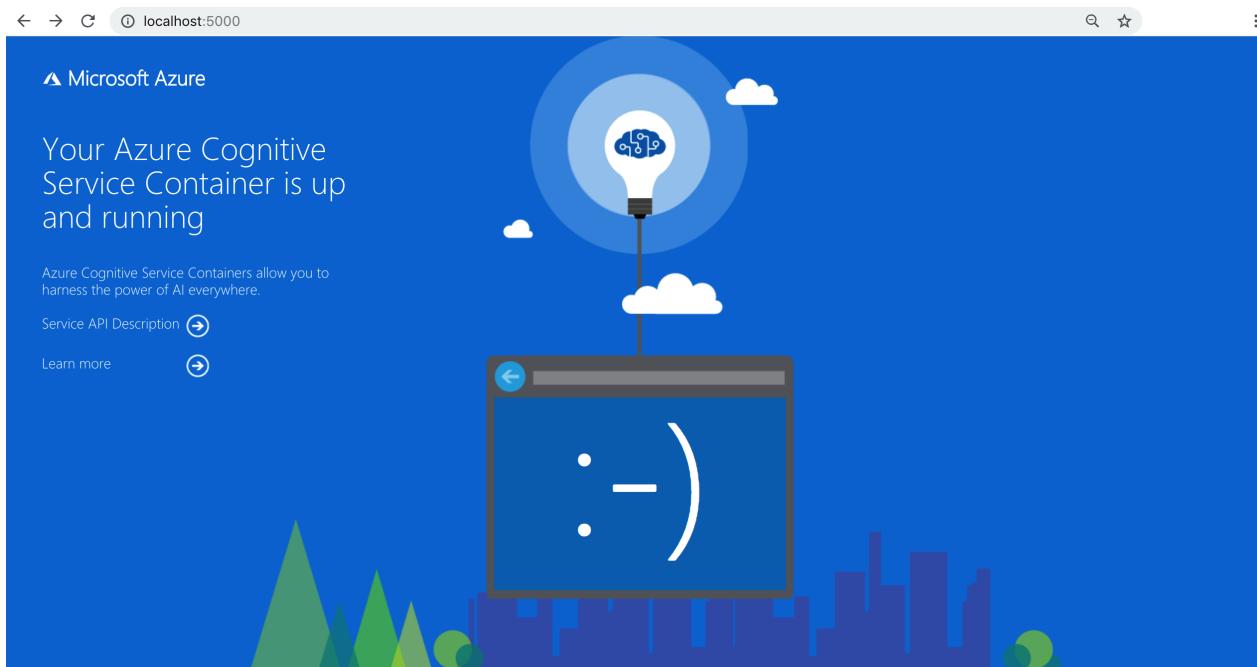
Import endpoint logs

After the log is uploaded, [review the endpoint](#) utterances in the LUIS portal.

Validate that a container is running

There are several ways to validate that the container is running.

REQUEST	PURPOSE
<code>http://localhost:5000/</code>	The container provides a home page.
<code>http://localhost:5000/status</code>	Requested with GET, to validate that the container is running without causing an endpoint query. This request can be used for Kubernetes liveness and readiness probes .
<code>http://localhost:5000/swagger</code>	The container provides a full set of documentation for the endpoints and a <code>Try it now</code> feature. With this feature, you can enter your settings into a web-based HTML form and make the query without having to write any code. After the query returns, an example CURL command is provided to demonstrate the HTTP headers and body format that's required.



Stop the container

To shut down the container, in the command-line environment where the container is running, press **Ctrl+C**.

Troubleshooting

If you run the container with an output [mount](#) and logging enabled, the container generates log files that are helpful to troubleshoot issues that happen while starting or running the container.

Billing

The LUIS container sends billing information to Azure, using a *Cognitive Services* resource on your Azure account.

Queries to the container are billed at the pricing tier of the Azure resource that's used for the `<ApiKey>`.

Azure Cognitive Services containers aren't licensed to run without being connected to the billing endpoint for metering. You must enable the containers to communicate billing information with the billing endpoint at all times. Cognitive Services containers don't send customer data, such as the image or text that's being analyzed, to Microsoft.

Connect to Azure

The container needs the billing argument values to run. These values allow the container to connect to the billing endpoint. The container reports usage about every 10 to 15 minutes. If the container doesn't connect to Azure within the allowed time window, the container continues to run but doesn't serve queries until the billing endpoint is restored. The connection is attempted 10 times at the same time interval of 10 to 15 minutes. If it can't connect to the billing endpoint within the 10 tries, the container stops running.

Billing arguments

For the `docker run` command to start the container, all three of the following options must be specified with valid values:

OPTION	DESCRIPTION
<code>ApiKey</code>	The API key of the Cognitive Services resource that's used to track billing information. The value of this option must be set to an API key for the provisioned resource that's specified in <code>Billing</code> .
<code>Billing</code>	The endpoint of the Cognitive Services resource that's used to track billing information. The value of this option must be set to the endpoint URI of a provisioned Azure resource.
<code>Eula</code>	Indicates that you accepted the license for the container. The value of this option must be set to <code>accept</code> .

For more information about these options, see [Configure containers](#).

Supported dependencies for `latest` container

The latest container, released at 2019 //Build, will support:

- **Bing spell check:** requests to the query prediction endpoint with the `&spellCheck=true&bing-spell-check-subscription-key={bingKey}` query string parameters. Use the [Bing Spell Check v7 tutorial](#) to learn more. If this feature is used, the container sends the utterance to your Bing Spell Check V7 resource.
- **New prebuilt domains:** these enterprise-focused domains include entities, example utterances, and patterns. Extend these domains for your own use.

Unsupported dependencies for `latest` container

If your LUIS app has unsupported dependencies, you won't be able to [export for container](#) until you remove the unsupported features. When you attempt to export for container, the LUIS portal reports the unsupported features you need to remove.

You can use a LUIS application if it **doesn't include** any of the following dependencies:

UNSUPPORTED APP CONFIGURATIONS	DETAILS
Unsupported container cultures	Dutch (nl-NL) Japanese (ja-JP) German is only supported with the 1.0.2 tokenizer .

UNSUPPORTED APP CONFIGURATIONS	DETAILS
Unsupported entities for all cultures	KeyPhrase prebuilt entity for all cultures
Unsupported entities for English (en-US) culture	GeographyV2 prebuilt entities
Speech priming	External dependencies are not supported in the container.
Sentiment analysis	External dependencies are not supported in the container.

Blog posts

- [Running Cognitive Services Containers](#)
- [Getting started with Cognitive Services Language Understanding container](#)

Developer samples

Developer samples are available at our [GitHub repository](#).

View webinar

Join the [webinar](#) to learn about:

- How to deploy Cognitive Services to any machine using Docker
- How to deploy Cognitive Services to AKS

Summary

In this article, you learned concepts and workflow for downloading, installing, and running Language Understanding (LUIS) containers. In summary:

- Language Understanding (LUIS) provides one Linux container for Docker providing endpoint query predictions of utterances.
- Container images are downloaded from the Microsoft Container Registry (MCR).
- Container images run in Docker.
- You can use REST API to query the container endpoints by specifying the host URI of the container.
- You must specify billing information when instantiating a container.

IMPORTANT

Cognitive Services containers are not licensed to run without being connected to Azure for metering. Customers need to enable the containers to communicate billing information with the metering service at all times. Cognitive Services containers do not send customer data (for example, the image or text that is being analyzed) to Microsoft.

Next steps

- Review [Configure containers](#) for configuration settings
- Refer to [Troubleshooting](#) to resolve issues related to LUIS functionality.
- Use more [Cognitive Services Containers](#)

Configure Language Understanding Docker containers

7/26/2019 • 8 minutes to read • [Edit Online](#)

The **Language Understanding** (LUIS) container runtime environment is configured using the `docker run` command arguments. LUIS has several required settings, along with a few optional settings. Several [examples](#) of the command are available. The container-specific settings are the input [mount settings](#) and the billing settings.

Configuration settings

This container has the following configuration settings:

REQUIRED	SETTING	PURPOSE
Yes	ApiKey	Used to track billing information.
No	ApplicationInsights	Allows you to add Azure Application Insights telemetry support to your container.
Yes	Billing	Specifies the endpoint URI of the service resource on Azure.
Yes	Eula	Indicates that you've accepted the license for the container.
No	Fluentd	Write log and, optionally, metric data to a Fluentd server.
No	Http Proxy	Configure an HTTP proxy for making outbound requests.
No	Logging	Provides ASP.NET Core logging support for your container.
Yes	Mounts	Read and write data from host computer to container and from container back to host computer.

IMPORTANT

The [ApiKey](#), [Billing](#), and [Eula](#) settings are used together, and you must provide valid values for all three of them; otherwise your container won't start. For more information about using these configuration settings to instantiate a container, see [Billing](#).

ApiKey setting

The [ApiKey](#) setting specifies the Azure resource key used to track billing information for the container. You must specify a value for the ApiKey and the value must be a valid key for the *Cognitive Services* resource specified for the [Billing](#) configuration setting.

This setting can be found in the following places:

- Azure portal: **Cognitive Services** Resource Management, under **Keys**
- LUIS portal: **Keys and Endpoint settings** page.

Do not use the starter key or the authoring key.

ApplicationInsights setting

The [ApplicationInsights](#) setting allows you to add [Azure Application Insights](#) telemetry support to your container. Application Insights provides in-depth monitoring of your container. You can easily monitor your container for availability, performance, and usage. You can also quickly identify and diagnose errors in your container.

The following table describes the configuration settings supported under the [ApplicationInsights](#) section.

REQUIRED	NAME	DATA TYPE	DESCRIPTION
No	InstrumentationKey	String	<p>The instrumentation key of the Application Insights instance to which telemetry data for the container is sent. For more information, see Application Insights for ASP.NET Core.</p> <p>Example: InstrumentationKey=123456789</p>

Billing setting

The `Billing` setting specifies the endpoint URI of the *Cognitive Services* resource on Azure used to meter billing information for the container. You must specify a value for this configuration setting, and the value must be a valid endpoint URI for a *Cognitive Services* resource on Azure. The container reports usage about every 10 to 15 minutes.

This setting can be found in the following places:

- Azure portal: **Cognitive Services** Overview, labeled `Endpoint`
- LUIS portal: **Keys and Endpoint settings** page, as part of the endpoint URI.

Remember to include the `luis/v2.0` routing in the URL as shown in the following table:

REQUIRED	NAME	DATA TYPE	DESCRIPTION
Yes	Billing	String	<p>Billing endpoint URI</p> <p>Example: Billing=https://westus.api.cognitive.mic</p>

Eula setting

The `Eula` setting indicates that you've accepted the license for the container. You must specify a value for this configuration setting, and the value must be set to `accept`.

REQUIRED	NAME	DATA TYPE	DESCRIPTION
Yes	Eula	String	<p>License acceptance</p> <p>Example: Eula=accept</p>

Cognitive Services containers are licensed under [your agreement](#) governing your use of Azure. If you do not have an existing agreement governing your use of Azure, you agree that your agreement governing use of Azure is the [Microsoft Online Subscription Agreement](#), which incorporates the [Online Services Terms](#). For previews, you also agree to the [Supplemental Terms of Use for Microsoft Azure Previews](#). By using the container you agree to these terms.

Fluentd settings

Fluentd is an open-source data collector for unified logging. The `Fluentd` settings manage the container's connection to a *Fluentd* server. The container includes a Fluentd logging provider, which allows your container to write logs and, optionally, metric data to a Fluentd server.

The following table describes the configuration settings supported under the `Fluentd` section.

NAME	DATA TYPE	DESCRIPTION
Host	String	The IP address or DNS host name of the Fluentd server.
Port	Integer	The port of the Fluentd server. The default value is 24224.
HeartbeatMs	Integer	The heartbeat interval, in milliseconds. If no event traffic has been sent before this interval expires, a heartbeat is sent to the Fluentd server. The default value is 60000 milliseconds (1 minute).

NAME	DATA TYPE	DESCRIPTION
<code>SendBufferSize</code>	Integer	The network buffer space, in bytes, allocated for send operations. The default value is 32768 bytes (32 kilobytes).
<code>TlsConnectionEstablishmentTimeoutMs</code>	Integer	The timeout, in milliseconds, to establish a SSL/TLS connection with the Fluentd server. The default value is 10000 milliseconds (10 seconds). If <code>UseTLS</code> is set to false, this value is ignored.
<code>UseTLS</code>	Boolean	Indicates whether the container should use SSL/TLS for communicating with the Fluentd server. The default value is false.

Http proxy credentials settings

If you need to configure an HTTP proxy for making outbound requests, use these two arguments:

NAME	DATA TYPE	DESCRIPTION
<code>HTTP_PROXY</code>	string	The proxy to use, for example, <code>http://proxy:8888</code> <code><proxy-url></code>
<code>HTTP_PROXY_CREDS</code>	string	Any credentials needed to authenticate against the proxy, for example, <code>username:password</code> .
<code><proxy-user></code>	string	The user for the proxy.
<code><proxy-password></code>	string	The password associated with <code><proxy-user></code> for the proxy.

```
docker run --rm -it -p 5000:5000 \
--memory 2g --cpus 1 \
--mount type=bind,src=/home/azureuser/output,target=/output \
<registry-location>/<image-name> \
Eula=accept \
Billing=<endpoint> \
ApiKey=<api-key> \
HTTP_PROXY=<proxy-url> \
HTTP_PROXY_CREDS=<proxy-user>:<proxy-password> \
```

Logging settings

The `Logging` settings manage ASP.NET Core logging support for your container. You can use the same configuration settings and values for your container that you use for an ASP.NET Core application.

The following logging providers are supported by the container:

PROVIDER	PURPOSE
<code>Console</code>	The ASP.NET Core <code>Console</code> logging provider. All of the ASP.NET Core configuration settings and default values for this logging provider are supported.
<code>Debug</code>	The ASP.NET Core <code>Debug</code> logging provider. All of the ASP.NET Core configuration settings and default values for this logging provider are supported.
<code>Disk</code>	The JSON logging provider. This logging provider writes log data to the output mount.

This container command stores logging information in the JSON format to the output mount:

```
docker run --rm -it -p 5000:5000 \
--memory 2g --cpus 1 \
--mount type=bind,src=/home/azureuser/output,target=/output \
<registry-location>/<image-name> \
Eula=accept \
Billing=<endpoint> \
ApiKey=<api-key> \
Logging:Disk:Format=json
```

This container command shows debugging information, prefixed with `dbug`, while the container is running:

```
docker run --rm -it -p 5000:5000 \
--memory 2g --cpus 1 \
<registry-location>/<image-name> \
Eula=accept \
Billing=<endpoint> \
ApiKey=<api-key> \
Logging:Console:LogLevel:Default=Debug
```

Disk logging

The `Disk` logging provider supports the following configuration settings:

NAME	DATA TYPE	DESCRIPTION
<code>Format</code>	String	The output format for log files. Note: This value must be set to <code>json</code> to enable the logging provider. If this value is specified without also specifying an output mount while instantiating a container, an error occurs.
<code>MaxFileSize</code>	Integer	The maximum size, in megabytes (MB), of a log file. When the size of the current log file meets or exceeds this value, a new log file is started by the logging provider. If -1 is specified, the size of the log file is limited only by the maximum file size, if any, for the output mount. The default value is 1.

For more information about configuring ASP.NET Core logging support, see [Settings file configuration](#).

Mount settings

Use bind mounts to read and write data to and from the container. You can specify an input mount or output mount by specifying the `--mount` option in the `docker run` command.

The LUIS container doesn't use input or output mounts to store training or service data.

The exact syntax of the host mount location varies depending on the host operating system. Additionally, the [host computer](#)'s mount location may not be accessible due to a conflict between permissions used by the docker service account and the host mount location permissions.

The following table describes the settings supported.

REQUIRED	NAME	DATA TYPE	DESCRIPTION
Yes	<code>Input</code>	String	The target of the input mount. The default value is <code>/input</code> . This is the location of the LUIS package files. Example: <code>--mount type=bind,src=c:\input,target=/input</code>
No	<code>Output</code>	String	The target of the output mount. The default value is <code>/output</code> . This is the location of the logs. This includes LUIS query logs and container logs. Example: <code>--mount type=bind,src=c:\output,target=/output</code>

Example docker run commands

The following examples use the configuration settings to illustrate how to write and use `docker run` commands. Once running, the container continues to run until you [stop](#) it.

- These examples use the directory off the `c:` drive to avoid any permission conflicts on Windows. If you need to use a specific directory as the input directory, you may need to grant the docker service permission.
- Do not change the order of the arguments unless you are very familiar with docker containers.
- If you are using a different operating system, use the correct console/terminal, folder syntax for mounts, and line continuation character for your system. These examples assume a Windows console with a line continuation character `^`. Because the container is a Linux operating system, the target mount uses a Linux-style folder syntax.

Remember to include the `luis/v2.0` routing in the URL as shown in the following table.

Replace `{argument_name}` with your own values:

PLACEHOLDER	VALUE	FORMAT OR EXAMPLE
<code>{API_KEY}</code>	The endpoint key of the trained LUIS application.	<code>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</code>
<code>{ENDPOINT_URL}</code>	The billing endpoint value is available on the Azure Cognitive Services Overview page.	<code>https://westus.api.cognitive.microsoft.com/luis/v2.0</code>

IMPORTANT

The `Eula`, `Billing`, and `ApiKey` options must be specified to run the container; otherwise, the container won't start. For more information, see [Billing](#). The ApiKey value is the **Key** from the Keys and Endpoints page in the LUIS portal and is also available on the Azure [Cognitive Services](#) resource keys page.

Basic example

The following example has the fewest arguments possible to run the container:

```
docker run --rm -it -p 5000:5000 --memory 4g --cpus 2 ^
--mount type=bind,src=c:\input,target=/input ^
--mount type=bind,src=c:\output,target=/output ^
mcr.microsoft.com/azure-cognitive-services/luis:latest ^
Eula=accept ^
Billing={ENDPOINT_URL} ^
ApiKey={API_KEY}
```

ApplicationInsights example

The following example sets the ApplicationInsights argument to send telemetry to Application Insights while the container is running:

```
docker run --rm -it -p 5000:5000 --memory 6g --cpus 2 ^
--mount type=bind,src=c:\input,target=/input ^
--mount type=bind,src=c:\output,target=/output ^
mcr.microsoft.com/azure-cognitive-services/luis:latest ^
Eula=accept ^
Billing={ENDPOINT_URL} ^
ApiKey={API_KEY} ^
InstrumentationKey={INSTRUMENTATION_KEY}
```

Logging example

The following command sets the logging level, `Logging:Console:LogLevel`, to configure the logging level to [Information](#).

```
docker run --rm -it -p 5000:5000 --memory 6g --cpus 2 ^
--mount type=bind,src=c:\input,target=/input ^
--mount type=bind,src=c:\output,target=/output ^
mcr.microsoft.com/azure-cognitive-services/luis:latest ^
Eula=accept ^
Billing={ENDPOINT_URL} ^
ApiKey={API_KEY} ^
Logging:Console:LogLevel:Default=Information
```

Next steps

- Review [How to install and run containers](#)
- Refer to [Troubleshooting](#) to resolve issues related to LUIS functionality.
- Use more [Cognitive Services Containers](#)

Deploy the Language Understanding (LUIS) container to Azure Container Instances

7/9/2019 • 2 minutes to read • [Edit Online](#)

Learn how to deploy the Cognitive Services [LUIS](#) container to Azure [Container Instances](#). This procedure demonstrates the creation of an Anomaly Detector resource. Then we discuss pulling the associated container image. Finally, we highlight the ability to exercise the orchestration of the two from a browser. Using containers can shift the developers' attention away from managing infrastructure to instead focusing on application development.

Prerequisites

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

Create a LUIS resource

1. Sign into the [Azure portal](#)
2. Click [Create Language Understanding](#)
3. Enter all required settings:

SETTING	VALUE
Name	Desired name (2-64 characters)
Subscription	Select appropriate subscription
Location	Select any nearby and available location
Pricing Tier	F0 - the minimal pricing tier
Resource Group	Select an available resource group

4. Click **Create** and wait for the resource to be created. After it is created, navigate to the resource page
5. Collect configured `endpoint` and an API key:

RESOURCE TAB IN PORTAL	SETTING	VALUE
Overview	Endpoint	Copy the endpoint. It looks similar to <code>https://luis.cognitiveservices.azure.com/luis/</code>
Keys	API Key	Copy 1 of the two keys. It is a 32 alphanumeric-character string with no spaces or dashes, <code>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</code> .

Create an Azure Container Instance resource

1. Go to the [Create](#) page for Container Instances.
2. On the **Basics** tab, enter the following details:

SETTING	VALUE
Subscription	Select your subscription.
Resource group	Select the available resource group or create a new one such as <code>cognitive-services</code> .
Container name	Enter a name such as <code>cognitive-container-instance</code> . The name must be in lower caps.
Location	Select a region for deployment.
Image type	<code>Public</code>
Image name	Enter the Cognitive Services container location. The location can be the same used in the <code>docker pull</code> command, refer to the container repositories and images for the available image names and their corresponding repository.
OS type	<code>Linux</code>
Size	Change size to the suggested recommendations for your specific Cognitive Service container: 2 CPU cores 4 GB

3. On the **Networking** tab, enter the following details:

SETTING	VALUE
Ports	Set the TCP port to <code>5000</code> . Exposes the container on port 5000.

4. On the **Advanced** tab, enter the required **Environment Variables** for the container billing settings of the ACI resource:

KEY	VALUE
<code>apikey</code>	Copied from the Keys page of the resource. It is a 32 alphanumeric-character string with no spaces or dashes, <code>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</code> .
<code>billing</code>	Copied from the Overview page of the resource.
<code>eula</code>	<code>accept</code>

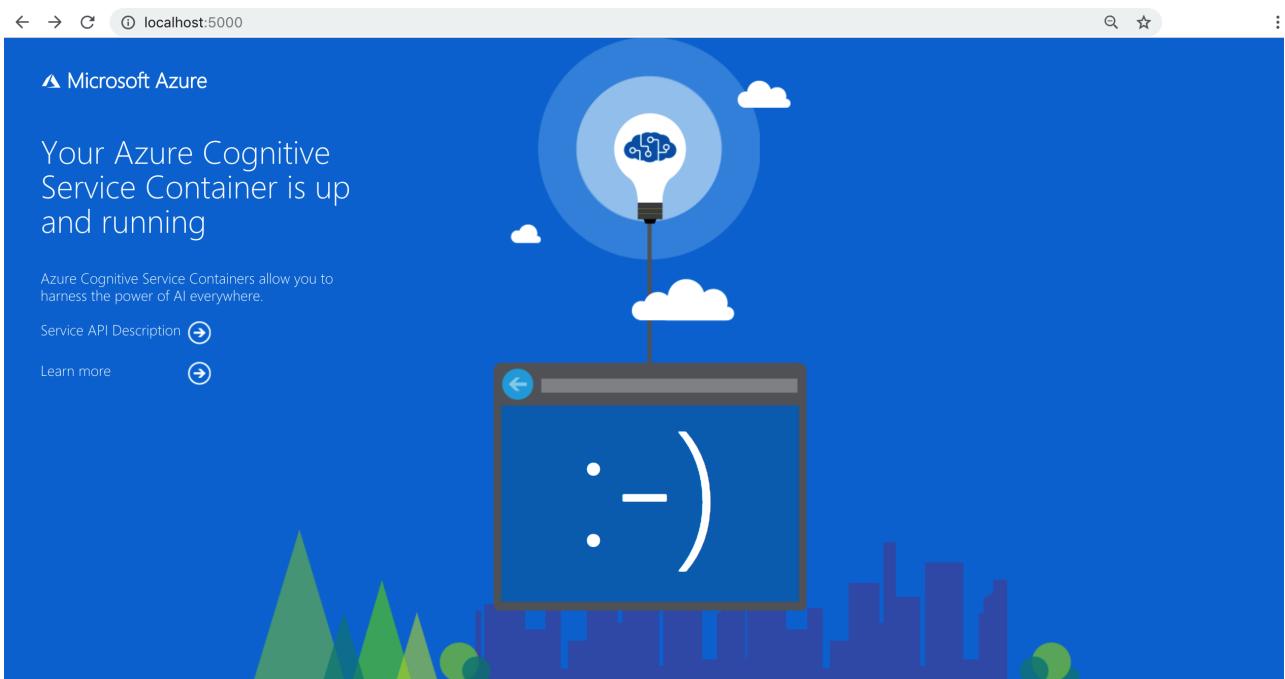
5. Click **Review and Create**

6. After validation passes, click **Create** to finish the creation process
7. When the resource is successfully deployed, it's ready

Validate that a container is running

There are several ways to validate that the container is running.

REQUEST	PURPOSE
<code>http://localhost:5000/</code>	The container provides a home page.
<code>http://localhost:5000/status</code>	Requested with GET, to validate that the container is running without causing an endpoint query. This request can be used for Kubernetes liveness and readiness probes .
<code>http://localhost:5000/swagger</code>	The container provides a full set of documentation for the endpoints and a <code>Try it now</code> feature. With this feature, you can enter your settings into a web-based HTML form and make the query without having to write any code. After the query returns, an example CURL command is provided to demonstrate the HTTP headers and body format that's required.



Next steps

Let's continue working with Azure Cognitive Services containers.

[Use more Cognitive Services Containers](#)

Export and delete your customer data in Language Understanding (LUIS) in Cognitive Services

7/26/2019 • 2 minutes to read • [Edit Online](#)

Delete customer data to ensure privacy and compliance.

Summary of customer data request features

Language Understanding Intelligent Service (LUIS) preserves customer content to operate the service, but the LUIS user has full control over viewing, exporting, and deleting their data. This can be done through the LUIS web portal or the [LUIS Authoring \(also known as Programmatic\) APIs](#).

NOTE

This article provides steps for how to delete personal data from the device or service and can be used to support your obligations under the GDPR. If you're looking for general info about GDPR, see the [GDPR section of the Service Trust portal](#).

Customer content is stored encrypted in Microsoft regional Azure storage and includes:

- User account content collected at registration
- Training data required to build the models
- Logged user queries used by [active learning](#) to help improve the model
 - Users can turn off query logging by appending `&log=false` to the request, details [here](#)

Deleting customer data

LUIS users have full control to delete any user content, either through the LUIS web portal or the LUIS Authoring (also known as Programmatic) APIs. The following table displays links assisting with both:

	USER ACCOUNT	APPLICATION	EXAMPLE UTTERANCE(S)	END-USER QUERIES
Portal	Link	Link	Link	Active learning utterances Logged Utterances
APIs	Link	Link	Link	Link

Exporting customer data

LUIS users have full control to view the data on the portal, however it must be exported through the LUIS Authoring (also known as Programmatic) APIs. The following table displays links assisting with data exports via the LUIS Authoring (also known as Programmatic) APIs:

	USER ACCOUNT	APPLICATION	UTTERANCE(S)	END-USER QUERIES
APIs	Link	Link	Link	Link

Location of active learning

To enable [active learning](#), users' logged utterances, received at the published LUIS endpoints, are stored in the following Azure geographies:

- [Europe](#)
- [Australia](#)
- [United States](#)

With the exception of active learning data (detailed below), LUIS follows the [data storage practices for regional services](#).

Europe

The [eu.luis.ai](#) portal and Europe Authoring (also known as Programmatic APIs) are hosted in Azure's Europe geography. The eu.luis.ai portal and Europe Authoring (also known as Programmatic APIs) support deployment of endpoints to the following Azure geographies:

- Europe
- France
- United Kingdom

When deploying to these Azure geographies, the utterances received by the endpoint from end users of your app will be stored in Azure's Europe geography for active learning. You can disable active learning, see [Disable active learning](#). To manage stored utterances, see [Delete utterance](#).

Australia

The [au.luis.ai](#) portal and Australia Authoring (also known as Programmatic APIs) are hosted in Azure's Australia geography. The au.luis.ai portal and Australia Authoring (also known as Programmatic APIs) support deployment of endpoints to the following Azure geographies:

- Australia

When deploying to these Azure geographies, the utterances received by the endpoint from end users of your app will be stored in Azure's Australia geography for active learning. You can disable active learning, see [Disable active learning](#). To manage stored utterances, see [Delete utterance](#).

United States

The [luis.ai](#) portal and United States Authoring (also known as Programmatic APIs) are hosted in Azure's United States geography. The luis.ai portal and United States Authoring (also known as Programmatic APIs) support deployment of endpoints to the following Azure geographies:

- Azure geographies not supported by the Europe or Australia authoring regions

When deploying to these Azure geographies, the utterances received by the endpoint from end users of your app will be stored in Azure's United States geography for active learning. You can disable active learning, see [Disable active learning](#). To manage stored utterances, see [Delete utterance](#).

Next steps

[LUIS regions reference](#)

Authoring and publishing regions and the associated keys

8/9/2019 • 3 minutes to read • [Edit Online](#)

Three authoring regions are supported by corresponding LUIS portals. To publish a LUIS app to more than one region, you need at least one key per region.

LUIS Authoring regions

There are three LUIS authoring portals, based on region. You must author and publish in the same region.

LUIS	AUTHORING REGION	AZURE REGION NAME
www.luis.ai	U.S. not Europe not Australia	<code>westus</code>
au.luis.ai	Australia	<code>australiaeast</code>
eu.luis.ai	Europe	<code>westeurope</code>

Authoring regions have [paired fail-over regions](#).

Publishing regions and Azure resources

The app is published to all regions associated with the LUIS resources added in the LUIS portal. For example, for an app created on www.luis.ai, if you create a LUIS or Cognitive Service resource in **westus** and [add it to the app as a resource](#), the app is published in that region.

Public apps

A public app is published in all regions so that a user with a region-based LUIS resource key can access the app in whichever region is associated with their resource key.

Publishing regions are tied to authoring regions

The authoring region app can only be published to a corresponding publish region. If your app is currently in the wrong authoring region, export the app, and import it into the correct authoring region for your publishing region.

LUIS apps created on <https://www.luis.ai> can be published to all endpoints except the [European](#) and [Australian](#) regions.

Publishing to Europe

To publish to the European regions, you create LUIS apps at <https://eu.luis.ai> only. If you attempt to publish anywhere else using a key in the Europe region, LUIS displays a warning message. Instead, use <https://eu.luis.ai>. LUIS apps created at <https://eu.luis.ai> don't automatically migrate to other regions. Export and then import the LUIS app in order to migrate it.

Europe publishing regions

GLOBAL REGION	AUTHORING API REGION & AUTHORING WEBSITE	PUBLISHING & QUERYING REGION API REGION NAME	ENDPOINT URL FORMAT
Europe	<code>westeurope</code> eu.luis.ai	France Central <code>francecentral</code>	https://francecentral.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY

GLOBAL REGION	AUTHORING API REGION & AUTHORIZING WEBSITE	PUBLISHING & QUERYING REGION API REGION NAME	ENDPOINT URL FORMAT
Europe	westeurope eu.luis.ai	North Europe northeurope	https://northeurope.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY
Europe	westeurope eu.luis.ai	West Europe westeurope	https://westeurope.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY
Europe	westeurope eu.luis.ai	UK South uksouth	https://uksouth.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY

Publishing to Australia

To publish to the Australian regions, you create LUIS apps at <https://au.luis.ai> only. If you attempt to publish anywhere else using a key in the Australian region, LUIS displays a warning message. Instead, use <https://au.luis.ai>. LUIS apps created at <https://au.luis.ai> don't automatically migrate to other regions. Export and then import the LUIS app in order to migrate it.

Australia publishing regions

GLOBAL REGION	AUTHORING API REGION & AUTHORIZING WEBSITE	PUBLISHING & QUERYING REGION API REGION NAME	ENDPOINT URL FORMAT
Australia	australiaeast au.luis.ai	Australia East australiaeast	https://australiaeast.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY

Publishing to other regions

To publish to the other regions, you create LUIS apps at <https://www.luis.ai> only.

Other publishing regions

GLOBAL REGION	AUTHORING API REGION & AUTHORIZING WEBSITE	PUBLISHING & QUERYING REGION API REGION NAME	ENDPOINT URL FORMAT
Africa	westus www.luis.ai	South Africa North southafricanorth	https://southafricanorth.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY
Asia	westus www.luis.ai	Central India centralindia	https://centralindia.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY

GLOBAL REGION	AUTHORING API REGION & AUTHORING WEBSITE	PUBLISHING & QUERYING REGION <small>API REGION NAME</small>	ENDPOINT URL FORMAT
Asia	westus <a data-bbox="536 242 636 271" href="http://www.luis.ai">www.luis.ai	East Asia eastasia	<a data-bbox="1160 208 1440 345" href="https://eastasia.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY">https://eastasia.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY
Asia	westus <a data-bbox="536 437 636 467" href="http://www.luis.ai">www.luis.ai	Japan East japaneast	<a data-bbox="1160 404 1440 541" href="https://japaneast.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY">https://japaneast.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY
Asia	westus <a data-bbox="536 617 636 646" href="http://www.luis.ai">www.luis.ai	Japan West japanwest	<a data-bbox="1160 583 1440 720" href="https://japanwest.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY">https://japanwest.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY
Asia	westus <a data-bbox="536 797 636 826" href="http://www.luis.ai">www.luis.ai	Korea Central koreacentral	<a data-bbox="1160 763 1440 900" href="https://koreacentral.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY">https://koreacentral.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY
Asia	westus <a data-bbox="536 977 636 1006" href="http://www.luis.ai">www.luis.ai	Southeast Asia southeastasia	<a data-bbox="1160 943 1440 1080" 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
North America	westus <a data-bbox="536 1156 636 1185" href="http://www.luis.ai">www.luis.ai	Canada Central canadacentral	<a data-bbox="1160 1123 1440 1260" href="https://canadacentral.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY">https://canadacentral.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY
North America	westus <a data-bbox="536 1336 636 1365" href="http://www.luis.ai">www.luis.ai	Central US centralus	<a data-bbox="1160 1302 1440 1439" href="https://centralus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY">https://centralus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY
North America	westus <a data-bbox="536 1516 636 1545" href="http://www.luis.ai">www.luis.ai	East US eastus	<a data-bbox="1160 1482 1440 1619" href="https://eastus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY">https://eastus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY
North America	westus <a data-bbox="536 1695 636 1724" href="http://www.luis.ai">www.luis.ai	East US 2 eastus2	<a data-bbox="1160 1662 1440 1799" 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
North America	westus <a data-bbox="536 1875 636 1904" href="http://www.luis.ai">www.luis.ai	North Central US northcentralus	<a data-bbox="1160 1841 1440 1978" href="https://northcentralus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY">https://northcentralus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY

GLOBAL REGION	AUTHORING API REGION & AUTHORIZING WEBSITE	PUBLISHING & QUERYING REGION API REGION NAME	ENDPOINT URL FORMAT
North America	westus www.luis.ai	South Central US southcentralus	https://southcentralus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY
North America	westus www.luis.ai	West Central US westcentralus	https://westcentralus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY
North America	westus www.luis.ai	West US westus	https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY
North America	westus www.luis.ai	West US 2 westus2	https://westus2.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY
South America	westus www.luis.ai	Brazil South brazilsouth	https://brazilsouth.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY

Endpoints

LUIS currently has 2 endpoints: one for authoring and one for query prediction analysis.

PURPOSE	URL
Authoring	https://{{region}}.api.cognitive.microsoft.com/luis/api/v2.0/apps/{{appID}}
Text analysis (query prediction)	https://{{region}}.api.cognitive.microsoft.com/luis/v2.0/apps/{{appID}}?q={{q}}[&timezoneOffset][&verbose][&spellCheck][&staging][&bing-speech-check-subscription-key][&log]

The following table explains the parameters, denoted with curly braces `{}`, in the previous table.

PARAMETER	PURPOSE
region	Azure region - authoring and publishing have different regions
appID	Luis app ID used in URL route and found on app dashboard
q	utterance text sent from client application such as chat bot

Failover regions

Each region has a secondary region to fail over to. Europe fails over inside Europe and Australia fails over inside Australia.

Authoring regions have paired fail-over regions.

Next steps

Boundaries for your LUIS model and keys

8/8/2019 • 2 minutes to read • [Edit Online](#)

Luis has several boundary areas. The first is the [model boundary](#), which controls intents, entities, and features in Luis. The second area is [quota limits](#) based on key type. A third area of boundaries is the [keyboard combination](#) for controlling the Luis website. A fourth area is the [world region mapping](#) between the Luis authoring website and the Luis [endpoint](#) APIs.

Model boundaries

If your app exceeds the Luis model limits and boundaries, consider using a [Luis dispatch](#) app or using a [Luis container](#).

Area	Limit
App name	*Default character max
Batch testing	10 datasets, 1000 utterances per dataset
Explicit list	50 per application
External entities	no limits
Intents	500 per application: 499 custom intents, and the required <i>None</i> intent. <i>Dispatch-based</i> application has corresponding 500 dispatch sources.
List entities	Parent: 50, child: 20,000 items. Canonical name is *default character max. Synonym values have no length restriction.
Machine-learned entities + roles: composite, simple, entity role	A limit of either 100 parent entities or 330 entities, whichever limit the user hits first. A role counts as an entity for the purpose of this boundary. An example is a composite with a simple entity which has 2 roles is: 1 composite + 1 simple + 2 roles = 4 of the 330 entities.
Preview - Dynamic list entities	2 lists of ~1k per query prediction endpoint request
Patterns	500 patterns per application. Maximum length of pattern is 400 characters. 3 Pattern.any entities per pattern Maximum of 2 nested optional texts in pattern
Pattern.any	100 per application, 3 pattern.any entities per pattern
Phrase list	10 phrase lists, 5,000 items per list
Prebuilt entities	no limit

AREA	LIMIT
Regular expression entities	20 entities 500 character max. per regular expression entity pattern
Roles	300 roles per application. 10 roles per entity
Utterance	500 characters
Utterances	15,000 per application - there is no limit on the number of utterances per intent
Versions	no limit
Version name	10 characters restricted to alphanumeric and period (.)

*Default character max is 50 characters.

Object naming

Do not use the following characters in the following names.

OBJECT	EXCLUDE CHARACTERS
Intent, entity, and role names	: \$
Version name	\ / : ? & = * + () % @ \$\br/>~ ! #

Key usage

Language Understand has separate keys, one type for authoring, and one type for querying the prediction endpoint. To learn more about the differences between key types, see [Authoring and query prediction endpoint keys in LUIS](#).

Key limits

The authoring key has different limits for authoring and endpoint. The LUIS service endpoint key is only valid for endpoint queries.

KEY	AUTHORING	ENDPOINT	PURPOSE
Language Understanding Authoring/Starter	1 million/month, 5/second	1 thousand/month, 5/second	Authoring your LUIS app
Language Understanding Subscription - F0 - Free tier	invalid	10 thousand/month, 5/second	Querying your LUIS endpoint
Language Understanding Subscription - S0 - Basic tier	invalid	50/second	Querying your LUIS endpoint
Cognitive Service Subscription - S0 - Standard tier	invalid	50/second	Querying your LUIS endpoint
Sentiment analysis integration	invalid	no charge	Adding sentiment information including key phrase data extraction
Speech integration	invalid	\$5.50 USD/1 thousand endpoint requests	Convert spoken utterance to text utterance and return LUIS results

Keyboard controls

KEYBOARD INPUT	DESCRIPTION
Control+E	switches between tokens and entities on utterances list

Website sign-in time period

Your sign-in access is for **60 minutes**. After this time period, you will get this error. You need to sign in again.

Application settings

8/9/2019 • 2 minutes to read • [Edit Online](#)

These application settings are stored in the [exported](#) app and [updated](#) with the REST APIs. Changing your app version settings resets your app training status to untrained.

SETTING	DEFAULT VALUE	NOTES
NormalizePunctuation	True	Removes punctuation.
NormalizeDiacritics	True	Removes diacritics.

Diacritics normalization

Turn on utterance normalization for diacritics to your LUIS JSON app file in the `settings` parameter.

```
"settings": [  
    {"name": "NormalizeDiacritics", "value": "true"}  
]
```

The following utterances show how diacritics normalization impacts utterances:

WITH DIACRITICS SET TO FALSE	WITH DIACRITICS SET TO TRUE
quiero tomar una piña colada	quiero tomar una pina colada

Language support for diacritics

Brazilian portuguese `pt-br` diacritics

DIACRITICS SET TO FALSE	DIACRITICS SET TO TRUE
á	a
â	a
ã	a
à	a
ç	c
é	e
ê	e
í	i

DIACRITICS SET TO FALSE ó ô õ ú**DIACRITICS SET TO TRUE** o o o uDutch nl-nl diacritics**DIACRITICS SET TO FALSE** á à é ë è ï í ó ö ú ü**DIACRITICS SET TO TRUE** a a e e e i i o o u uFrench fr- diacritics

This includes both french and canadian subcultures.

DIACRITICS SET TO FALSE é à è ù**DIACRITICS SET TO TRUE** e a e u

DIACRITICS SET TO FALSE â ê î ô û ç è ï ü ÿ**DIACRITICS SET TO TRUE** a e i o u c e i u y**German de-de diacritics****DIACRITICS SET TO FALSE** ä ö ü**DIACRITICS SET TO TRUE** a o u**Italian it-it diacritics****DIACRITICS SET TO FALSE** à è é ì î ò ó**DIACRITICS SET TO TRUE** a e e i i o o

DIACRITICS SET TO FALSE

ù

ú

DIACRITICS SET TO TRUE

u

u

Spanish [es-] diacritics

This includes both spanish and canadian mexican.

DIACRITICS SET TO FALSE

á

é

í

ó

ú

ü

ñ

DIACRITICS SET TO TRUE

a

e

i

o

u

u

u

Punctuation normalization

Turn on utterance normalization for punctuation to your LUIS JSON app file in the `settings` parameter.

```
"settings": [
    {"name": "NormalizePunctuation", "value": "true"}
]
```

The following utterances show how diacritics impacts utterances:

WITH DIACRITICS SET TO FALSE

Hmm..... I will take the cappuccino

WITH DIACRITICS SET TO TRUE

Hmm I will take the cappuccino

Punctuation removed

The following punctuation is removed with `NormalizePunctuation` is set to true.

PUNCTUATION

-

.

'

PUNCTUATION

"

\

/

?

!

-

,

;

:

(

)

[

{

}

+

i

Entities per culture in your LUIS model

8/9/2019 • 4 minutes to read • [Edit Online](#)

Language Understanding (LUIS) provides prebuilt entities. When a prebuilt entity is included in your application, LUIS includes the corresponding entity prediction in the endpoint response. All example utterances are also labeled with the entity. The behavior of prebuilt entities **can't** 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.

CULTURE	SUBCULTURES	NOTES
Chinese	zh-CN	
Dutch	nl-NL	
English	en-US (American)	
French	fr-CA (Canada), fr-FR (France),	
German	de-DE	
Italian	it-IT	
Japanese	ja-JP	
Korean	ko-KR	
Portuguese	pt-BR (Brazil)	
Spanish	es-ES (Spain), es-MX (Mexico)	
Turkish	turkish	No prebuilt entities supported in Turkish

Chinese entity support

The following entities are supported:

PREBUILT ENTITY	ZH-CN
Age: year month week day	✓
Currency (money): dollar fractional unit (ex: penny)	✓

PREBUILT ENTITY	ZH-CN
DatetimeV2: date daterange time timerange	✓
Dimension: volume area weight information (ex: bit/byte) length (ex: meter) speed (ex: mile per hour)	✓
Email	✓
GeographyV2	-
KeyPhrase	-
Number	✓
Ordinal	✓
OrdinalV2	-
Percentage	✓
PersonName	✓
Phonenumber	✓
Temperature: fahrenheit kelvin rankine delisle celsius	✓
URL	✓

Dutch entity support

The following entities are supported:

PREBUILT ENTITY	NL-NL
Age: year month week day	✓

PREBUILT ENTITY	NL-NL
Currency (money): dollar fractional unit (ex: penny)	✓
Datetime	-
Dimension: volume area weight information (ex: bit/byte) length (ex: meter) speed (ex: mile per hour)	✓
Email	✓
GeographyV2	-
KeyPhrase	✓
Number	✓
Ordinal	✓
OrdinalV2	-
Percentage	✓
PersonName	-
Phonenumber	✓
Temperature: fahrenheit kelvin rankine delisle celsius	✓
URL	✓

English (American) entity support

The following entities are supported:

PREBUILT ENTITY	EN-US
Age: year month week day	✓

PREBUILT ENTITY	EN-US
Currency (money): dollar fractional unit (ex: penny)	✓
DatetimeV2: date daterange time timerange	✓
Dimension: volume area weight information (ex: bit/byte) length (ex: meter) speed (ex: mile per hour)	✓
Email	✓
GeographyV2	✓
KeyPhrase	✓
Number	✓
Ordinal	✓
OrdinalV2	✓
Percentage	✓
PersonName	✓
Phonenumber	✓
Temperature: fahrenheit kelvin rankine delisle celsius	✓
URL	✓

French (France) entity support

The following entities are supported:

PREBUILT ENTITY	FR-FR
Age: year month week day	✓
Currency (money): dollar fractional unit (ex: penny)	✓
DatetimeV2: date daterange time timerange	✓
Dimension: volume area weight information (ex: bit/byte) length (ex: meter) speed (ex: mile per hour)	✓
Email	✓
GeographyV2	-
KeyPhrase	✓
Number	✓
Ordinal	✓
OrdinalV2	-
Percentage	✓
PersonName	-
Phonenumber	✓
Temperature: fahrenheit kelvin rankine delisle celsius	✓
URL	✓

French (Canadian) entity support

The following entities are supported:

PREBUILT ENTITY	FR-CA
Age: year month week day	✓
Currency (money): dollar fractional unit (ex: penny)	✓
DatetimeV2: date daterange time timerange	✓
Dimension: volume area weight information (ex: bit/byte) length (ex: meter) speed (ex: mile per hour)	✓
Email	✓
GeographyV2	-
KeyPhrase	✓
Number	✓
Ordinal	✓
OrdinalV2	-
Percentage	✓
PersonName	-
Phonenumber	✓
Temperature: fahrenheit kelvin rankine delisle celsius	✓
URL	✓

German entity support

The following entities are supported:

PREBUILT ENTITY	DE-DE
Age: year month week day	✓
Currency (money): dollar fractional unit (ex: penny)	✓
DatetimeV2: date daterange time timerange	✓
Dimension: volume area weight information (ex: bit/byte) length (ex: meter) speed (ex: mile per hour)	✓
Email	✓
GeographyV2	-
KeyPhrase	✓
Number	✓
Ordinal	✓
OrdinalV2	-
Percentage	✓
PersonName	-
Phonenumber	✓
Temperature: fahrenheit kelvin rankine delisle celsius	✓
URL	✓

Italian entity support

The following entities are supported:

PREBUILT ENTITY	IT-IT
Age: year month week day	✓
Currency (money): dollar fractional unit (ex: penny)	✓
Datetime	-
Dimension: volume area weight information (ex: bit/byte) length (ex: meter) speed (ex: mile per hour)	✓
Email	✓
GeographyV2	-
KeyPhrase	✓
Number	✓
Ordinal	✓
OrdinalV2	-
Percentage	✓
PersonName	-
Phonenumber	✓
Temperature: fahrenheit kelvin rankine delisle celsius	✓
URL	✓

Japanese entity support

The following entities are supported:

PREBUILT ENTITY	JA-JP
Age: year month week day	✓
Currency (money): dollar fractional unit (ex: penny)	✓
Datetime	-
Dimension: volume area weight information (ex: bit/byte) length (ex: meter) speed (ex: mile per hour)	✓
Email	✓
GeographyV2	-
KeyPhrase	✓
Number	✓
Ordinal	✓
OrdinalV2	-
Percentage	✓
PersonName	-
Phonenumber	✓
Temperature: fahrenheit kelvin rankine delisle celsius	✓
URL	✓

Korean entity support

The following entities are supported:

PREBUILT ENTITY	KO-KR
Age: year month week day	-
Currency (money): dollar fractional unit (ex: penny)	-
Datetime	-
Dimension: volume area weight information (ex: bit/byte) length (ex: meter) speed (ex: mile per hour)	-
Email	✓
GeographyV2	-
KeyPhrase	✓
Number	-
Ordinal	-
OrdinalV2	-
Percentage	-
PersonName	-
Phonenumber	✓
Temperature: fahrenheit kelvin rankine delisle celsius	-
URL	✓

Portuguese (Brazil) entity support

The following entities are supported:

PREBUILT ENTITY	PT-BR
Age: year month week day	✓
Currency (money): dollar fractional unit (ex: penny)	✓
DatetimeV2: date daterange time timerange	✓
Dimension: volume area weight information (ex: bit/byte) length (ex: meter) speed (ex: mile per hour)	✓
Email	✓
GeographyV2	-
KeyPhrase	✓
Number	✓
Ordinal	✓
OrdinalV2	-
Percentage	✓
PersonName	-
Phonenumber	✓
Temperature: fahrenheit kelvin rankine delisle celsius	✓
URL	✓

Spanish (Spain) entity support

The following entities are supported:

PREBUILT ENTITY	ES-ES
Age: year month week day	✓
Currency (money): dollar fractional unit (ex: penny)	✓
DatetimeV2: date daterange time timerange	✓
Dimension: volume area weight information (ex: bit/byte) length (ex: meter) speed (ex: mile per hour)	✓
Email	✓
GeographyV2	-
KeyPhrase	✓
Number	✓
Ordinal	✓
OrdinalV2	-
Percentage	✓
PersonName	-
Phonenumber	✓
Temperature: fahrenheit kelvin rankine delisle celsius	✓
URL	✓

Spanish (Mexico) entity support

The following entities are supported:

PREBUILT ENTITY	ES-MX
Age: year month week day	-
Currency (money): dollar fractional unit (ex: penny)	-
DatetimeV2: date daterange time timerange	-
Dimension: volume area weight information (ex: bit/byte) length (ex: meter) speed (ex: mile per hour)	-
Email	✓
GeographyV2	-
KeyPhrase	✓
Number	✓
Ordinal	-
OrdinalV2	-
Percentage	-
PersonName	-
Phonenumber	✓
Temperature: fahrenheit kelvin rankine delisle celsius	-
URL	✓

See notes on [Deprecated prebuilt entities](#)

KeyPhrase is not available in all subcultures of Portuguese (Brazil) - [pt-BR](#).

Turkish entity support

There are no prebuilt entities supported in Turkish.

Contribute to prebuilt entity cultures

The prebuilt entities are developed in the Recognizers-Text open-source project. [Contribute](#) to the project. This project includes examples of currency per culture.

GeographyV2 and PersonName are not included in the Recognizers-Text project. For issues with these prebuilt entities, please open a [support request](#).

Next steps

Learn about the [number](#), [datetimeV2](#), and [currency](#) entities.

Age prebuilt entity for a LUIS app

8/9/2019 • 2 minutes to read • [Edit Online](#)

The prebuilt age entity captures the age value both numerically and in terms of days, weeks, months, and years. Because this entity is already trained, you do not need to add example utterances containing age to the application intents. Age entity is supported in [many cultures](#).

Types of age

Age is managed from the [Recognizers-text](#) GitHub repository

Resolution for prebuilt age entity

API version 2.x

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

```
{
  "query": "A 90 day old utilities bill is quite late.",
  "topScoringIntent": {
    "intent": "None",
    "score": 0.8236133
  },
  "entities": [
    {
      "entity": "90 day old",
      "type": "builtin.age",
      "startIndex": 2,
      "endIndex": 11,
      "resolution": {
        "unit": "Day",
        "value": "90"
      }
    }
  ]
}
```

Preview API version 3.x

The following JSON is with the `verbose` parameter set to `false`:

```
{
  "query": "A 90 day old utilities bill is quite late.",
  "prediction": {
    "normalizedQuery": "a 90 day old utilities bill is quite late.",
    "topIntent": "None",
    "intents": {
      "None": {
        "score": 0.558252
      }
    },
    "entities": {
      "age": [
        {
          "number": 90,
          "unit": "Day"
        }
      ]
    }
  }
}
```

The following JSON is with the `verbose` parameter set to `true`:

```
{
  "query": "A 90 day old utilities bill is quite late.",
  "prediction": {
    "normalizedQuery": "a 90 day old utilities bill is quite late.",
    "topIntent": "None",
    "intents": {
      "None": {
        "score": 0.558252
      }
    },
    "entities": {
      "age": [
        {
          "number": 90,
          "unit": "Day"
        }
      ],
      "$instance": {
        "age": [
          {
            "type": "builtin.age",
            "text": "90 day old",
            "startIndex": 2,
            "length": 10,
            "modelTypeId": 2,
            "modelType": "Prebuilt Entity Extractor"
          }
        ]
      }
    }
  }
}
```

Next steps

Learn about the [currency](#), [datetimeV2](#), and [dimension](#) entities.

Currency prebuilt entity for a LUIS app

8/9/2019 • 2 minutes to read • [Edit Online](#)

The prebuilt currency entity detects currency in many denominations and countries/regions, regardless of LUIS app culture. Because this entity is already trained, you do not need to add example utterances containing currency to the application intents. Currency entity is supported in [many cultures](#).

Types of currency

Currency is managed from the [Recognizers-text](#) GitHub repository

Resolution for currency entity

API version 2.x

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.currency",
      "startIndex": 23,
      "endIndex": 28,
      "resolution": {
        "unit": "Dollar",
        "value": "10.99"
      }
    }
  ]
}
```

Preview API version 3.x

The following JSON is with the `verbose` parameter set to `false`:

```
{  
    "query": "search for items under $10.99",  
    "prediction": {  
        "normalizedQuery": "search for items under $10.99",  
        "topIntent": "None",  
        "intents": {  
            "None": {  
                "score": 0.605889857  
            }  
        },  
        "entities": {  
            "money": [  
                {  
                    "number": 10.99,  
                    "unit": "Dollar"  
                }  
            ]  
        }  
    }  
}
```

The following JSON is with the `verbose` parameter set to `true`:

```
{  
    "query": "search for items under $10.99",  
    "prediction": {  
        "normalizedQuery": "search for items under $10.99",  
        "topIntent": "None",  
        "intents": {  
            "None": {  
                "score": 0.605889857  
            }  
        },  
        "entities": {  
            "money": [  
                {  
                    "number": 10.99,  
                    "unit": "Dollar"  
                }  
            ],  
            "$instance": {  
                "money": [  
                    {  
                        "type": "builtin.currency",  
                        "text": "$10.99",  
                        "startIndex": 23,  
                        "length": 6,  
                        "modelTypeId": 2,  
                        "modelType": "Prebuilt Entity Extractor"  
                    }  
                ]  
            }  
        }  
    }  
}
```

Next steps

Learn about the [datetimeV2](#), [dimension](#), and [email](#) entities.

DatetimeV2 prebuilt entity for a LUIS app

8/9/2019 • 5 minutes to read • [Edit Online](#)

The **datetimeV2** prebuilt entity extracts date and time values. These values resolve in a standardized format for client programs to consume. When an utterance has a date or time that isn't complete, LUIS includes *both past and future values* in the endpoint response. Because this entity is already trained, you do not need to add example utterances containing datetimeV2 to the application intents.

Types of datetimeV2

DatetimeV2 is managed from the [Recognizers-text](#) GitHub repository

Example JSON

The following example JSON response has a `datetimeV2` entity with a subtype of `datetime`. For examples of other types of datetimeV2 entities, see [Subtypes of datetimeV2](#).

```
"entities": [
  {
    "entity": "8am on may 2nd 2019",
    "type": "builtin.datetimeV2.datetime",
    "startIndex": 0,
    "endIndex": 18,
    "resolution": {
      "values": [
        {
          "timex": "2019-05-02T08",
          "type": "datetime",
          "value": "2019-05-02 08:00:00"
        }
      ]
    }
  }
]
```

JSON property descriptions

PROPERTY NAME	PROPERTY TYPE AND DESCRIPTION
Entity	string - Text extracted from the utterance with type of date, time, date range, or time range.
type	string - One of the subtypes of datetimeV2
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	Has a <code>values</code> array that has one, two, or four values of resolution .

PROPERTY NAME	PROPERTY TYPE AND DESCRIPTION
end	The end value of a time, or date range, in the same format as <code>value</code> . Only used if <code>type</code> is <code>daterange</code> , <code>timerange</code> , or <code>datetimerange</code>

Subtypes of datetimeV2

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

- `date`
- `time`
- `daterange`
- `timerange`
- `datetimerange`
- `duration`
- `set`

Values of resolution

- 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 datetimeV2 value is ambiguous. Ambiguity includes lack of specific year, time, or time range. See [Ambiguous dates](#) for examples. When the time is ambiguous for A.M. or P.M., both values are included.
- The array has four elements if the utterance has two elements with ambiguity. This ambiguity includes elements that have:
 - A date or date range that is ambiguous as to year
 - A time or time range that is ambiguous as to A.M. or P.M. For example, 3:00 April 3rd.

Each element of the `values` array may have the following fields:

PROPERTY NAME	PROPERTY DESCRIPTION
timex	time, date, or date range expressed in TIMEX format that follows the ISO 8601 standard and the TIMEX3 attributes for annotation using the TimeML language. This annotation is described in the TIMEX guidelines .
type	The subtype, which can be one of the following items: <code>datetime</code> , <code>date</code> , <code>time</code> , <code>daterange</code> , <code>timerange</code> , <code>datetimerange</code> , <code>duration</code> , <code>set</code> .
value	Optional. A datetime object in the Format yyyy:MM:dd (date), HH:mm:ss (time) yyyy:MM:dd 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> .

Valid date values

The **datetimeV2** supports dates between the following ranges:

MIN	MAX
1st January 1900	31st December 2099

Ambiguous dates

If the date can be in the past or future, LUIS provides both values. An example is an utterance that includes the month and date without the 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.
- When 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". This resolution assumes that today's date is a date between May 2nd 2017 and May 1st 2018. Fields with `x` in the `timex` field are parts of the date that aren't 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": "2019-05-02"
        },
        {
          "timex": "XXXX-05-02",
          "type": "date",
          "value": "2020-05-02"
        }
      ]
    }
  }
]
```

Date range resolution examples for numeric date

The `datetimev2` entity extracts 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 next year. In the `timex` field, the `xxxx` values indicate the ambiguity of the year. `P3D` indicates the time period is three 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": "2019-05-02",
          "end": "2019-05-05"
        }
      ]
    }
  }
]
```

Date range resolution examples for day of week

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

```
"entities": [
  {
    "entity": "tuesday to thursday",
    "type": "builtin.datetimeV2.daterange",
    "startIndex": 0,
    "endIndex": 19,
    "resolution": {
      "values": [
        {
          "timex": "(XXXX-WXX-2,XXXX-WXX-4,P2D)",
          "type": "daterange",
          "start": "2019-04-30",
          "end": "2019-05-02"
        }
      ]
    }
  }
]
```

Ambiguous time

The values array has two time elements if the time, or time range is ambiguous. When there's an ambiguous time, values have both the A.M. and P.M. times.

Time range resolution example

The following example shows how LUIS uses **datetimeV2** to resolve the utterance that has a time range.

```

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

```

Preview API version 3.x

DatetimeV2 JSON response has changed in the API V3.

Changes from API V2:

- `datetimeV2.timex.type` property is no longer returned because it is returned at the parent level, `datetimev2.type`.
- The `datetimeV2.timex` property has been renamed to `datetimeV2.value`.

For the utterance, `8am on may 2nd 2017`, the V3 version of DatetimeV2 is:

```

{
  "query": "8am on may 2nd 2017",
  "prediction": {
    "normalizedQuery": "8am on may 2nd 2017",
    "topIntent": "None",
    "intents": {
      "None": {
        "score": 0.6826963
      }
    },
    "entities": {
      "datetimeV2": [
        {
          "type": "datetime",
          "values": [
            {
              "timex": "2017-05-02T08",
              "value": "2017-05-02 08:00:00"
            }
          ]
        }
      ]
    }
  }
}

```

The following JSON is with the `verbose` parameter set to `false`:

```
{  
    "query": "8am on may 2nd 2017",  
    "prediction": {  
        "normalizedQuery": "8am on may 2nd 2017",  
        "topIntent": "None",  
        "intents": {  
            "None": {  
                "score": 0.6826963  
            }  
        },  
        "entities": {  
            "datetimeV2": [  
                {  
                    "type": "datetime",  
                    "values": [  
                        {  
                            "timex": "2017-05-02T08",  
                            "value": "2017-05-02 08:00:00"  
                        }  
                    ]  
                }  
            ],  
            "$instance": {  
                "datetimeV2": [  
                    {  
                        "type": "builtin.datetimeV2.datetime",  
                        "text": "8am on may 2nd 2017",  
                        "startIndex": 0,  
                        "length": 19,  
                        "modelTypeId": 2,  
                        "modelType": "Prebuilt Entity Extractor",  
                        "recognitionSources": [  
                            "model"  
                        ]  
                    }  
                ]  
            }  
        }  
    }  
}
```

Deprecated prebuilt datetime

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

To replace `datetime` with `datetimeV2` in your LUIS app, complete the following steps:

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**.

Next steps

Learn about the [dimension](#), [email](#) entities, and [number](#).

Dimension prebuilt entity for a LUIS app

8/9/2019 • 2 minutes to read • [Edit Online](#)

The prebuilt dimension entity detects various types of dimensions, regardless of the LUIS app culture. Because this entity is already trained, you do not need to add example utterances containing dimensions to the application intents. Dimension entity is supported in [many cultures](#).

Types of dimension

Dimension is managed from the [Recognizers-text](#) GitHub repository

Resolution for dimension entity

API version 2.x

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

```
{
  "query": "it takes more than 10 1/2 miles of cable and wire to hook it all up , and 23 computers.",
  "topScoringIntent": {
    "intent": "None",
    "score": 0.762141049
  },
  "intents": [
    {
      "intent": "None",
      "score": 0.762141049
    }
  ],
  "entities": [
    {
      "entity": "10 1/2 miles",
      "type": "builtin.dimension",
      "startIndex": 19,
      "endIndex": 30,
      "resolution": {
        "unit": "Mile",
        "value": "10.5"
      }
    }
  ]
}
```

Preview API version 3.x

The following JSON is with the `verbose` parameter set to `false`:

```
{
  "query": "it takes more than 10 1/2 miles of cable and wire to hook it all up , and 23 computers.",
  "prediction": {
    "normalizedQuery": "it takes more than 10 1/2 miles of cable and wire to hook it all up , and 23 computers.",
    "topIntent": "None",
    "intents": {
      "None": {
        "score": 0.400049
      }
    },
    "entities": {
      "dimension": [
        {
          "number": 10.5,
          "unit": "Mile"
        }
      ]
    }
  }
}
```

The following JSON is with the `verbose` parameter set to `true`:

```
{
  "query": "it takes more than 10 1/2 miles of cable and wire to hook it all up , and 23 computers.",
  "prediction": {
    "normalizedQuery": "it takes more than 10 1/2 miles of cable and wire to hook it all up , and 23 computers.",
    "topIntent": "None",
    "intents": {
      "None": {
        "score": 0.400049
      }
    },
    "entities": {
      "dimension": [
        {
          "number": 10.5,
          "unit": "Mile"
        }
      ],
      "$instance": {
        "dimension": [
          {
            "type": "builtin.dimension",
            "text": "10 1/2 miles",
            "startIndex": 19,
            "length": 12,
            "modelTypeId": 2,
            "modelType": "Prebuilt Entity Extractor"
          }
        ]
      }
    }
  }
}
```

Next steps

Learn about the [email](#), [number](#), and [ordinal](#) entities.

Deprecated prebuilt entities in a LUIS app

8/9/2019 • 3 minutes to read • [Edit Online](#)

The following prebuilt entities are deprecated and can't be added to new LUIS apps.

- **Datetime**: Existing LUIS apps that use **datetime** should be migrated to **datetimeV2**, although the datetime entity continues to function in pre-existing apps that use it.
- **Geography**: Existing LUIS apps that use **geography** is supported until December 2018.
- **Encyclopedia**: Existing LUIS apps that use **encyclopedia** is supported until December 2018.

Geography culture

Geography is available only in the `en-us` locale.

3 Geography subtypes

PREBUILT ENTITY	EXAMPLE UTTERANCE	JSON
<code>builtin.geography.city</code>	<code>seattle</code>	<pre>{ "type": "builtin.geography.city", "entity": "seattle" }</pre>
<code>builtin.geography.city</code>	<code>paris</code>	<pre>{ "type": "builtin.geography.city", "entity": "paris" }</pre>
<code>builtin.geography.country</code>	<code>australia</code>	<pre>{ "type": "builtin.geography.country", "entity": "australia" }</pre>
<code>builtin.geography.country</code>	<code>japan</code>	<pre>{ "type": "builtin.geography.country", "entity": "japan" }</pre>
<code>builtin.geography.pointOfInterest</code>	<code>amazon river</code>	<pre>{ "type": "builtin.geography.pointOfInterest", "entity": "amazon river" }</pre>
<code>builtin.geography.pointOfInterest</code>	<code>sahara desert</code>	<pre>{ "type": "builtin.geography.pointofInterest", "entity": "sahara desert" }</pre>

Encyclopedia culture

Encyclopedia is available only in the `en-us` locale.

Encyclopedia subtypes

Encyclopedia built-in entity includes over 100 sub-types in the following table: In addition, encyclopedia entities often map to multiple types. For example, the query Ronald Reagan yields:

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

PREBUILT ENTITY	PREBUILT 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.master	bruce lee
builtin.encyclopedia.people.person	builtin.encyclopedia.architecture.architect	james gallier
builtin.encyclopedia.people.person	builtin.encyclopedia.geography.mountain	jean couzy
builtin.encyclopedia.people.person	builtin.encyclopedia.celebrities.celebrity	angeline 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

PREBUILT ENTITY	PREBUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.people.person	builtin.encyclopedia.american_football james franklin	
builtin.encyclopedia.people.person	builtin.encyclopedia.cricket.coach	andy flower
builtin.encyclopedia.people.person	builtin.encyclopedia.ice_hockey.coach	david quinn
builtin.encyclopedia.people.person	builtin.encyclopedia.ice_hockey.player	vincent lecavalier
builtin.encyclopedia.people.person	builtin.encyclopedia.government.politi	harold nicolson
builtin.encyclopedia.people.person	builtin.encyclopedia.government.us_pre	barack obama
builtin.encyclopedia.people.person	builtin.encyclopedia.government.us_vic	dick cheney
builtin.encyclopedia.organization.orga	builtin.encyclopedia.organization.orga	united nations
builtin.encyclopedia.organization.orga	builtin.encyclopedia.sports.league	american league
builtin.encyclopedia.organization.orga	builtin.encyclopedia.ice_hockey.confer	western hockey league
builtin.encyclopedia.organization.orga	builtin.encyclopedia.baseball.division	american league east
builtin.encyclopedia.organization.orga	builtin.encyclopedia.baseball.league	major league baseball
builtin.encyclopedia.organization.orga	builtin.encyclopedia.basketball.confer	national basketball league
builtin.encyclopedia.organization.orga	builtin.encyclopedia.basketball.divisi	pacific division
builtin.encyclopedia.organization.orga	builtin.encyclopedia.soccer.league	premier league
builtin.encyclopedia.organization.orga	builtin.encyclopedia.american_football	afc north
builtin.encyclopedia.organization.orga	builtin.encyclopedia.broadcast.broadcast	nebraska educational telecommunications
builtin.encyclopedia.organization.orga	builtin.encyclopedia.broadcast.tv_stat	abc
builtin.encyclopedia.organization.orga	builtin.encyclopedia.broadcast.tv_chan	cnbc world
builtin.encyclopedia.organization.orga	builtin.encyclopedia.broadcast.radio_s	bbc radio 1
builtin.encyclopedia.organization.orga	builtin.encyclopedia.business.operatio	bank of china
builtin.encyclopedia.organization.orga	builtin.encyclopedia.music.record_labe	pixar
builtin.encyclopedia.organization.orga	builtin.encyclopedia.aviation.airline	air france
builtin.encyclopedia.organization.orga	builtin.encyclopedia.automotive.compan	general motors

PREBUILT ENTITY	PREBUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.organization.orga	builtin.encyclopedia.music.musical_in	gibson guitar corporation
builtin.encyclopedia.organization.orga	builtin.encyclopedia.tv.network	cartoon network
builtin.encyclopedia.organization.orga	builtin.encyclopedia.education.educati	cornwall hill college
builtin.encyclopedia.organization.orga	builtin.encyclopedia.education.school	boston arts academy
builtin.encyclopedia.organization.orga	builtin.encyclopedia.education.univers	johns hopkins university
builtin.encyclopedia.organization.orga	builtin.encyclopedia.sports.team	united states national handball team
builtin.encyclopedia.organization.orga	builtin.encyclopedia.basketball.team	chicago bulls
builtin.encyclopedia.organization.orga	builtin.encyclopedia.sports.profession	boston celtics
builtin.encyclopedia.organization.orga	builtin.encyclopedia.cricket.team	mumbai indians
builtin.encyclopedia.organization.orga	builtin.encyclopedia.baseball.team	houston astros
builtin.encyclopedia.organization.orga	builtin.encyclopedia.american_football	green bay packers
builtin.encyclopedia.organization.orga	builtin.encyclopedia.ice_hockey.team	hamilton bulldogs
builtin.encyclopedia.organization.orga	builtin.encyclopedia.soccer.team	fc bayern munich
builtin.encyclopedia.organization.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.championsh	super bowl xxxix
builtin.encyclopedia.time.event	builtin.encyclopedia.award.competition	eurovision song contest 2003
builtin.encyclopedia.tv.series_episode	builtin.encyclopedia.tv.series_episode	the magnificent seven
builtin.encyclopedia.tv.series_episode	builtin.encyclopedia.tv.multipart_tv_e	the deadly assassin
builtin.encyclopedia.commerce.consumer	builtin.encyclopedia.commerce.consumer	nokia lumia 620
builtin.encyclopedia.commerce.consumer	builtin.encyclopedia.music.album	dance pool
builtin.encyclopedia.commerce.consumer	builtin.encyclopedia.automotive.model	pontiac fiero
builtin.encyclopedia.commerce.consumer	builtin.encyclopedia.computer.computer	toshiba satellite

PREBUILT ENTITY	PREBUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.commerce.consumer	builtin.encyclopedia.computer.web_brow	internet explorer
builtin.encyclopedia.commerce.brand	builtin.encyclopedia.commerce.brand	diet coke
builtin.encyclopedia.commerce.brand	builtin.encyclopedia.automotive.make	chrysler
builtin.encyclopedia.music.artist	builtin.encyclopedia.music.artist	michael jackson
builtin.encyclopedia.music.artist	builtin.encyclopedia.music.group	the yardbirds
builtin.encyclopedia.music.music_video	builtin.encyclopedia.music.music_video	the beatles anthology
builtin.encyclopedia.theater.play	builtin.encyclopedia.theater.play	camelot
builtin.encyclopedia.sports.fight_song	builtin.encyclopedia.sports.fight_song	the cougar song
builtin.encyclopedia.film.series	builtin.encyclopedia.film.series	the twilight saga
builtin.encyclopedia.tv.program	builtin.encyclopedia.tv.program	late night with david letterman
builtin.encyclopedia.radio.radio_progr	builtin.encyclopedia.radio.radio_progr	grand ole opry
builtin.encyclopedia.film.film	builtin.encyclopedia.film.film	alice in wonderland
builtin.encyclopedia.cricket.tournamen	builtin.encyclopedia.cricket.tournamen	cricket world cup
builtin.encyclopedia.government.govern	builtin.encyclopedia.government.govern	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.divisi	builtin.encyclopedia.ice_hockey.divisi	hockeyallsvenskan
builtin.encyclopedia.architecture.styl	builtin.encyclopedia.architecture.styl	spanish colonial revival architecture
builtin.encyclopedia.broadcast.produce	builtin.encyclopedia.broadcast.produce	columbia tristar television
builtin.encyclopedia.book.author	builtin.encyclopedia.book.author	adam maxwell
builtin.encyclopedia.religion.founding	builtin.encyclopedia.religion.founding	gautama buddha
builtin.encyclopedia.martial_arts.mart	builtin.encyclopedia.martial_arts.mart	american kenpo
builtin.encyclopedia.sports.school	builtin.encyclopedia.sports.school	yale university
builtin.encyclopedia.business.product_	builtin.encyclopedia.business.product_	canon powershot

PREBUILT ENTITY	PREBUILT ENTITY (SUB-TYPES)	EXAMPLE UTTERANCE
builtin.encyclopedia.internet.website	builtin.encyclopedia.internet.website	bing
builtin.encyclopedia.time.holiday	builtin.encyclopedia.time.holiday	easter
builtin.encyclopedia.food.candy_bar	builtin.encyclopedia.food.candy_bar	cadbury dairy milk
builtin.encyclopedia.finance.stock_exc	builtin.encyclopedia.finance.stock_exc	tokyo stock exchange
builtin.encyclopedia.film.festival	builtin.encyclopedia.film.festival	berlin international film festival

Next steps

Learn about the [dimension](#), [email](#) entities, and [number](#).

Email prebuilt entity for a LUIS app

8/9/2019 • 2 minutes to read • [Edit Online](#)

Email extraction includes the entire email address from an utterance. Because this entity is already trained, you do not need to add example utterances containing email to the application intents. Email entity is supported in `en-us` culture only.

Resolution for prebuilt email

API version 2.x

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

```
{
  "query": "please send the information to patti.owens@microsoft.com",
  "topScoringIntent": {
    "intent": "None",
    "score": 0.811592042
  },
  "intents": [
    {
      "intent": "None",
      "score": 0.811592042
    }
  ],
  "entities": [
    {
      "entity": "patti.owens@microsoft.com",
      "type": "builtin.email",
      "startIndex": 31,
      "endIndex": 55,
      "resolution": {
        "value": "patti.owens@microsoft.com"
      }
    }
  ]
}
```

Preview API version 3.x

The following JSON is with the `verbose` parameter set to `false`:

```
{  
    "query": "please send the information to patti.owens@microsoft.com",  
    "prediction": {  
        "normalizedQuery": "please send the information to patti.owens@microsoft.com",  
        "topIntent": "None",  
        "intents": {  
            "None": {  
                "score": 0.5023781  
            }  
        },  
        "entities": {  
            "email": [  
                "patti.owens@microsoft.com"  
            ]  
        }  
    }  
}
```

The following JSON is with the `verbose` parameter set to `true`:

```
{  
    "query": "please send the information to patti.owens@microsoft.com",  
    "prediction": {  
        "normalizedQuery": "please send the information to patti.owens@microsoft.com",  
        "topIntent": "None",  
        "intents": {  
            "None": {  
                "score": 0.5023781  
            }  
        },  
        "entities": {  
            "email": [  
                "patti.owens@microsoft.com"  
            ],  
            "$instance": {  
                "email": [  
                    {  
                        "type": "builtin.email",  
                        "text": "patti.owens@microsoft.com",  
                        "startIndex": 31,  
                        "length": 25,  
                        "modelTypeId": 2,  
                        "modelType": "Prebuilt Entity Extractor"  
                    }  
                ]  
            }  
        }  
    }  
}
```

Next steps

Learn about the [number](#), [ordinal](#), and [percentage](#).

GeographyV2 prebuilt entity for a LUIS app

8/9/2019 • 2 minutes to read • [Edit Online](#)

The prebuilt geographyV2 entity detects places. Because this entity is already trained, you do not need to add example utterances containing GeographyV2 to the application intents. GeographyV2 entity is supported in English culture.

Subtypes

The geographical locations have subtypes:

SUBTYPE	PURPOSE
poi	point of interest
city	name of city
countryRegion	name of country or region
continent	name of continent
state	name of state or province

Resolution for GeographyV2 entity

API version 2.x

The following example shows the resolution of the **builtin.geographyV2** entity.

```
{
  "query": "Carol is visiting the sphinx in gizah egypt in africa before heading to texas",
  "topScoringIntent": {
    "intent": "None",
    "score": 0.8008023
  },
  "intents": [
    {
      "intent": "None",
      "score": 0.8008023
    }
  ],
  "entities": [
    {
      "entity": "the sphinx",
      "type": "builtin.geographyV2.poi",
      "startIndex": 18,
      "endIndex": 27
    },
    {
      "entity": "gizah",
      "type": "builtin.geographyV2.city",
      "startIndex": 32,
      "endIndex": 36
    },
    {
      "entity": "egypt",
      "type": "builtin.geographyV2.countryRegion",
      "startIndex": 38,
      "endIndex": 42
    },
    {
      "entity": "africa",
      "type": "builtin.geographyV2.continent",
      "startIndex": 47,
      "endIndex": 52
    },
    {
      "entity": "texas",
      "type": "builtin.geographyV2.state",
      "startIndex": 72,
      "endIndex": 76
    },
    {
      "entity": "carol",
      "type": "builtin.personName",
      "startIndex": 0,
      "endIndex": 4
    }
  ]
}
```

Preview API version 3.x

The following JSON is with the `verbose` parameter set to `false`:

```
{  
  "query": "Carol is visiting the sphinx in gizah egypt in africa before heading to texas",  
  "prediction": {  
    "normalizedQuery": "carol is visiting the sphinx in gizah egypt in africa before heading to texas",  
    "topIntent": "None",  
    "intents": {  
      "None": {  
        "score": 0.5115521  
      }  
    },  
    "entities": {  
      "geographyV2": [  
        "the sphinx",  
        "gizah",  
        "egypt",  
        "africa",  
        "texas"  
      ]  
    }  
  }  
}
```

The following JSON is with the `verbose` parameter set to `true`:

```
{  
    "query": "Carol is visiting the sphinx in gizah egypt in africa before heading to texas",  
    "prediction": {  
        "normalizedQuery": "carol is visiting the sphinx in gizah egypt in africa before heading to texas",  
        "topIntent": "None",  
        "intents": {  
            "None": {  
                "score": 0.5115521  
            }  
        },  
        "entities": {  
            "geographyV2": [  
                "the sphinx",  
                "gizah",  
                "egypt",  
                "africa",  
                "texas"  
            ],  
            "$instance": {  
                "geographyV2": [  
                    {  
                        "type": "builtin.geographyV2",  
                        "text": "the sphinx",  
                        "startIndex": 18,  
                        "length": 10,  
                        "modelTypeId": 2,  
                        "modelType": "Prebuilt Entity Extractor"  
                    },  
                    {  
                        "type": "builtin.geographyV2",  
                        "text": "gizah",  
                        "startIndex": 32,  
                        "length": 5,  
                        "modelTypeId": 2,  
                        "modelType": "Prebuilt Entity Extractor"  
                    },  
                    {  
                        "type": "builtin.geographyV2",  
                        "text": "egypt",  
                        "startIndex": 38,  
                        "length": 5,  
                        "modelTypeId": 2,  
                        "modelType": "Prebuilt Entity Extractor"  
                    },  
                    {  
                        "type": "builtin.geographyV2",  
                        "text": "africa",  
                        "startIndex": 47,  
                        "length": 6,  
                        "modelTypeId": 2,  
                        "modelType": "Prebuilt Entity Extractor"  
                    },  
                    {  
                        "type": "builtin.geographyV2",  
                        "text": "texas",  
                        "startIndex": 72,  
                        "length": 5,  
                        "modelTypeId": 2,  
                        "modelType": "Prebuilt Entity Extractor"  
                    }  
                ]  
            }  
        }  
    }  
}
```

Next steps

Learn about the [email](#), [number](#), and [ordinal](#) entities.

keyPhrase prebuilt entity for a LUIS app

8/9/2019 • 2 minutes to read • [Edit Online](#)

keyPhrase extracts a variety of key phrases from an utterance. You do not need to add example utterances containing keyPhrase to the application. keyPhrase entity is supported in [many cultures](#) as part of the [text analytics](#) features.

Resolution for prebuilt keyPhrase entity

API version 2.x

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

```
{  
  "query": "where is the educational requirements form for the development and engineering group",  
  "topScoringIntent": {  
    "intent": "GetJobInformation",  
    "score": 0.182757929  
  },  
  "entities": [  
    {  
      "entity": "development",  
      "type": "builtin.keyPhrase",  
      "startIndex": 51,  
      "endIndex": 61  
    },  
    {  
      "entity": "educational requirements",  
      "type": "builtin.keyPhrase",  
      "startIndex": 13,  
      "endIndex": 36  
    }  
  ]  
}
```

Next steps

Learn about the [percentage](#), [number](#), and [age](#) entities.

Number prebuilt entity for a LUIS app

8/9/2019 • 2 minutes to read • [Edit Online](#)

There are many ways in which numeric values are used to quantify, express, and describe pieces of information. This article covers only some of the possible examples. LUIS interprets the variations in user utterances and returns consistent numeric values. Because this entity is already trained, you do not need to add example utterances containing number to the application intents.

Types of number

Number is managed from the [Recognizers-text](#) GitHub repository

Examples of number resolution

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.

Resolution for prebuilt number

API version 2.x

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": {  
                "subtype": "integer",  
                "value": "24"  
            }  
        }  
    ]  
}
```

Preview API version 3.x

The following JSON is with the `verbose` parameter set to `false`:

```
{  
    "query": "order two dozen eggs",  
    "prediction": {  
        "normalizedQuery": "order two dozen eggs",  
        "topIntent": "None",  
        "intents": {  
            "None": {  
                "score": 0.7124502  
            }  
        },  
        "entities": {  
            "number": [  
                24  
            ]  
        }  
    }  
}
```

The following JSON is with the `verbose` parameter set to `true`:

```
{  
    "query": "order two dozen eggs",  
    "prediction": {  
        "normalizedQuery": "order two dozen eggs",  
        "topIntent": "None",  
        "intents": {  
            "None": {  
                "score": 0.7124502  
            }  
        },  
        "entities": {  
            "number": [  
                24  
            ],  
            "$instance": {  
                "number": [  
                    {  
                        "type": "builtin.number",  
                        "text": "two dozen",  
                        "startIndex": 6,  
                        "length": 9,  
                        "modelTypeId": 2,  
                        "modelType": "Prebuilt Entity Extractor"  
                    }  
                ]  
            }  
        }  
    }  
}
```

Next steps

Learn about the [currency](#), [ordinal](#), and [percentage](#).

Ordinal prebuilt entity for a LUIS app

8/9/2019 • 2 minutes to read • [Edit Online](#)

Ordinal number is a numeric representation of an object inside a set: `first`, `second`, `third`. Because this entity is already trained, you do not need to add example utterances containing ordinal to the application intents. Ordinal entity is supported in [many cultures](#).

Types of ordinal

Ordinal is managed from the [Recognizers-text](#) GitHub repository

Resolution for prebuilt ordinal entity

API version 2.x

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"
      }
    }
  ]
}
```

Preview API version 3.x

The following JSON is with the `verbose` parameter set to `false`:

```
{
  "query": "Order the second option",
  "prediction": {
    "normalizedQuery": "order the second option",
    "topIntent": "None",
    "intents": {
      "None": {
        "score": 0.7124502
      }
    },
    "entities": {
      "ordinal": [
        {
          "offset": 2,
          "relativeTo": "start"
        }
      ]
    }
  }
}
```

The following JSON is with the `verbose` parameter set to `true`:

```
{
  "query": "Order the second option",
  "prediction": {
    "normalizedQuery": "order the second option",
    "topIntent": "None",
    "intents": {
      "None": {
        "score": 0.7124502
      }
    },
    "entities": {
      "ordinal": [
        {
          "offset": 2,
          "relativeTo": "start"
        }
      ],
      "$instance": {
        "ordinal": [
          {
            "type": "builtin.ordinal",
            "text": "second",
            "startIndex": 10,
            "length": 6,
            "modelTypeId": 2,
            "modelType": "Prebuilt Entity Extractor",
            "recognitionSources": [
              "model"
            ]
          }
        ]
      }
    }
  }
}
```

Next steps

Learn about the [OrdinalV2](#), [phone number](#), and [temperature](#) entities.

Ordinal V2 prebuilt entity for a LUIS app

8/9/2019 • 2 minutes to read • [Edit Online](#)

Ordinal V2 number expands [Ordinal](#) to provide relative references such as `next`, `last`, and `previous`. These are not extracted using the ordinal prebuilt entity.

Resolution for prebuilt ordinal V2 entity

API version 2.x

The following example shows the resolution of the **builtin.ordinalV2** entity.

```
{
  "query": "what is the second to last choice in the list",
  "topScoringIntent": {
    "intent": "None",
    "score": 0.823669851
  },
  "intents": [
    {
      "intent": "None",
      "score": 0.823669851
    }
  ],
  "entities": [
    {
      "entity": "the second to last",
      "type": "builtin.ordinalV2.relative",
      "startIndex": 8,
      "endIndex": 25,
      "resolution": {
        "offset": "-1",
        "relativeTo": "end"
      }
    }
  ]
}
```

Preview API version 3.x

The following JSON is with the `verbose` parameter set to `false`:

```
{
  "query": "what is the second to last choice in the list",
  "prediction": {
    "normalizedQuery": "what is the second to last choice in the list",
    "topIntent": "None",
    "intents": {
      "None": {
        "score": 0.823669851
      }
    },
    "entities": {
      "ordinalV2": [
        {
          "offset": -1,
          "relativeTo": "end"
        }
      ]
    }
  }
}
```

The following JSON is with the `verbose` parameter set to `true`:

```
{
  "query": "what is the second to last choice in the list",
  "prediction": {
    "normalizedQuery": "what is the second to last choice in the list",
    "topIntent": "None",
    "intents": {
      "None": {
        "score": 0.823669851
      }
    },
    "entities": {
      "ordinalV2": [
        {
          "offset": -1,
          "relativeTo": "end"
        }
      ],
      "$instance": {
        "ordinalV2": [
          {
            "type": "builtin.ordinalV2.relative",
            "text": "the second to last",
            "startIndex": 8,
            "length": 18,
            "modelTypeId": 2,
            "modelType": "Prebuilt Entity Extractor",
            "recognitionSources": [
              "model"
            ]
          }
        ]
      }
    }
  }
}
```

Next steps

Learn about the [percentage](#), [phone number](#), and [temperature](#) entities.

Percentage prebuilt entity for a LUIS app

8/9/2019 • 2 minutes to read • [Edit Online](#)

Percentage numbers can appear as fractions, `3 1/2`, or as percentage, `2%`. Because this entity is already trained, you do not need to add example utterances containing percentage to the application intents. Percentage entity is supported in [many cultures](#).

Types of percentage

Percentage is managed from the [Recognizers-text](#) GitHub repository

Resolution for prebuilt percentage entity

API version 2.x

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
    }
  ],
  "entities": [
    {
      "entity": "2%",
      "type": "builtin.percentage",
      "startIndex": 36,
      "endIndex": 37,
      "resolution": {
        "value": "2%"
      }
    }
  ]
}
```

Preview API version 3.x

The following JSON is with the `verbose` parameter set to `false`:

```
{  
    "query": "set a trigger when my stock goes up 2%",  
    "prediction": {  
        "normalizedQuery": "set a trigger when my stock goes up 2%",  
        "topIntent": "None",  
        "intents": {  
            "None": {  
                "score": 0.541765451  
            }  
        },  
        "entities": {  
            "percentage": [  
                2  
            ]  
        }  
    }  
}
```

The following JSON is with the `verbose` parameter set to `true`:

```
{  
    "query": "set a trigger when my stock goes up 2%",  
    "prediction": {  
        "normalizedQuery": "set a trigger when my stock goes up 2%",  
        "topIntent": "None",  
        "intents": {  
            "None": {  
                "score": 0.541765451  
            }  
        },  
        "entities": {  
            "percentage": [  
                2  
            ],  
            "$instance": {  
                "percentage": [  
                    {  
                        "type": "builtin.percentage",  
                        "text": "2%",  
                        "startIndex": 36,  
                        "length": 2,  
                        "modelTypeId": 2,  
                        "modelType": "Prebuilt Entity Extractor"  
                    }  
                ]  
            }  
        }  
    }  
}
```

Next steps

Learn about the [ordinal](#), [number](#), and [temperature](#) entities.

PersonName prebuilt entity for a LUIS app

8/9/2019 • 2 minutes to read • [Edit Online](#)

The prebuilt personName entity detects people names. Because this entity is already trained, you do not need to add example utterances containing personName to the application intents. personName entity is supported in English and Chinese [cultures](#).

Resolution for personName entity

API version 2.x

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

```
{  
    "query": "Is Jill Jones in Cairo?",  
    "topScoringIntent": {  
        "intent": "WhereIsEmployee",  
        "score": 0.762141049  
    },  
    "entities": [  
        {  
            "entity": "Jill Jones",  
            "type": "builtin.personName",  
            "startIndex": 3,  
            "endIndex": 12  
        }  
    ]  
}
```

Preview API version 3.x

The following JSON is with the `verbose` parameter set to `false`:

```
{  
    "query": "Is Jill Jones in Cairo?",  
    "prediction": {  
        "normalizedQuery": "is jill jones in cairo?",  
        "topIntent": "None",  
        "intents": {  
            "None": {  
                "score": 0.6544678  
            }  
        },  
        "entities": {  
            "personName": [  
                "Jill Jones"  
            ]  
        }  
    }  
}
```

The following JSON is with the `verbose` parameter set to `true`:

```
{  
    "query": "Is Jill Jones in Cairo?",  
    "prediction": {  
        "normalizedQuery": "is jill jones in cairo?",  
        "topIntent": "None",  
        "intents": {  
            "None": {  
                "score": 0.6544678  
            }  
        },  
        "entities": {  
            "personName": [  
                "Jill Jones"  
            ],  
            "$instance": {  
                "personName": [  
                    {  
                        "type": "builtin.personName",  
                        "text": "Jill Jones",  
                        "startIndex": 3,  
                        "length": 10,  
                        "modelTypeId": 2,  
                        "modelType": "Prebuilt Entity Extractor"  
                    }  
                ]  
            }  
        }  
    }  
}
```

Next steps

Learn about the [email](#), [number](#), and [ordinal](#) entities.

Phone number prebuilt entity for a LUIS app

8/9/2019 • 2 minutes to read • [Edit Online](#)

The `phononenumber` entity extracts a variety of phone numbers including country code. Because this entity is already trained, you do not need to add example utterances to the application. The `phononenumber` entity is supported in `en-us` culture only.

Types of a phone number

`Phonenumber` is managed from the [Recognizers-text](#) GitHub repository

Resolution for this prebuilt entity

API version 2.x

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

```
{
  "query": "my mobile is 1 (800) 642-7676",
  "topScoringIntent": {
    "intent": "None",
    "score": 0.8448457
  },
  "intents": [
    {
      "intent": "None",
      "score": 0.8448457
    }
  ],
  "entities": [
    {
      "entity": "1 (800) 642-7676",
      "type": "builtin.phonenumber",
      "startIndex": 13,
      "endIndex": 28,
      "resolution": {
        "score": "1",
        "value": "1 (800) 642-7676"
      }
    }
  ]
}
```

Preview API version 3.x

The following JSON is with the `verbose` parameter set to `false`:

```
{  
    "query": "my mobile is 1 (800) 642-7676",  
    "prediction": {  
        "normalizedQuery": "my mobile is 1 (800) 642-7676",  
        "topIntent": "None",  
        "intents": {  
            "None": {  
                "score": 0.592748761  
            }  
        },  
        "entities": {  
            "phonenumber": [  
                "1 (800) 642-7676"  
            ]  
        }  
    }  
}
```

The following JSON is with the `verbose` parameter set to `true`:

```
{  
    "query": "my mobile is 1 (800) 642-7676",  
    "prediction": {  
        "normalizedQuery": "my mobile is 1 (800) 642-7676",  
        "topIntent": "None",  
        "intents": {  
            "None": {  
                "score": 0.592748761  
            }  
        },  
        "entities": {  
            "phonenumber": [  
                "1 (800) 642-7676"  
            ],  
            "$instance": {  
                "phonenumber": [  
                    {  
                        "type": "builtin.phonenumber",  
                        "text": "1 (800) 642-7676",  
                        "startIndex": 13,  
                        "length": 16,  
                        "score": 1,  
                        "modelTypeId": 2,  
                        "modelType": "Prebuilt Entity Extractor"  
                    }  
                ]  
            }  
        }  
    }  
}
```

Next steps

Learn about the [percentage](#), [number](#), and [temperature](#) entities.

Temperature prebuilt entity for a LUIS app

8/9/2019 • 2 minutes to read • [Edit Online](#)

Temperature extracts a variety of temperature types. Because this entity is already trained, you do not need to add example utterances containing temperature to the application. Temperature entity is supported in [many cultures](#).

Types of temperature

Temperature is managed from the [Recognizers-text](#) GitHub repository

Resolution for prebuilt temperature entity

API version 2.x

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

```
{
  "query": "set the temperature to 30 degrees",
  "topScoringIntent": {
    "intent": "None",
    "score": 0.85310787
  },
  "intents": [
    {
      "intent": "None",
      "score": 0.85310787
    }
  ],
  "entities": [
    {
      "entity": "30 degrees",
      "type": "builtin.temperature",
      "startIndex": 23,
      "endIndex": 32,
      "resolution": {
        "unit": "Degree",
        "value": "30"
      }
    }
  ]
}
```

Preview API version 3.x

The following JSON is with the `verbose` parameter set to `false`:

```
{
  "query": "set the temperature to 30 degrees",
  "prediction": {
    "normalizedQuery": "set the temperature to 30 degrees",
    "topIntent": "None",
    "intents": {
      "None": {
        "score": 0.656305432
      }
    },
    "entities": {
      "temperature": [
        {
          "number": 30,
          "unit": "Degree"
        }
      ]
    }
  }
}
```

The following JSON is with the `verbose` parameter set to `true`:

```
{
  "query": "set the temperature to 30 degrees",
  "prediction": {
    "normalizedQuery": "set the temperature to 30 degrees",
    "topIntent": "None",
    "intents": {
      "None": {
        "score": 0.656305432
      }
    },
    "entities": {
      "temperature": [
        {
          "number": 30,
          "unit": "Degree"
        }
      ],
      "$instance": {
        "temperature": [
          {
            "type": "builtin.temperature",
            "text": "30 degrees",
            "startIndex": 23,
            "length": 10,
            "modelTypeId": 2,
            "modelType": "Prebuilt Entity Extractor"
          }
        ]
      }
    }
  }
}
```

Next steps

Learn about the [percentage](#), [number](#), and [age](#) entities.

URL prebuilt entity for a LUIS app

8/9/2019 • 2 minutes to read • [Edit Online](#)

URL entity extracts URLs with domain names or IP addresses. Because this entity is already trained, you do not need to add example utterances containing URLs to the application. URL entity is supported in `en-us` culture only.

Types of URLs

Url is managed from the [Recognizers-text](#) GitHub repository

Resolution for prebuilt URL entity

API version 2.x

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

```
{
  "query": "https://www.luis.ai is a great cognitive services example of artificial intelligence",
  "topScoringIntent": {
    "intent": "None",
    "score": 0.781975448
  },
  "intents": [
    {
      "intent": "None",
      "score": 0.781975448
    }
  ],
  "entities": [
    {
      "entity": "https://www.luis.ai",
      "type": "builtin.url",
      "startIndex": 0,
      "endIndex": 17
    }
  ]
}
```

Preview API version 3.x

The following JSON is with the `verbose` parameter set to `false`:

```
{
  "query": "https://www.luis.ai is a great cognitive services example of artificial intelligence",
  "prediction": {
    "normalizedQuery": "https://www.luis.ai is a great cognitive services example of artificial
intelligence",
    "topIntent": "None",
    "intents": {
      "None": {
        "score": 0.421936184
      }
    },
    "entities": {
      "url": [
        "https://www.luis.ai"
      ]
    }
  }
}
```

The following JSON is with the `verbose` parameter set to `true`:

```
{
  "query": "https://www.luis.ai is a great cognitive services example of artificial intelligence",
  "prediction": {
    "normalizedQuery": "https://www.luis.ai is a great cognitive services example of artificial
intelligence",
    "topIntent": "None",
    "intents": {
      "None": {
        "score": 0.421936184
      }
    },
    "entities": {
      "url": [
        "https://www.luis.ai"
      ],
      "$instance": {
        "url": [
          {
            "type": "builtin.url",
            "text": "https://www.luis.ai",
            "startIndex": 0,
            "length": 19,
            "modelTypeId": 2,
            "modelType": "Prebuilt Entity Extractor"
          }
        ]
      }
    }
  }
}
```

Next steps

Learn about the [ordinal](#), [number](#), and [temperature](#) entities.

Prebuilt domain reference for your LUIS app

8/9/2019 • 3 minutes to read • [Edit Online](#)

This reference provides information about the [prebuilt domains](#), which are prebuilt collections of intents and entities that LUIS offers.

[Custom domains](#), by contrast, start with no intents and models. You can add any prebuilt domain intents and entities to a custom model.

Supported domains across cultures

The only supported culture is english.

ENTITY TYPE	DESCRIPTION
Calendar	Calendar is anything about personal meetings and appointments, <i>not</i> public events (such as world cup schedules, Seattle event calendars) or generic calendars (such as what day is it today, what does fall begin, when is Labor Day).
Communication	Requests to make calls, send texts or instant messages, find and add contacts and various other communication-related requests (generally outgoing). Contact name only queries do not belong to Communication domain.
Email	Email is a subdomain of the Communication domain. It mainly contains requests to send and receive messages through emails.
HomeAutomation	The HomeAutomation domain provides intents and entities related to controlling smart home devices. It mainly supports the control command related to lights and air conditioner but it has some generalization abilities for other electric appliances.
Notes	Note domain provides intents and entities for creating notes and writing down items for users.
Places	Places include businesses, institutions, restaurants, public spaces and addresses. The domain supports place finding and asking about the information of a public place such as location, operating hours and distance.
RestaurantReservation	Restaurant reservation domain supports intents for handling reservations for restaurants.
ToDo	ToDo domain provides types of task lists for users to add, mark and delete their todo items.

ENTITY TYPE	DESCRIPTION
Utilities	Utilities domain is a general domain among all LUIS prebuilt models which contains common intents and utterances in difference scenarios.
Weather	Weather domain focuses on checking weather condition and advisories with location and time or checking time by weather conditions.
Web	The Web domain provides the intent and entities for searching for a website.

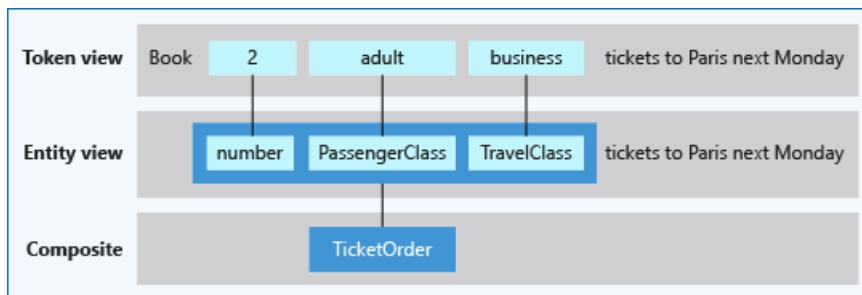
Composite entity

7/26/2019 • 2 minutes to read • [Edit Online](#)

A composite entity is made up of other entities, such as prebuilt entities, simple, regular expression, and list entities. The separate entities form a whole entity.

This entity is a good fit when the data:

- Are related to each other.
- Are related to each other in the context of the utterance.
- Use a variety of entity types.
- Need to be grouped and processed by the client application as a unit of information.
- Have a variety of user utterances that require machine-learning.



Example JSON

Consider a composite entity of prebuilt `number` and `Location::ToLocation` with the following utterance:

`book 2 tickets to paris`

Notice that `2`, the number, and `paris`, the ToLocation have words between them that are not part of any of the entities. The green underline, used in a labeled utterance in the [LUIS](#) website, indicates a composite entity.

`book 2 tickets to paris`

Composite entities are returned in a `compositeEntities` array and all entities within the composite are also returned in the `entities` array:

```

"entities": [
    {
        "entity": "2 tickets to cairo",
        "type": "ticketInfo",
        "startIndex": 0,
        "endIndex": 17,
        "score": 0.67200166
    },
    {
        "entity": "2",
        "type": "builtin.number",
        "startIndex": 0,
        "endIndex": 0,
        "resolution": {
            "subtype": "integer",
            "value": "2"
        }
    },
    {
        "entity": "cairo",
        "type": "builtin.geographyV2",
        "startIndex": 13,
        "endIndex": 17
    }
],
"compositeEntities": [
    {
        "parentType": "ticketInfo",
        "value": "2 tickets to cairo",
        "children": [
            {
                "type": "builtin.geographyV2",
                "value": "cairo"
            },
            {
                "type": "builtin.number",
                "value": "2"
            }
        ]
    }
]

```

DATA OBJECT	ENTITY NAME	VALUE
Prebuilt Entity - number	"builtin.number"	"2"
Prebuilt Entity - GeographyV2	"Location::ToLocation"	"paris"

Next steps

In this [tutorial](#), add a **composite entity** to bundle extracted data of various types into a single containing entity. By bundling the data, the client application can easily extract related data in different data types.

List entity

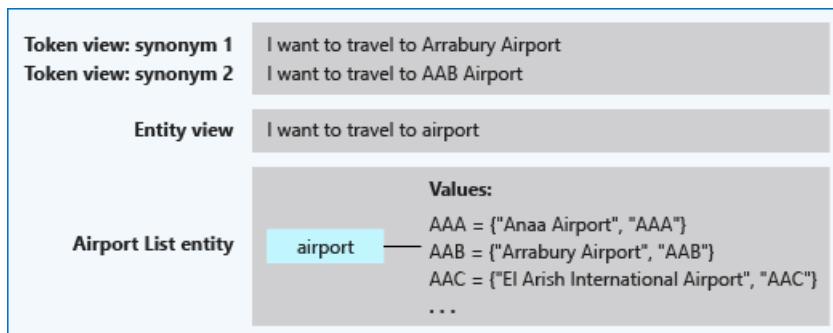
7/26/2019 • 2 minutes to read • [Edit Online](#)

List entities represent a fixed, closed set of related words along with their synonyms. LUIS does not discover additional values for list entities. Use the **Recommend** feature to see suggestions for new words based on the current list. If there is more than one list entity with the same value, each entity is returned in the endpoint query.

A list entity isn't machine-learned. It is an exact text match. LUIS marks any match to an item in any list as an entity in the response.

The entity is a good fit when the text data:

- Are a known set.
- Doesn't change often. If you need to change the list often or want the list to self-expand, a simple entity boosted with a phrase list is a better choice.
- The set doesn't exceed the maximum LUIS [boundaries](#) for this entity type.
- The text in the utterance is an exact match with a synonym or the canonical name. LUIS doesn't use the list beyond exact text matches. Fuzzy matching, case-insensitivity, stemming, plurals, and other variations are not resolved with a list entity. To manage variations, consider using a [pattern](#) with the optional text syntax.



Example JSON

Suppose the app has a list, named `cities`, allowing for variations of city names including city of airport (Sea-tac), airport code (SEA), postal zip code (98101), and phone area code (206).

LIST ITEM	ITEM SYNONYMS
Seattle	sea-tac, sea, 98101, 206, +1
Paris	cdg, roissy, ory, 75001, 1, +33
book 2 tickets to paris	

In the previous utterance, the word `paris` is mapped to the `Paris` item as part of the `cities` list entity. The list entity matches both the item's normalized name as well as the item synonyms.

```
"entities": [
  {
    "entity": "paris",
    "type": "Cities",
    "startIndex": 18,
    "endIndex": 22,
    "resolution": {
      "values": [
        "Paris"
      ]
    }
  }
]
```

Another example utterance, using a synonym for Paris:

```
book 2 tickets to roissy
```

```
"entities": [
  {
    "entity": "roissy",
    "type": "Cities",
    "startIndex": 18,
    "endIndex": 23,
    "resolution": {
      "values": [
        "Paris"
      ]
    }
  }
]
```

DATA OBJECT	ENTITY NAME	VALUE
Simple Entity	Customer	bob jones

Next steps

In this [tutorial](#), learn how to use a **list entity** to extract exact matches of text from a list of known items.

Pattern.any entity

7/26/2019 • 2 minutes to read • [Edit Online](#)

Pattern.any is a variable-length placeholder used only in a pattern's template utterance to mark where the entity begins and ends.

Pattern.any entities need to be marked in the [Pattern](#) template examples, not the intent user examples.

The entity is a good fit when:

- The ending of the entity can be confused with the remaining text of the utterance.

Usage

Given a client application that searches for books based on title, the pattern.any extracts the complete title. A template utterance using pattern.any for this book search is `Was {BookTitle} written by an American this year[?]`.

In the following table, each row has two versions of the utterance. The top utterance is how LUIS initially sees the utterance. It isn't clear where the book title begins and ends. The bottom utterance uses a Pattern.any entity to mark the beginning and end of the entity.

UTTERANCE WITH ENTITY IN BOLD

Was The Man Who Mistook His Wife for a Hat and Other Clinical Tales written by an American this year?

Was **The Man Who Mistook His Wife for a Hat and Other Clinical Tales** written by an American this year?

Was Half Asleep in Frog Pajamas written by an American this year?

Was **Half Asleep in Frog Pajamas** written by an American this year?

Was The Particular Sadness of Lemon Cake: A Novel written by an American this year?

Was **The Particular Sadness of Lemon Cake: A Novel** written by an American this year?

Was There's A Wocket In My Pocket! written by an American this year?

Was **There's A Wocket In My Pocket!** written by an American this year?

Example JSON

```
{  
  "query": "where is the form Understand your responsibilities as a member of the community and who needs to sign it after I read it?",  
  "topScoringIntent": {  
    "intent": "FindForm",  
    "score": 0.999999464  
  },  
  "intents": [  
    {  
      "intent": "FindForm",  
      "score": 0.999999464  
    },  
    {  
      "intent": "GetEmployeeBenefits",  
      "score": 4.883697E-06  
    },  
    {  
      "intent": "None",  
      "score": 1.02040713E-06  
    },  
    {  
      "intent": "GetEmployeeOrgChart",  
      "score": 9.278342E-07  
    },  
    {  
      "intent": "MoveAssetsOrPeople",  
      "score": 9.278342E-07  
    }  
  "entities": [  
    {  
      "entity": "understand your responsibilities as a member of the community",  
      "type": "FormName",  
      "startIndex": 18,  
      "endIndex": 78,  
      "role": ""  
    }  
}
```

Next steps

In this [tutorial](#), use the **Pattern.any** entity to extract data from utterances where the utterances are well-formatted and where the end of the data may be easily confused with the remaining words of the utterance.

Regular expression entity

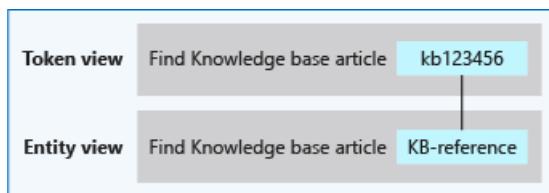
7/26/2019 • 2 minutes to read • [Edit Online](#)

A regular expression entity extracts an entity based on a regular expression pattern you provide.

A regular expression is best for raw utterance text. It ignores case and ignores cultural variant. Regular expression matching is applied after spell-check alterations at the character level, not the token level. If the regular expression is too complex, such as using many brackets, you're not able to add the expression to the model. Uses part but not all of the [.NET Regex](#) library.

The entity is a good fit when:

- The data are consistently formatted with any variation that is also consistent.
- The regular expression does not need more than 2 levels of nesting.



Usage considerations

Regular expressions may match more than you expect to match. An example of this is numeric word matching such as `one` and `two`. An example is the following regex, which matches the number `one` along with other numbers:

```
(plus )?(zero|one|two|three|four|five|six|seven|eight|nine)(\s+  
(zero|one|two|three|four|five|six|seven|eight|nine))*
```

This regex expression also matches any words that end with these numbers, such as `phone`. In order to fix issues like this, make sure the regex matches takes into account word boundaries. The regex to use word boundaries for this example is used in the following regex:

```
\b(plus )?(zero|one|two|three|four|five|six|seven|eight|nine)(\s+  
(zero|one|two|three|four|five|six|seven|eight|nine))*\b
```

Example JSON

When using `kb[0-9]{6}`, as the regular expression entity definition, the following JSON response is an example utterance with the returned regular expression entities for the query `When was kb123456 published?`:

```
{  
  "query": "when was kb123456 published?",  
  "topScoringIntent": {  
    "intent": "FindKBArticle",  
    "score": 0.933641255  
  },  
  "intents": [  
    {  
      "intent": "FindKBArticle",  
      "score": 0.933641255  
    },  
    {  
      "intent": "None",  
      "score": 0.04397359  
    }  
  ],  
  "entities": [  
    {  
      "entity": "kb123456",  
      "type": "KB number",  
      "startIndex": 9,  
      "endIndex": 16  
    }  
  ]  
}
```

Next steps

In this [tutorial](#), create an app to extract consistently-formatted data from an utterance using the **Regular Expression** entity.

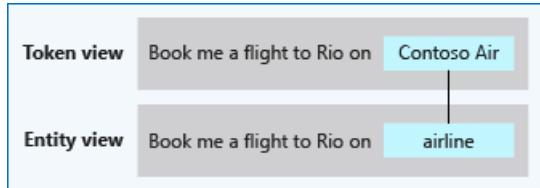
Simple entity

7/26/2019 • 2 minutes to read • [Edit Online](#)

A simple entity is a generic entity that describes a single concept and is learned from the machine-learned context. Because simple entities are generally names such as company names, product names, or other categories of names, add a [phrase list](#) when using a simple entity to boost the signal of the names used.

The entity is a good fit when:

- The data aren't consistently formatted but indicate the same thing.



Example JSON

```
Bob Jones wants 3 meatball pho
```

In the previous utterance, `Bob Jones` is labeled as a simple `Customer` entity.

The data returned from the endpoint includes the entity name, the discovered text from the utterance, the location of the discovered text, and the score:

```
"entities": [
  {
    "entity": "bob jones",
    "type": "Customer",
    "startIndex": 0,
    "endIndex": 8,
    "score": 0.473899543
  }
]
```

DATA OBJECT	ENTITY NAME	VALUE
Simple Entity	<code>Customer</code>	<code>bob jones</code>

Next steps

In this [tutorial](#), extract machine-learned data of employment job name from an utterance using the **Simple entity**. To increase the extraction accuracy, add a [phrase list](#) of terms specific to the simple entity.

Common API response codes and their meaning

8/9/2019 • 2 minutes to read • [Edit Online](#)

The [authoring](#) and [endpoint](#) APIs return HTTP response codes. While response messages include information specific to a request, the HTTP response status code is general.

Common status codes

The following table lists some of the most common HTTP response status codes for the [authoring](#) and [endpoint](#) APIs:

CODE	API	EXPLANATION
400	Authoring, Endpoint	request's parameters are incorrect meaning the required parameters are missing, malformed, or too large
400	Authoring, Endpoint	request's body is incorrect meaning the JSON is missing, malformed, or too large
401	Authoring	used endpoint subscription key, instead of authoring key
401	Authoring, Endpoint	invalid, malformed, or empty key
401	Authoring, Endpoint	key doesn't match region
401	Authoring	you are not the owner or collaborator
401	Authoring	invalid order of API calls
403	Authoring, Endpoint	total monthly key quota limit exceeded
409	Endpoint	application is still loading
410	Endpoint	application needs to be retrained and republished
414	Endpoint	query exceeds maximum character limit
429	Authoring, Endpoint	Rate limit is exceeded (requests/second)

Next steps

- REST API [authoring](#) and [endpoint](#) documentation

Language and region support for LUIS

8/9/2019 • 4 minutes to read • [Edit Online](#)

LUIS has a variety of features within the service. Not all features are at the same language parity. Make sure the features you are interested in are supported in the language culture you are targeting. A LUIS app is culture-specific and cannot be changed once it is set.

Multi-language LUIS apps

If you need a multi-language LUIS client application such as a chatbot, you have a few options. If LUIS supports all the languages, you develop a LUIS app for each language. Each LUIS app has a unique app ID, and endpoint log. If you need to provide language understanding for a language LUIS does not support, you can use [Microsoft Translator API](#) to translate the utterance into a supported language, submit the utterance to the LUIS endpoint, and receive the resulting scores.

Languages supported

LUIS understands utterances in the following languages:

LANGUAGE	LOCALE	PREBUILT DOMAIN	PREBUILT ENTITY	PHRASE LIST RECOMMENDATIONS	**TEXT ANALYTICS (SENTIMENT AND KEYWORDS)
American English	en-US	✓	✓	✓	✓
*Chinese	zh-CN	✓	✓	✓	-
Dutch	nl-NL	-	-	-	✓
French (France)	fr-FR	-	✓	✓	✓
French (Canada)	fr-CA	-	-	-	✓
German	de-DE	-	✓	✓	✓
Italian	it-IT	-	✓	✓	✓
*Japanese	ja-JP	-	✓	✓	Key phrase only
Korean	ko-KR	-	-	-	Key phrase only
Portuguese (Brazil)	pt-BR	-	✓	✓	not all sub-cultures
Spanish (Spain)	es-ES	-	✓	✓	✓
Spanish (Mexico)	es-MX	-	-	✓	✓
Turkish	tr-TR	-	-	-	Sentiment only

Language support varies for [prebuilt entities](#) and [prebuilt domains](#).

*Chinese support notes

- In the `zh-cn` culture, LUIS expects the simplified Chinese character set instead of the traditional character set.
- The names of intents, entities, features, and regular expressions may be in Chinese or Roman characters.
- See the [prebuilt domains reference](#) for information on which prebuilt domains are supported in the `zh-cn` culture.

*Japanese support notes

- Because LUIS does not provide syntactic analysis and will not understand the difference between Keigo and informal Japanese, you need to incorporate the different levels of formality as training examples for your applications.
 - ござります is not the same as です.
 - です is not the same as だ.

**Text analytics support notes

Text analytics includes keyPhrase prebuilt entity and sentiment analysis. Only Portuguese is supported for subcultures: `pt-PT` and `pt-BR`. All other cultures are supported at the primary culture level. Learn more about Text Analytics [supported languages](#).

Speech API supported languages

See Speech [Supported languages](#) for Speech dictation mode languages.

Bing Spell Check supported languages

See Bing Spell Check [Supported languages](#) for a list of supported languages and status.

Rare or foreign words in an application

In the `en-us` culture, LUIS learns to distinguish most English words, including slang. In the `zh-cn` culture, LUIS learns to distinguish most Chinese characters. If you use a rare word in `en-us` or character in `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-interchangeable, to indicate that the set of rare words forms a class that LUIS should learn to recognize, but they are not synonyms or interchangeable with each other.

Hybrid languages

Hybrid languages combine words from two cultures such as English and Chinese. These languages are not supported in LUIS because an app is based on a single culture.

Tokenization

To perform machine learning, LUIS breaks an utterance into [tokens](#) based on culture.

LANGUAGE	EVERY SPACE OR SPECIAL CHARACTER	CHARACTER LEVEL	COMPOUND WORDS	TOKENIZED ENTITY RETURNED
Chinese		✓		✓
Dutch			✓	✓
English (en-us)	✓			
French (fr-FR)	✓			

LANGUAGE	EVERY SPACE OR SPECIAL CHARACTER	CHARACTER LEVEL	COMPOUND WORDS	TOKENIZED ENTITY RETURNED
French (fr-CA)	✓			
German			✓	✓
Italian	✓			
Japanese				✓
Korean		✓		✓
Portuguese (Brazil)	✓			
Spanish (es-ES)	✓			
Spanish (es-MX)	✓			

Custom tokenizer versions

The following cultures have custom tokenizer versions:

CULTURE	VERSION	PURPOSE
German de-de	1.0.0	<p>Tokenizes words by splitting them using a machine learning-based tokenizer that tries to break down composite words into their single components.</p> <p>If a user enters <code>Ich fahre einen krankenwagen</code> as an utterance, it is turned to <code>Ich fahre einen kranken wagen</code>. Allowing the marking of <code>kranken</code> and <code>wagen</code> independently as different entities.</p>
German de-de	1.0.2	<p>Tokenizes words by splitting them on spaces.</p> <p>If a user enters <code>Ich fahre einen krankenwagen</code> as an utterance, it remains a single token. Thus <code>krankenwagen</code> is marked as a single entity.</p>

Migrating between tokenizer versions

Tokenization happens at the app level. There is no support for version-level tokenization.

[Import the file as a new app](#), instead of a version. This action means the new app has a different app ID but uses the tokenizer version specified in the file.

Language understanding glossary of common vocabulary and concepts

7/30/2019 • 5 minutes to read • [Edit Online](#)

The Language Understanding (LUIS) glossary explains terms that you might encounter as you work with the LUIS API Service.

Active version

The active LUIS version is the version that receives any changes to the model. In the [LUIS](#) portal, if you want to make changes to a version that is not the active version, you need to first set that version as active.

Authoring

Authoring is the ability to create, manage and deploy a [LUIS app](#), either using the [LUIS](#) portal or the [authoring APIs](#).

Authoring Key

Previously named "Programmatic" key. Used to author the app. Not used for production-level endpoint queries. For more information, see [Key limits](#).

Batch text JSON file

Batch testing is the ability to validate a current LUIS app's model with a consistent and known test set of user utterances. The batch test is defined in a [JSON formatted file](#).

See also:

- [Concepts](#)
- [How-to](#)
- [Tutorial]luis-tutorial-batch-testing.md)

Collaborator

A collaborator is not the [owner](#) of the app, but has the same permissions to add, edit, and delete the intents, entities, utterances.

Currently editing

Same as [active version](#)

Domain

In the LUIS context, a **domain** is an area of knowledge. Your domain is specific to your app area of knowledge. This can be a general area such as the travel agent app. A travel agent app can also be specific to just the areas of information for your company such as specific geographical locations, languages, and services.

Endpoint

The [LUIS endpoint](#) URL is where you submit LUIS queries after the [LUIS app](#) is authored and published. The endpoint URL contains the region of the published app as well as the app ID. You can find the endpoint on the [Keys and endpoints](#) page of your app, or you can get the endpoint URL from the [Get App Info API](#).

An example endpoint looks like:

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0/apps/<appID>?subscription-key=<subscriptionID>&verbose=true&timezoneOffset=0&q=<utterance>
```

QUERYSTRING PARAMETER	DESCRIPTION
region	published region
appId	LUIS app ID
subscriptionID	LUIS endpoint (subscription) key created in Azure portal
q	utterance
timezoneOffset	minutes

Entity

Entities are important words in [utterances](#) that describe information relevant to the [intent](#), and sometimes they are essential to it. An entity is essentially a datatype in LUIS.

F-measure

In [batch testing](#), a measure of the test's accuracy.

False negative (FN)

In [batch testing](#), the data points represent utterances in which your app incorrectly predicted the absence of the target intent/entity.

False positive (FP)

In [batch testing](#), the data points represent utterances in which your app incorrectly predicted the existence of the target intent/entity.

Features

In machine learning, a [feature](#) is a distinguishing trait or attribute of data that your system observes.

Intent

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 LUIS, the intent prediction is based on the entire utterance. Entities, by comparison, are pieces of an utterance.

Labeling

Labeling, or marking, is the process of associating a word or phrase in an intent's [utterance](#) with an [entity](#) (datatype).

LUIS app

A LUIS app is a trained data model for natural language processing including [intents](#), [entities](#), and labeled [utterances](#).

Owner

Each app has one owner who is the person that created the app. The owner can add [collaborators](#).

Patterns

The previous Pattern feature is replaced with [Patterns](#). Use patterns to improve prediction accuracy by providing fewer training examples.

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). An interchangeable list is treated as synonyms.

Prebuilt domain

A [prebuilt domain](#) is a LUIS app configured for a specific domain such as home automation (HomeAutomation) or restaurant reservations (RestaurantReservation). The intents, utterances, and entities are configured for this domain.

Prebuilt entity

A [prebuilt entity](#) is an entity LUIS provides for common types of information such as number, URL, and email. You choose to add a prebuilt entity to your application.

Precision

In [batch testing](#), precision (also called positive predictive value) is the fraction of relevant utterances among the retrieved utterances.

Programmatic key

Renamed to [authoring key](#).

Publish

Publishing means making a LUIS [active version](#) available on either the staging or production [endpoint](#).

Quota

Luis quota is the limitation of the [Azure subscription tier](#). The Luis quota can be limited by both requests per second (HTTP Status 429) and total requests in a month (HTTP Status 403).

Recall

In [batch testing](#), recall (also known as sensitivity), is the ability for Luis to generalize.

Semantic dictionary

A semantic dictionary is provided on the List entity page as well as the Phrase list page. The semantic dictionary provides suggestions of words based on the current scope.

Sentiment Analysis

Sentiment analysis provides positive or negative values of the utterances provided by [Text Analytics](#).

Speech priming

Speech priming allows your speech service to be primed with your LUIS model.

Spelling correction

Enable Bing spell checker to correct misspelled words in the utterances before prediction.

Starter key

Same as [programmatic key](#), renamed to Authoring key.

Subscription key

The subscription key is the **endpoint** key associated with the LUIS service [you created in Azure](#). This key is not the [authoring key](#). If you have an endpoint key, it should be used for any endpoint requests instead of the authoring key. You can see your current endpoint key inside the endpoint URL at the bottom of [Keys and endpoints page](#) in [LUIS](#) website. It is the value of **subscription-key** name/value pair.

Test

Testing a LUIS app means passing an utterance to LUIS and viewing the JSON results.

Timezone offset

The endpoint includes timezoneOffset. This is the number in minutes you want to add or remove from the datetimeV2 prebuilt entity. For example, if the utterance is "what time is it now?", the datetimeV2 returned is the current time for the client request. If your client request is coming from a bot or other application that is not the same as your bot's user, you should pass in the offset between the bot and the user.

See [Change time zone of prebuilt datetimeV2 entity](#).

Token

A token is the smallest unit that can be labeled in an entity. Tokenization is based on the application's [culture](#).

Train

Training is the process of teaching LUIS about any changes to the [active version](#) since the last training.

True negative (TN)

In [batch testing](#), the data points represent utterances in which your app correctly predicted the absence of the target intent/entity.

True positive (TP)

In [batch testing](#), the data points represent utterances in which your app correctly predicted the existence of the

target intent/entity.

Utterance

An utterance is a natural language phrase such as "book 2 tickets to Seattle next Tuesday". Example utterances are added to the intent.

Version

A LUIS [version](#) is a specific data model associated with a LUIS app ID and the published endpoint. Every LUIS app has at least one version.

Language Understanding Frequently Asked Questions (FAQ)

8/9/2019 • 12 minutes to read • [Edit Online](#)

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

What's new

[Learn more](#) about what's new in Language Understanding.

Authoring

What are the LUIS best practices?

Start with the [Authoring Cycle](#), then read the [best practices](#).

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

The best way to build your app is through an [incremental process](#).

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

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

Is it important to train the None intent?

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

How can I correct spelling mistakes in utterances?

See the [Bing Spell Check API V7](#) tutorial. LUIS enforces limits imposed by Bing Spell Check API V7.

How do I edit my LUIS app programmatically?

To edit your LUIS app programmatically, use the [Authoring API](#). See [Call LUIS authoring API](#) and [Build a LUIS app programmatically using Node.js](#) for examples of how to call the Authoring API. The Authoring API requires that you use an [authoring key](#) rather than an endpoint key. Programmatic authoring allows up to 1,000,000 calls per month and five transactions per second. For more info on the keys you use with LUIS, see [Manage keys](#).

Where is the Pattern feature that provided regular expression matching?

The previous **Pattern feature** is currently deprecated, replaced by [Patterns](#).

How do I use an entity to pull out the correct data?

See [entities](#) and [data extraction](#).

Should variations of an example utterance include punctuation?

Either add the different variations as example utterances to the intent or add the pattern of the example utterance with the [syntax to ignore](#) the punctuation.

Does LUIS currently support Cortana?

Cortana prebuilt apps were deprecated in 2017. They are no longer supported.

How do I transfer ownership of a LUIS app?

To transfer a LUIS app to a different Azure subscription, export the LUIS app and import it using a new account. Update the LUIS app ID in the client application that calls it. The new app may return slightly different LUIS scores from the original app.

A prebuilt entity is tagged in an example utterance instead of my custom entity. How do I fix this?

See [Troubleshooting prebuilt entities](#).

I tried to import an app or version file but I got an error, what happened?

Read more about [version import errors](#) and [app import errors](#).

Collaborating

How do I give collaborators access to LUIS with Azure Active Directory (Azure AD) or Role-based access control (RBAC)?

See [Azure Active Directory resources](#) and [Azure Active Directory tenant user](#) to learn how to give collaborators access.

Endpoint

My endpoint query returned unexpected results. What should I do?

Unexpected query prediction results are based on the state of the published model. To correct the model, you may need to change the model, train, and publish again.

Correcting the model starts with [active learning](#).

You can remove non-deterministic training by updating the [application version settings API](#) in order to use all training data.

Review the [best practices](#) for other tips.

Why does LUIS add spaces to the query around or in the middle of words?

Luis [tokenizes](#) the utterance based on the [culture](#). Both the original value and the tokenized value are available for [data extraction](#).

How do I create and assign a LUIS endpoint key?

Create the [endpoint key](#) in Azure for your [service level](#). Assign the key on the [Keys and endpoints](#) page. There is no corresponding API for this action. Then you must change the HTTP request to the endpoint to [use the new endpoint key](#).

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 (less than 50%) does not necessarily mean that LUIS has low confidence. Providing more training data can help increase the [score](#) of the most-likely intent.

Why don't I see my endpoint hits in my app's Dashboard?

The total endpoint hits in your app's Dashboard are updated periodically, but the metrics associated with your LUIS endpoint key in the Azure portal are updated more frequently.

If you don't see updated endpoint hits in the Dashboard, sign in to the Azure portal, and find the resource associated with your LUIS endpoint key, and open **Metrics** to select the **Total Calls** metric. If the endpoint key is used for more than one LUIS app, the metric in the Azure portal shows the aggregate number of calls from all LUIS apps that use it.

Is there a PowerShell command get to the endpoint quota?

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

You can use a PowerShell command to see the endpoint quota:

```
Get-AzCognitiveServicesAccountUsage -ResourceGroupName <your-resource-group> -Name <your-resource-name>
```

My LUIS app was working yesterday but today I'm getting 403 errors. I didn't change the app. How do I fix it?

Follow these [instructions](#) to create a LUIS endpoint key and assign it to the app. Then you must change the client application's HTTP request to the endpoint to [use the new endpoint key](#). If you created a new resource in a different region, change the HTTP client request's region too.

How do I secure my LUIS endpoint?

See [Securing the endpoint](#).

Working within LUIS limits

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

See the [boundaries](#) reference.

I want to build a LUIS app with more than the maximum number of intents. What should I do?

See [Best practices for intents](#).

I want to build an app in LUIS with more than the maximum number of entities. What should I do?

See [Best practices for entities](#)

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

For the maximum length of a [phrase list](#), see the [boundaries](#) reference.

What are the limits on example utterances?

See the [boundaries](#) reference.

Testing and training

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

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

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

See the [Batch testing](#) tutorial.

When an app is exported then reimported into a new app (with a new app ID), the LUIS prediction scores are different. Why does this happen?

See [Prediction differences between copies of same app](#).

Some utterances go to the wrong intent after I made changes to my app. The issue seems to disappear at random. How do I fix it?

See [Train with all data](#).

App publishing

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

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

The screenshot shows the Azure portal interface. On the left, there's a sidebar with various service icons: New, Dashboard, All resources, Resource groups, App Services, SQL databases, SQL data warehouses, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts, Virtual networks, and Azure Active Directory. The 'Azure Active Directory' icon is highlighted with a red box. The main area has a title 'MANAGE' and a list of options: Users and groups, Enterprise applications, Devices (Preview), App registrations, Application proxy, Licenses, Azure AD Connect, Domain names, Mobility (MDM and MAM), Company branding, User settings, and Properties. The 'Properties' option is also highlighted with a red box. On the right, a 'Save' and 'Discard' button are at the top. Below them, the 'Name' field contains 'Contoso'. Under 'Country or region', 'United States' is listed. In the 'Location' section, 'Asia, United States, Europe datacenters' is shown. The 'Notification language' is set to 'English'. A toggle switch for 'Global admin can manage Azure Subscriptions' is set to 'No'. The 'Directory ID' field contains a long GUID value 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx' and is also highlighted with a red box.

Why are there more endpoint keys assigned to my app than I assigned?

Each LUIS app has the authoring/starter key in the endpoint list as a convenience. This key allows only a few endpoint hits so you can try out LUIS.

If your app existed before LUIS was generally available (GA), LUIS endpoint keys in your subscription are assigned automatically. This was done to make GA migration easier. Any new LUIS endpoint keys in the Azure portal are *not* automatically assigned to LUIS.

Key management

How do I know what key I need, where I get it, and what I do with it?

See [Authoring and query prediction endpoint keys in LUIS](#) to learn about the differences between the [authoring key](#) and the [endpoint prediction key](#).

I got an error about being out of quota. How do I fix it?

See, [Fix HTTP status code 403 and 429](#) to learn more.

I need to handle more endpoint queries. How do I do that?

See, [Fix HTTP status code 403 and 429](#) to learn more.

App management

How do I download a log of user utterances?

By default, your LUIS app logs utterances from users. To download a log of utterances that users send to your LUIS app, go to **My Apps**, and select the app. In the contextual toolbar, select **Export Endpoint Logs**. The log is formatted as a comma-separated value (CSV) file.

How can I disable the logging of utterances?

You can turn off the logging of user utterances by setting `log=false` in the Endpoint URL that your client application uses to query LUIS. However, turning off logging disables your LUIS app's ability to suggest utterances or improve performance that's based on [active learning](#). If you set `log=false` because of data-privacy concerns, you can't download a record of those user utterances from LUIS or use those utterances to improve your app.

Logging is the only storage of utterances.

Why don't I want all my endpoint utterances logged?

If you are using your log for prediction analysis, do not capture test utterances in your log.

Data management

Can I delete data from LUIS?

- You can always delete example utterances used for training LUIS. If you delete an example utterance from your LUIS app, it is removed from the LUIS web service and is unavailable for export.
- You can delete utterances from the list of user utterances that LUIS suggests in the [Review endpoint utterances](#) page. Deleting utterances from this list prevents them from being suggested, but doesn't delete them from logs.
- If you delete an account, all apps are deleted, along with their example utterances and logs. The data is retained on the servers for 60 days before it is deleted permanently.

How does Microsoft manage data I send to LUIS?

The [Trust Center](#) explains our commitments and your options for data management and access in Azure Services.

Language and translation support

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

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

App notification

Why did I get an email saying I'm almost out of quota?

Your authoring/starter key is only allowed 1000 endpoint queries a month. Create a LUIS endpoint key (free or paid) and use that key when making endpoint queries. If you are making endpoint queries from a bot or another client application, you need to change the LUIS endpoint key there.

Bots

My LUIS bot isn't working. What do I do?

The first issue is to isolate if the issue is related to LUIS or happens outside the LUIS middleware.

Resolve issue in LUIS

Pass the same utterance to LUIS from the [LUIS endpoint](#). If you receive an error, resolve the issue in LUIS until the

error is no longer returned. Common errors include:

- Out of call volume quota. Quota will be replenished in <time>. - This issue indicates you either need to change from an authoring key to an [endpoint key](#) or you need to change [service tiers](#).

Resolve issue in Azure Bot Service

If you are using the Azure Bot Service and the issue is that the **Test in Web Chat** returns

Sorry, my bot code is having an issue, check your logs:

1. In the Azure portal, for your bot, from the **Bot management** section, select **Build**.
2. Open the online code editor.
3. In the top, blue navigation bar, select the bot name (the second item to the right).
4. In the resulting drop-down list, select **Open Kudu Console**.
5. Select **LogFiles**, then select **Application**. Review all log files. If you don't see the error in the application folder, review all log files under **LogFiles**.
6. Remember to rebuild your project if you are using a compiled language such as C#.

TIP

The console can also install packages.

Resolve issue while debugging on local machine with Bot Framework.

To learn more about local debugging of a bot, see [Debug a bot](#).

Integrating LUIS

Where is my LUIS app created during the Azure web app bot subscription process?

If you select a LUIS template, and select the **Select** button in the template pane, the left-side pane changes to include the template type, and asks in what region to create the LUIS template. The web app bot process doesn't create a LUIS subscription though.



What LUIS regions support Bot Framework speech priming?

[Speech priming](#) is only supported for LUIS apps in the central (US) instance.

API Programming Strategies

How do I programmatically get the LUIS region of a resource?

Use the LUIS sample to [find region](#) programmatically using C# or Node.js.

LUIS service

Is Language Understanding (LUIS) available on-premises or in private cloud?

Yes, you can use the LUIS [container](#) for these scenarios if you have the necessary connectivity to meter usage.

Migrating to the next version

How do I migrate to preview V3 API?

See [API v2 to v3 Migration guide for LUIS apps](#)

Build 2019 Conference announcements

The following features were released at the Build 2019 Conference:

- [Preview of V3 API migration guide](#)
- [Improved analytics dashboard](#)
- [Improved prebuilt domains](#)
- [Dynamic list entities](#)
- [External entities](#)

Videos:

- [How to use Azure Conversational AI to scale your business for the next generation](#)

Next steps

To learn more about LUIS, see the following resources:

- [Stack Overflow questions tagged with LUIS](#)
- [MSDN Language Understanding Intelligent Services \(LUIS\) Forum](#)