

Language Understanding (LUIS) documentation

Learn how Language Understanding enables your applications to understand what a person wants in their own words.

About Language Understanding (LUIS)

OVERVIEW

[What is language understanding?](#)

[LUIS portal ↗](#)

CONCEPT

[Plan your app](#)

[Iterative app design](#)

[DevOps practices](#)

WHAT'S NEW

[What's new?](#)

VIDEO

[Ignite 2019: Advanced Natural Language Understanding \(NLU\) models using LUIS ↗](#)

Create intents, entities, and features

CONCEPT

[Intents](#)

[Entities](#)

[Utterances](#)

[Phrase list features](#)

QUICKSTART

[Portal: Create a new LUIS app](#)

[Using C# SDK](#)

[Using Python SDK](#)

HOW-TO GUIDE

[Create a new LUIS app](#)

[Add intent with example utterances](#)

[Add entities](#)

[Add phrase list feature](#)

Version, train, test, and publish

CONCEPT

[Versioning](#)

[Interactive testing](#)

[Batch testing](#)

QUICKSTART

[Using SDK](#)

HOW-TO GUIDE

[Version your app](#)

[Perform interactive testing](#)

[Perform batch testing](#)

[Train your app](#)

[Publish prediction endpoint](#)

Get prediction from endpoint

QUICKSTART

Using SDK

QUICKSTART

[With C# using REST](#)

[With Go using REST](#)

[With Java using REST](#)

[With Node.js using REST](#)

[With Python using REST](#)

Reference

REFERENCE

[Developer resources](#)

[App limitations](#)

[Prebuilt entities](#)

[Custom Entities](#)

[Prebuilt domains](#)

Help and feedback

REFERENCE

[Support and help options](#)

[Troubleshooting](#)

What is Language Understanding (LUIS)?

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

ⓘ Note

As of July 2023, Azure AI services encompass all of what were previously known as Azure Cognitive Services and Azure Applied AI Services. There are no changes to pricing. The names "Cognitive Services" and "Azure Applied AI" continue to be used in Azure billing, cost analysis, price list, and price APIs. There are no breaking changes to application programming interfaces (APIs) or SDKs.

Language Understanding (LUIS) is a cloud-based conversational AI 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. LUIS provides access through its [custom portal](#), [APIs](#) and [SDK client libraries](#).

For first time users, follow these steps to [sign in to LUIS portal](#) To get started, you can try a LUIS [prebuilt domain app](#).

This documentation contains the following article types:

- **Quickstarts** are getting-started instructions to guide you through making requests to the service.
- **How-to guides** contain instructions for using the service in more specific or customized ways.
- **Concepts** provide in-depth explanations of the service functionality and features.
- **Tutorials** are longer guides that show you how to use the service as a component in broader business solutions.

What does LUIS Offer

- **Simplicity:** LUIS offloads you from the need of in-house AI expertise or any prior machine learning knowledge. With only a few clicks you can build your own conversational AI application. You can build your custom application by following one of our [quickstarts](#), or you can use one of our [prebuilt domain](#) apps.
- **Security, Privacy and Compliance:** LUIS is backed by Azure infrastructure, which offers enterprise-grade security, privacy, and compliance. Your data remains yours; you can delete your data at any time. Your data is encrypted while it's in storage. Learn more about this [here](#).
- **Integration:** easily integrate your LUIS app with other Microsoft services like [Microsoft Bot framework](#), [QnA Maker](#), and [Speech service](#).

Luis Scenarios

- [Build an enterprise-grade conversational bot](#): This reference architecture describes how to build an enterprise-grade conversational bot (chatbot) using the Azure Bot Framework.
- [Commerce Chatbot](#): Together, the Azure AI Bot Service and Language Understanding service enable developers to create conversational interfaces for various scenarios like banking, travel, and entertainment.
- [Controlling IoT devices using a Voice Assistant](#): Create seamless conversational interfaces with all of your internet-accessible devices—from your connected television or fridge to devices in a connected power plant.

Application Development life cycle



- **Plan:** Identify the scenarios that users might use your application for. Define the actions and relevant information that needs to be recognized.
- **Build:** Use your authoring resource to develop your app. Start by defining [intents](#) and [entities](#). Then, add training [utterances](#) for each intent.
- **Test and Improve:** Start testing your model with other utterances to get a sense of how the app behaves, and you can decide if any improvement is needed. You can improve your application by following these [best practices](#).
- **Publish:** Deploy your app for prediction and query the endpoint using your prediction resource. Learn more about authoring and prediction resources [here](#).

- **Connect:** Connect to other services such as [Microsoft Bot framework](#), [QnA Maker](#), and [Speech service](#).
- **Refine:** [Review endpoint utterances](#) to improve your application with real life examples

Learn more about planning and building your application [here](#).

Next steps

- [What's new](#) with the service and documentation
- [Build a LUIS app](#)
- [API reference](#)
- [Best practices](#)
- [Developer resources](#) for LUIS.
- [Plan your app with intents](#) and [entities](#).

Language and region support for LUIS

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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.

Multilingual LUIS apps

If you need a multilingual 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 the [Translator service](#) to translate the utterance into a supported language, submit the utterance to the LUIS endpoint, and receive the resulting scores.

ⓘ Note

A newer version of Language Understanding capabilities is now available as part of Azure AI Language. For more information, see [Azure AI Language Documentation](#). For language understanding capabilities that support multiple languages within the Language Service, see [Conversational Language Understanding](#).

Languages supported

LUIS understands utterances in the following languages:

Language	Locale	Prebuilt domain	Prebuilt entity	Phrase list recommendations	**Sentiment analysis and key phrase extraction
Arabic (preview - modern standard Arabic)	ar-AR	-	-	-	-
*Chinese	zh-CN	✓	✓	✓	-
Dutch	nl-NL	✓	-	-	✓
English (United States)	en-US	✓	✓	✓	✓
English (UK)	en-GB	✓	✓	✓	✓
French (Canada)	fr-CA	-	-	-	✓
French (France)	fr-FR	✓	✓	✓	✓
German	de-DE	✓	✓	✓	✓
Gujarati (preview)	gu-IN	-	-	-	-
Hindi (preview)	hi-IN	-	✓	-	-
Italian	it-IT	✓	✓	✓	✓
*Japanese	ja-JP	✓	✓	✓	Key phrase only
Korean	ko-KR	✓	-	-	Key phrase only
Marathi (preview)	mr-IN	-	-	-	-
Portuguese (Brazil)	pt-BR	✓	✓	✓	not all sub-cultures
Spanish (Mexico)	es-MX	-	✓	✓	✓
Spanish (Spain)	es-ES	✓	✓	✓	✓
Tamil (preview)	ta-IN	-	-	-	-
Telugu (preview)	te-IN	-	-	-	-
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.
 - `でござります` is not the same as `です`.
 - `です` is not the same as `だ`.

**Language service support notes

The Language service 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.

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.

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
Arabic	✓		
Chinese		✓	
Dutch	✓		✓
English (en-us)	✓		
English (en-GB)	✓		
French (fr-FR)	✓		
French (fr-CA)	✓		
German	✓		✓
Gujarati	✓		
Hindi	✓		
Italian	✓		
Japanese			✓
Korean		✓	
Marathi	✓		
Portuguese (Brazil)	✓		
Spanish (es-ES)	✓		
Spanish (es-MX)	✓		
Tamil	✓		
Telugu	✓		
Turkish	✓		

Custom tokenizer versions

The following cultures have custom tokenizer versions:

Culture	Version	Purpose
German de-de	1.0.0	Tokenizes words by splitting them using a machine learning-based tokenizer that tries to break down composite words into their single components. 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.
German de-de	1.0.2	Tokenizes words by splitting them on spaces. 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.
Dutch nl-nl	1.0.0	Tokenizes words by splitting them using a machine learning-based tokenizer that tries to break down composite words into their single components. If a user enters <code>Ik ga naar de kleuterschool</code> as an utterance, it is turned to <code>Ik ga naar de kleuter school</code> . Allowing the marking of <code>kleuter</code> and <code>school</code> independently as different entities.
Dutch nl-nl	1.0.1	Tokenizes words by splitting them on spaces. If a user enters <code>Ik ga naar de kleuterschool</code> as an utterance, it remains a single token. Thus <code>kleuterschool</code> is marked as a single entity.

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.

What's new in Language Understanding

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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

Release notes

February 2022

- LUIS containers can be used in [disconnected environments](#).

January 2022

- [Updated text recognizer](#) ↗ to v1.8.2
- Added [English \(UK\)](#) to supported languages.

December 2021

- [Updated text recognizer](#) ↗ to v1.8.1
- Jio India west publishing region

November 2021

- [Azure Role-based access control \(RBAC\) article](#)

August 2021

- Norway East [publishing region](#).
- West US 3 [publishing region](#).

April 2021

- Switzerland North [authoring region](#).

January 2021

- The V3 prediction API now supports the [Bing Spellcheck API](#).
- The regional portals (au.luis.ai and eu.luis.ai) have been consolidated into a single portal and URL. If you were using one of these portals, you will be automatically re-directed to luis.ai.

December 2020

- All LUIS users are required to [migrate to a LUIS authoring resource](#)
- New [evaluation endpoints](#) that allow you to submit batch tests using the REST API, and get accuracy results for your intents and entities. Available starting with the v3.0-preview LUIS Endpoint.

June 2020

- [Preview 3.0 Authoring SDK](#) -
 - Version 3.2.0-preview.3 - [.NET - NuGet](#) ↗
 - Version 4.0.0-preview.3 - [JS - NPM](#) ↗
- Applying DevOps practices with LUIS
 - Concepts
 - [DevOps practices for LUIS](#)
 - [Continuous Integration and Continuous Delivery workflows for LUIS DevOps](#)
 - [Testing for LUIS DevOps](#)
 - How-to
 - [Apply DevOps to LUIS app development using GitHub Actions](#)
 - [Complete code GitHub repo](#) ↗

May 2020 - //Build

- Released as **generally available (GA)**:
 - [Language Understanding container](#)
 - Preview portal promoted to [current portal](#) ↗, [previous](#) ↗ portal still available
 - New machine-learning entity creation and labeling experience
 - [Upgrade process](#) from composite and simple entities to machine-learning entities

- [Setting](#) support for normalizing word variants
- Preview Authoring API changes
 - App schema 7.x for nested machine-learning entities
 - [Migration to required feature](#)
- New resources for developers
 - [Continuous integration tools](#)
 - Workshop - learn best practices for *Natural Language Understanding (NLU)* using LUIS
- [Customer managed keys](#) - encrypt all the data you use in LUIS by using your own key
- [AI show](#) (video) - see the new features in LUIS

March 2020

- TLS 1.2 is now enforced for all HTTP requests to this service. For more information, see Azure AI services [security](#).

November 4, 2019 - Ignite

- Video - [Advanced Natural Language Understanding \(NLU\) models using LUIS and Azure AI services | BRK2188](#)
- Improved developer productivity
 - General availability of our [prediction endpoint V3](#).
 - Ability to import and export apps with [.luis](#) ([LUDown](#)) format. This paves the way for an effective CI/CD process.
- Language expansion
 - [Arabic and Hindi](#) in public preview.
- Prebuilt models
 - [Prebuilt domains](#) is now generally available (GA)
 - Japanese [prebuilt entities](#) - age, currency, number, and percentage are not supported in V3.
 - Italian [prebuilt entities](#) - age, currency, dimension, number, and percentage resolution changed from V2.
- Enhanced user experience in [preview.luis.ai portal](#) - revamped labeling experience to enable building and debugging complex models. Try the preview portal tutorials:
 - [Intents only](#)
 - [Decomposable machine-learning entity](#)

- Advance language understanding capabilities - [building sophisticated language models](#) with less effort.
- Define machine learning features at the model level and enable models to be used as signals to other models, for example using entities as features to intents and to other entities.
- New, expanded [limits](#) - higher maximum for phrase lists and total phrases, new model as a feature limits
- Extract information from text in the format of deep hierarchy structure, making conversation applications more powerful.

```

Address Entity → " Carlos Rossi, Avenida João Jorge, 112, ap. 31, Vila Industrial, Campinas – SP, 13035-680, Brazil"
Name → Carlos Rossi
  First Name → Carlos
  Last Name → Rossi
Street → Avenida João Jorge, 112, ap. 31
  Street Name → Avenida João Jorge
  Street Number → 112
  Apartment → ap. 31
Neighborhood → Vila Industrial
State → Campinas – SP
  Municipality → Campinas
  State Abbreviate → SP
Postal Code → 13035-680

```

September 3, 2019

- Azure authoring resource - [migrate now](#).
 - 500 apps per Azure resource
 - 100 versions per app
- Turkish support for prebuilt entities
- Italian support for datetimeV2

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 Ignite videos

[Advanced Natural Language Understanding \(NLU\) models using LUIS and Azure AI services | BRK2188 ↗](#)

2019 Build videos

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

Service updates

[Azure update announcements for Azure AI services ↗](#)

Language Understanding Frequently Asked Questions (FAQ)

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

What are the maximum limits for LUIS application?

LUIS has several limit areas. The first is the model limit, which controls intents, entities, and features in LUIS. The second area is quota limits based on key type. A third area of limits 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. See [LUIS limits](#) for more details.

What is the difference between Authoring and Prediction keys?

An authoring resource lets you create, manage, train, test, and publish your applications. A prediction resource lets you query your prediction endpoint beyond the 1,000 requests provided by the authoring resource. See [Authoring and query prediction endpoint keys in LUIS](#) to learn about the differences between the authoring key and the prediction runtime key.

Does LUIS support speech to text?

Yes, [Speech to text](#) is provided as an integration with LUIS.

What are Synonyms and word variations?

LUIS has little or no knowledge of the broader *NLP* aspects, such as semantic similarity, without explicit identification in examples. For example, the following tokens (words) are three different things until they are used in similar contexts in the examples provided:

- Buy
- Buying
- Bought

For semantic similarity Natural Language Understanding (NLU), you can use [Conversation Language Understanding](#)

What are the Authoring and prediction pricing?

Language Understand has separate resources, one type for authoring, and one type for querying the prediction endpoint, each has their own pricing. See [Resource usage and limits](#)

What are the supported regions?

See [region support ↗](#)

How does LUIS store data?

LUIS stores data encrypted in an Azure data store corresponding to the region specified by the key. Data used to train the model such as entities, intents, and utterances will be saved in LUIS for the lifetime of the application. If an owner or contributor deletes the app, this data will be deleted with it. If an application hasn't been used in 90 days, it will be deleted. See [Data retention](#) to know more details about data storage

Does LUIS support Customer-Managed Keys (CMK)?

The Language Understanding service automatically encrypts your data when it is persisted to the cloud. The Language Understanding service encryption protects your data and helps you meet your organizational security and compliance commitments. See the [CMK article](#) for more details about customer-managed keys.

Is it important to train the None intent?

Yes, it is good to train your **None** intent with utterances, especially as you add more labels to other intents. See [none intent](#) for details.

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](#).

Should variations of an example utterance include punctuation?

Use one of the following solutions:

- Ignore [punctuation](#)
- Add the different variations as example utterances to the intent
- Add the pattern of the example utterance with the [syntax to ignore](#) the punctuation.

Why is my app is getting different scores every time I train?

Enable or disable the use non-deterministic training option. When disabled, training will use all available data. When enabled (by default), training will use a random sample each time the app is trained, to be used as a negative for the intent. To make sure that you are getting same scores every time, make sure you train your LUIS app with all your data. See the [training article](#) for more information.

I received an HTTP 403 error status code. How do I fix it? Can I handle more requests per second?

You get 403 and 429 error status codes when you exceed the transactions per second or transactions per month for your pricing tier. Increase your pricing tier, or use Language Understanding Docker [containers](#).

When you use all of the free 1000 endpoint queries or you exceed your pricing tier's monthly transactions quota, you will 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](#), navigate to your Language Understanding resource, and select **Resource Management**, then select **Pricing tier**, and change your pricing tier. You don't need to change 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.

An HTTP 429 error code is returned when your transactions per second exceed 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.

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](#).

What do I do when I expect LUIS requests to go beyond the quota?

Luis has a monthly quota and 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.

Can I Use multiple apps with same app definition?

Yes, export the original LUIS app and 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.

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 apps, make sure to [train with all data](#).

Designate a single main app. Any utterances that are suggested for review should be added to the main 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 main app to the other apps. 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 the app.

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.

What are the supported languages?

See [supported languages](#), for multilingual NLU, consider using the new [Conversation Language Understanding \(CLU\)](#) feature of the Language Service.

Is Language Understanding (LUIS) available on-premises or in a private cloud?

Yes, you can use the LUIS [container](#) for these scenarios if you have the necessary connectivity to meter usage.

How do I integrate LUIS with Azure AI Bot Services?

Use this [tutorial](#) to integrate LUIS app with a Bot

Quickstart: Build your app in LUIS portal

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

In this quickstart, create a LUIS app using the prebuilt home automation domain for turning lights and appliances on and off. This prebuilt domain provides intents, entities, and example utterances for you. Next, try customizing your app by adding more intents and entities. When you're finished, you'll have a LUIS endpoint running in the cloud.

Sign in to LUIS portal

A new user to LUIS needs to follow this procedure:

1. Sign in to the [LUIS portal](#), select your country/region and agree to the terms of use. If you see **My Apps** instead, a LUIS resource already exists and you should skip ahead to create an app. If not, start by using an Azure resource, this allows you to link your LUIS account with a new or existing Azure Authoring resource.
2. In the **Choose an authoring** window that appears, find your Azure subscription, and LUIS authoring resource. If you don't have a resource, you can create a new one.

Choose an authoring resource

X

Switching your authoring resource will also switch to your LUIS apps. You can switch back at any time. [Learn more about LUIS resources in Azure.](#)

Azure directory ?

Microsoft

Azure Subscription *

Select azure subscription ...



LUIS authoring resource* ?

Select a LUIS authoring resource ...



[Create a new LUIS authoring resource](#)

Done

Cancel

When you create a new authoring resource, provide the following information:

- **Tenant name** - the tenant your Azure subscription is associated with.
- **Azure subscription name** - the subscription that will be billed for the resource.
- **Azure resource group name** - a custom resource group name you choose or create. Resource groups allow you to group Azure resources for access and management.
- **Azure resource name** - a custom name you choose, used as part of the URL for your authoring and prediction endpoint queries.
- **Pricing tier** - the pricing tier determines the maximum transaction per second and month.

Select subscription and authoring resource

When you select a specific subscription and authoring resource, you will see a list of apps associated with it.

The screenshot shows the Microsoft Azure Cognitive Services Language Understanding (LUIS) interface. At the top, it says "Cognitive Services | Language Understanding". Below that, "My LUIS / Conversation" is selected. The main heading is "Conversation apps". It displays an "Azure subscription: mySubscription / Authoring resource: myResource" section with a link to "Choose a different authoring resource". Below this are buttons for "+ New app", "Rename", "Export", "Import logs", "Export logs", and "Delete". A search bar is at the top right. The main area has columns for "Name", "Last modified", "Culture", and "Endpoint hits". A message says "You don't have any apps yet." There is also a magnifying glass icon with a plus sign.

Create a new app

You can create and manage your applications on [My Apps](#).

Create an application

To create an application, click **+ New app**.

In the window that appears, enter the following information:

Name	Description
Name	A name for your app. For example, "home automation".
Culture	The language that your app understands and speaks.
Description	A description for your app.
Prediction resource	The prediction resource that will receive queries.

Select **Done**.

ⓘ Note

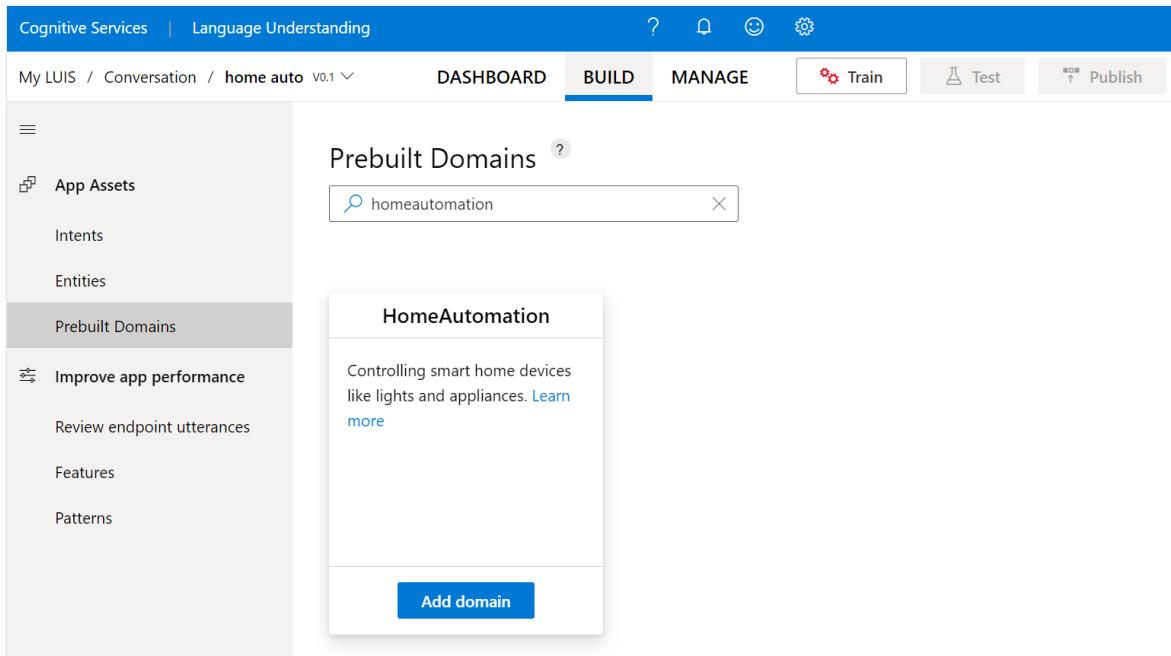
The culture cannot be changed once the application is created.

Add prebuilt domain

LUIS offers a set of prebuilt domains that can help you get started with your application. A prebuilt domain app is already populated with [intents](#), [entities](#) and [utterances](#).

1. In the left navigation, select **Prebuilt domains**.
2. Search for **HomeAutomation**.

3. Select **Add domain** on the HomeAutomation card.



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

Check out intents and entities

1. Select **Intents** in the left navigation menu to see the HomeAutomation domain intents. It has example utterances, such as `HomeAutomation.QueryState` and `HomeAutomation.SetDevice`.

! 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.

2. Select the **HomeAutomation.TurnOff** intent. The intent contains a list of example utterances that are labeled with entities.

The screenshot shows the Microsoft LUIS portal interface. The top navigation bar includes 'Cognitive Services | Language Understanding', 'DASHBOARD', 'BUILD' (which is selected), 'MANAGE', 'Train', 'Test', and 'Publish'. The left sidebar has sections for 'App Assets', 'Intents' (selected), 'Entities', 'Prebuilt Domains', 'Improve app performance', 'Review endpoint utterances', 'Features', and 'Patterns'. The main content area is titled 'HomeAutomation.TurnOff' and shows 'Machine learning features' with a '+ Add feature' button. Below is the 'Examples' section with a search bar and 'View options' dropdown. It lists two examples: 'turn the light off' and 'turn the blue lights off'. Each example shows the input text and the predicted entities underlined.

3. If you want to view the entities for the app, select **Entities**. If you select one of the entities, such as **HomeAutomation.DeviceName** you will see a list of values associated with it.

The screenshot shows the Microsoft LUIS portal interface. The top navigation bar includes 'Cognitive Services | home automation', 'DASHBOARD', 'BUILD' (selected), 'MANAGE', 'Train', 'Test', and 'Publish'. The left sidebar has sections for 'App Assets', 'Intents', 'Entities' (selected), 'Prebuilt Domains', 'Improve app performance', 'Review endpoint utterances', 'Features', and 'Patterns'. The main content area is titled 'Entities' and shows a table of entities. The table has columns for 'Name ↑' and 'Type'. The entities listed are:

Name ↑	Type
datetimeV2	Prebuilt
HomeAutomation.DeviceName	List
HomeAutomation.DeviceType	List
HomeAutomation.Location	Machine learned
HomeAutomation.NumericalChange	Machine learned
HomeAutomation.Quantifier	List
HomeAutomation.Settings	Machine learned
HomeAutomation.SettingsType	List

Train the LUIS app

After your application is populated with intents, entities, and utterances, you need to train the application so that the changes you made can be reflected.

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



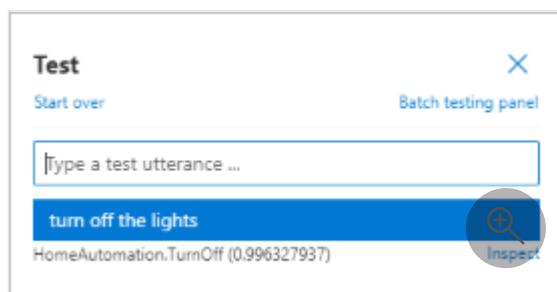
2. Training is complete when the **Train** button is disabled.

Test your app

Once you've trained your app, you can test it.

1. Select **Test** from the top-right navigation.
2. Type a test utterance into the interactive test pane, and press Enter. For example, *Turn off the lights*.

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



3. Select **Inspect** to view more information about the prediction.

The screenshot shows two side-by-side pages from the LUIS platform. On the left is the 'Test' pane, which includes a text input field ('Type a test utterance ...') containing 'turn off the lights', and a results section showing 'HomeAutomation.TurnOff (0.9975709)' with an 'Inspect' link. On the right is the 'Version: 0.1' summary page, which displays the user input 'turn off the lights', the top scoring intent 'HomeAutomation.TurnOff (0.9975709)', and various analysis sections like 'ML Entities', 'Composite Entities', 'Other entities', and 'Top matched pattern(s)'. A 'Sentiment' section is also present with a search icon.

4. Close the test pane.

Customize your application

Besides the prebuilt domains LUIS allows you to create your own custom applications or to customize on top of prebuilt ones.

Create Intents

To add more intents to your app

1. Select **Intents** in the left navigation menu.
2. Select **Create**
3. Enter the intent name, `HomeAutomation.AddDeviceAlias`, and then select **Done**.

Create Entities

To add more entities to your app

1. Select **Entities** in the left navigation menu.
2. Select **Create**
3. Enter the entity name, `HomeAutomation.DeviceAlias`, select machine learned from type and then select **Create**.

Add example utterances

Example utterances are text that a user enters in a chat bot or other client applications. They map the intention of the user's text to a LUIS intent.

On the **Intents** page for `HomeAutomation.AddDeviceAlias`, add the following example utterances under **Example Utterance**,

#	Example utterances
1	<code>Add alias to my fan to be wind machine</code>
2	<code>Alias lights to illumination</code>
3	<code>nickname living room speakers to our speakers a new fan</code>
4	<code>rename living room tv to main tv</code>

For best results make sure that example utterances vary in the following ways:

- Utterance length
- punctuation
- Word choice
- Verb tense (is, was, will be)
- Word order

Label example utterances

Labeling your utterances is needed because you added an ML entity. Labeling is used by your application to learn how to extract the ML entities you created.

To label your utterances, you have two options: Entity palette labeling and inline labeling.

Entity palette labeling

When you select the @ icon on top right, the entity palette opens in the right side of the page.

1. Select the entity you want to start labeling.

The screenshot shows the Microsoft LUIS portal interface. The top navigation bar has 'DASHBOARD', 'BUILD' (which is highlighted in blue), and 'MANAGE' tabs. On the left, there's a sidebar with 'App Assets', 'Intents' (which is selected and highlighted in grey), 'Entities', 'Prebuilt Domains', 'Improve app performance', 'Review endpoint utterances', 'Features', and 'Patterns'. The main area is titled 'Examples' and contains a search bar, a 'View options' dropdown, and a '@' icon. Below this is a section for 'Example user input' with a text box containing 'Type an example of what a user might say and hit Enter.' Underneath are two examples of user input: 'nickname living room speakers to our rename living room' and 'alias lights to illumination'. To the right is a 'Label entities' pane listing various entities like 'HomeAutomation.DeviceAlias', 'HomeAutomation.DeviceName (List)', etc., each with a small blue or grey circular icon. A large red circle highlights the '@' icon at the top right of the main interface area.

2. Highlight the text you want to label with this entity.

This screenshot shows the Microsoft LUIS portal interface after step 1. The 'BUILD' tab is still selected. In the main area, the 'alias lights to [illumination]' example is shown. When the word 'lights' is highlighted, a context menu appears with a list of entities: 'HomeAutomation.DeviceAlias' (selected and highlighted in blue), 'HomeAutomation.DeviceName (List)', 'HomeAutomation.DeviceType (List)', 'HomeAutomation.Quantifier (List)', 'HomeAutomation.SettingType (List)', 'datetimeV2 PREBUILT', 'HomeAutomation.Location', 'HomeAutomation.NumericalC...', 'HomeAutomation.Settings', and 'HomeAutomation.Unit'. A red circle highlights the 'HomeAutomation.DeviceAlias' entry in this list. The rest of the interface remains similar to the first screenshot.

Inline labeling

1. Highlight the text you want to label.

2. Select the entity you want to label the text with from the menu that pops up.

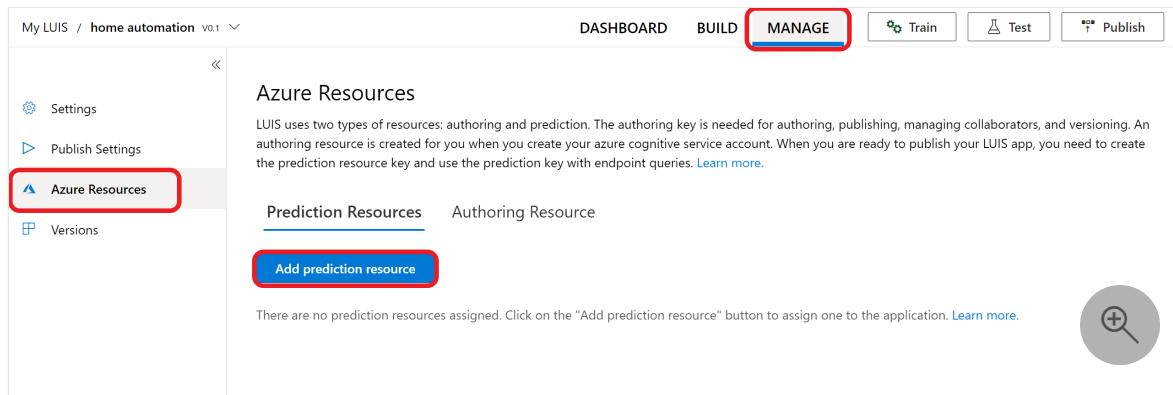
This screenshot shows the Microsoft LUIS portal interface after step 2. The 'BUILD' tab is still selected. The 'alias lights to [illumination]' example is shown with 'lights' highlighted. A context menu is open, showing the same list of entities as the previous screenshot, with 'HomeAutomation.DeviceAlias' selected. The rest of the interface is consistent with the previous screenshots.

Create Prediction resource

At this point, you have completed authoring your application. You need to create a prediction resource to publish your application in order to receive predictions in a chat bot or other client applications through the prediction endpoint.

To create a Prediction resource from the LUIS portal

1. Go to the **Manage** tab in your application.
2. In the left navigation, select **Azure Resources**.
3. select **Add Prediction resource** button.



If you already have a prediction resource, add it. Otherwise, select **Create a new prediction resource**.

Add a prediction resource

X

Your application needs a Language Understanding resource for you to access your language model. [Learn more about resources in Azure.](#)

Azure directory

Microsoft

Note: To switch to another directory, use the top right user avatar.

Azure subscription *

Prediction resource* ?

Loading ...

[Create a new prediction resource](#)

Done

Cancel 

Publish the app to get the endpoint URL

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

1. Select Publish in the top-right navigation.



2. Select the Production slot, then select Done.

Choose your publishing slot and settings

X

Staging slot

Production Slot

Last Published: 08/04/2021

Sentiment Analysis: Off

Speech Priming: Off

[Change settings](#)

[Done](#)

[Cancel](#)

3. Select **Access your endpoint URLs** in the notification to go to the **Azure Resources** page. You will only be able to see the URLs if you have a prediction resource associated with the app. You can also find the **Azure Resources** page by clicking **Manage**.

 Publish app 'prebuilt-app' completed 11:46 AM 

[Access your endpointUrls](#)

Query the V3 API prediction endpoint

1. In the LUIS portal, in the **Manage** section (top-right menu), on the **Azure Resources** page (left menu), on the **Prediction Resources** tab, copy the **Example Query** at the bottom of the page. The URL has your app ID, key, and slot name. The V3 prediction endpoint URL has the form of: `https://YOUR-RESOURCE-NAME.api.cognitive.microsoft.com/luis/prediction/v3.0/apps/APP-ID/slots/SLOT-NAME/predict?subscription-key=YOUR-PREDICTION-KEY&<optional-name-value-pairs>&query=YOUR_QUERY_HERE`

The screenshot shows the Microsoft Azure Cognitive Services Language Understanding (LUIS) interface. The top navigation bar includes 'Cognitive Services' and 'Language Understanding'. The main menu has options like 'DASHBOARD', 'BUILD', 'MANAGE', 'Train', 'Test', and 'Publish'. On the left, a sidebar titled 'Azure Resources' lists 'Settings', 'Publish Settings', 'Azure Resources' (which is selected), and 'Versions'. The main content area is titled 'MyResource' and shows its configuration details: Resource group: myResourceGroup, Location: eastus, Primary Key: <your-key-will-appear-here>, Secondary Key: <your-key-will-appear-here>, Endpoint URL: <your-endpoint-will-appear-here>, and Pricing Tier: F0 (Free). Below these details is an 'Example Query:' input field containing a single dot ('.') and a 'Change query parameters' link. A magnifying glass icon is located in the bottom right corner of this section.

Paste the URL into a new browser tab. If you don't see the URL, you don't have a prediction resource and will need to create one.

2. In the browser address bar, for the query string, make sure the following values are in the URL. If they are not in the query string, add them:

- `verbose=true`
- `show-all-intents=true`

3. In the browser address bar, go to the end of the URL and enter *turn off the living room light* for the query string, then press Enter.

```
{  
  "query": "turn off the living room light",  
  "prediction": {  
    "topIntent": "HomeAutomation.TurnOff",  
    "intents": {  
      "HomeAutomation.TurnOff": {  
        "score": 0.969448864  
      },  
      "HomeAutomation.QueryState": {  
        "score": 0.0122336326  
      },  
      "HomeAutomation.TurnUp": {  
        "score": 0.006547436  
      },  
    }  
  }  
}
```

```
"HomeAutomation.TurnDown": {
    "score": 0.0050634006
},
"HomeAutomation.SetDevice": {
    "score": 0.004951761
},
"HomeAutomation.TurnOn": {
    "score": 0.00312553928
},
"None": {
    "score": 0.000552945654
}
},
"entities": {
    "HomeAutomation.Location": [
        "living room"
    ],
    "HomeAutomation.DeviceName": [
        [
            [
                "living room light"
            ]
        ],
        "HomeAutomation.DeviceType": [
            [
                [
                    "light"
                ]
            ],
            "$instance": {
                "HomeAutomation.Location": [
                    {
                        "type": "HomeAutomation.Location",
                        "text": "living room",
                        "startIndex": 13,
                        "length": 11,
                        "score": 0.902181149,
                        "modelTypeId": 1,
                        "modelType": "Entity Extractor",
                        "recognitionSources": [
                            "model"
                        ]
                    }
                ],
                "HomeAutomation.DeviceName": [
                    {
                        "type": "HomeAutomation.DeviceName",
                        "text": "living room light",
                        "startIndex": 13,
                        "length": 17,
                        "modelTypeId": 5,
                        "modelType": "List Entity Extractor",
                        "recognitionSources": [
                            "model"
                        ]
                    }
                ],
                [
                    ...
                ]
            }
        ],
        ...
    ]
}
```

```
"HomeAutomation.DeviceType": [
  {
    "type": "HomeAutomation.DeviceType",
    "text": "light",
    "startIndex": 25,
    "length": 5,
    "modelTypeId": 5,
    "modelType": "List Entity Extractor",
    "recognitionSources": [
      "model"
    ]
  }
]
```

Learn more about the [V3 prediction endpoint](#).

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

- [Iterative app design](#)
- [Best practices](#)

Quickstart: Language Understanding (LUIS) client libraries and REST API

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Create and query an Azure LUIS artificial intelligence (AI) app with the LUIS SDK client libraries with this quickstart using C#, Python, or JavaScript. You can also use cURL to send requests using the REST API.

Language Understanding (LUIS) enables you to apply natural language processing (NLP) to a user's conversational, natural language text to predict overall meaning, and pull out relevant, detailed information.

- The **authoring** client library and REST API allows you to create, edit, train, and publish your LUIS app.
- The **prediction runtime** client library and REST API allows you to query the published app.

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

- Create an app
- Add an intent, a machine-learned entity, with an example utterance
- Train and publish app
- Query prediction runtime

[Reference documentation](#) | [Authoring](#) ↗ and [Prediction](#) ↗ Library source code | [Authoring](#) ↗ and [Prediction](#) ↗ NuGet | [C# Sample](#) ↗

Prerequisites

- The current version of [.NET Core](#) ↗ and [.NET Core CLI](#).
- Azure subscription - [Create one for free](#) ↗
- Once you have your Azure subscription, [create a Language Understanding authoring resource](#) ↗ in the Azure portal to get your key and endpoint. Wait for it

to deploy and click the **Go to resource** button.

- You will need the key and endpoint from the resource you [create](#) to connect your application to Language Understanding authoring. You'll paste your key and endpoint into the code below later in the quickstart. You can use the free pricing tier ([F0](#)) to try the service.

Setting up

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`.

```
.NET CLI
```

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

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

```
.NET CLI
```

```
cd language-understanding-quickstart
```

3. You can build the application with:

```
.NET CLI
```

```
dotnet build
```

The build output should contain no warnings or errors.

```
Console
```

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

Install the NuGet libraries

Within the application directory, install the Language Understanding (LUIS) client libraries for .NET with the following commands:

.NET CLI

```
dotnet add package Microsoft.Azure.CognitiveServices.Language.LUIS.Authoring  
--version 3.2.0-preview.3  
dotnet add package Microsoft.Azure.CognitiveServices.Language.LUIS.Runtime -  
-version 3.1.0-preview.1
```

Authoring Object model

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

Code examples for authoring

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

- Apps - [create](#), [delete](#), [publish](#)
- Example utterances - [add](#), [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

Prediction Object model

The Language Understanding (LUIS) prediction runtime client is a [LuisRuntimeClient](#) object that authenticates to Azure, which contains your resource key.

Code examples for prediction runtime

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

- Prediction by [staging or production slot](#)
- Prediction by [version](#)

Code examples

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

- [Create an app](#)
- [Add intent](#)
- [Add entities](#)
- [Add example utterances](#)
- [Train the app](#)
- [Publish the app](#)
- [Prediction by slot](#)

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:

C#

```
using System;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.Azure.CognitiveServices.Language.LUIS.Authoring;
using Microsoft.Azure.CognitiveServices.Language.LUIS.Authoring.Models;
using Microsoft.Azure.CognitiveServices.Language.LUIS.Runtime;
using Microsoft.Azure.CognitiveServices.Language.LUIS.Runtime.Models;
using Newtonsoft.Json;
```

Add boilerplate code

1. Change the signature of the `Main` method to allow async calls:

C#

```
public static async Task Main()
```

2. Add the rest of the code in the `Main` method of the `Program` class unless otherwise specified.

Create variables for the app

Create two sets of variables: the first set you change, the second set leave as they appear in the code sample.

ⓘ Important

Remember to remove the key from your code when you're done, and never post it publicly. For production, use a secure way of storing and accessing your credentials like [Azure Key Vault](#). See the Azure AI services [security](#) article for more information.

1. Create variables to hold your authoring key and resource names.

C#

```
var key = "PASTE_YOUR_LUIS_AUTHORING_SUBSCRIPTION_KEY_HERE";  
  
var authoringEndpoint = "PASTE_YOUR_LUIS_AUTHORING_ENDPOINT_HERE";  
var predictionEndpoint = "PASTE_YOUR_LUIS_PREDICTION_ENDPOINT_HERE";
```

2. Create variables to hold your endpoints, app name, version, and intent name.

C#

```
var appName = "Contoso Pizza Company";  
var versionId = "0.1";  
var intentName = "OrderPizzaIntent";
```

Authenticate the client

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

C#

```
var credentials = new  
Microsoft.Azure.CognitiveServices.Language.LUIS.Authoring.ApiKeyServiceClien  
tCredentials(key);  
var client = new LUISAuthoringClient(credentials) { Endpoint =  
authoringEndpoint };
```

Create a LUIS app

A LUIS app contains the natural language processing (NLP) model including intents, entities, and example utterances.

Create a [ApplicationCreateObject](#). The name and language culture are required properties. Call the [Apps.AddAsync](#) method. The response is the app ID.

```
C#
```

```
var newApp = new ApplicationCreateObject
{
    Culture = "en-us",
    Name = appName,
    InitialVersionId = versionId
};

var appId = await client.Apps.AddAsync(newApp);
```

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.

The `intentName` value is hard-coded to `OrderPizzaIntent` as part of the variables in the [Create variables for the app](#) section.

```
C#
```

```
await client.Model.AddIntentAsync(appId, versionId, new ModelCreateObject()
{
    Name = intentName
});
```

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](#).

It is important to know that entities are not marked with an intent. They can and usually do apply to many intents. Only the 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.

The entity creation code creates a machine-learning entity with subentities and features applied to the `Quantity` subentities.

Pizza order [Edit](#)

Machine learned

Schema and features Examples Roles

An ML entity can be composed of smaller subentities, each of which can have its own ML features. [Learn more about ML features](#).

Name	Machine learning features
Pizza order	+ Add feature
Pizza	+ Add feature
Quantity	@ number * × + Add feature
Type	+ Add feature
Size	+ Add feature
Toppings	+ Add feature
Type	+ Add feature
Quantity	@ number × QuantityPhraselist × + Add feature

C#

```
// Add Prebuilt entity
await client.Model.AddPrebuiltAsync(appId, versionId, new[] { "number" });

// Define ml entity with children and grandchildren
var mlEntityDefinition = new EntityModelCreateObject
{
    Name = "Pizza order",
    Subentities = new[]
    {
        new EntityModelCreateObject
        {
            Name = "Pizza",
            Subentities = new[]
            {
                new EntityModelCreateObject
                {
                    Name = "Quantity",
                    MachineLearningFeatures = new[]
                    {
                        "number"
                    }
                }
            }
        }
    }
}
```

```

    Children = new[]
    {
        new ChildEntityModelCreateObject
        {
            Name = "Pizza",
            Children = new[]
            {
                new ChildEntityModelCreateObject { Name = "Quantity" },
                new ChildEntityModelCreateObject { Name = "Type" },
                new ChildEntityModelCreateObject { Name = "Size" }
            }
        },
        new ChildEntityModelCreateObject
        {
            Name = "Toppings",
            Children = new[]
            {
                new ChildEntityModelCreateObject { Name = "Type" },
                new ChildEntityModelCreateObject { Name = "Quantity" }
            }
        }
    }
};

// Add ML entity
var mlEntityId = await client.Model.AddEntityAsync(appId, versionId,
mlEntityDefinition); ;

// Add phraselist feature
var phraselistId = await client.Features.AddPhraseListAsync(appId,
versionId, new PhraselistCreateObject
{
    EnabledForAllModels = false,
    IsExchangeable = true,
    Name = "QuantityPhraselist",
    Phrases = "few,more,extra"
});

// Get entity and subentities
var model = await client.Model.GetEntityAsync(appId, versionId, mlEntityId);
var toppingQuantityId = GetModelGrandchild(model, "Toppings", "Quantity");
var pizzaQuantityId = GetModelGrandchild(model, "Pizza", "Quantity");

// add model as feature to subentity model
await client.Features.AddEntityFeatureAsync(appId, versionId,
pizzaQuantityId, new ModelFeatureInformation { ModelName = "number",
IsRequired = true });
await client.Features.AddEntityFeatureAsync(appId, versionId,
toppingQuantityId, new ModelFeatureInformation { ModelName = "number"});

// add phrase list as feature to subentity model
await client.Features.AddEntityFeatureAsync(appId, versionId,
toppingQuantityId, new ModelFeatureInformation { FeatureName =
"QuantityPhraselist" });

```

Use the following method to the class to find the Quantity subentity's ID, in order to assign the features to that subentity.

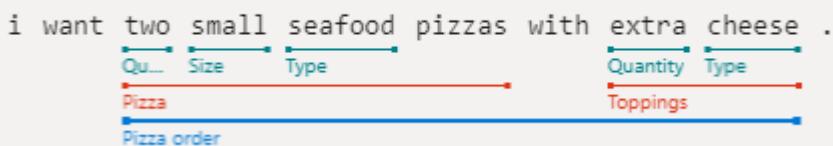
```
C#
```

```
static Guid GetModelGrandchild(NDepthEntityExtractor model, string
childName, string grandchildName)
{
    return model.Children.
        Single(c => c.Name == childName).
        Children.
        Single(c => c.Name == grandchildName).Id;
}
```

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.



i want two small seafood pizzas with extra cheese .
Qu... Size Type
Pizza
Quantity Type
Toppings
Pizza order

Call [Examples.AddAsync](#) with the app ID, version ID, and the example.

```
C#
```

```
// Define labeled example
var labeledExampleUtteranceWithMLEntity = new ExampleLabelObject
{
    Text = "I want two small seafood pizzas with extra cheese.",
    IntentName = intentName,
    EntityLabels = new[]
    {
        new EntityLabelObject
        {
            StartCharIndex = 7,
            EndCharIndex = 48,
            EntityName = "Pizza order",
        }
    }
}
```

```

        Children = new[]
        {
            new EntityLabelObject
            {
                StartCharIndex = 7,
                EndCharIndex = 30,
                EntityName = "Pizza",
                Children = new[]
                {
                    new EntityLabelObject { StartCharIndex = 7,
EndCharIndex = 9, EntityName = "Quantity" },
                    new EntityLabelObject { StartCharIndex = 11,
EndCharIndex = 15, EntityName = "Size" },
                    new EntityLabelObject { StartCharIndex = 17,
EndCharIndex = 23, EntityName = "Type" }
                }
            },
            new EntityLabelObject
            {
                StartCharIndex = 37,
                EndCharIndex = 48,
                EntityName = "Toppings",
                Children = new[]
                {
                    new EntityLabelObject { StartCharIndex = 37,
EndCharIndex = 41, EntityName = "Quantity" },
                    new EntityLabelObject { StartCharIndex = 43,
EndCharIndex = 48, EntityName = "Type" }
                }
            }
        },
    });
};

// Add an example for the entity.
// Enable nested children to allow using multiple models with the same name.
// The quantity subentity and the phraselist could have the same exact name
if this is set to True
await client.Examples.AddAsync(appId, versionId,
labeledExampleUtteranceWithMLEntity, enableNestedChildren: true);

```

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.

C#

```
await client.Train.TrainVersionAsync(appId, versionId);
while (true)
{
    var status = await client.Train.GetStatusAsync(appId, versionId);
    if (status.All(m => m.Details.Status == "Success"))
    {
        // Assumes that we never fail, and that eventually we'll always
        // succeed.
        break;
    }
}
```

Publish app to production slot

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.

C#

```
await client.Apps.PublishAsync(appId, new ApplicationPublishObject {
    VersionId = versionId, IsStaging=false});
```

Authenticate the prediction runtime client

Use an [ApiKeyServiceClientCredentials](#) object with your key, and use it with your endpoint to create an [LuisRuntimeClient](#) object.

⊗ Caution

This quickstart uses the authoring key as part of the runtime credentials. The authoring key is allowed to query the runtime with a few queries. For staging and production-level code, replace the authoring key with a prediction runtime key.

C#

```
var runtimeClient = new LuisRuntimeClient(credentials) { Endpoint =
predictionEndpoint };
```

Get prediction from runtime

Add the following code to create the request to the prediction runtime.

The user utterance is part of the [PredictionRequest](#) object.

The `GetSlotPredictionAsync` method needs several parameters such as the app ID, the slot name, the prediction request object to fulfill the request. The other options such as verbose, show all intents, and log are optional.

C#

```
// Production == slot name
var request = new PredictionRequest { Query = "I want two small pepperoni
pizzas with more salsa" };
var prediction = await
runtimeClient.Prediction.GetSlotPredictionAsync(appId, "Production",
request);
Console.WriteLine(JsonConvert.SerializeObject(prediction, Formatting.Indented));
```

The prediction response is a JSON object including the intent and any entities found.

JSON

```
{
  "query": "I want two small pepperoni pizzas with more salsa",
  "prediction": {
    "topIntent": "OrderPizzaIntent",
    "intents": {
      "OrderPizzaIntent": {
        "score": 0.753606856
      },
      "None": {
        "score": 0.119097039
      }
    },
    "entities": {
      "Pizza order": [
        {
          "Pizza": [
            {
              "Quantity": [
                2
              ]
            }
          ]
        }
      ]
    }
  }
}
```

```
        ],
        "Type": [
            "pepperoni"
        ],
        "Size": [
            "small"
        ],
        "$instance": {
            "Quantity": [
                {
                    "type": "builtin.number",
                    "text": "two",
                    "startIndex": 7,
                    "length": 3,
                    "score": 0.968156934,
                    "modelTypeId": 1,
                    "modelType": "Entity Extractor",
                    "recognitionSources": [
                        "model"
                    ]
                }
            ],
            "Type": [
                {
                    "type": "Type",
                    "text": "pepperoni",
                    "startIndex": 17,
                    "length": 9,
                    "score": 0.9345611,
                    "modelTypeId": 1,
                    "modelType": "Entity Extractor",
                    "recognitionSources": [
                        "model"
                    ]
                }
            ],
            "Size": [
                {
                    "type": "Size",
                    "text": "small",
                    "startIndex": 11,
                    "length": 5,
                    "score": 0.9592077,
                    "modelTypeId": 1,
                    "modelType": "Entity Extractor",
                    "recognitionSources": [
                        "model"
                    ]
                }
            ]
        }
    ],
    "Toppings": [
    {
```

```
        "Type": [
            "salsa"
        ],
        "Quantity": [
            "more"
        ],
        "$instance": {
            "Type": [
                {
                    "type": "Type",
                    "text": "salsa",
                    "startIndex": 44,
                    "length": 5,
                    "score": 0.7292897,
                    "modelTypeId": 1,
                    "modelType": "Entity Extractor",
                    "recognitionSources": [
                        "model"
                    ]
                }
            ],
            "Quantity": [
                {
                    "type": "Quantity",
                    "text": "more",
                    "startIndex": 39,
                    "length": 4,
                    "score": 0.9320932,
                    "modelTypeId": 1,
                    "modelType": "Entity Extractor",
                    "recognitionSources": [
                        "model"
                    ]
                }
            ]
        }
    ],
    "$instance": {
        "Pizza": [
            {
                "type": "Pizza",
                "text": "two small pepperoni pizzas",
                "startIndex": 7,
                "length": 26,
                "score": 0.812199831,
                "modelTypeId": 1,
                "modelType": "Entity Extractor",
                "recognitionSources": [
                    "model"
                ]
            }
        ],
        "Toppings": [
            {
                "type": "Toppings",
                "text": "pepperoni",
                "startIndex": 13,
                "length": 10,
                "score": 0.812199831,
                "modelTypeId": 1,
                "modelType": "Entity Extractor",
                "recognitionSources": [
                    "model"
                ]
            }
        ],
        "Dish": [
            {
                "type": "Dish",
                "text": "two small pepperoni pizzas",
                "startIndex": 7,
                "length": 26,
                "score": 0.812199831,
                "modelTypeId": 1,
                "modelType": "Entity Extractor",
                "recognitionSources": [
                    "model"
                ]
            }
        ],
        "Food": [
            {
                "type": "Food",
                "text": "two small pepperoni pizzas",
                "startIndex": 7,
                "length": 26,
                "score": 0.812199831,
                "modelTypeId": 1,
                "modelType": "Entity Extractor",
                "recognitionSources": [
                    "model"
                ]
            }
        ]
    }
]
```

```
        "type": "Toppings",
        "text": "more salsa",
        "startIndex": 39,
        "length": 10,
        "score": 0.7250252,
        "modelTypeId": 1,
        "modelType": "Entity Extractor",
        "recognitionSources": [
            "model"
        ]
    }
]
],
{
    "$instance": {
        "Pizza order": [
            {
                "type": "Pizza order",
                "text": "two small pepperoni pizzas with more
salsa",
                "startIndex": 7,
                "length": 42,
                "score": 0.769223332,
                "modelTypeId": 1,
                "modelType": "Entity Extractor",
                "recognitionSources": [
                    "model"
                ]
            }
        ]
    }
}
}
```

Run the application

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

.NET CLI

```
dotnet run
```

Clean up resources

You can delete the app from the [LUIS portal](#) and delete the Azure resources from the [Azure portal](#).

If you're using the REST API, delete the `ExampleUtterances.JSON` file from the file system when you're done with the quickstart.

Troubleshooting

- Authenticating to the client library - authentication errors usually indicate that the wrong key & endpoint were used. This quickstart uses the authoring key and endpoint for the prediction runtime as a convenience, but will only work if you haven't already used the monthly quota. If you can't use the authoring key and endpoint, you need to use the prediction runtime key and endpoint when accessing the prediction runtime SDK client library.
- Creating entities - if you get an error creating the nested machine-learning entity used in this tutorial, make sure you copied the code and didn't alter the code to create a different entity.
- Creating example utterances - if you get an error creating the labeled example utterance used in this tutorial, make sure you copied the code and didn't alter the code to create a different labeled example.
- Training - if you get a training error, this usually indicates an empty app (no intents with example utterances), or an app with intents or entities that are malformed.
- Miscellaneous errors - because the code calls into the client libraries with text and JSON objects, make sure you haven't changed the code.

Other errors - if you get an error not covered in the preceding list, let us know by giving feedback at the bottom on this page. Include the programming language and version of the client libraries you installed.

Next steps

- [Iterative app development for LUIS](#)

Use cases for Language Understanding

Article • 07/18/2022

What is a Transparency Note?

An AI system includes not only the technology, but also the people who will use it, the people who will be affected by it, and the environment in which it is deployed. Creating a system that is fit for its intended purpose requires an understanding of how the technology works, its capabilities and limitations, and how to achieve the best performance.

Microsoft provides *Transparency Notes* to help you understand how our AI technology works. This includes the choices system owners can make that influence system performance and behavior, and the importance of thinking about the whole system, including the technology, the people, and the environment. You can use Transparency Notes when developing or deploying your own system, or share them with the people who will use or be affected by your system.

Transparency Notes are part of a broader effort at Microsoft to put our AI principles into practice. To find out more, see [Microsoft AI Principles](#).

Introduction to Language Understanding

[Language Understanding \(LUIS\)](#) is a cloud-based conversational AI service that applies custom machine-learning intelligence to a user's natural language text. It predicts the overall meaning of an input text and pulls out specific information from it. LUIS needs to be integrated with a client application, which can be any conversational application that communicates with a user in natural language to complete a task. The most common client application is a chat bot.

Client applications use the output returned by LUIS to make a decision or perform an action about how to fulfill the user's requests. For example, a user types "I want to order a Pizza" in a chat bot which is sent to LUIS for interpretation. LUIS analyzes the input text and returns its interpretation in a form that can be processed by the chat bot, linking the input text with a preconfigured action to order the Pizza for the user. LUIS only provides the intelligence to understand the input text for the client application and doesn't perform any actions. LUIS currently [supports multiple languages](#).

The basics of Language Understanding

LUIS is the natural language understanding component in an end-to-end conversational application that predicts the overall intention of an incoming text and extracts important information from it. The service enables its users to [customize domain-specific LUIS applications](#) where they can iteratively train, test, and publish these applications. Users of the service need to provide and label training data relevant to the domain of the client application being built. The quality of the provided training data is important and needs to be similar to the expected user input. LUIS provides a [web portal](#) to simplify the customization experience for domain experts and nontechnical users.

For more information, see:

- [Create a new LUIS app in the LUIS portal](#)
- [Add intents to determine user intention of utterances](#)
- [Add entities to extract data](#)

Language Understanding terminology

The following terms are commonly used within LUIS:

Term	Definition
Authoring	Authoring is the phase prior to publishing a LUIS application. Everything from creating the application, creating intent and entity models, adding example utterances, labeling utterances, training, testing and publishing the application is part of the authoring phase. All this can be done through the LUIS custom portal or through REST APIs.
Utterances	Utterances represent the input text from an end-user that LUIS needs to interpret. Developers add example utterances as training data to each intent and label them with the intents and entities they want extracted to train LUIS. It's important to capture different example utterances with varied terminologies when they are added for each intent. For example, "I wanted to order a large cheese pizza" would be an example utterance in an application that orders pizza. Learn more
Intents	Intents are tasks or actions that the user wants to perform. Intent models understand and classify the overall meaning and intention of an input text. Developers define a set of intents to trigger an action users want to take in the client application. For example, intents for an application that orders pizza could be, "Make Order", "Edit Order", or "Cancel Order". Learn more
Entities	Entities represent a word or phrase in an utterance that is relevant to the user's intent. Entity models extract different types of entities, as defined by developers. In an example utterance "I wanted to order a large cheese pizza", developers could define a "size" entity to extract "large" and a "type" entity to extract "cheese" from the utterance. Developers define entities to extract key data from user utterances in LUIS apps. When authoring the LUIS app, developers label a word or multiple

Term	Definition
	<p>words they want extracted within the example utterances with a specific entity.</p> <p>Learn more</p>
Prebuilt entities	<p>Prebuilt entities are pre-trained models that can recognize common types of information, such as names, geographies, dates, times, numbers, and measurements. When a prebuilt entity is included in a LUIS application, its predictions are included in the published application. Prebuilt entities cannot be modified.</p>
Prebuilt domains	<p>Prebuilt domains are pre-trained, ready-to-use LUIS applications that contains prebuilt intent and entity models, along with labeled example utterances. LUIS offers multiple prebuilt application domains that can be added such as home automation or restaurant reservation. Prebuilt domains are fully customizable. Developers can add, edit, and delete intents, entities, or example utterances and retrain the application. Learn more</p>

Example use cases

LUIS can be used in multiple scenarios across a variety of industries. Some examples are:

- **Use in an end-to-end conversational bot.** Use LUIS to build and train a custom natural language understanding model based on a specific domain and the expected users' utterances. Integrate it with any end-to-end conversational bot so that it can process and analyze incoming text in real time to identify the intention of the text and extract important information from it. Have the bot perform the desired action based on the intention and extracted information.
- **Human assistant bots.** One example for a human assistant bot is to help staff improve customer engagements by triaging customer queries and assigning them to the appropriate support engineer. Another example would be a human resources bot in an enterprise that allows employees to communicate in natural language and receive guidance based on the query.
- **Command and control application.** Integrating a client application with a speech to text component, users can speak a command in natural language for LUIS to process, identify intent, and extract information from the text for the client application to perform an action. This use case has many applications, such as to stop, play, forward, and rewind a song or turn lights on or off.

Considerations when choosing a use case

- **Don't use LUIS for decisions that may have serious adverse impacts**, such as use cases that include identifying whether to accept or reject an insurance claim based

on a user's description of an incident. Additionally, it is advisable to include human review of decisions that have the potential for serious impacts on individuals.

- **Avoid creating custom entities that extract unnecessary or sensitive information.** It's your responsibility to ensure that the entities being created only extract the necessary information for your end-to-end scenario. Avoid extracting sensitive user information if it is not required for your scenario. For example, if your scenario requires extracting your user's city and country, create entities that extract only the city and country from a user's address, and don't create one that would extract their full address. To ensure that your model is inclusive, make sure you represent a variety of cities, countries, and address formats within your training data (example utterances).
- **Avoid storing users' personal data.** LUIS doesn't retain end user data by default, but LUIS customers have the option to retain the data if they opt to, which should then be relayed appropriately to the end users. If you do opt to retain user data, avoid storing private user data (like name, date of birth, or other identifying information) or sending requests to LUIS that contain private user data. You can deploy a private user data detector or a profanity filter on the original text prior to sending it to the LUIS model.

Next steps

- [Introduction to Language Understanding](#)
- [Characteristics and limitations for using Language Understanding](#)
- [Data, privacy and security](#)
- [Microsoft AI principles](#)
- [Building responsible bots](#)

Characteristics and limitations for using Language Understanding

Article • 07/18/2022

Performance with Language Understanding (LUIS) will vary based on the scenario, input data, and enabled features. The following sections are designed to help the reader understand key concepts about performance as they apply to LUIS.

Understand and measure performance

The performance of LUIS is measured by examining the predicted intents and entities for a user's utterances and how well the system recognizes the custom natural language processing (NLP) concepts (at a threshold value in comparison with a human judge). Comparing the human judge's performance with the custom recognized intents and entities allows the developer to classify the events into two kinds of correct (or "true") events and two kinds of incorrect (or "false") events. The following table shows the options by using a "Make call" intent as an example.

Term	Correct/Incorrect	Definition	Example
True positive	Correct	The system returns the same results that would be expected from a human judge.	For the utterance "Make a phone call to Sarah," the system correctly predicts the intent as "Make call."
True negative	Correct	The system doesn't return a result, which aligns with what would be expected from a human judge.	For the utterance "Turn off the lights," the system doesn't predict this utterance as a "Make call" intent and predicts it as a "None" intent.
False positive	Incorrect	The system returns an incorrect result where a human judge wouldn't.	For the utterance "Turn off the lights," the system incorrectly predicts the intent as "Make call."
False negative	Incorrect	The system doesn't return a result when a human judge would return a correct result.	For the utterance "I need to call Sarah to tell her that I am late," the system incorrectly predicts this utterance as a "None" intent.

Errors that happen with LUIS are mostly dependent on the utterances provided as training data for each intent during the authoring phase. Any application of LUIS will experience both false negative and false positive errors. Developers need to consider

how each type of error will affect the overall system and carefully think through scenarios where true events won't be recognized and incorrect events will be recognized. Customers should assess the downstream effects that will be in the implementation and understand the consequences of both types of errors on their client application. Developers should create ways to identify, report, and respond to each type of error.

Given that a LUIS prediction response triggers the client application to perform a specific action, if the prediction is incorrect, the client application will perform an action different than the intended one. For example, suppose the user utterance was "I need to make a phone call to Sarah" and the predicted intent from LUIS was a "None" intent. In this case, the client application won't go through the "making a call" logic and will be unable to perform the expected action properly. Handling this error from the client application side is the short-term solution. The system would expect another clear input from the user similar to the utterances given to the system during authoring like "Make call."

The more reliable solution is to review the user's traffic in LUIS and add the incorrectly predicted utterances to the correct intent to be retrained and republished. Developers may want to give the end-users a way to report errors like these, so that they can improve the LUIS model over time. Developers need to periodically plan to review the performance of the deployed system to ensure errors are being handled appropriately.

System limitations and best practices for enhancing system performance

- **Understand service limitations:** There are some service limitations such as number of intents per application and number of example utterances per intent. [Learn more on system limitations](#).
- **Plan application schema:** Think about end-user utterances and the main actions that the client application will perform based on these utterances. Developers need to plan the LUIS application schema accordingly with the intents, example utterances, and entities to be extracted. This step is essential, and we advise all LUIS users to do it. [Learn more](#).
- **Quality of training data:** The quality of example utterances provided during training impacts the end results. Carefully choose realistic example utterances for training the model. Capture a variety of different example utterances expected to be sent from the end-users with varied terminologies and contextual differences. Beware of using sensitive information in example utterances that are added during authoring. For example, don't add a real credit card number in an example

utterance. Example utterances are saved in LUIS storage accounts to train models. [Learn more](#).

- **Build models by using real world data:** Avoid using automatically generated data because then the model learns a fixed grammar, which diminishes the model's ability to generalize across different ways of speaking. A good practice is to deploy a simple model and start collecting data that's used in training the final model. This practice helps give an understanding of how users are shifting and how they might express different things over time.
- **Regularly review endpoint utterances:** Enable [active learning feature](#) to review endpoint utterances. This feature gives the customer insight on how the model is performing. Based on performance, developers can modify example utterances, retrain, and republish the application to improve the model's prediction accuracy. If the customer decides to enable this feature, it is advised to inform the end-users that their utterances are being saved during processing. [Learn more](#).
- **Provide secondary paths:** LUIS may not perform well when used with speech to text in situations with a lot of background noise or for people with speaking difficulties and speech impairments. Ensure there's always a secondary path for users to enact commands when LUIS does not perform as expected.

General guidelines to understand and improve performance

The following guidelines helps understand and improve performance in LUIS.

Understand confidence scores

Depending on system configuration, LUIS might return a confidence score for the detected intent and entity models as part of the system's prediction response. In the latest [V3 prediction endpoint](#), if the user enables the "show-all-intent" flag in the prediction endpoint, LUIS returns all intents that were created along with the confidence score for each intent. If the flag is disabled, only the top scoring intent returns along with its score.

The top two intents can have a very small score difference between them. LUIS doesn't indicate this proximity. It only returns the scores for each intent or the score of the top intent depending on whether the flag is enabled or not. For entities, the "verbose" flag must be enabled to return the scores of the detected entities along with an array of detailed information per entity. If the flag is disabled, only the detected entities return.

These scores serve as an indicator of how confident the service is with the system's response. A higher value indicates that the service is more confident that the result is accurate.

A confidence score is between zero (0) and one (1). A highly confident LUIS score is 0.99. A low confidence score is 0.01. The returned score is directly affected by the data provided during the authoring of the application.

If the user's input is similar to the trained utterances, a higher score intent is returned and more accurate entities are extracted. If user input is different than the utterances provided during the authoring phase, scores are lower.

To obtain an accurate prediction and a high confidence score, provide multiple variations of an utterance with the same meaning.

Set confidence score thresholds

Developers may choose to make decisions in the system based on the intent confidence score the system returns. The confidence score threshold the system uses can be adjusted to meet the required needs. If it's more important to identify all potential intents of the text, use a lower threshold. This means that you might get more false positives but fewer false negatives.

If it's more important for the system to recognize only true intents of the text being analyzed, use a higher threshold. When using a higher threshold, you might get fewer false positives but more false negatives.

Different intent scenarios call for different approaches. For example, if there is an intent for greetings or starting the bot, a developer may accept a lower threshold (0.40, for example) as they would want to accept multiple variations of the incoming text. But if there is an intent for a specific action like making a phone call, the developer would probably want to set a higher threshold (0.95, for example) to ensure accuracy of the predicted text.

Returning and reviewing all intent scores is a good way to verify that the correct intent is identified, and also that the next identified intent's score is significantly and consistently lower.

If multiple intents have close prediction scores, the developer might want to review and add to more example utterances for each intent. Define a threshold for the delta score between the top two intents. If the difference is lower than the threshold defined, make programmatic choices about handling such cases.

It's critical to test the system with any thresholds being added by using real data that the system will process in production to determine the effects of various threshold values. At any point, you can always continue to add example utterances with a wider variety of contextual differences and republish the application.

Different training sessions can result in different predictions

When training an intent model in a LUIS application, the system chooses a set of random example utterances from all the other intents created as negative examples. With each training iteration, the random set of negative examples changes. This turnover affects the scores returned from the system on the same example utterance being predicted.

This difference occurs because there's non-deterministic training. In other words, there's an element of randomness. This randomness can also cause an overlap of an utterance to more than one intent. This means that the top intent for the same utterance can change based on training. For example, the top score could become the second top score, and the second top score could become the first top score.

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 two intents. The two intents should have about the same number of example utterances. A rule of thumb for separation to prevent inversion because of training is a 15 percent difference in scores.

Customers can choose to turn off the non-deterministic training. The system will then train an intent with all the data provided to the other intents as negative examples instead of training by using a small random percentage of the other intent's data.

Improve accuracy by using patterns when several utterances are similar

A [Pattern](#) in LUIS is a template utterance assigned to an intent. It contains syntax to identify entities and ignorable text.

Patterns are another feature that can be used to increase the confidence scores for intent and entity prediction without providing many more utterances. Use patterns when an intent score is low or if the correct intent's score is close to the top scoring intent.

Be aware that setting an intent for a template utterance in a pattern isn't a guarantee of the intent prediction, but it's a strong signal. Understand how [to can use patterns in](#)

LUIS application.

Review incorrect predictions to improve performance

LUIS customers can improve the application's prediction accuracy if the active learning feature is enabled. If the "log" flag is enabled in the prediction API, LUIS logs the user queries and selects those that need validation to be added to a review list. The utterances are added to the review list when the top firing intent has a low confidence score or the top two intents' confidence scores are too close.

The app owner or contributor reviews and validates the selected utterances, which include the correct intent and any entities within the intent. If an utterance wasn't correctly predicted, the user has the option to add this utterance as an example utterance of the correct intent and extract the correct entities from it. If scores of the same top two intents are close for a number of endpoint utterances, add more example utterances to each intent with a wider variety of contextual differences, train and publish the app again.

Reviewing suggested utterances should be part of the regular maintenance for the LUIS application to ensure that it keeps returning correct predictions of a high score. Learn about [fixing unsure predictions by reviewing endpoint utterances](#).

Quality of the incoming text to the system will affect results

LUIS only processes text. The fidelity and formatting of the incoming text will affect the performance of the system. Make sure to consider the following:

- Speech transcription quality might affect the quality of the results. If the source data is voice, make sure to use the highest quality combination of automatic and human transcription to ensure the best performance. Consider using custom speech models to obtain better quality results.
- Lack of standard punctuation or casing might affect the quality of the results. If using a speech system, like Azure AI Speech to Text, be sure to select the option to include punctuation.
- Frequent misspellings in the training data might affect the confidence of the response. Consider using a spell-checking service to correct misspelled words. You can easily integrate with [Bing Spell Check](#).
- Spell-checking could be introduced, but it might not always be the best solution to include. In all cases, the use of actual data, including spelling mistakes, would be best.

- The training data for LUIS models is provided by the application owner. Data that most closely resembles the training data yields the best performance.

Performance varies across features and languages

LUIS has a variety of features within the service. Not all features are at the same language parity. For example, language support varies for prebuilt entities and prebuilt domains. You might find that performance for a particular feature isn't consistent with another feature. Also, you might find that for a particular feature that performance isn't consistent across various languages. Understand [language support across all LUIS features](#).

Next steps

- [Introduction to Language Understanding](#)
- [Language Understanding transparency note](#)
- [Microsoft AI principles](#)
- [Building responsible bots](#)

Guidance for integration and responsible use with Language Understanding

Article • 07/18/2022

Microsoft works to help customers responsibly develop and deploy solutions by using Language Understanding (LUIS). Our principled approach upholds personal agency and dignity by considering the AI system's:

- Fairness, reliability, and safety.
- Privacy and security.
- Inclusiveness.
- Transparency.
- Human accountability.

These considerations reflect our commitment to developing responsible AI.

General guidelines for integration and responsible use principles

When you get ready to integrate and responsibly use AI-powered products or features, the following activities help to set you up for success:

- **Understand what it can do.** Fully assess the potential of any AI system to understand its capabilities and limitations. Understand how it will perform in your particular scenario and context by thoroughly testing it with real-life conditions and data.
- **Respect an individual's right to privacy.** Only collect data and information from individuals for lawful and justifiable purposes. Only use data and information that you have consent to use for this purpose.
- **Obtain legal review.** Obtain appropriate legal advice to review your solution, particularly if you will use it in sensitive or high-risk applications. Understand what restrictions you might need to work within and your responsibility to resolve any issues that might come up in the future.
- **Have a human in the loop.** Keep a human in the loop, and include human oversight as a consistent pattern area to explore. Ensure constant human oversight of the AI-powered product or feature. Maintain the role of humans in decision

making. Make sure you can have real-time human intervention in the solution to prevent harm and manage where the AI model doesn't perform as required.

- **Maintain security.** Ensure your solution is secure and has adequate controls to preserve the integrity of your content and prevent unauthorized access.
- **Build trust with affected stakeholders.** Communicate the expected benefits and potential risks to affected stakeholders. Help people understand why the data is needed and how the use of the data will lead to their benefit. Describe data handling in an understandable way.
- **Create a customer feedback loop.** Provide a feedback channel that allows users and individuals to report issues with the service after it's deployed. After you've deployed an AI-powered product or feature, it requires ongoing monitoring and improvement. Be ready to implement any feedback and suggestions for improvement. Establish channels to collect questions and concerns from affected stakeholders. People who might be directly or indirectly impacted by the system include employees, visitors, and the general public. For example, consider using:
 - Feedback features built into app experiences.
 - An easy-to-remember email address for feedback.
 - Anonymous feedback boxes placed in semi-private spaces.
 - Knowledgeable representatives in the lobby.
- **Always plan to have the user confirm an action before being processed.** Plan to have your user confirm an action before being processed by your client application to avoid incorrect responses that might come from the LUIS application. For example, suppose your LUIS application is integrated in an HR bot and LUIS returns that your user requests five days off. Have the user confirm that this request is the correct action to be taken before processing it.
- **Always plan to have a correction path for the user.** After a certain action is taken by the client application, show a confirmation message to the user of the action that was processed. Plan that the response of the LUIS application might not be accurate and that your user might end up in an error state. In that case, always have a fallback plan or a correction path that the user can use to exit from that state. For example, if your LUIS application is integrated in an HR bot and the user requested five days off, show a confirmation message that days were deducted from the balance. Include the start and end dates with a link to delete the request or to update the dates.

Next steps

- [Introduction to Language Understanding](#)
- [Language Understanding transparency note](#)
- [Microsoft AI principles](#) ↗

- Building responsible bots ↗

Data and privacy for Language Understanding

Article • 07/18/2022

What data does Language Understanding process?

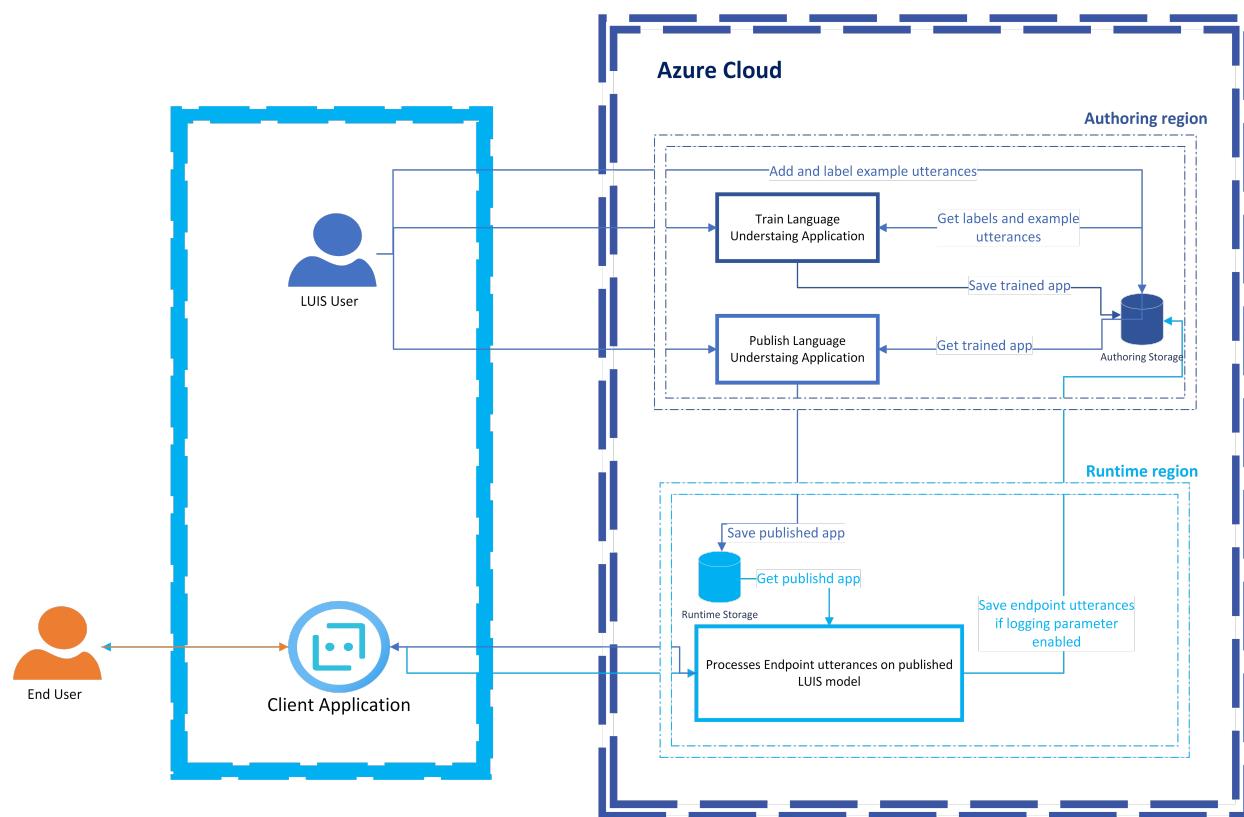
Language Understanding (LUIS) processes the following types of data:

- **Example utterances:** Example utterances are the labeled text utterances provided by the customer to train the custom LUIS model. Providing the example utterances and labeling them are prerequisites before training the model. LUIS users can provide example utterances interactively through the [LUIS web portal](#) or programmatically by using LUIS APIs.
- **Natural language understanding (NLU) LUIS model:** Based on the user's request to train the model, LUIS processes the provided example utterances and labels to output a trained NLU model. The trained model is overridden every time the LUIS user requests another training after having done updates to the example utterances and labels. A good practice is to [clone the current active model](#) to a different version of the app before making changes to the model. After the user is satisfied with a model, the user requests to publish the NLU model to receive user text utterances at runtime.
- **Endpoint utterances:** Endpoint utterances are the user's text utterances sent from a customer's client application, for example, a chat bot, and received by the trained NLU LUIS model. Endpoint utterances are processed in real time in the Microsoft Azure cloud, unless they're running on a [LUIS hosted container](#). Output of the processed data contains predictions for the overall meaning and extractions of the detailed information of the incoming text based on the customer's customized model. Output is then returned to the client application to perform an action to fulfill the user's request.
- **Logged endpoint utterances:** Endpoint utterances can be logged to help LUIS customers [improve the performance of the LUIS application](#). LUIS users have the option to [turn off or turn on query logging](#) as needed.

LUIS doesn't collect customer data to improve its machine-learned models or for product improvement purposes. We use aggregate telemetry, such as which APIs are used and the number of calls from each subscription and resource, for service monitoring purposes.

How does LUIS process data?

The following diagram illustrates how your data is processed.



How is data retained, and what customer controls are available?

Luis is a data processor for GDPR purposes. In compliance with GDPR policies, Luis users have full control to view, export, or delete any user content either through the [Luis web portal](#) or programmatically by using Luis APIs. For more information, see [Export and delete your customer data](#).

Customer controls include:

- Example utterances and labels provided by the Luis user as prerequisite to train the Luis model are saved until the customer deletes this data.
- Trained Luis models persist in Azure Storage accounts until the customer deletes the Luis application. The model is overridden each time the user retrains the model.
- Published Luis models persist in Azure Storage accounts until the customer deletes the Luis application. The model is overridden each time the user republishes the model.
- Endpoint utterances are saved only if the Luis customer (model owner) enables logging them for [review to enhance the performance of the Luis application](#).

When utterances are logged, they're saved for only 30 days. Otherwise, text utterances are only processed in memory in real time to output the LUIS response and never saved.

Optional: Security for customers' data

Azure services are implemented while maintaining appropriate technical and organizational measures to protect customer data in the cloud.

To maintain security and data governance requirements, run Docker containers for LUIS in your own environment. For more information on installing and running LUIS containers, see [Install and run Docker containers for LUIS](#). For more information on the security model of Azure AI services, see [Azure AI services container security](#). You're responsible for securing and maintaining the equipment and infrastructure required to operate LUIS's containers located on your premises.

To learn more about Microsoft's privacy and security commitments, see the [Microsoft Trust Center](#).

Next steps

- [Introduction to Language Understanding](#)
- [Language Understanding transparency note](#)
- [Microsoft AI principles](#)
- [Building responsible bots](#)

Sign in to the LUIS portal and create an app

Article • 07/18/2023

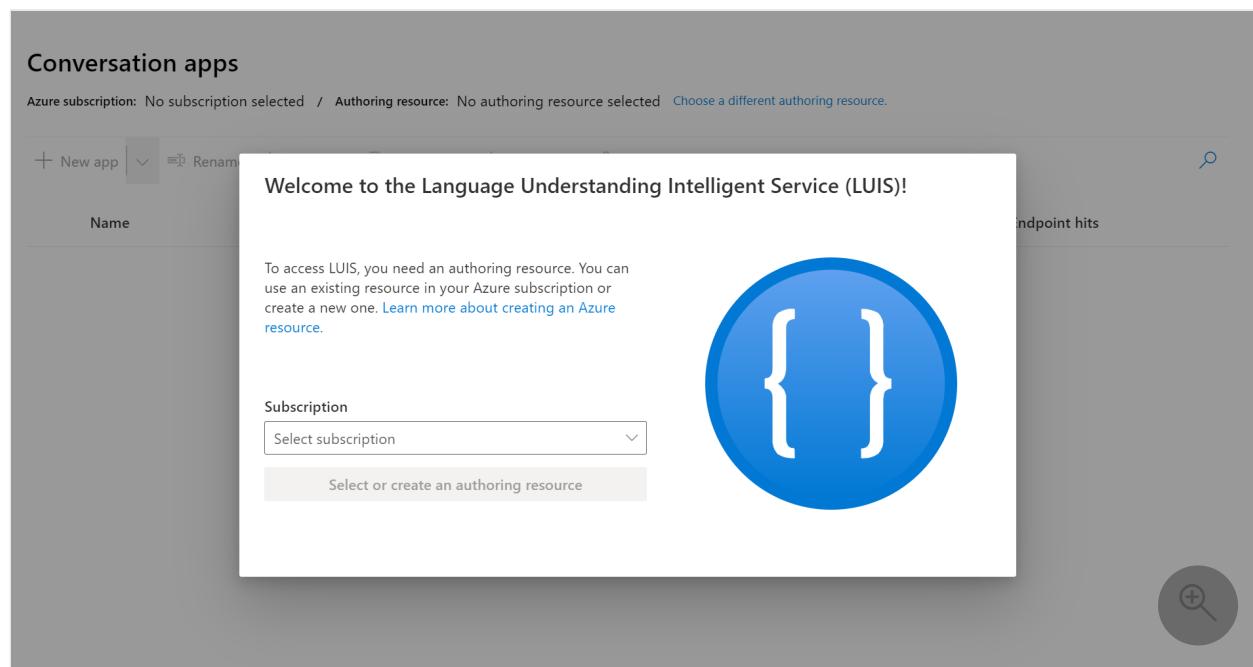
ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

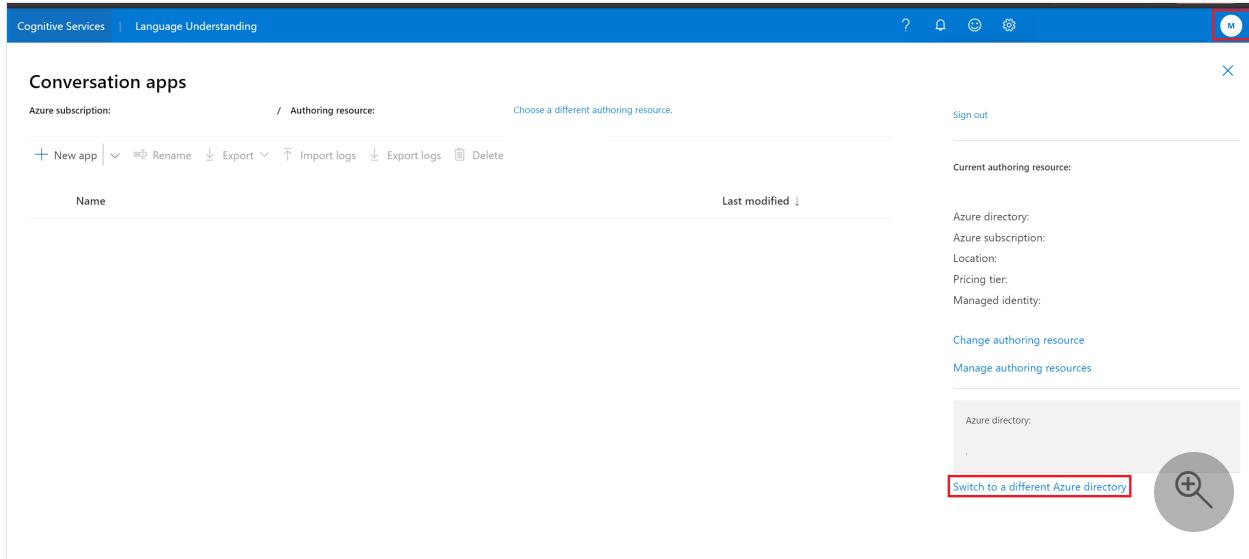
Use this article to get started with the LUIS portal, and create an authoring resource. After completing the steps in this article, you will be able to create and publish LUIS apps.

Access the portal

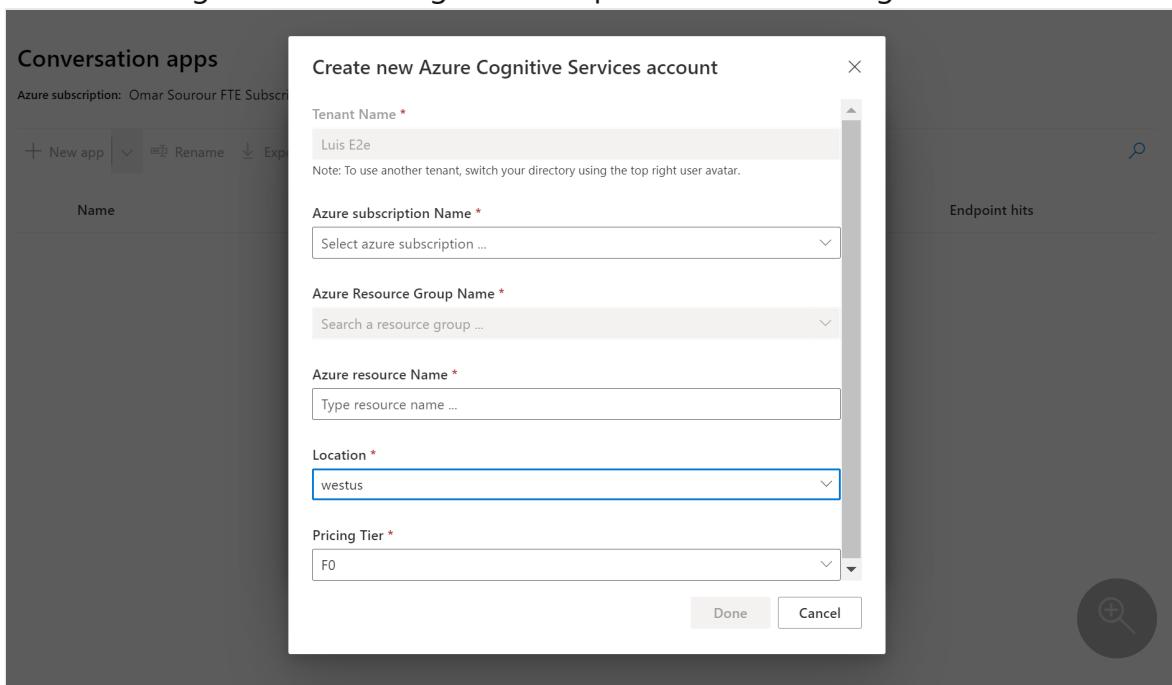
1. To get started with LUIS, go to the [LUIS Portal](#). If you do not already have a subscription, you will be prompted to go create a [free account](#) and return back to the portal.
2. Refresh the page to update it with your newly created subscription
3. Select your subscription from the dropdown list



4. If your subscription lives under another tenant, you will not be able to switch tenants from the existing window. You can switch tenants by closing this window and selecting the avatar containing your initials in the top-right section of the screen. Select **Choose a different authoring resource** from the top to reopen the window.



5. If you have an existing LUIS authoring resource associated with your subscription, choose it from the dropdown list. You can view all applications that are created under this authoring resource.
6. If not, then select **Create a new authoring resource** at the bottom of this modal.
7. When creating a new authoring resource, provide the following information:



- **Tenant Name** - the tenant your Azure subscription is associated with. You will not be able to switch tenants from the existing window. You can switch tenants by closing this window and selecting the avatar at the top-right corner of the screen,

containing your initials. Select **Choose a different authoring resource** from the top to reopen the window.

- **Azure Resource group name** - a custom resource group name you choose in your subscription. Resource groups allow you to group Azure resources for access and management. If you currently do not have a resource group in your subscription, you will not be allowed to create one in the LUIS portal. Go to [Azure portal](#) to create one then go to LUIS to continue the sign-in process.
- **Azure Resource name** - a custom name you choose, used as part of the URL for your authoring transactions. Your resource name can only include alphanumeric characters, `-`, and can't start or end with `-`. If any other symbols are included in the name, creating a resource will fail.
- **Location** - Choose to author your applications in one of the [three authoring locations](#) that are currently supported by LUIS including: West US, West Europe and East Australia
- **Pricing tier** - By default, F0 authoring pricing tier is selected as it is the recommended. Create a [customer managed key](#) from the Azure portal if you are looking for an extra layer of security.

8. Now you have successfully signed in to LUIS. You can now start creating applications.

Note

- When creating a new resource, make sure that the resource name only includes alphanumeric characters, `'-'`, and can't start or end with `'-'`. Otherwise, it will fail.

Create a new LUIS app

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 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 .lu or .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:

- [Add application](#) - start with an empty app and create intents, utterances, and entities.
- [Add prebuilt application](#) - start with a prebuilt domain, including intents, utterances, and entities.

Create new app in LUIS using portal

1. On My Apps page, select your **Subscription**, and **Authoring resource** then select **+ New App**.

Language Understanding

My LUIS / Conversation

Conversation apps

Azure subscription: MySubscription / Authoring resource: my-resource [Choose a different authoring resource](#).

+ New app | Rename | Export | Import logs | Export logs | Delete |

Name	Last modified	Culture
Conversation	1 hour ago	English

You don't have any apps yet.

2. In the dialog box, enter the name of your application, such as Pizza Tutorial.

Create new app

Name *
Pizza Tutorial

Culture *
English

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

Description
Type app description ...

Prediction resource
Choose prediction resource ...

Note: Only resources allowed for use in this region are displayed.

Done **Cancel**

3. Choose your application culture, and then select **Done**. The description and prediction resource are optional at this point. You can set them at any time in the **Manage** section of the portal.

! Note

The culture cannot be changed once the application is created.

After the app is created, the LUIS portal shows the **Intents** list with the **None** intent already created for you. You now have an empty app.

The screenshot shows the Microsoft LUIS portal interface. At the top, there's a navigation bar with 'My LUIS / Pizza Tutorial v0.1' on the left and 'DASHBOARD', 'BUILD' (which is highlighted in blue), 'MANAGE', 'Train', 'Test', and 'Publish' buttons on the right. On the left side, there's a sidebar with links like 'App Assets', 'Intents' (which is selected and highlighted in grey), 'Entities', 'Prebuilt Domains', 'Improve app performance', 'Review endpoint utterances', 'Features', and 'Patterns'. The main content area is titled 'Intents' with a help icon. It includes buttons for '+ Create', '+ Add prebuilt domain intent', 'Rename', and 'Delete'. A search bar says 'Search for an intent by name ...'. Below these are three columns: 'Name ↑', 'Examples', and 'Features'. There is one row in the table with the intent name 'None' in blue, 0 examples, and a '+ Add feature' button. In the bottom right corner of the main area, there's a circular button with a magnifying glass icon.

Next steps

If your app design includes intent detection, [create new intents](#), and add example utterances. If your app design is only data extraction, add example utterances to the None intent, then [create entities](#), and label the example utterances with those entities.

Add intents to determine user intention of utterances

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Add [intents](#) to your LUIS app to identify groups of questions or commands that have the same intention.

Add an intent to your app

1. Sign in to the [LUIS portal](#), and select your **Subscription** and **Authoring** resource to see the apps assigned to that authoring resource.
2. Open your app by selecting its name on the **My Apps** page.
3. Select **Build** from the top navigation bar, then select **Intents** from the left panel.
4. On the **Intents** page, select **+ Create**.
5. In the **Create new intent** dialog box, enter the intent name, for example *ModifyOrder*, and select **Done**.

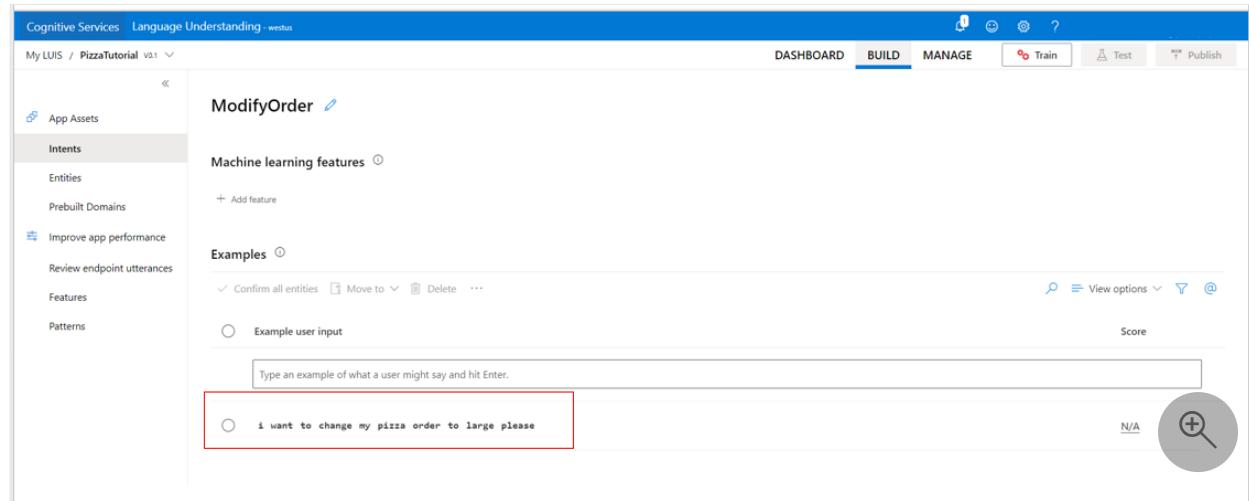


The intent needs [example utterances](#) in order to predict utterances at the published prediction endpoint.

Add an example utterance

Example utterances are text examples of user questions or commands. To teach Language Understanding (LUIS) when to predict the intent, you need to add example utterances. Carefully consider each utterance you add. Each utterance added should be different than the examples that are already added to the intent..

On the intent details page, enter a relevant utterance you expect from your users, such as *"I want to change my pizza order to large please"* in the text box below the intent name, and then press Enter.



Luis converts all utterances to lowercase and adds spaces around **tokens**, such as hyphens.

Intent prediction errors

An intent prediction error is determined when an utterance is not predicted with the trained app for the intent.

1. To find utterance prediction errors and fix them, use the **Incorrect** and **Unclear** filter options.

The screenshot shows the Microsoft LUIS portal's Intent details page for the 'ModifyOrder' intent. On the left, there's a sidebar with options like App Assets, Intents, Entities, Prebuilt Domains, Improve app performance, Review endpoint utterances, Features, and Patterns. The main area shows the 'ModifyOrder' intent with its machine learning features and examples. Examples include 'Example user input' and 'book a flight please'. Each example has a 'Score' (e.g., 0.192), 'Nearest intent' (e.g., 0.35 None), and a 'Difference' score (-0.16). A red box highlights the 'Difference' section in the sidebar, which includes checkboxes for 'Unclear' (checked), 'Correct' (unchecked), and 'Incorrect' (checked). The sidebar also shows filters for 'Intent score' (0-1), 'Entities' (No items available), and 'Nearest intent' (None). There are buttons for 'Train', 'Test', and 'Publish' at the top right.

2. To display the score value on the Intent details page, select **Show details intent scores** from the **View** menu.

When the filters and view are applied and there are example utterances with errors, the example utterance list will show the utterances and the issues.

Each row shows the current training's prediction score for the example utterance, and the nearest other intent score, which is the difference between these two scores.

💡 Tip

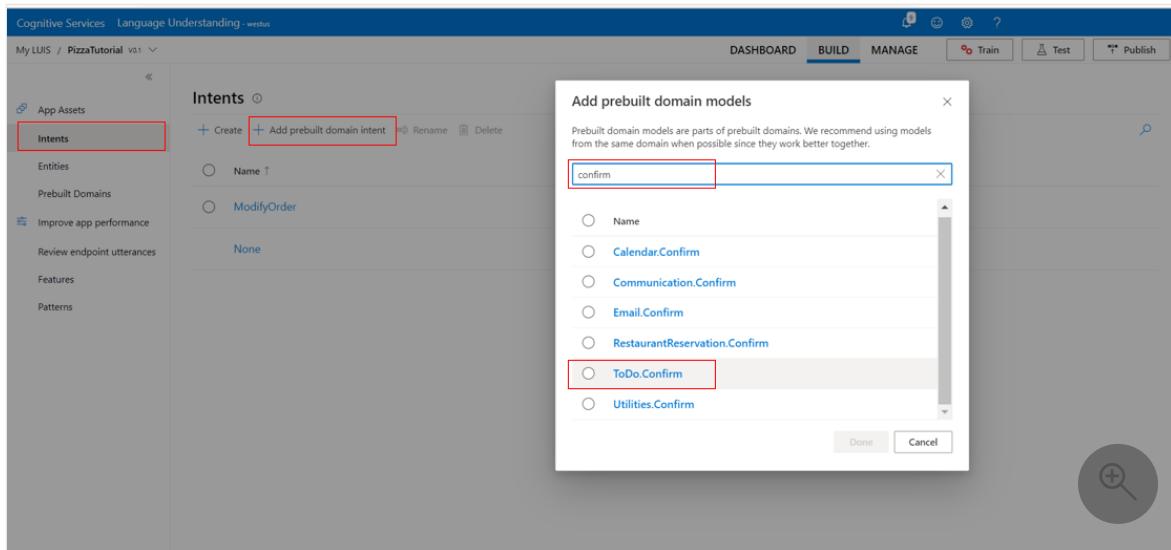
To fix intent prediction errors, use the **Summary dashboard**. The summary dashboard provides analysis for the active version's last training and offers the top suggestions to fix your model.

Add a prebuilt intent

Now imagine you want to quickly create a confirmation intent. You can use one of the prebuilt intents to create a confirmation intent.

1. On the **Intents** page, select **Add prebuilt domain intent** from the toolbar above the intents list.

2. Select an intent from the pop-up dialog.



3. Select the **Done** button.

Next steps

- Add entities
- Label entities
- Train and test

Add entities to extract data

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Create entities to extract key data from user utterances in Language Understanding (LUIS) apps. Extracted entity data is used by your client application to fulfill customer requests.

The entity represents a word or phrase inside the utterance that you want extracted. Entities describe information relevant to the intent, and sometimes they are essential for your app to perform its task.

How to create a new entity

The following process works for [machine learned entities](#), [list entities](#), and [regular expression entities](#).

1. Sign in to the [LUIS portal](#), and select your **Subscription** and **Authoring resource** to see the apps assigned to that authoring resource.
2. Open your app by selecting its name on [My Apps](#) page.
3. Select **Build** from the top navigation menu, then select **Entities** from the left panel, [Select + Create](#), then select the entity type.
4. Continue configuring the entity. Select **Create** when you are done.

Create a machine learned entity

Following the pizza example, we would need to create a "PizzaOrder" entity to extract pizza orders from utterances.

1. Select **Build** from the top navigation menu, then select **Entities** from the left panel
2. In the **Create an entity type** dialog box, enter the name of the entity and select **Machine learned**, select. To add sub-entities, select **Add structure**. Then select **Create**.

Create an entity

X

Name *

PizzaOrder

Type: Machine Learned List Regex Pattern.Any

Machine learned entities are learned from context. Use an ML entity to identify data that are not always well-formatted but have the same meaning. An ML entity can be composed of smaller subentities, each of which can have its own properties.

Example [Show subentities](#)

"Book 2 adult business tickets to Arrabury Airport"

TicketOrder

Add structure

Next

Cancel

A pizza order might include many details, like quantity and type. To add these details, we would create a subentity.

3. In **Add subentities**, add a subentity by selecting the + on the parent entity row.

Add subentities (optional)

X

An ML entity can be composed of smaller subentities, each of which can have its own properties. You can add subentities when you create your ML entity, and you can add subentities to existing ML entities.

Note: Once you have created your entity you will not be able to rearrange subentities.

PizzaOrder

+ ^

Quantity

+ □ ^

Type

TypeName

+ □ ^

Crust

+ □ ^

AdditionalToppings

+ □ ^

Back

Create

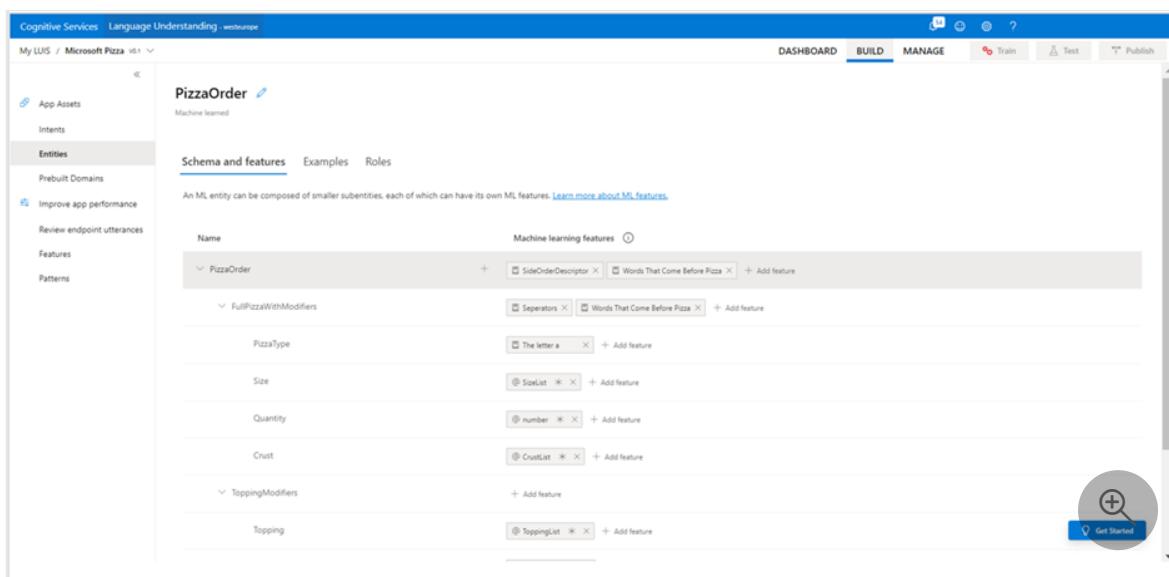
Cancel

4. Select **Create** to finish the creation process.

Add a feature to a machine learned entity

Some entities include many details. Imagine a "PizzaOrder" entity, it may include "*ToppingModifiers*" or "*FullPizzaWithModifiers*". These could be added as features to a machine learned entity.

1. Select **Build** from the top navigation bar, then select **Entities** from the left panel.
2. Add a feature by selecting **+ Add feature** on the entity or subentity row.
3. Select one of the existing entities and phrase lists.
4. If the entity should only be extracted if the feature is found, select the asterisk for that feature.



Create a regular expression entity

For extracting structured text or a predefined sequence of alphanumeric values, use regular expression entities. For example, *OrderNumber* could be predefined to be exactly 5 characters with type numbers ranging between 0 and 9.

1. Select **Build** from the top navigation bar, then select **Intents** from the left panel
2. Select **+ Create**.
3. In the **Create an entity type** dialog box, enter the name of the entity and select **RegEx**, enter the regular expression in the **Regex** field and select **Create**.

Create an entity

X

Name *

OrderNumber

Type: Machine learned List Regex Pattern.any

Regex entities extract an entity based on a regular expression pattern you provide. Regex entities are a good fit for the data that are consistently formatted with any variation that is also consistent.

Example

"Book 2 adult business tickets on flight SU 2996"

flightNumber

"flightNumber" is defined as: `flight [A-Z]{2} [0-9]{4}`

Regex *

[0-9]{5}

Create

Cancel

Create a list entity

List entities represent a fixed, closed set of related words. While you, as the author, can change the list, LUIS won't grow or shrink the list. You can also import to an existing list entity using a [list entity .json format](#).

Use the procedure to create a list entity. Once the list entity is created, you don't need to label example utterances in an intent. List items and synonyms are matched using exact text. A "Size" entity could be of type list, and it will include different sizes like "small", "medium", "large" and "family".

1. From the **Build** section, select **Entities** in the left panel, and then select **+ Create**.
2. In the **Create an entity type** dialog box, enter the name of the entity, such as *Size* and select **List**.
3. In the **Create a list entity** dialog box, in the **Add new sublist....**, enter the list item name, such as *large*. Also, you can add synonyms to a list item like *huge* and *mega* for item *large*.

The screenshot shows the Microsoft LUIS Entities page. On the left, a sidebar lists categories: App Assets, Intents, Entities (selected), Prebuilt Domains, Improve app performance, Review endpoint utterances, Features, and Patterns. The main area is titled 'Size' with a 'List' sub-type. It has tabs for 'List items' (selected), 'Examples', and 'Roles'. Below this, a note states: 'List entities represent a fixed, closed set of related words along with their synonyms. List entities are extracted by an exact text match and can be resolved to a normalized value.' A link 'Learn more about List entities' is provided. There are buttons for 'Import values' and 'Delete'. A search bar is at the top right. The 'List items' section contains four entries: 'Normalized values ↑' (with a dropdown menu 'Type in a list item ...'), 'Family' (with a dropdown menu 'Type in value ...'), 'Large' (with a dropdown menu 'Type in value ...'), and 'Medium' (with a dropdown menu 'Type in value ...'). A large circular button with a plus sign and a magnifying glass icon is in the bottom right corner.

4. When you are finished adding list items and synonyms, select **Create**.

When you are done with a group of changes to the app, remember to **Train** the app. Do not train the app after a single change.

ⓘ Note

This procedure demonstrates creating and labeling a list entity from an example utterance in the **Intent detail** page. You can also create the same entity from the **Entities** page.

Add a prebuilt domain entity

1. Select **Entities** in the left side.
2. On the **Entities** page, select **Add prebuilt domain entity**.
3. In **Add prebuilt domain models** dialog box, select the prebuilt domain entity.
4. Select **Done**. After the entity is added, you do not need to train the app.

Add a prebuilt entity

To recognize common types of information, add a [prebuilt entity](#)

1. Select **Entities** in the left side.
2. On the **Entities** page, select **Add prebuilt entity**.
3. In **Add prebuilt entities** dialog box, select the prebuilt entity.

Add prebuilt entities

X

When you add a built-in entity, its predictions will be available to you while labeling utterances.



Search built-in entities ...



Name



age



Age of a person or thing

10-month-old, 19 years old, 58 year-old



datetimeV2



Dates and times, resolved to a canonical form

June 23, 1976, Jul 11 2012, 7 AM, 6:49 PM, tomorrow at 7 AM



dimension



Spacial dimensions, including length, distance, area, and volume

2 miles, 650 square kilometres, 9,350 feet



email



Email addresses

Done

Cancel



4. Select Done. After the entity is added, you do not need to train the app.

Add a role to distinguish different contexts

A role is a named subtype of an entity, based on context. In the following utterance, there are two locations, and each is specified semantically by the words around it such as to and from:

Pick up the pizza order from Seattle and deliver to New York City.

In this procedure, add origin and destination roles to a prebuilt geographyV2 entity.

1. From the Build section, select Entities in the left panel.
2. Select + Add prebuilt entity. Select geographyV2 then select Done. A prebuilt entity will be added to the app.

If you find that your pattern, when it includes a Pattern.any, extracts entities incorrectly, use an [explicit list](#) to correct this problem.

1. Select the newly added prebuilt geographyV2 entity from the Entities page list of entities.
2. To add a new role, select + next to **No roles added**.
3. In the **Type role...** textbox, enter the name of the role Origin then enter. Add a second role name of Destination then enter.

The screenshot shows the Microsoft LUIS portal interface. The top navigation bar includes 'Cognitive Services', 'Language Understanding', and 'My LUIS / Microsoft Pizza v3.1'. The main menu on the left has 'Entities' selected, with other options like 'App Assets', 'Intents', 'Prebuilt Domains', 'Improve app performance', 'Review endpoint utterances', 'Features', and 'Patterns'. The central panel displays the 'geographyV2' entity, which is a 'Prebuilt' entity. Under the 'Roles' tab, there are three input fields: 'Origin', 'Destination', and 'Type role...'. The 'Type role...' field is highlighted with a red box. A tooltip above the 'Type role...' field explains: 'A role is a named alias for an entity based on context within the utterance. A role can be used in both example utterances and patterns.' A large search icon is located in the bottom right corner of the main panel area.

The role is added to the prebuilt entity but isn't added to any utterances using that entity.

Create a pattern.any entity

Patterns are designed to improve accuracy when multiple utterances are very similar. A pattern allows you to gain more accuracy for an intent without providing several more utterances. The [Pattern.any](#) entity is only available with patterns. See the [patterns article](#) for more information.

Next steps

- [Label your example utterances](#)
- [Train and test your application](#)

How to label example utterances

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Labeling an entity in an example utterance gives LUIS an example of what the entity is and where the entity can appear in the utterance. You can label machine-learned entities and subentities.

You only label machine-learned entities and sub-entities. Other entity types can be added as features to them when applicable.

Label example utterances from the Intent detail page

To label examples of entities within the utterance, select the utterance's intent.

1. Sign in to the [LUIS portal](#), and select your **Subscription** and **Authoring resource** to see the apps assigned to that authoring resource.
2. Open your app by selecting its name on **My Apps** page.
3. Select the Intent that has the example utterances you want to label for extraction with an entity.
4. Select the text you want to label then select the entity.

Two techniques to label entities

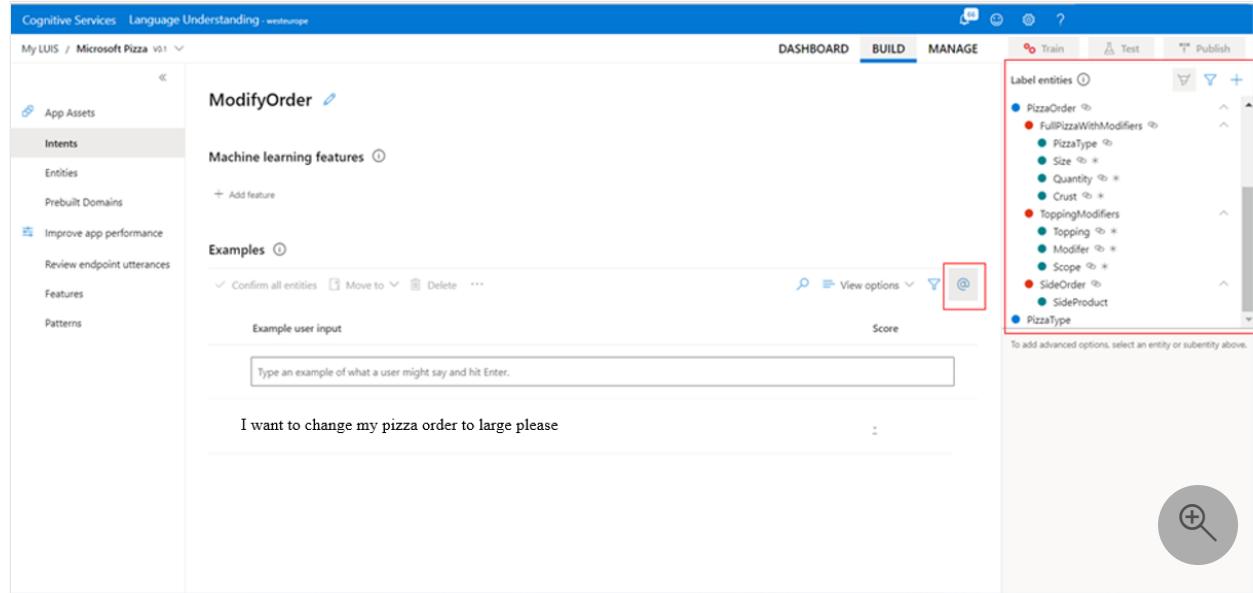
Two labeling techniques are supported on the Intent detail page.

- Select entity or subentity from [Entity Palette](#) then select within example utterance text. This is the recommended technique because you can visually verify you are working with the correct entity or subentity, according to your schema.
- Select within the example utterance text first. A menu will appear with labeling choices.

Label with the Entity Palette visible

After you've [planned your schema with entities](#), keep the **Entity palette** visible while labeling. The **Entity palette** is a reminder of what entities you planned to extract.

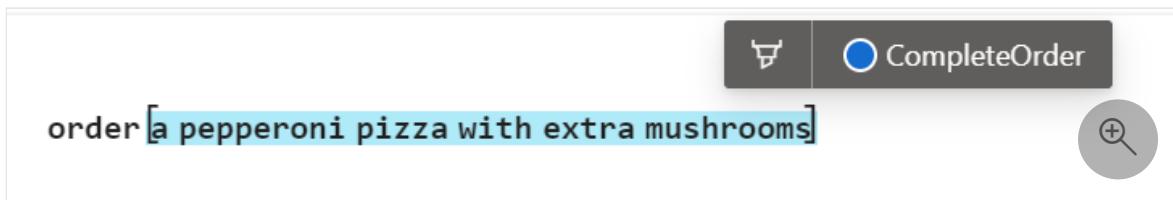
To access the **Entity Palette**, select the @ symbol in the contextual toolbar above the example utterance list.



Label entity from Entity Palette

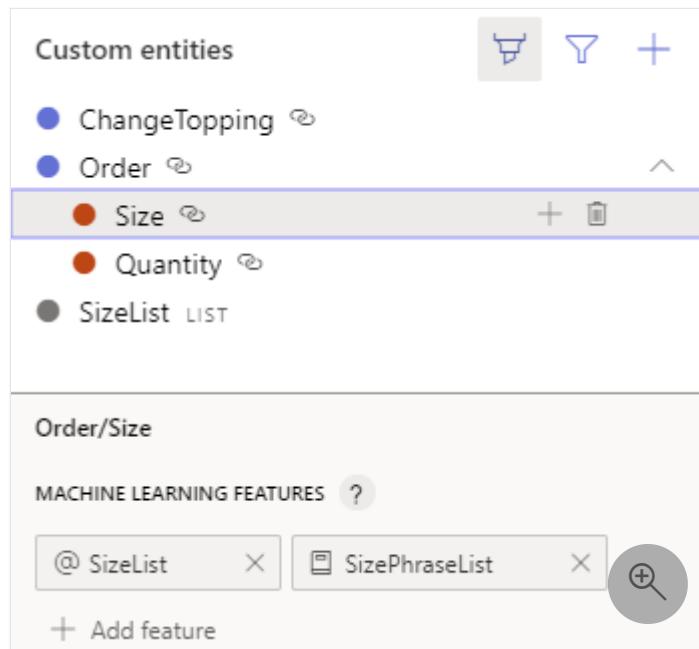
The entity palette offers an alternative to the previous labeling experience. It allows you to brush over text to instantly label it with an entity.

1. Open the entity palette by selecting on the @ symbol at the top right of the utterance table.
2. Select the entity from the palette that you want to label. This action is visually indicated with a new cursor. The cursor follows the mouse as you move in the LUIS portal.
3. In the example utterance, *paint* the entity with the cursor.



Add entity as a feature from the Entity Palette

The Entity Palette's lower section allows you to add features to the currently selected entity. You can select from all existing entities and phrase lists or create a new phrase list.

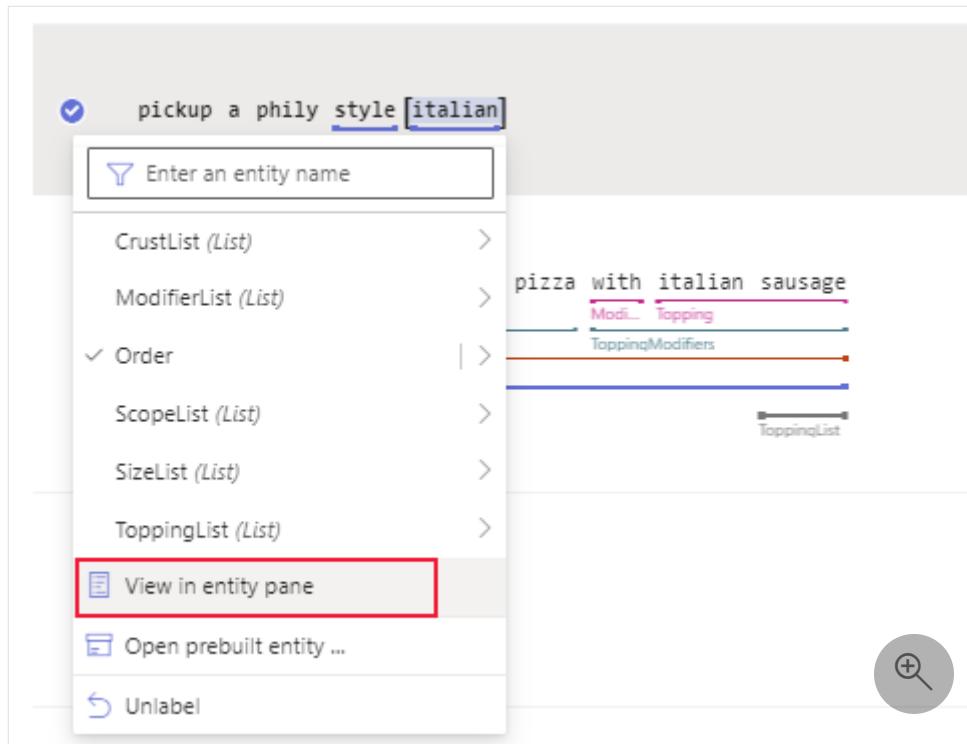


Label text with a role in an example utterance

Tip

Roles can be replaced by labeling with subentities of a machine-learning entities.

1. Go to the Intent details page, which has example utterances that use the role.
2. To label with the role, select the entity label (solid line under text) in the example utterance, then select **View in entity pane** from the drop-down list.



The entity palette opens to the right.

3. Select the entity, then go to the bottom of the palette and select the role.

Custom entities

- CrustList LIST
- ModifierList LIST
- Order ↗
 - FullPizzaWithModifiers ↗
 - PizzaType ↗
 - Size ↗ *
 - Quantity ↗ *
 - Crust ↗ *
 - ToppingModifiers
 - Topping ↗ *
 - Modifier ↗ *
 - Scope ↗ *
 - SideOrder ↗
 - SideProduct
- ScopeList LIST
- SizeList LIST
- ToppingList LIST

Order

ROLE

Select or create a role

MACHINE LEARNING FEATURES ?

- SideOrderDescriptor X
- Words That Come Before Pizza X

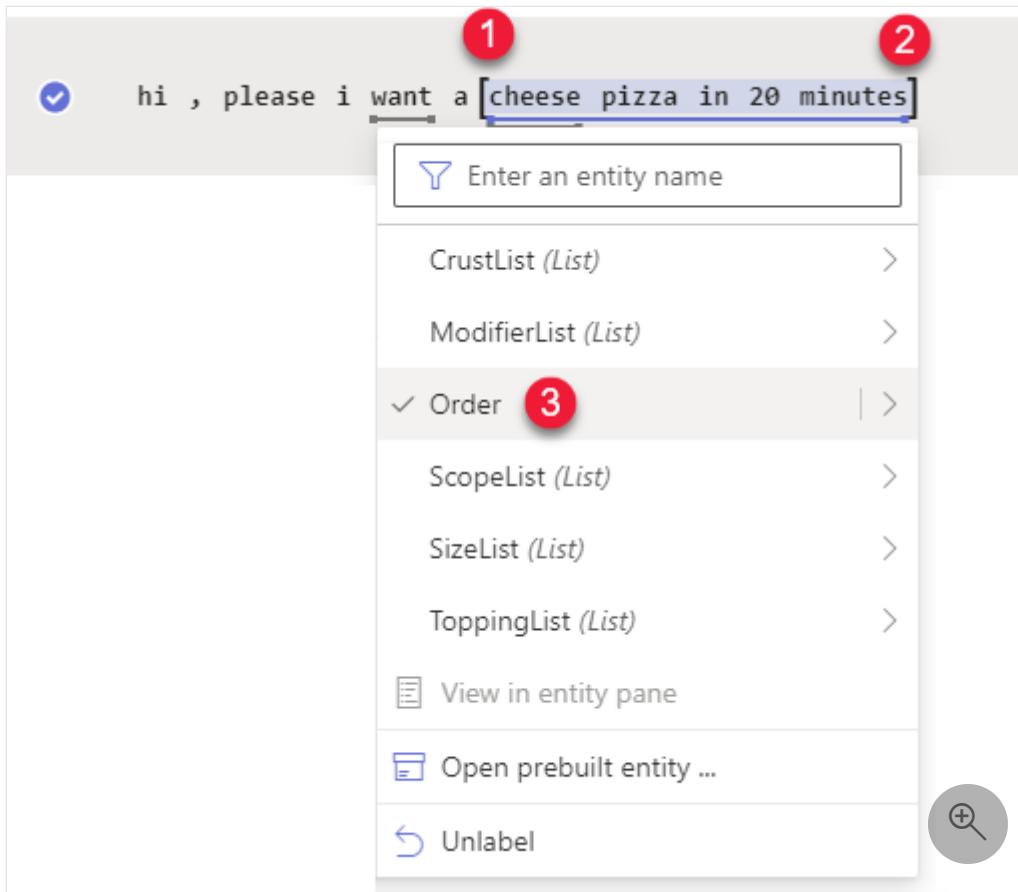
+ Add feature

Label entity from in-place menu

Labeling in-place allows you to quickly select the text within the utterance and label it. You can also create a machine learning entity or list entity from the labeled text.

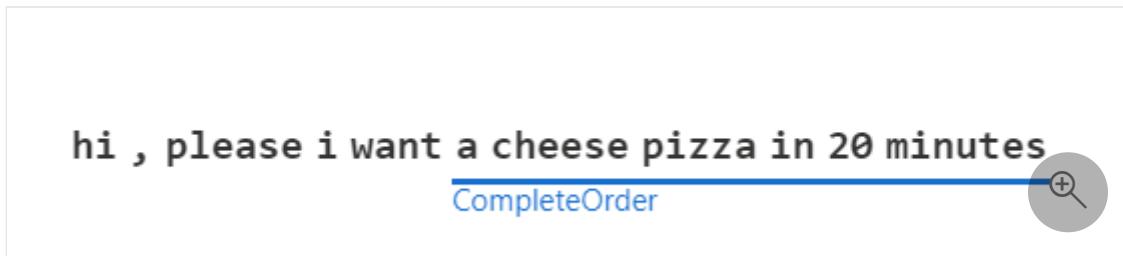
Consider the example utterance: "hi, please i want a cheese pizza in 20 minutes".

Select the left-most text, then select the right-most text of the entity. In the menu that appears, pick the entity you want to label.



Review labeled text

After labeling, review the example utterance and ensure the selected span of text has been underlined with the chosen entity. The solid line indicates the text has been labeled.



Confirm predicted entity

If there is a dotted-lined box around the span of text, it indicates the text is predicted but *not labeled yet*. To turn the prediction into a label, select the utterance row, then select **Confirm entities** from the contextual toolbar.

ⓘ Note

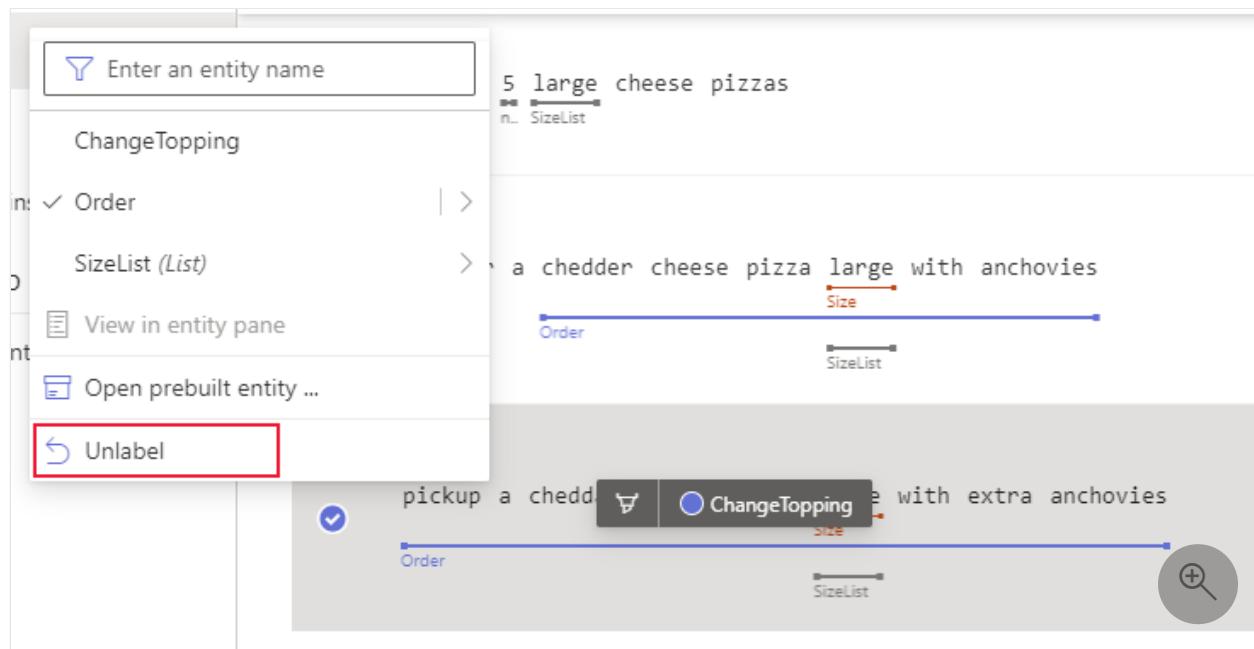
You do not need to label for punctuation. Use **application settings** to control how punctuation impacts utterance predictions.

Unlabel entities

ⓘ Note

Only machine learned entities can be unlabeled. You can't label or unlabel regular expression entities, list entities, or prebuilt entities.

To unlabel an entity, select the entity and select **Unlabel** from the in-place menu.



Automatic labeling for parent and child entities

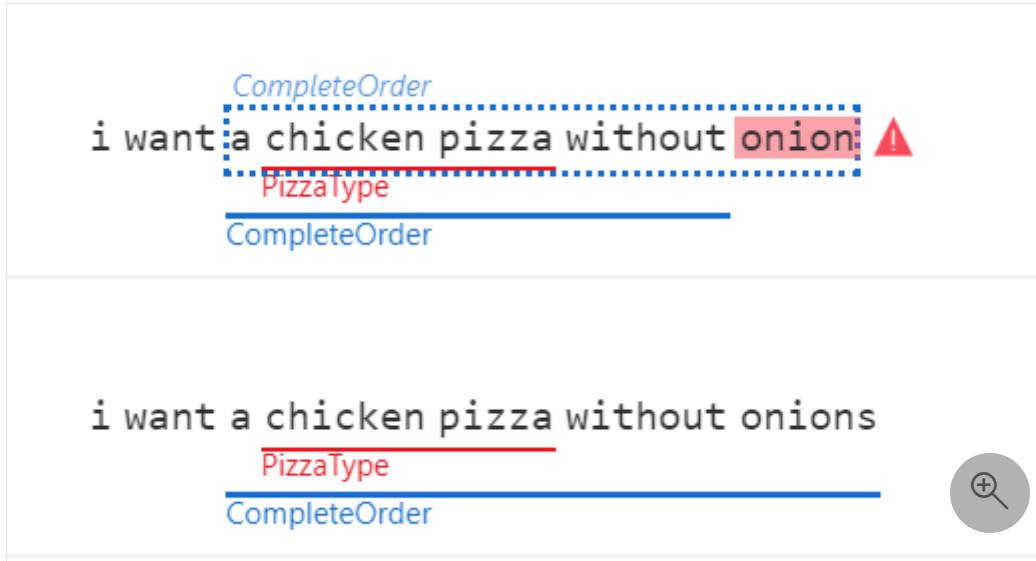
If you are labeling for a subentity, the parent will be labeled automatically.

Automatic labeling for non-machine learned entities

Non-machine learned entities include prebuilt entities, regular expression entities, list entities, and pattern.any entities. These are automatically labeled by LUIS so they are not required to be manually labeled by users.

Entity prediction errors

Entity prediction errors indicate the predicted entity doesn't match the labeled entity. This is visualized with a caution indicator next to the utterance.



Next steps

[Train and test your application](#)

Train and test your LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Training is the process of teaching your Language Understanding (LUIS) app to extract intent and entities from user utterances. Training comes after you make 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, you should make updates to the LUIS app, then train and test again.

Training is applied to the active version in the LUIS portal.

How to train interactively

Before you start training your app in the [LUIS portal](#), make sure every intent has at least one utterance. You must train your LUIS app at least once to test it.

1. Access your app by selecting its name on the [My Apps](#) page.
2. In your app, select **Train** in the top-right part of the screen.
3. When training is complete, a notification appears at the top of the browser.

ⓘ Note

The training dates and times are in GMT + 2.

Start the training process

💡 Tip

You do not need to train after every single change. Training should be done after a group of changes are applied to the model, or if you want to test or publish the

app.

To train your app in the LUIS portal, you only need to select the **Train** button on the top-right corner of the screen.

Training with the REST APIs is a two-step process.

1. Send an HTTP POST [request for training ↗](#).
2. Request the [training status ↗](#) with an HTTP GET request.

In order to know when training is complete, you must poll the status until all models are successfully trained.

Test Your application

Testing is the process of providing sample utterances to LUIS and getting a response of recognized intents and entities. You can test your LUIS app interactively one utterance at a time, or provide a set of utterances. While testing, you can compare the current active model's prediction response to the published model's prediction response.

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.

Interactive testing

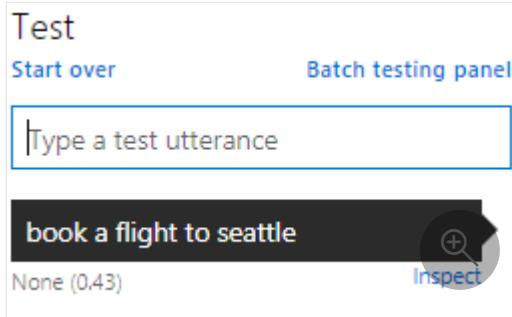
Interactive testing is done from the **Test** panel of the LUIS portal. You can enter an utterance to see how intents and entities are identified and scored. If LUIS isn't predicting an utterance's intents and entities as you would expect, copy the utterance to the **Intent** page as a new utterance. Then label parts of that utterance for entities to train your LUIS app.

See [batch testing](#) if you are testing more than one utterance at a time, and the [Prediction scores](#) article to learn more about prediction scores.

Test an utterance

The test utterance should not be exactly the same as any example utterances in the app. The test utterance should include word choice, phrase length, and entity usage you expect for a user.

1. Sign in to the [LUIS portal](#), and select your **Subscription** and **Authoring resource** to see the apps assigned to that authoring resource.
2. Open your app by selecting its name on **My Apps** page.
3. Select **Test** in the top-right corner of the screen for your app, and a panel will slide into view.



4. Enter an utterance in the text box and press the enter button on the keyboard. You can test a single utterance in the **Test** box, or multiple utterances as a batch in the **Batch testing panel**.
5. The utterance, its top intent, and score are added to the list of utterances under the text box. In the above example, this is displayed as 'None (0.43)'.

Inspect the prediction

Inspect the test result details in the **Inspect** panel.

1. With the **Test** panel open, select **Inspect** for an utterance you want to compare. **Inspect** is located next to the utterance's top intent and score. Refer to the above image.
2. The **Inspection** panel will appear. The panel includes the top scoring intent and any identified entities. The panel shows the prediction of the selected utterance.

Test

[Start over](#) [Batch testing panel](#)

Type a test utterance ...

hello

None (0.8511093) [Inspect](#)

Version: 0.1 [X](#)

[Start over](#) [Compare with published](#)

User input

hello

Top scoring intent

None (0.8511093) [Add to example utterances](#)

ML Entities [Disable required features](#)

No predicted n-depth entities

Composite Entities

No predicted composite entities

Other entities

No predicted entities

Top matched pattern(s)

No matched patterns

Sentiment 

Enable sentiment analysis to get sentiment score

 **Tip**

From the inspection panel, you can add the test utterance to an intent by selecting [Add to example utterances](#).

Change deterministic training settings using the version settings API

Use the [Version settings API](#) with the UseAllTrainingData set to *true* to turn off deterministic training.

Change deterministic training settings using the LUIS portal

Log into the [LUIS portal](#) and select your app. Select **Manage** at the top of the screen, then select **Settings**. Enable or disable the **use non-deterministic training** option. When disabled, training will use all available data. Training will only use a *random* sample of data from other intents as negative data when training each intent

The screenshot shows the LUIS portal interface under the 'MANAGE' tab. On the left, a sidebar has 'Settings' selected. In the main area, there's an 'App description (optional)' input field, an 'App ID' section, and a 'Culture' setting (en-us). Below these are sections for 'Make endpoints public' (set to 'Off'), 'Version Settings', and 'Normalize punctuation' (set to 'Off'). A red box highlights the 'Use non-deterministic training' section, which has a toggle switch set to 'On (recommended)'. There are also sections for 'Normalize word forms' (set to 'Off') and a search icon.

View sentiment results

If sentiment analysis is configured on the [Publish](#) page, the test results will include the sentiment found in the utterance.

Correct matched pattern's intent

If you are using [Patterns](#) and the utterance matched is a pattern, but the wrong intent was predicted, select the [Edit](#) link by the pattern and 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**.

Note

Any testing against the published model is deducted from your Azure subscription quota balance.

The screenshot shows two panels: 'Test' on the left and 'Published' on the right. Both panels have a header bar with 'Start over' and 'Compare with published' buttons, and a 'Show JSON view' button in the top right corner of the 'Published' panel. The 'Test' panel has a text input field containing 'Type a test utterance ...' and a bolded message 'is href-123234 available in french?'. Below this is a 'FindForm (0.979601562)' entry with an 'Inspect' link. The 'Published' panel also has a text input field with the same message. It lists several sections: 'User input' (is href-123234 available in french?), 'Top scoring intent' (FindForm (0.979601562)), 'ML Entities' (HR Request: href - 123234 available in french), 'Composite Entities' (No predicted composite entities), 'Other entities' (FormNumber: href - 123234), 'Top matched pattern(s)' (No matched patterns), and 'Sentiment' (neutral (1)).

View endpoint JSON in test panel

You can view the endpoint JSON returned for the comparison by selecting the **Show JSON view** in the top-right corner of the panel.

Next steps

If testing requires testing a batch of utterances, See [batch testing](#).

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.

- [Improve your application](#)
- [Publishing your application](#)

Batch testing with a set of example utterances

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Batch testing validates your active trained version to measure its prediction accuracy. A batch test helps you view the accuracy of each intent and entity in your active version. 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 or labeling entities within the utterance.

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.

The batch JSON file you use should include utterances with top-level machine-learning entities labeled including start and end position. The utterances should not be part of the examples already in the app. They should be utterances you want to positively predict for intent and entities.

You can separate out tests by intent and/or entity or have all the tests (up to 1000 utterances) in the same file.

Common errors importing a batch

If you run into errors uploading your batch file to LUIS, check for the following common issues:

- More than 1,000 utterances in a batch file

- 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 labels starting or ending on a space.

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.

LUIS portal

Batch testing using the LUIS portal

Import and train an example app

Import an app that takes a pizza order such as `1 pepperoni pizza on thin crust`.

1. Download and save [app JSON file](#).
2. Sign in to the [LUIS portal](#), and select your **Subscription** and **Authoring resource** to see the apps assigned to that authoring resource.
3. Select the arrow next to **New app** and click **Import as JSON** to import the JSON into a new app. Name the app `Pizza app`.
4. Select **Train** in the top-right corner of the navigation to train the app.

Roles in batch testing

⊗ Caution

Entity roles are not supported in batch testing.

Batch test file

The example JSON includes one utterance with a labeled entity to illustrate what a test file looks like. In your own tests, you should have many utterances with correct intent and machine-learning entity labeled.

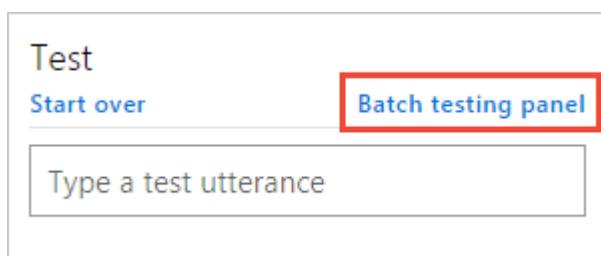
1. Create `pizza-with-machine-learned-entity-test.json` in a text editor or [download](#) it.
2. In the JSON-formatted batch file, add an utterance with the **Intent** you want predicted in the test.

```
JSON

[
  {
    "text": "I want to pick up 1 cheese pizza",
    "intent": "ModifyOrder",
    "entities": [
      {
        "entity": "Order",
        "startPos": 18,
        "endPos": 31
      },
      {
        "entity": "ToppingList",
        "startPos": 20,
        "endPos": 25
      }
    ]
  }
]
```

Run the batch

1. Select **Test** in the top navigation bar.
2. Select **Batch testing panel** in the right-side panel.



3. Select **Import**. In the dialog box that 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](#).

4. Choose the file location of the `pizza-with-machine-learned-entity-test.json` file.
5. Name the dataset `pizza test` and select **Done**.
6. Select the **Run** button.
7. After the batch test completes, you can see the following columns:

Column	Description
State	Status of the test. See results is only visible after the test is completed.
Name	The name you have given to the test.
Size	Number of tests in this batch test file.
Last Run	Date of last run of this batch test file.
Last result	Number of successful predictions in the test.

8. To view detailed results of the test, select **See results**.

 **Tip**

- Selecting **Download** will download the same file that you uploaded.
- If you see the batch test failed, at least one utterance intent did not match the prediction.

Review batch results for intents

To review the batch test results, select **See results**. The test results show graphically how the test utterances were predicted against the active version.

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.

1. Select the **ModifyOrder** intent in the filter list. The utterance is predicted as a **True Positive** meaning the utterance successfully matched its positive prediction listed in the batch file.



The green checkmarks in the filters list also indicate the success of the test for each intent. All the other intents are listed with a 1/1 positive score because the utterance was tested against each intent, as a negative test for any intents not listed in the batch test.

2. Select the **Confirmation** intent. This intent isn't listed in the batch test so this is a negative test of the utterance that is listed in the batch test.

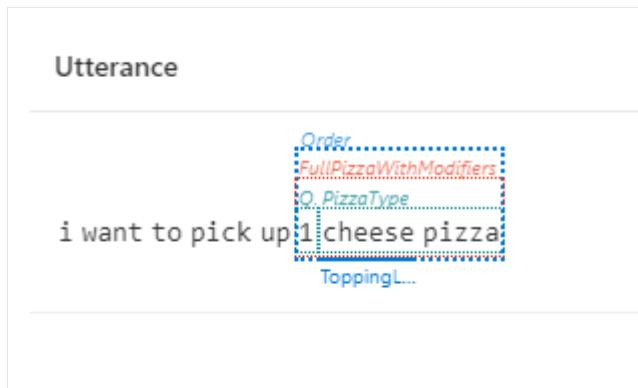


The negative test was successful, as noted with the green text in the filter, and the grid.

Review batch test results for entities

The **ModifyOrder** entity, as a machine entity with subentities, displays if the top-level entity matched and how the subentities are predicted.

1. Select the **ModifyOrder** entity in the filter list then select the circle in the grid.
2. The entity prediction displays below the chart. The display includes solid lines for predictions that match the expectation and dotted lines for predictions that don't match the expectation.



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.

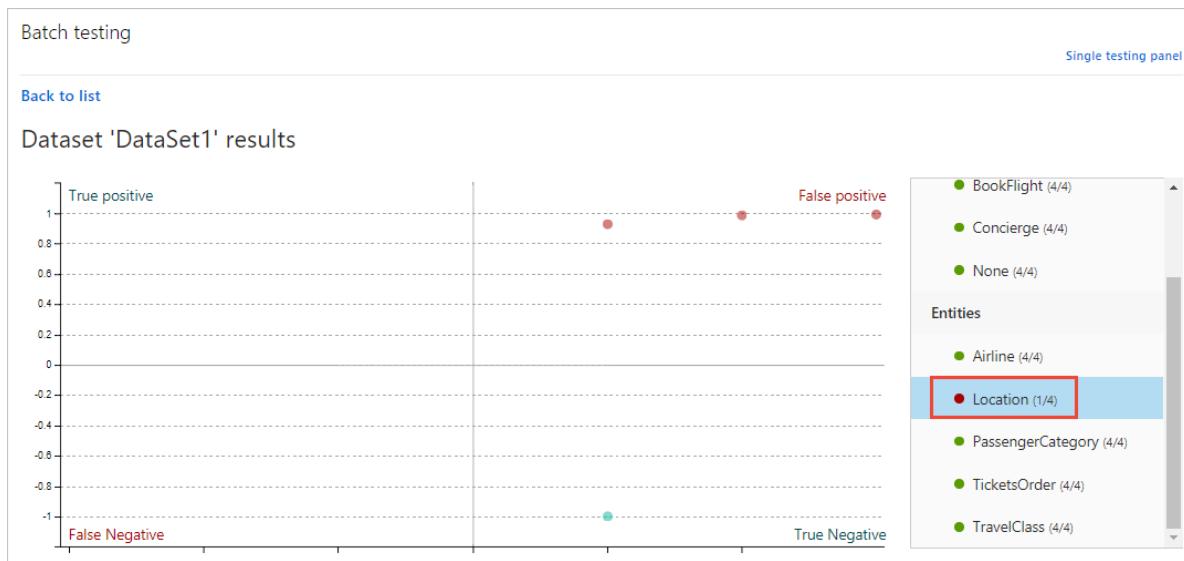
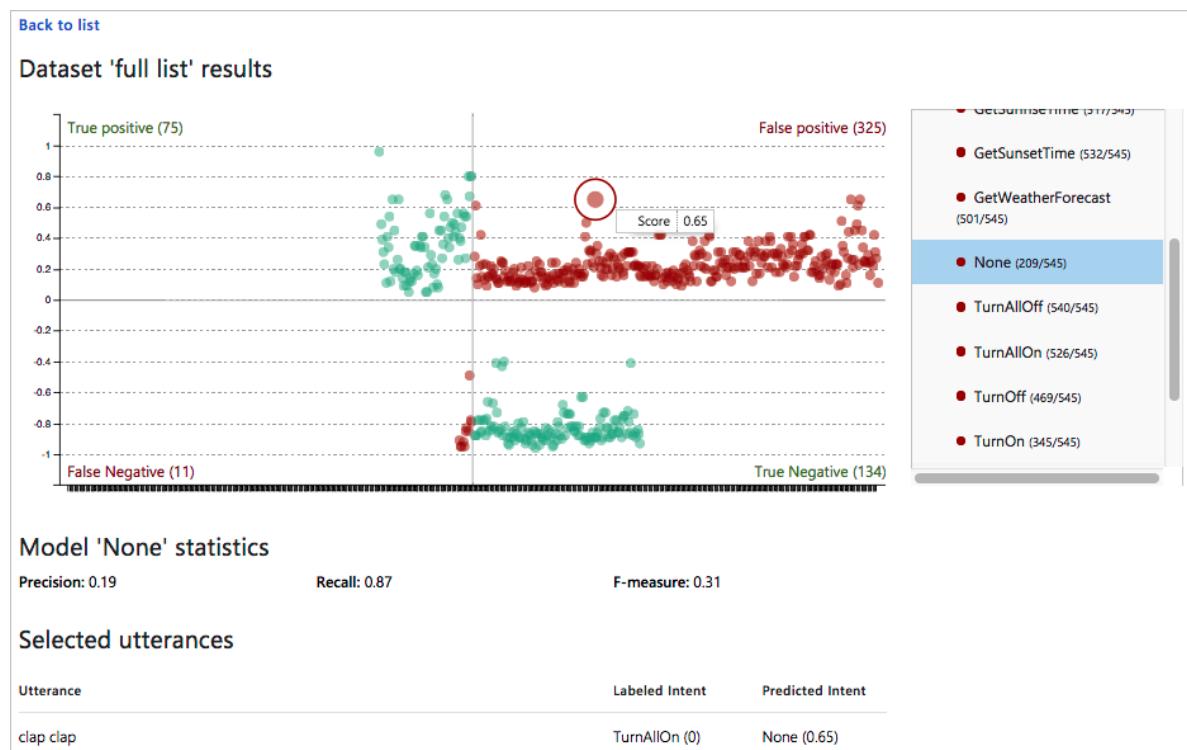


Chart result examples

The chart in the LUIS portal, you can perform the following actions:

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.

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](#)

Publish your active, trained app

Article • 07/18/2023

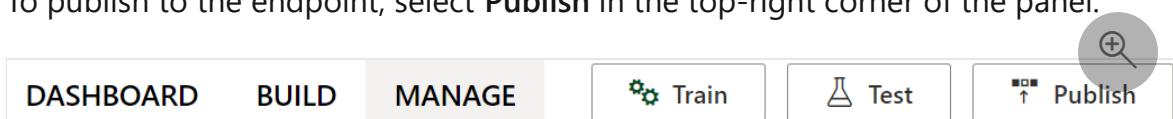
ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

When you finish building, training, and testing your active LUIS app, you make it available to your client application by publishing it to an endpoint.

Publishing

1. Sign in to the [LUIS portal](#), and select your **Subscription** and **Authoring resource** to see the apps assigned to that authoring resource.
2. Open your app by selecting its name on **My Apps** page.
3. To publish to the endpoint, select **Publish** in the top-right corner of the panel.



4. Select your settings for the published prediction endpoint, then select **Publish**.

Staging Slot

Last Published: 1/13/2021

Sentiment Analysis: Off

Speech Priming: Off



Publishing slots

Select the correct slot when the pop-up window displays:

- Staging
- Production

By using both publishing slots, you can have two different versions of your app available at the published endpoints, or the same version on two different endpoints.

Publish in more than one region

The app is published to all regions associated with the LUIS prediction resources. You can find your LUIS prediction resources in the LUIS portal by clicking **Manage** from the top navigation menu, and selecting [Azure Resources](#).

For example, if you add 2 prediction resources to an application in two regions, `westus` and `eastus`, and add these to the app as resources, the app is published in both regions. For more information about LUIS regions, see [Regions](#).

Configure publish settings

After you select the slot, configure the publish settings for:

- Sentiment analysis: Sentiment analysis allows LUIS to integrate with the Language service to provide sentiment and key phrase analysis. You do not have to provide a Language service key and there is no billing charge for this service to your Azure account. See [Sentiment analysis](#) for more information about the sentiment analysis JSON endpoint response.
- Speech priming: Speech priming is the process of sending the LUIS model output to the Speech service prior to converting the text to speech. This allows the speech service to provide speech conversion more accurately for your model. This allows for Speech and LUIS requests and responses in one call by making one speech call and getting back a LUIS response. It provides less latency overall.

After you publish, these settings are available for review from the **Manage** section's **Publish settings** page. You can change the settings with every publish. If you cancel a publish, any changes you made during the publish are also canceled.

When your app is published

When your app is successfully published, a success notification will appear at the top of the browser. The notification also includes a link to the endpoints.

If you need the endpoint URL, select the link or select **Manage** in the top menu, then select **Azure Resources** in the left menu.

Next steps

- See [Manage keys](#) to add keys to LUIS.
- See [Train and test your app](#) for instructions on how to test your published app in the test console.

How to update the LUIS model with REST APIs

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

In this article, you will add example utterances to a Pizza 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.

[Reference documentation ↗](#) | [Sample ↗](#)

Prerequisites

- [.NET Core 3.1 ↗](#)
- [Visual Studio Code ↗](#)

Example utterances JSON file

The example utterances follow 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. If you begin or end the label at a space in the text, the API call to add the utterances fails.

JSON

```
[  
  {  
    "text": "order a pizza",  
    "intentName": "OrderPizza",  
    "entityLabels": []  
  }]
```

```
"intentName": "ModifyOrder",
"entityLabels": [
  {
    "entityName": "Order",
    "startCharIndex": 6,
    "endCharIndex": 12
  }
]
},
{
  "text": "order a large pepperoni pizza",
  "intentName": "ModifyOrder",
  "entityLabels": [
    {
      "entityName": "Order",
      "startCharIndex": 6,
      "endCharIndex": 28
    },
    {
      "entityName": "FullPizzaWithModifiers",
      "startCharIndex": 6,
      "endCharIndex": 28
    },
    {
      "entityName": "PizzaType",
      "startCharIndex": 14,
      "endCharIndex": 28
    },
    {
      "entityName": "Size",
      "startCharIndex": 8,
      "endCharIndex": 12
    }
  ]
},
{
  "text": "I want two large pepperoni pizzas on thin crust",
  "intentName": "ModifyOrder",
  "entityLabels": [
    {
      "entityName": "Order",
      "startCharIndex": 7,
      "endCharIndex": 46
    },
    {
      "entityName": "FullPizzaWithModifiers",
      "startCharIndex": 7,
      "endCharIndex": 46
    },
    {
      "entityName": "PizzaType",
      "startCharIndex": 17,
      "endCharIndex": 32
    },
    {
      "entityName": "Size"
    }
  ]
}
```

```
        "entityName": "Size",
        "startCharIndex": 11,
        "endCharIndex": 15
    },
{
    "entityName": "Quantity",
    "startCharIndex": 7,
    "endCharIndex": 9
},
{
    "entityName": "Crust",
    "startCharIndex": 37,
    "endCharIndex": 46
}
]
}
```

Create Pizza app

Create the pizza app.

1. Select [pizza-app-for-luis-v6.json](#) to bring up the GitHub page for the `pizza-app-for-luis.json` file.
2. Right-click or long tap the **Raw** button and select **Save link as** to save the `pizza-app-for-luis.json` to your computer.
3. Sign into the [LUIS portal](#).
4. Select [My Apps](#).
5. On the **My Apps** page, select **+ New app for conversation**.
6. Select **Import as JSON**.
7. In the **Import new app** dialog, select the **Choose File** button.
8. Select the `pizza-app-for-luis.json` file you downloaded, then select **Open**.
9. In the **Import new app** dialog **Name** field, enter a name for your Pizza app, then select the **Done** button.

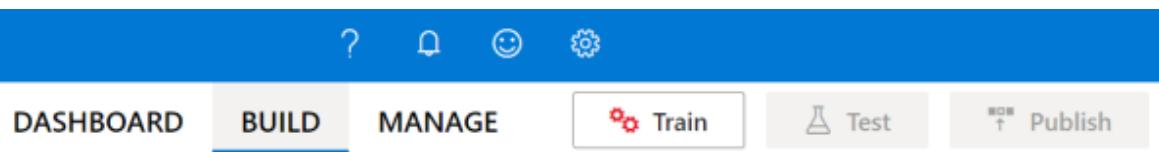
The app will be imported.

If you see a dialog **How to create an effective LUIS app**, close the dialog.

Train and publish the Pizza app

You should see the **Intents** page with a list of the intents in the Pizza app.

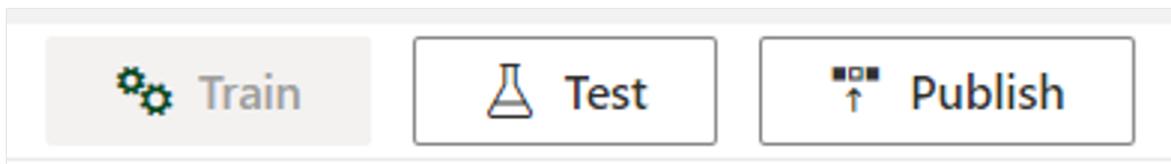
1. In the top-right side of the LUIS website, select the **Train** button.



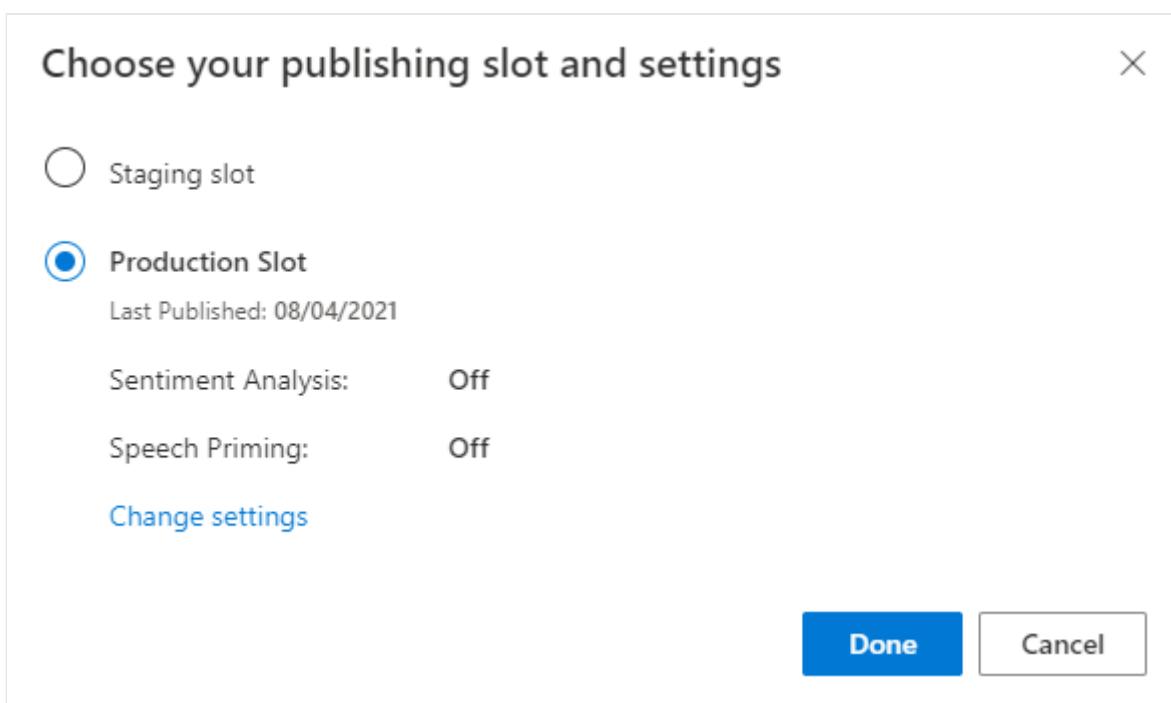
2. Training is complete when the **Train** button is disabled.

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

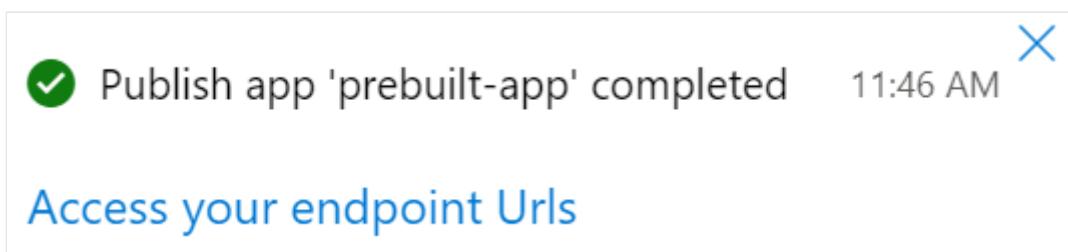
1. Select **Publish** in the top-right navigation.



2. Select the **Production** slot, then select **Done**.



3. Select **Access your endpoint URLs** in the notification to go to the **Azure Resources** page. You will only be able to see the URLs if you have a prediction resource associated with the app. You can also find the **Azure Resources** page by clicking **Manage**.



Add an authoring resource to the Pizza app

1. Select **MANAGE**.
2. Select **Azure Resources**.
3. Select **Authoring Resource**.
4. Select **Change authoring resource**.

If you have an authoring resource, enter the **Tenant Name**, **Subscription Name**, and **Luis resource name** of your authoring resource.

If you do not have an authoring resource:

1. Select **Create new resource**.
2. Enter a **Tenant Name**, **Resource Name**, **Subscription Name**, and **Azure Resource Group Name**.

Your Pizza app is now ready to use.

Record the access values for your Pizza app

To use your new Pizza app, you will need the app ID, authoring key, and authoring endpoint of your Pizza app. To get predictions, you will need your separate prediction endpoint and prediction key.

To find these values:

1. From the **Intents** page, select **MANAGE**.
2. From the **Application Settings** page, record the **App ID**.
3. Select **Azure Resources**.
4. Select **Authoring Resource**.
5. From the **Authoring Resource** and **Prediction Resources** tabs, record the **Primary Key**. This value is your authoring key.
6. Record the **Endpoint URL**. This value is your authoring endpoint.

Change model programmatically

1. Create a new console application targeting the C# language, with a project and folder name of `csharp-model-with-rest`.

Console

```
dotnet new console -lang C# -n csharp-model-with-rest
```

2. Change to the `csharp-model-with-rest` directory you created, and install required dependencies with these commands:

```
Console

cd csharp-model-with-rest
dotnet add package System.Net.Http
dotnet add package JsonFormatterPlus
```

3. Overwrite Program.cs with the following code:

```
C#

// This quickstart shows how to add utterances to a LUIS model using
// the REST APIs.
//

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;

namespace AddUtterances
{
    class Program
    {
        // Values to modify.

        // YOUR-APP-ID: The App ID GUID found on the www.luis.ai
        Application Settings page.
        static string appID = "PASTE_YOUR_LUIS_APP_ID_HERE";

        // YOUR-AUTHORING-KEY: Your LUIS authoring key, 32 character
        value.
        static string authoringKey =
        "PASTE_YOUR_LUIS_AUTHORING_SUBSCRIPTION_KEY_HERE";

        // YOUR-AUTHORING-ENDPOINT: Replace this endpoint with your
        authoring key endpoint.
        // For example, "https://your-resource-
        name.cognitiveservices.azure.com/"
        static string authoringEndpoint =
        "PASTE_YOUR_LUIS_AUTHORING_ENDPOINT_HERE";
```

```

// NOTE: Replace this your version number.
static string appVersion = "0.1";
///////////

    static string host = String.Format("{0}luis/authoring/v3.0-
preview/apps/{1}/versions/{2}/", authoringEndpoint, appID, appVersion);

    // GET request with authentication
    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);
        }
    }

    // POST request with authentication
    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 utterances as string with POST request
    async static Task AddUtterances(string utterances)
    {
        string uri = host + "examples";

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

    // Train app after adding utterances

```

```

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

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

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

    // Add utterances, train, check status
    static void Main(string[] args)
    {
        string utterances = @"
[
{
    'text': 'order a pizza',
    'intentName': 'ModifyOrder',
    'entityLabels': [
        {
            'entityName': 'Order',
            'startCharIndex': 6,
            'endCharIndex': 12
        }
    ]
},
{
    'text': 'order a large pepperoni pizza',
    'intentName': 'ModifyOrder',
    'entityLabels': [
        {
            'entityName': 'Order',
            'startCharIndex': 6,
            'endCharIndex': 28
        },
        {
            'entityName': 'FullPizzaWithModifiers',
            'startCharIndex': 6,
            'endCharIndex': 28
        },
        {
            'entityName': 'PizzaType',
            'startCharIndex': 14,
            'endCharIndex': 28
        },
        {
            'entityName': 'Large',
            'startCharIndex': 14,
            'endCharIndex': 28
        }
]
}
";
        await Train();
        await Status();
    }
}

```

```

        'entityName': 'Size',
        'startCharIndex': 8,
        'endCharIndex': 12
    }
]
},
{
    'text': 'I want two large pepperoni pizzas on thin
crust',
    'intentName': 'ModifyOrder',
    'entityLabels': [
        {
            'entityName': 'Order',
            'startCharIndex': 7,
            'endCharIndex': 46
        },
        {
            'entityName': 'FullPizzaWithModifiers',
            'startCharIndex': 7,
            'endCharIndex': 46
        },
        {
            'entityName': 'PizzaType',
            'startCharIndex': 17,
            'endCharIndex': 32
        },
        {
            'entityName': 'Size',
            'startCharIndex': 11,
            'endCharIndex': 15
        },
        {
            'entityName': 'Quantity',
            'startCharIndex': 7,
            'endCharIndex': 9
        },
        {
            'entityName': 'Crust',
            'startCharIndex': 37,
            'endCharIndex': 46
        }
    ]
}
";
}

AddUtterances(utterances).Wait();
Train().Wait();
Status().Wait();
}
}
}

```

4. Replace the values starting with `YOUR-` with your own values.

Information	Purpose
YOUR-APP-ID	Your LUIS app ID.
YOUR-AUTHORING-KEY	Your 32 character authoring key.
YOUR-AUTHORING-ENDPOINT	Your authoring URL endpoint. For example, https://replace-with-your-resource-name.api.cognitive.microsoft.com/ . You set your resource name when you created the resource.

Assigned keys and resources are visible in the LUIS portal in the Manage section, on the **Azure resources** page. The app ID is available in the same Manage section, on the **Application Settings** page.

ⓘ Important

Remember to remove the key from your code when you're done, and never post it publicly. For production, use a secure way of storing and accessing your credentials like [Azure Key Vault](#). See the Azure AI services [security](#) article for more information.

5. Build the console application.

Console

```
dotnet build
```

6. Run the console application.

Console

```
dotnet run
```

7. Review the authoring response:

Console

```
Added utterances.
[
  {
    "value": {
      "ExampleId": 1137150691,
      "UtteranceText": "order a pizza"
    },
]
```

```
        "hasError": false
    },
    {
        "value": {
            "ExampleId": 1137150692,
            "UtteranceText": "order a large pepperoni pizza"
        },
        "hasError": false
    },
    {
        "value": {
            "ExampleId": 1137150693,
            "UtteranceText": "i want two large pepperoni pizzas on thin
crust"
        },
        "hasError": false
    }
]
Sent training request.
{
    "statusId": 9,
    "status": "Queued"
}
Requested training status.
[
    {
        "modelId": "edb46abf-0000-41ab-beb2-a41a0fe1630f",
        "details": {
            "statusId": 9,
            "status": "Queued",
            "exampleCount": 0
        }
    },
    {
        "modelId": "a5030be2-616c-4648-bf2f-380fa9417d37",
        "details": {
            "statusId": 9,
            "status": "Queued",
            "exampleCount": 0
        }
    },
    {
        "modelId": "3f2b1f31-a3c3-4fbe-8182-e9d9dbc120b9",
        "details": {
            "statusId": 9,
            "status": "Queued",
            "exampleCount": 0
        }
    },
    {
        "modelId": "e4b6704b-1636-474c-9459-fe9ccbeba51c",
        "details": {
            "statusId": 9,
            "status": "Queued",
            "exampleCount": 0
        }
    }
]
```

```
        }
    },
{
    "modelId": "031d3777-2a00-4a7a-9323-9a3280a30000",
    "details": {
        "statusId": 9,
        "status": "Queued",
        "exampleCount": 0
    }
},
{
    "modelId": "9250e7a1-06eb-4413-9432-ae132ed32583",
    "details": {
        "statusId": 9,
        "status": "Queued",
        "exampleCount": 0
    }
}
]
```

Clean up resources

When you are finished with this quickstart, delete the project folder from the file system.

Next steps

[Best practices for an app](#)

How to query the prediction runtime with user text

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

To understand what a LUIS prediction endpoint returns, view a prediction result in a web browser.

Prerequisites

In order to query a public app, you need:

- Your Language Understanding (LUIS) resource information:
 - **Prediction key** - which can be obtained from [LUIS Portal](#). If you do not already have a subscription to create a key, you can register for a [free account](#).
 - **Prediction endpoint subdomain** - the subdomain is also the **name** of your LUIS resource.
- A LUIS app ID - use the public IoT app ID of `df67dcdb-c37d-46af-88e1-8b97951ca1c2`. The user query used in the quickstart code is specific to that app. This app should work with any prediction resource other than the Europe or Australia regions, since it uses "westus" as the authoring region.

Use the browser to see predictions

1. Open a web browser.
2. Use the complete URLs below, replacing `YOUR-KEY` with your own LUIS Prediction key. The requests are GET requests and include the authorization, with your LUIS Prediction key, as a query string parameter.

V3 prediction request

The format of the V3 URL for a **GET** endpoint (by slots) request is:

```
https://YOUR-LUIS-ENDPOINT-
SUBDOMAIN.api.cognitive.microsoft.com/luis/prediction/v3.0/apps/df67dcdb-
c37d-46af-88e1-8b97951ca1c2/slots/production/predict?query=turn on all
lights&subscription-key=YOUR-LUIS-PREDICTION-KEY
```

3. 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.Operation` entity with the value `on`.

V3 prediction response

JSON

```
{
  "query": "turn on all lights",
  "prediction": {
    "topIntent": "HomeAutomation.TurnOn",
    "intents": {
      "HomeAutomation.TurnOn": {
        "score": 0.5375382
      }
    },
    "entities": {
      "HomeAutomation.Operation": [
        "on"
      ]
    }
  }
}
```

4. To see all the intents, add the appropriate query string parameter.

V3 prediction endpoint

Add `show-all-intents=true` to the end of the querystring to **show all intents**, and `verbose=true` to return all detailed information for entities.

```
https://YOUR-LUIS-ENDPOINT-
SUBDOMAIN.api.cognitive.microsoft.com/luis/prediction/v3.0/apps/df67dcdb-
c37d-46af-88e1-8b97951ca1c2/slots/production/predict?query=turn on all
lights&subscription-key=YOUR-LUIS-PREDICTION-KEY&show-all-intents=true&verbose=true
```

```
lights&subscription-key=YOUR-LUIS-PREDICTION-KEY&show-all-intents=true&verbose=true
```

JSON

```
{  
    "query": "turn off the living room light",  
    "prediction": {  
        "topIntent": "HomeAutomation.TurnOn",  
        "intents": {  
            "HomeAutomation.TurnOn": {  
                "score": 0.5375382  
            },  
            "None": {  
                "score": 0.08687421  
            },  
            "HomeAutomation.TurnOff": {  
                "score": 0.0207554  
            }  
        },  
        "entities": {  
            "HomeAutomation.Operation": [  
                "on"  
            ]  
        }  
    }  
}
```

Next steps

- V3 prediction endpoint
- Custom subdomains
- Use the client libraries or REST API

How to use the Dashboard to improve your app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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	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.

Issue	Chart color	Explanation
Incorrect predictions	Red	This occurs when an example utterance is not predicted for the labeled intent (the intent it is in).

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.

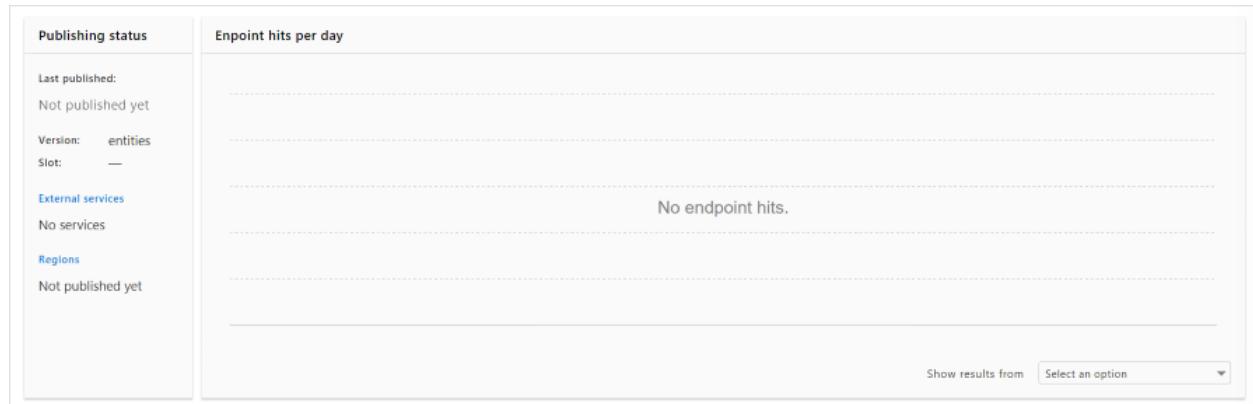
Before app is trained

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.



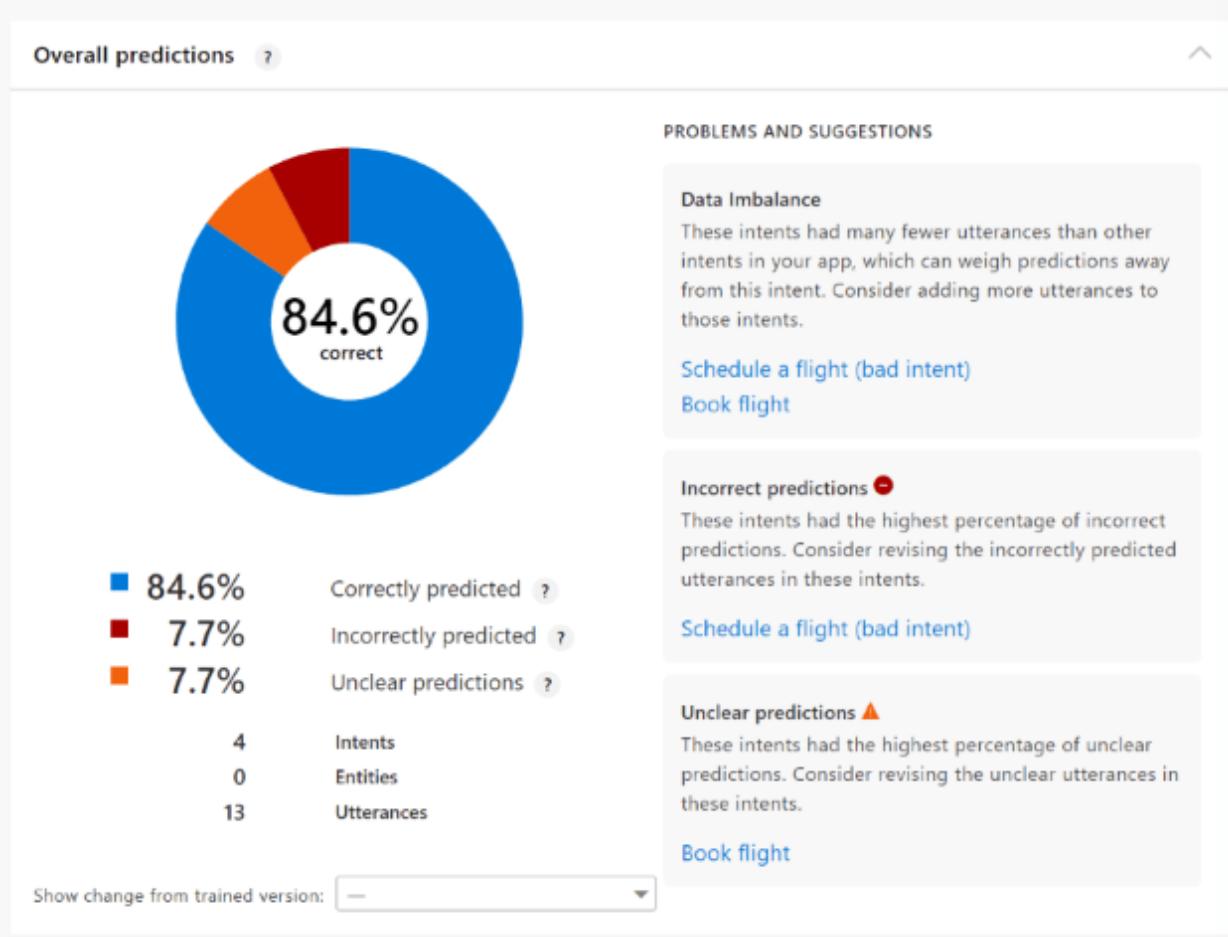
This also shows any external services, published regions, and aggregated endpoint hits.

Review training evaluation

The **Training evaluation** card contains the aggregated summary of your app's overall accuracy by area. The score indicates intent quality.

Training evaluation ?

Active version: dataimbal – trained Apr 24, 2019, 5:44:58 AM



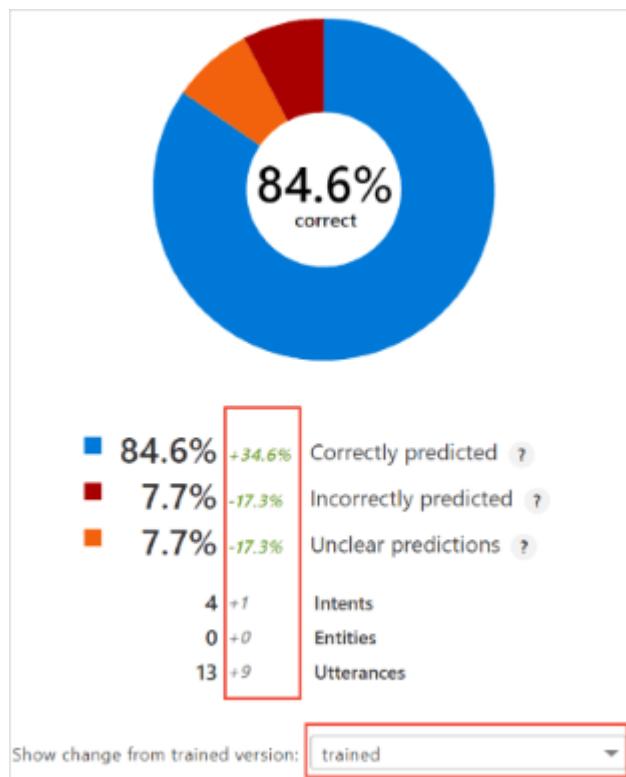
The chart indicates the correctly predicted intents and the problem areas with different colors. As you improve the app with the suggestions, this score increases.

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. If this is the case, the different data must be 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.

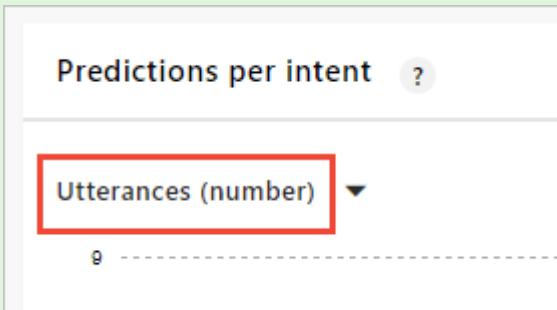
To fix this issue:

- 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.



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](#).

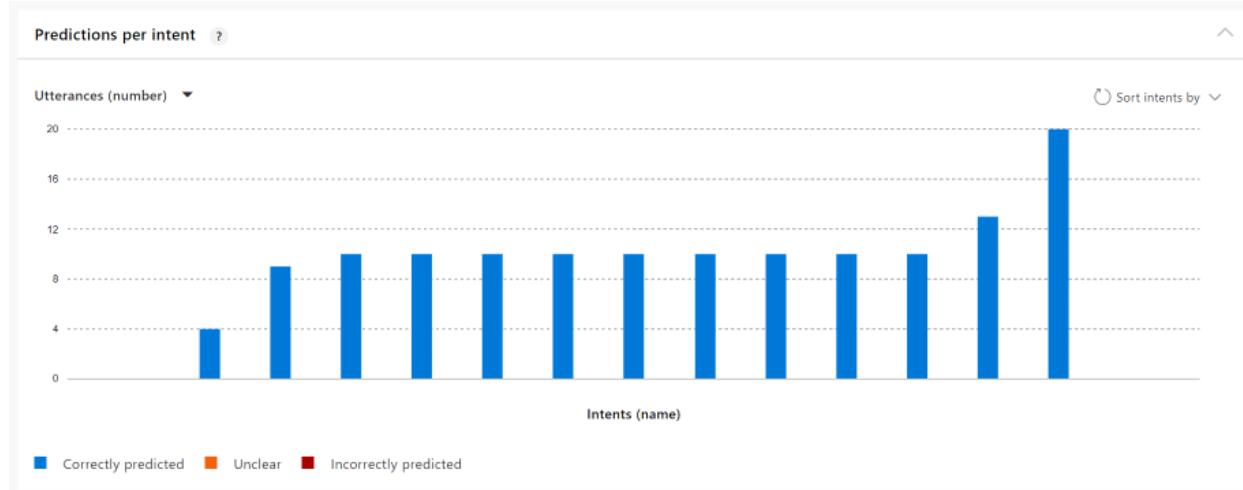
To fix this issue:

- Edit utterances to be more specific to the intent and train again.
- Combine intents if utterances are too closely aligned and train again.

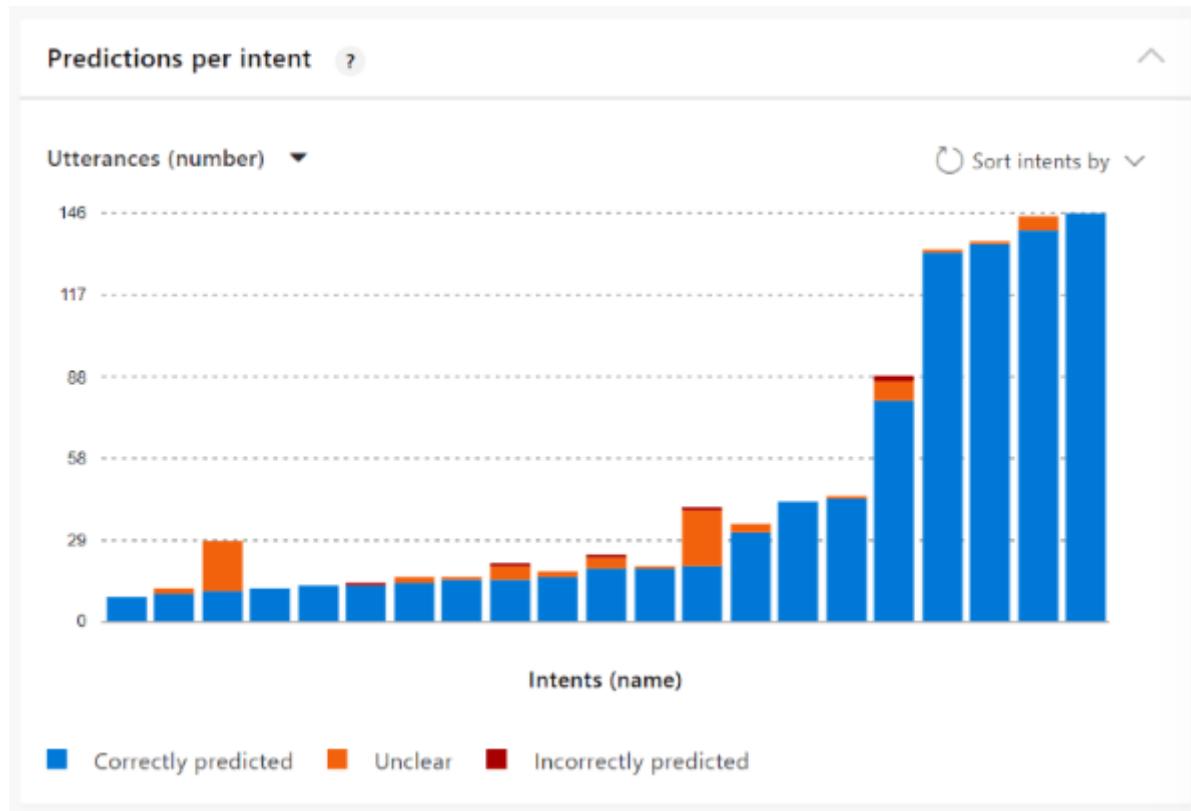
Utterances per intent

This card shows the overall app health across the intents. As you fix intents and retrain, continue to glance at this card for issues.

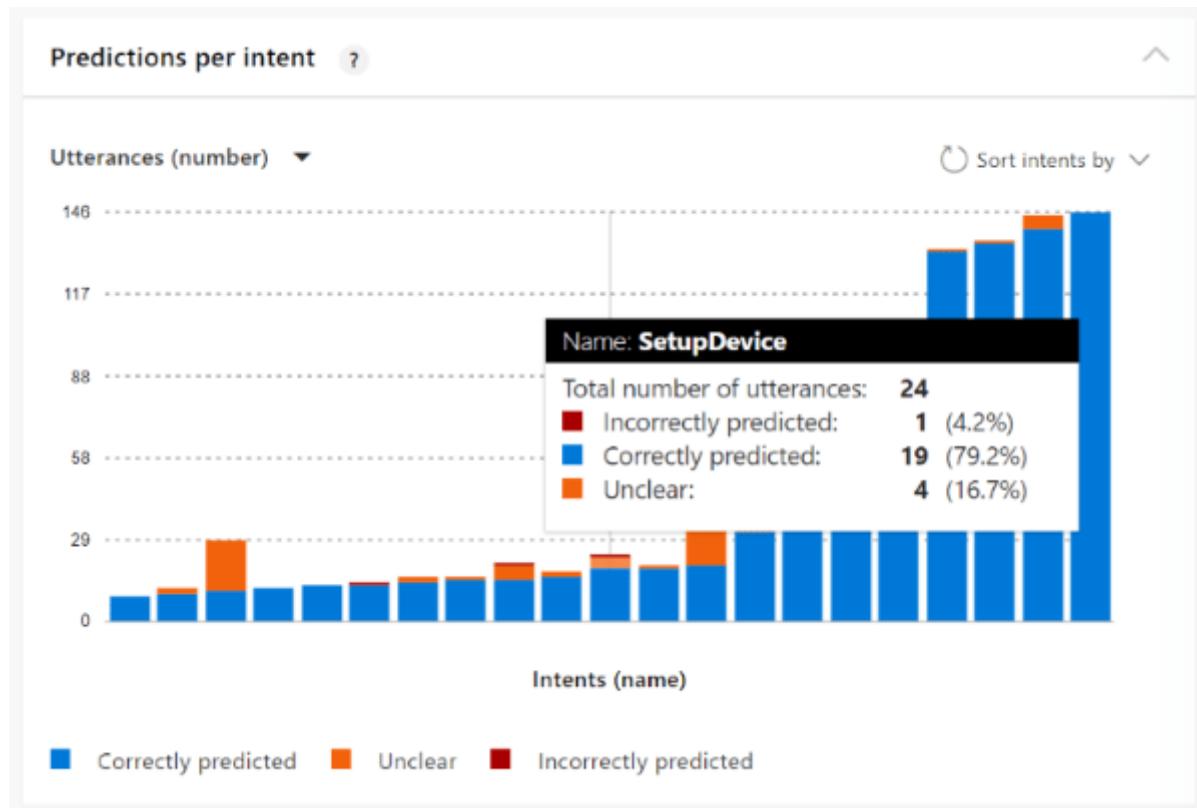
The following chart shows a well-balanced app with almost no issues to fix.



The following chart shows a poorly balanced app with many issues to fix.



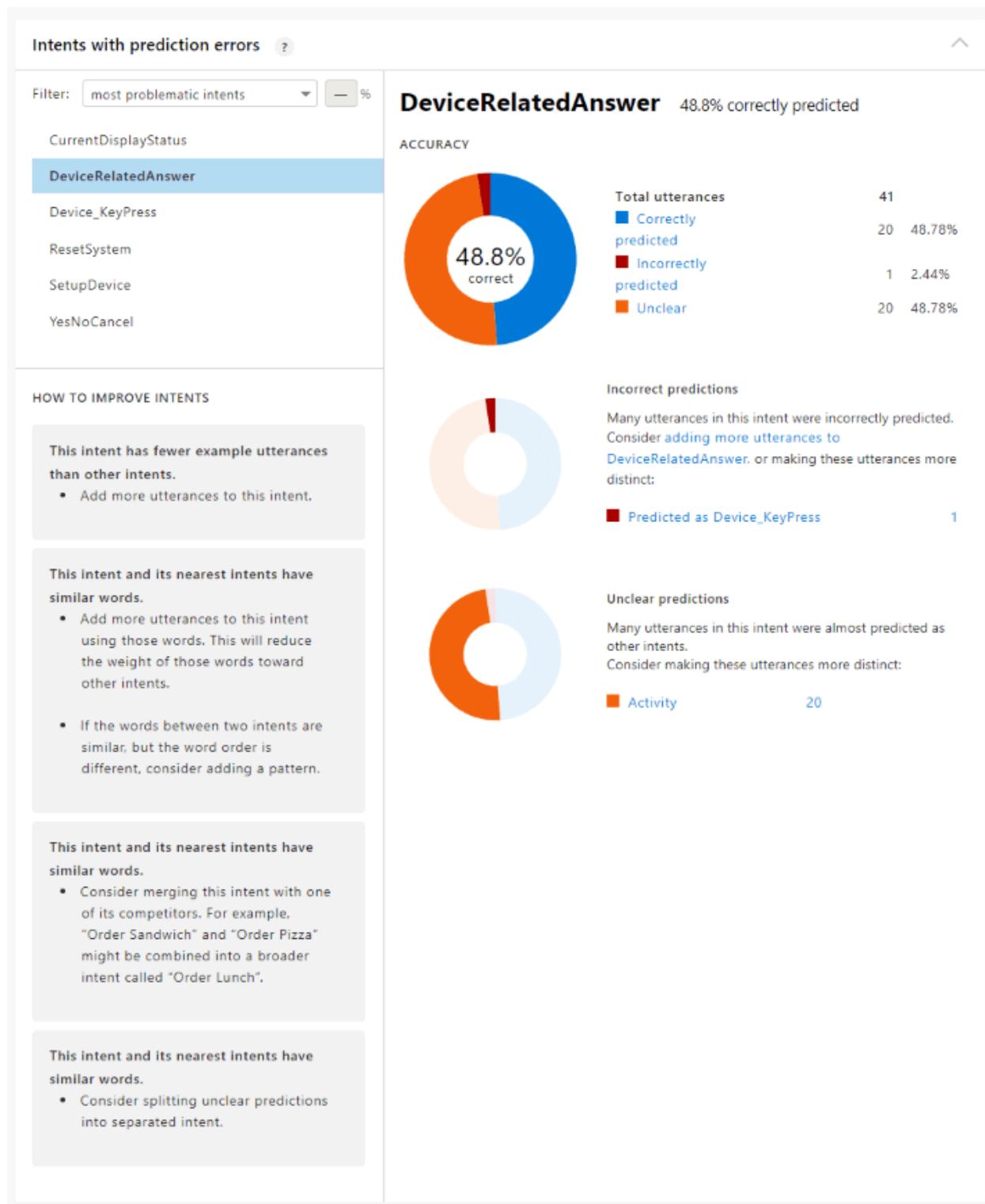
Hover over each intent's bar to get information about the intent.



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.
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](#)

How to improve a LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Use this article to learn how you can improve your LUIS apps, such as reviewing for correct predictions, and working with optional text in utterances.

Active Learning

The process of reviewing endpoint utterances for correct predictions is called Active learning. Active learning captures queries that are sent to the endpoint, and selects user utterances that it is unsure of. You review these utterances to select the intent and mark the entities for these real-world utterances. Then you can accept these changes into your app's example utterances, then [train](#) and [publish](#) the app. This helps LUIS identify utterances more accurately.

Log user queries to enable active learning

To enable active learning, you must log user queries. This is accomplished by calling the [endpoint query](#) with the `log=true` query string parameter and value.

ⓘ Note

To disable active learning, don't log user queries. You can change the query parameters by setting `log=false` in the endpoint query or omit the `log` parameter because the default value is `false` for the V3 endpoint.

Use the LUIS portal to construct the correct endpoint query.

1. Sign in to the [LUIS portal](#), and select your **Subscription** and **Authoring resource** to see the apps assigned to that authoring resource.
2. Open your app by selecting its name on **My Apps** page.

3. Go to the **Manage** section, then select **Azure resources**.
4. For the assigned prediction resource, select **Change query parameters**

Cognitive Services Language Understanding - westus

DASHBOARD BUILD MANAGE Train Test Publish

Azure Resources

Prediction Resources Authoring Resource

Add prediction resource

BaherLuisTest

Un-assign key

Resource group: XXXX
Location: westus2
Primary Key: XXXXXXXXXXXXXXXX
Secondary Key: XXXXXXXXXXXXXXXX
Endpoint URL: https://AuthoringResource.cognitiveservices.azure.com/
Pricing Tier: F0 (Free)
Example Query: https://AuthoringResource.cognitiveservices.azure.com/luis/prediction/v3.0/apps/APPID/slots/production/predict?verbose=true&show-all-intents=true&log=true&subscription-key=<Subscription-Key>&query=YOUR_QUERY_HERE

Change query parameters

5. Toggle **Save logs** then save by selecting **Done**.

Change request parameters for this endpoint url X

[Learn more about LUIS request parameters.](#)

Publishing slot

Production

Include scores for all intents

On

Include more entities details

On

Save logs

On

Enable spell check ⓘ

Off Enter your spell check key

Done **Cancel**

This action changes the example URL by adding the `log=true` query string parameter. Copy and use the changed example query URL when making prediction queries to the runtime endpoint.

Correct predictions to align utterances

Each utterance has a suggested intent displayed in the **Predicted Intent** column, and the suggested entities in dotted bounding boxes.

The screenshot shows the 'Review Endpoint Utterances' page. On the left, there's a sidebar with navigation links: App Assets, Intents, Entities, Prebuilt Domains, Improve app performance, Review endpoint utterances (which is selected), Features, and Patterns. The main area has a title 'Review Endpoint Utterances' with a help icon. Below it is a 'Filter' dropdown set to 'CancelOrder'. There are buttons for 'Confirm all entity predictions' (with a checkmark), 'Save', and 'Discard'. A table lists three utterances:

Utterance	Predicted Intent
" pick up 3 hawaiian pizzas and 2 double crust cheese pizzas with a side of bread sticks "	ModifyOrder (1)
" give me pizza "	ModifyOrder (0.022057...)
" i want pizza "	ModifyOrder (0.0587631)

Each utterance row contains a detailed breakdown of entities with dotted boxes around them. For the first utterance, the breakdown is:

Order
FullPizzaWithModifiers
Q. PizzaType
3 hawaiian pizzas and 2 double crust cheese pizzas with a side of bread sticks

For the third utterance, the breakdown is:

Order
FullPizzaWithModifiers
Size
Q. PizzaType
ToppingsModifiers
large cheese pizza with extra sauce

On the right side of the table, there's a large circular button with a magnifying glass and a plus sign (+).

If you agree with the predicted intent and entities, select the check mark next to the utterance. If the check mark is disabled, this means that there is nothing to confirm. If you disagree with the suggested intent, select the correct intent from the predicted intent's drop-down list. If you disagree with the suggested entities, start labeling them. After you are done, select the check mark next to the utterance to confirm what you labeled. Select **save utterance** to move it from the review list and add it its respective intent.

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 it.

Working with optional text and prebuilt entities

Suppose you have a Human Resources app that handles queries about an organization's personnel. It might allow for current and future dates in the utterance text - text that uses `s`, `'s`, and `?`.

If you create an "*OrganizationChart*" intent, you might consider the following example utterances:

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"
- A date: "March 3", "now", "in a month"

LUIS needs these to make predictions correctly. Notice that the last two examples in the table use almost the same text except for "in" and "on".

Using patterns, the following example template utterances would allow for optional information:

Intent	Example utterances with optional text and prebuilt entities
OrgChart-Manager	Who was {EmployeeListEntity}['s] manager [[on]{datetimeV2}?)
OrgChart-Manager	Who is {EmployeeListEntity}['s] manager [[on]{datetimeV2}?)

The optional square brackets syntax "[]" lets you add optional text to the template utterance and can be nested in a second level "[[]]" and include entities or text.

⊗ Caution

Remember that entities are found first, then the pattern is matched.

Next Steps:

To test how performance improves, 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

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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.

ⓘ Note

- After you add, edit, remove, or reassign a pattern, [train](#) and [publish](#) your app for your changes to affect endpoint queries.
- Patterns only include machine-learning entity parents, not subentities.

Add template utterance using correct syntax

1. Sign in to the [LUIS portal](#) , and select your **Subscription** and **Authoring resource** to see the apps assigned to that authoring resource.
2. Open your app by selecting its name on **My Apps** page.
3. Select **Patterns** in the left panel, under **Improve app performance**.
4. Select the correct intent for the pattern.
5. 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.

Patterns ?

ModifyOrder ▾

Pick up a {}

Pattern	Entities
	CrustList
	ModifierList
	number
	Order
	ScopeList
	SizeList
	ToppingList

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.

Patterns ?

ModifyOrder ▾

Pick up from {Deliver:

Roles

Deliver:Origin

Deliver:Destination

After you select the correct entity, finish entering the pattern, and then select Enter. When you are done entering patterns, [train](#) your app.

Pattern

pick up from `Deliver:Origin` and deliver to `Deliver:Destination`

Create a pattern.any entity

`Pattern.any` entities are only valid in [patterns](#), not intents' example utterances. 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.

1. Sign in to the [LUIS portal](#), and select your **Subscription** and **Authoring resource** to see the apps assigned to that authoring resource.
2. Open your app by selecting its name on **My Apps** page.
3. From the **Build** section, select **Entities** in the left panel, and then select **+ Create**.
4. In the **Choose an entity type** dialog box, enter the entity name in the **Name** box, and select **Pattern.Any** as the **Type** then select **Create**.

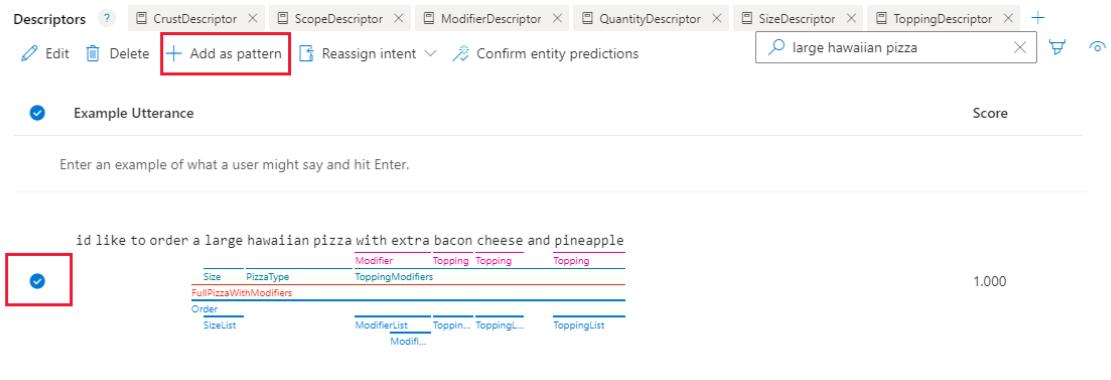
Once you [create a pattern utterance](#) using this entity, the entity is extracted with a combined machine-learning and text-matching algorithm.

Adding example utterances as pattern

If you want to add a pattern for an entity, the *easiest* way is to create the pattern from the Intent details page. This ensures your syntax matches the example utterance.

1. Sign in to the [LUIS portal](#), and select your **Subscription** and **Authoring resource** to see the apps assigned to that authoring resource.
2. Open your app by selecting its name on **My Apps** page.
3. On the **Intents** list page, select the intent name of the example utterance you want to create a template utterance from.
4. On the Intent details page, select the row for the example utterance you want to use as the template utterance, then select **+ Add as pattern** from the context toolbar.

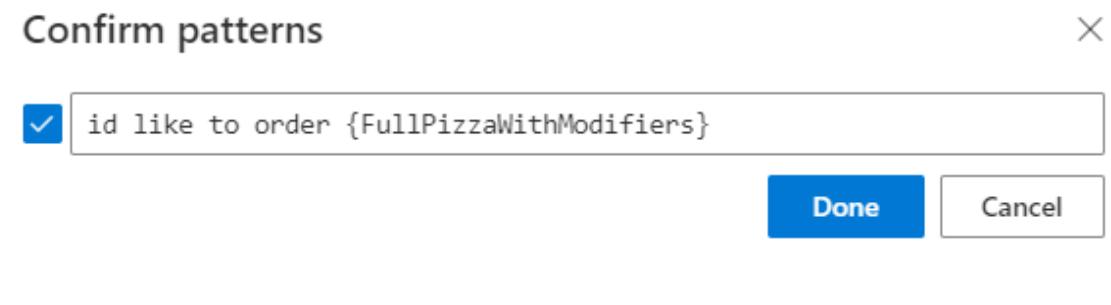
ModifyOrder



The screenshot shows the 'ModifyOrder' interface. At the top, there's a navigation bar with 'Descriptors' and several entity types like 'CrustDescriptor', 'ScopeDescriptor', etc. Below the navigation is a toolbar with 'Edit', 'Delete', '+ Add as pattern' (which is highlighted with a red box), 'Reassign intent', and 'Confirm entity predictions'. A search bar contains the text 'large hawaiian pizza'. Underneath, there's a section for 'Example Utterance' with a placeholder 'Enter an example of what a user might say and hit Enter.' Below this is a table showing a sample utterance: 'id like to order a large hawaiian pizza with extra bacon cheese and pineapple'. The table has columns for 'Size', 'PizzaType', 'Modifier', 'Topping', 'Topping', and 'Topping'. The 'Modifier' column contains 'FullPizzaWithModifiers'. The table is annotated with red boxes: one around the first column header 'Size', one around the 'Modifier' column header, and one around the first row of the table itself. To the right of the table is a 'Score' of '1.000'. At the bottom of the interface is a footer with links like 'Entity types', 'Entity patterns', 'Entity samples', 'Entity training', and 'Entity metrics'.

The utterance must include an entity in order to create a pattern from the utterance.

5. In the pop-up box, select **Done** on the **Confirm patterns** page. You don't need to define the entities' subentities, or features. You only need to list the machine-learning entity.



6. If you need to edit the template, such as selecting text as optional, with the [] (square) brackets, you need to make this edit from the **Patterns** page.
7. In the navigation bar, select **Train** to train the app with the new pattern.

Use the OR operator and groups

The following two patterns can be combined into a single pattern using the group "()" and OR "|" syntax.

Intent	Example utterances with optional text and prebuilt entities
OrgChart-Manager	"who will be {EmployeeListEntity}['s] manager [[in]{datetimeV2}?]"
OrgChart-Manager	"who will be {EmployeeListEntity}['s] manager [[on]{datetimeV2}?]"

The new template utterance will be:

"who (was | is | will be) {EmployeeListEntity}'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.

Template utterances

Due to the nature of the Human Resource subject domain, there are a few common ways of asking about employee relationships in organizations. Such as the following 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 needing to supply several more utterance examples.

💡 Tip

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.

Template utterance examples for this intent would include:

Template utterances examples	syntax meaning
Who does {EmployeeListEntity} report to[?]	interchangeable: {EmployeeListEntity} ignore: [?]
Who reports to {EmployeeListEntity}[?]	interchangeable: {EmployeeListEntity} ignore: [?]

The "*{EmployeeListEntity}*" syntax marks the entity location within the template utterance and which entity it is. The optional syntax, "[?]", marks words or **punctuation** that is optional. LUIS matches the utterance, ignoring the optional text inside the brackets.

ⓘ Important

While the syntax looks like a regular expression, it is not a regular expression. Only the curly bracket, "{}", and square bracket, "[]", syntax is supported. They can be

nested up to two levels.

For a pattern to be matched to an utterance, *first* the entities within the utterance must match the entities in the template utterance. This means the entities need to have enough examples in example utterances with a high degree of prediction before patterns with entities are successful. The template doesn't help predict entities, however. The template only predicts intents.

ⓘ Note

While patterns allow you to provide fewer example utterances, if the entities are not detected, the pattern will not match.

Add phrase list as a feature

Features help LUIS by providing hints that certain words and phrases are part of an app domain vocabulary.

1. Sign in to the [LUIS portal](#), and select your **Subscription** and **Authoring resource** to see the apps assigned to that authoring resource.
2. Open your app by selecting its name on **My Apps** page.
3. Select **Build**, then select **Features** in your app's left panel.
4. On the **Features** page, select **+ Create**.
5. In the **Create new phrase list feature** dialog box, enter a name such as **Pizza Toppings**. In the **Value** box, enter examples of toppings, such as *Ham*. You can type one value at a time, or a set of values separated by commas, and then press **Enter**.

Create new phrase list feature ×

Global On

Name *

Values *
 Search

These values are interchangeable

Create Cancel

6. Keep the **These values are interchangeable** selector enabled if the phrases can be used interchangeably. The interchangeable phrase list feature serves as a list of synonyms for training. Non-interchangeable phrase lists serve as separate features for training, meaning that features are similar but the intent changes when you swap phrases.
7. The phrase list can apply to the entire app with the **Global** setting, or to a specific model (intent or entity). If you create the phrase list as a *feature* from an intent or entity, the toggle is not set to global. In this case, the toggle specifies that the feature is local only to that model, therefore, *not global* to the application.
8. Select **Done**. The new feature is added to the **ML Features** page.

① Note

- You can delete, or deactivate a phrase list from the contextual toolbar on the **ML Features** page.
- A phrase list should be applied to the intent or entity it is intended to help but there may be times when a phrase list should be applied to the entire app as a **Global** feature. On the **Machine Learning** Features page, select the phrase list, then select **Make global** in the top contextual toolbar.

Add entity as a feature to an intent

To add an entity as a feature to an intent, select the intent from the Intents page, then select **+ Add feature** above the contextual toolbar. The list will include all phrase lists and entities that can be applied as features.

To add an entity as a feature to another entity, you can add the feature either on the Intent detail page using the [Entity Palette](#) or you can add the feature on the Entity detail page.

Next steps

- [Train and test your app after improvement.](#)

Correct misspelled words with Bing Resource

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

The V3 prediction API now supports the [Bing Spellcheck API](#). Add spell checking to your application by including the key to your Bing search resource in the header of your requests. You can use an existing Bing resource if you already own one, or [create a new one](#) to use this feature.

Prediction output example for a misspelled query:

JSON

```
{  
  "query": "bouk me a fliht to kayro",  
  "prediction": {  
    "alteredQuery": "book me a flight to cairo",  
    "topIntent": "book a flight",  
    "intents": {  
      "book a flight": {  
        "score": 0.9480589  
      }  
      "None": {  
        "score": 0.0332136229  
      }  
    },  
    "entities": {}  
  }  
}
```

Corrections to spelling are made before the LUIS user utterance prediction. You can see any changes to the original utterance, including spelling, in the response.

Create Bing Search Resource

To create a Bing Search resource in the Azure portal, follow these instructions:

1. Log in to the [Azure portal](#).
2. Select **Create a resource** in the top left corner.
3. In the search box, enter **Bing Search v7** and select the service.
4. An information panel appears to the right containing information including the Legal Notice. Select **Create** to begin the subscription creation process.

The screenshot shows the Azure portal interface for the Bing Search v7 service. At the top, there's a header with the service name 'Bing Search v7' and a Microsoft logo. Below the header, there's a large blue icon of a magnifying glass. To the right of the icon, the service name 'Bing Search v7' is displayed again with a 'Microsoft' badge underneath. A blue 'Create' button is prominently featured. Below these elements, there are navigation links: 'Overview' (which is underlined, indicating it's the current page), 'Plans', 'Usage Information + Support', and 'Reviews'. The main content area contains descriptive text about the Bing Search APIs v7, mentioning intelligent search across billions of webpages, images, videos, and news. It highlights features like location-based customization, relevancy, and safe search levels. A 'Legal Notice' section follows, which includes a note from Microsoft about data usage and a link to 'Search Services Terms'. The overall layout is clean and professional, typical of the Azure developer portal.

5. In the next panel, enter your service settings. Wait for service creation process to finish.
6. After the resource is created, go to the **Keys and Endpoint** blade on the left.
7. Copy one of the keys to be added to the header of your prediction request. You will only need one of the two keys.

Adding the key to the endpoint URL

For each query you want to apply spelling correction on, the endpoint query needs the Bing Spellcheck resource key passed in the query header parameter. 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 in the header's request for spelling corrections to work properly. You must set the value with **mkt-bing-spell-check-key** to the key value.

Header Key	Header Value
mkt-bing-spell-check-key	Keys found in Keys and Endpoint blade of your resource

Send misspelled utterance to LUIS

1. Add a misspelled utterance in the prediction query you will be sending such as "How far is the mountainn?". In English, `mountain`, with one `n`, is the correct spelling.
2. 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.

JSON

```
{
  "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 Search API v7 service, you need to add the correct and incorrect spelling.

Two solutions are:

- Label example utterances that have all the different spellings so that LUIS can learn proper spelling as well as typos. This option requires more labeling effort than using a spell checker.
- Create a phrase list with all variations of the word. With this solution, you do not need to label the word variations in the example utterances.

Next steps

[Learn more about example utterances](#)

How to recognize intents from speech using the Speech SDK for C#

Article • 07/18/2023

The Azure AI 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've 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 guide, 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 configuration 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 items before you begin this guide:

- A LUIS account. You can get one for free through the [LUIS portal](#).
- [Visual Studio 2019](#) (any edition).

LUIS and speech

LUIS integrates with the Speech service to recognize intents from speech. You don't need a Speech service subscription, just LUIS.

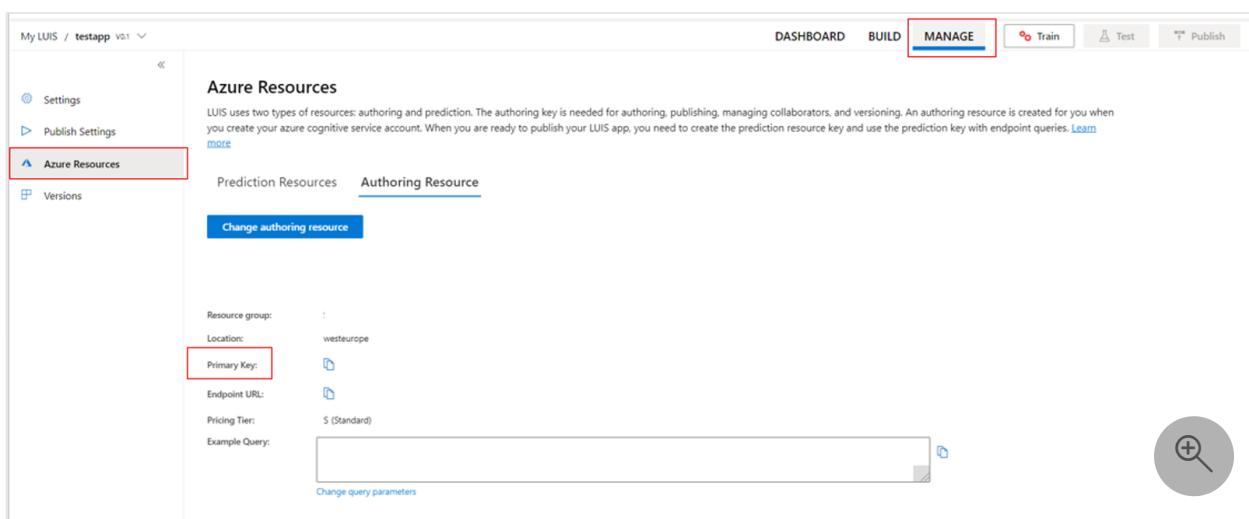
LUIS uses two kinds of keys:

Key type	Purpose
Authoring	Lets you create and modify LUIS apps programmatically
Prediction	Used to access the LUIS application in runtime

For this guide, you need the prediction key type. This guide uses the example Home Automation LUIS app, which you can create by following the [Use prebuilt Home automation app](#) quickstart. If you've created a LUIS app of your own, you can use it instead.

When you create a LUIS app, LUIS automatically generates an authoring key so you can test the app using text queries. This key doesn't enable the Speech service integration and won't work with this guide. Create a LUIS resource in the Azure dashboard and assign it to the LUIS app. You can use the free subscription tier for this guide.

After you create 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, select **Azure Resources** in the sidebar.



On the **Azure Resources** page:

Select the icon next to a key to copy it to the clipboard. (You may use either key.)

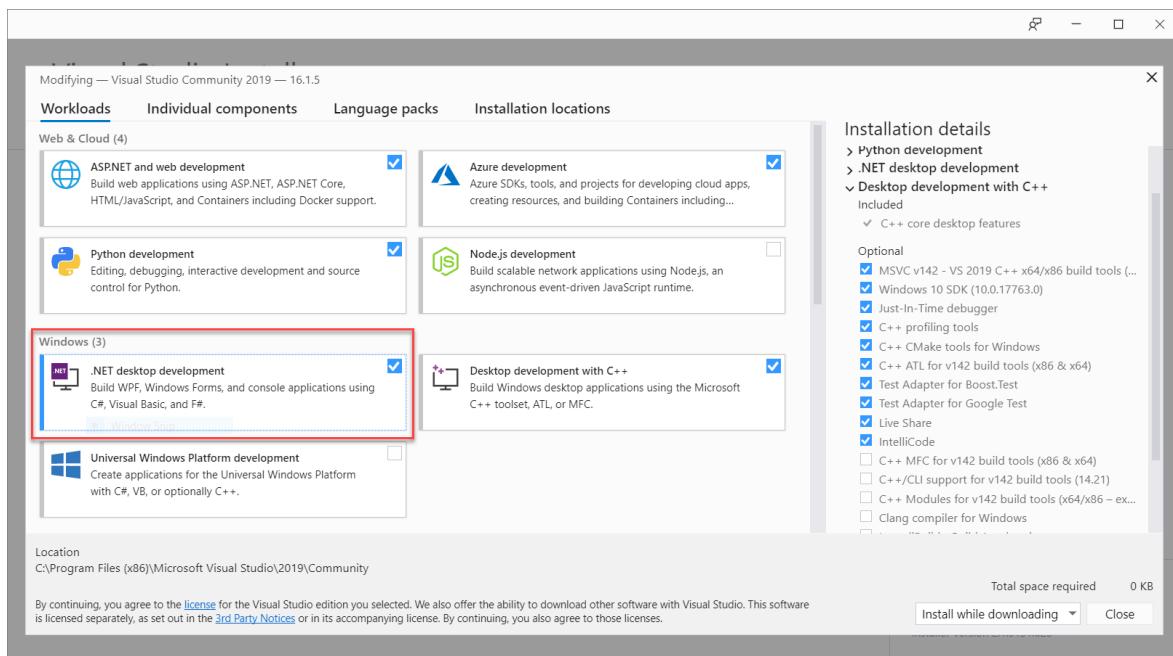
Create the project and add the workload

To create a Visual Studio project for Windows development, you need to create the project, set up Visual Studio for .NET desktop development, install the Speech SDK, and choose the target architecture.

To start, create the project in Visual Studio, and make sure that Visual Studio is set up for .NET desktop development:

1. Open Visual Studio 2019.
2. In the Start window, select **Create a new project**.
3. In the **Create a new project** window, choose **Console App (.NET Framework)**, and then select **Next**.
4. In the **Configure your new project** window, enter *helloworld* in **Project name**, choose or create the directory path in **Location**, and then select **Create**.
5. From the Visual Studio menu bar, select **Tools > Get Tools and Features**, which opens Visual Studio Installer and displays the **Modifying** dialog box.
6. Check whether the **.NET desktop development** workload is available. If the workload hasn't been installed, select the check box next to it, and then select **Modify** to start the installation. It may take a few minutes to download and install.

If the check box next to **.NET desktop development** is already selected, select **Close** to exit the dialog box.

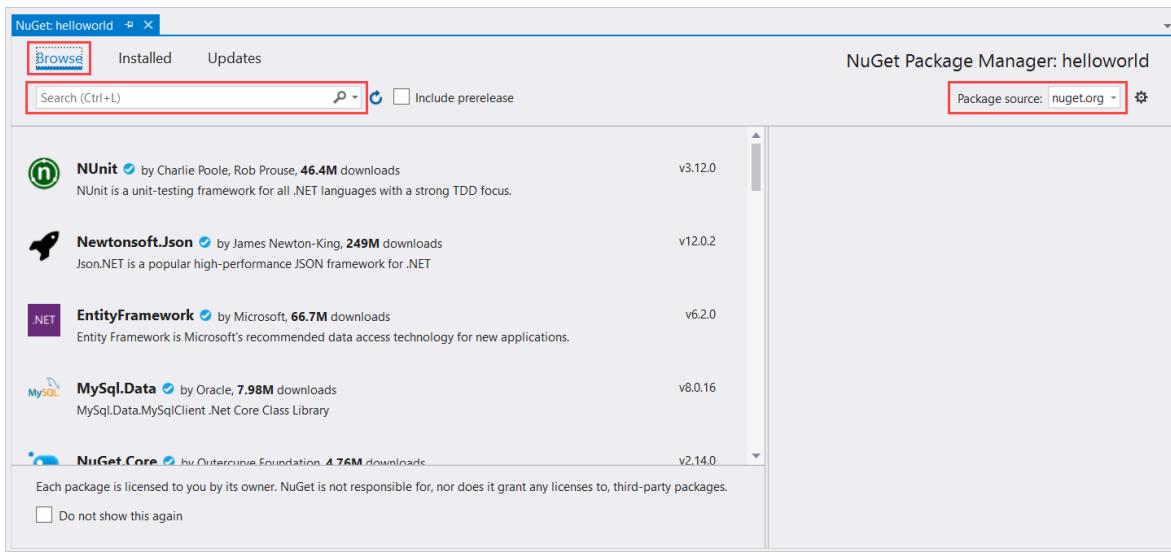


7. Close Visual Studio Installer.

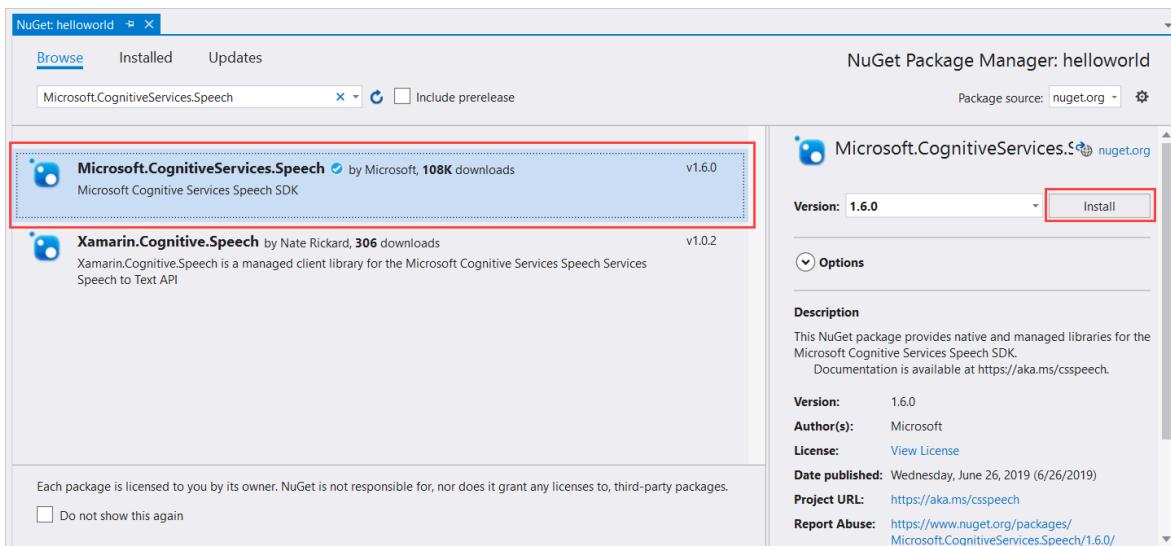
Install the Speech SDK

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 the **helloworld** project, 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* and select **Enter**.
5. From the search results, select the **Microsoft.CognitiveServices.Speech** package, and then select **Install** to install the latest stable version.



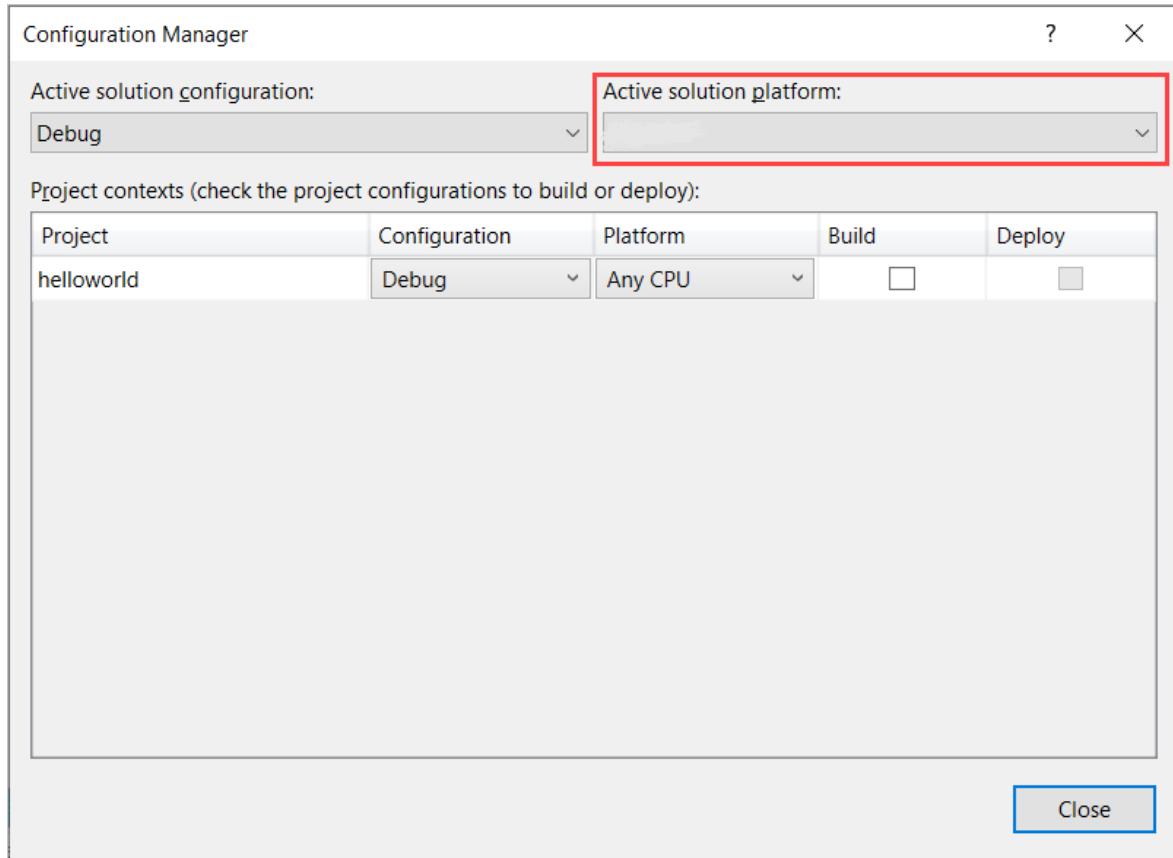
6. Accept all agreements and licenses to start the installation.

After the package is installed, a confirmation appears in the **Package Manager Console** window.

Choose the target architecture

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

Next, you add code to the project.

1. From Solution Explorer, open the file Program.cs.
2. Replace the block of `using` statements at the beginning of the file with the following declarations:

```
C#  
  
using System;  
using System.Threading.Tasks;
```

```
using Microsoft.CognitiveServices.Speech;
using Microsoft.CognitiveServices.Speech.Audio;
using Microsoft.CognitiveServices.Speech.Intent;
```

3. Replace the provided `Main()` method, with the following asynchronous equivalent:

C#

```
public static async Task Main()
{
    await RecognizeIntentAsync();
    Console.WriteLine("Please press Enter to continue.");
    Console.ReadLine();
}
```

4. Create an empty asynchronous method `RecognizeIntentAsync()`, as shown here:

C#

```
static async Task RecognizeIntentAsync()
{}
```

5. In the body of this new method, add this code:

C#

```
// 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($"      Language Understanding JSON:
{result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceR
esponse_JsonResult)}.");
    }
    else if (result.Reason == ResultReason.RecognizedSpeech)
    {
        Console.WriteLine($"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?" );
    }
}
}
```

6. Replace the placeholders in this method with your LUIS resource key, region, and app ID as follows.

Placeholder	Replace with
YourLanguageUnderstandingSubscriptionKey	Your LUIS resource key. Again, you must get this item from your Azure dashboard. You can find it on your app's Azure Resources page (under Manage) in the LUIS portal .
YourLanguageUnderstandingServiceRegion	The short identifier for the region your LUIS resource 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 in the LUIS portal .

With these changes made, you can build (**Control+Shift+B**) and run (**F5**) the application. When you're prompted, try saying "Turn off the lights" into your PC's microphone. The application displays the result in the console window.

The following sections include a discussion of the code.

Create an intent recognizer

First, you need to create a speech configuration from your LUIS prediction key and region. You can use speech configurations to create recognizers for the various capabilities of the Speech SDK. The speech configuration has multiple ways to specify the resource you want to use; here, we use `FromSubscription`, which takes the resource key and region.

Note

Use the key and region of your LUIS resource, not a Speech resource.

Next, create an intent recognizer using `new IntentRecognizer(config)`. Since the configuration already knows which resource to use, you don't need to specify the key 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. You don't have to add all the app's intents if you don't need to recognize them all in your application.

To add intents, you must provide 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.

<code>AddIntent()</code> argument	Purpose
<code>intentName</code>	The name of the intent as defined in the LUIS app. This value must match the LUIS intent name exactly.
<code>intentID</code>	An ID assigned to a recognized intent by the Speech SDK. This value can be whatever you like; it doesn't 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 off a device. The lines below add these intents to the recognizer; replace the three `AddIntent` lines in the `RecognizeIntentAsync()` method with this code.

C#

```
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 (for example, <code>IntermediateResultReceived</code>) when results are available.

The application uses single-shot mode and so calls `RecognizeOnceAsync()` to begin recognition. The result is an `IntentRecognitionResult` object containing information about the intent recognized. You extract the LUIS JSON response by using the following expression:

C#

```
result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResponse.JsonResult)
```

The application doesn't parse the JSON result. It only displays the JSON text in the console window.

```
Say something...
RECOGNIZED: Text=Hey turn off the light.
Intent Id: off.
Language Understanding JSON: {
  "query": "Hey turn off the light",
  "topScoringIntent": {
    "intent": "HomeAutomation.TurnOff",
    "score": 0.997960269
  },
  "entities": [
    {
      "entity": "light",
      "type": "HomeAutomation.DeviceType",
      "startIndex": 17,
      "endIndex": 21,
      "resolution": {
        "values": [
          "light"
        ]
      }
    }
  ]
}.
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 application before creating the recognizer to recognize intents in German.

For more information, see [LUIS language support](#).

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()` method to start recognition instead of `RecognizeOnceAsync()`.

The other capability is reading the audio containing the speech to be processed from a WAV file. Implementation 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, delete or comment out the body of the `RecognizeIntentAsync()` method, and add the following code in its place.

C#

```
// 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("YourAudioFile.wav"))
{
    using (var recognizer = new IntentRecognizer(config, audioInput))
    {
        // The TaskCompletionSource to stop recognition.
```

```

    var stopRecognition = new TaskCompletionSource<int>
(TaskCreationOptions.RunContinuationsAsynchronously);

        // 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($"    Language Understanding JSON:
{e.Result.Properties.GetProperty(PropertyId.LanguageUnderstandingServiceResp
onse_JsonResult)}.");
    }
    else if (e.Result.Reason == ResultReason.RecognizedSpeech)
    {
        Console.WriteLine($"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)
    {
        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.
    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 prediction key, region, and app ID and to add the Home Automation intents, as before. Change `whatstheweatherlike.wav` to the name of your recorded audio file. Then build, copy the audio file to the build directory, and run the application.

For example, if you say "Turn off the lights", pause, and then say "Turn on the lights" in your recorded audio file, console output similar to the following may appear:

```
Say something...
```

```
Session started event.  
RECOGNIZING: Text=turn  
RECOGNIZING: Text=turn off  
RECOGNIZING: Text=turn off the  
RECOGNIZING: Text=turn off the light  
RECOGNIZING: Text=turn off the lights  
RECOGNIZED: Text=Turn off the lights.  
    Intent Id: off.  
    Language Understanding JSON: {  
        "query": "turn off the lights",  
        "topScoringIntent": {  
            "intent": "HomeAutomation.TurnOff",  
            "score": 0.997679055  
        },  
        "entities": [  
            {  
                "entity": "lights",  
                "type": "HomeAutomation.DeviceType",  
                "startIndex": 13,  
                "endIndex": 18,  
                "resolution": {  
                    "values": [  
                        "light"  
                    ]  
                }  
            }  
        ]  
    }.  
RECOGNIZING: Text=turn  
RECOGNIZING: Text=turn on  
RECOGNIZING: Text=turn on the  
RECOGNIZING: Text=turn on the light  
RECOGNIZING: Text=turn on the lights  
RECOGNIZED: Text=Turn on the lights.  
    Intent Id: on.  
    Language Understanding JSON: {  
        "query": "turn on the lights",  
        "topScoringIntent": {  
            "intent": "HomeAutomation.TurnOn",  
            "score": 0.987454832  
        },  
        "entities": [  
            {  
                "entity": "lights",  
                "type": "HomeAutomation.DeviceType",  
                "startIndex": 12,  
                "endIndex": 17,  
                "resolution": {  
                    "values": [  
                        "light"  
                    ]  
                }  
            }  
        ]  
    }.  
NOMATCH: Speech could not be recognized.  
NOMATCH: Speech could not be recognized.  
CANCELED: Reason=EndOfStream  
  
Session stopped event.  
  
Stop recognition.  
Please press Enter to continue.
```

The Speech SDK team actively maintains a large set of examples in an open-source repository. For the sample source code repository, see the [Azure AI services Speech SDK](#)

on GitHub  . There are samples for C#, C++, Java, Python, Objective-C, Swift, JavaScript, UWP, Unity, and Xamarin. Look for the code from this article in the `samples/csharp/sharedcontent/console` folder.

Next steps

 [Quickstart: Recognize speech from a microphone](#)

Combine LUIS and question answering capabilities

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Azure AI services provides two natural language processing services, [Language Understanding](#) (LUIS) and question answering, each with a different purpose.

Understand when to use each service and how they complement each other.

Natural language processing (NLP) allows your client application, such as a chat bot, to work with your users' natural language.

When to use each feature

LUIS and question answering solve different problems. LUIS determines the intent of a user's text (known as an utterance), while question answering determines the answer to a user's text (known as a query).

To pick the correct service, you need to understand the user text coming from your client application, and what information it needs to get from the Azure AI service features.

As an example, if your chat bot receives the text "How do I get to the Human Resources building on the Seattle north campus", use the table 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 would be classified as matching a "FindLocation" intent.
Question answering	Returns the answer to the question from a custom knowledge base. For example, this text would be determined as a question, with the static text answer being "Get on the #9 bus and get off at Franklin street".

Create an orchestration project

Orchestration helps you connect more than one project and service together. Each connection in the orchestration is represented by a type and relevant data. The intent needs to have a name, a project type (LUIS, question answering, or conversational language understanding), and a project you want to connect to by name.

You can use orchestration workflow to create new orchestration projects. See [orchestration workflow](#) for more information.

Set up orchestration between Azure AI services features

To use an orchestration project to connect LUIS, question answering, and conversational language understanding, you need:

- A language resource in [Language Studio](#) or the Azure portal.
- To change your LUIS authoring resource to the Language resource. You can also optionally export your application from LUIS, and then [import it into conversational language understanding](#).

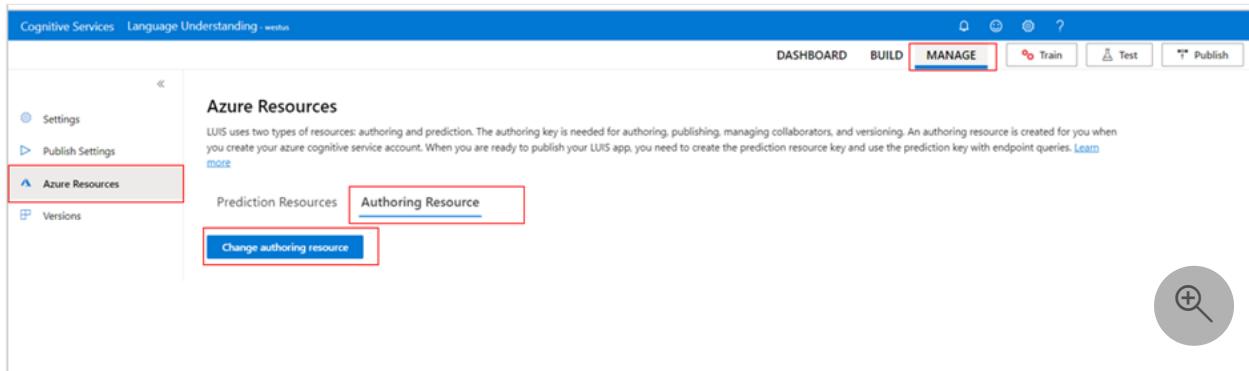
ⓘ Note

Luis can be used with Orchestration projects in West Europe only, and requires the authoring resource to be a Language resource. You can either import the application in the West Europe Language resource or change the authoring resource from the portal.

Change a LUIS resource to a language resource:

You need to follow the following steps to change LUIS authoring resource to a Language resource

1. Log in to the [LUIS portal](#).
2. From the list of LUIS applications, select the application you want to change to a Language resource.
3. From the menu at the top of the screen, select **Manage**.
4. From the left Menu, select **Azure resource**
5. Select **Authoring resource**, then change your LUIS authoring resource to the Language resource.



Next steps

- Conversational language understanding documentation

Migrate from Language Understanding (LUIS) to conversational language understanding (CLU)

Article • 07/18/2023

[Conversational language understanding \(CLU\)](#) is a cloud-based AI offering in Azure AI Language. It's the newest generation of [Language Understanding \(LUIS\)](#) and offers backwards compatibility with previously created LUIS applications. CLU employs state-of-the-art machine learning intelligence to allow users to build a custom natural language understanding model for predicting intents and entities in conversational utterances.

CLU offers the following advantages over LUIS:

- Improved accuracy with state-of-the-art machine learning models for better intent classification and entity extraction. LUIS required more examples to generalize certain concepts in intents and entities, while CLU's more advanced machine learning reduces the burden on customers by requiring significantly less data.
- Multilingual support for model learning and training. Train projects in one language and immediately predict intents and entities across 96 languages.
- Ease of integration with different CLU and [custom question answering](#) projects using [orchestration workflow](#).
- The ability to add testing data within the experience using Language Studio and APIs for model performance evaluation prior to deployment.

To get started, you can [create a new project](#) or [migrate your LUIS application](#).

Comparison between LUIS and CLU

The following table presents a side-by-side comparison between the features of LUIS and CLU. It also highlights the changes to your LUIS application after migrating to CLU. Select the linked concept to learn more about the changes.

LUIS features	CLU features	Post migration
Machine-learned and Structured ML entities	Learned entity components	Machine-learned entities without subentities will be transferred as CLU entities. Structured ML entities will only transfer leaf nodes (lowest level subentities that do not have their own subentities) as entities in CLU. The name of the entity in CLU

LUIS features	CLU features	Post migration
		will be the name of the subentity concatenated with the parent. For example, <i>Order.Size</i>
List, regex, and prebuilt entities	List, regex, and prebuilt entity components	List, regex, and prebuilt entities will be transferred as entities in CLU with a populated entity component based on the entity type.
<code>Pattern.Any</code> entities	Not currently available	<code>Pattern.Any</code> entities will be removed.
Single culture for each application	Multilingual models enable multiple languages for each project.	The primary language of your project will be set as your LUIS application culture. Your project can be trained to extend to different languages.
Entity roles	Roles are no longer needed.	Entity roles will be transferred as entities.
Settings for: normalize punctuation, normalize diacritics, normalize word form, use all training data	Settings are no longer needed.	Settings will not be transferred.
Patterns and phrase list features	Patterns and Phrase list features are no longer needed.	Patterns and phrase list features will not be transferred.
Entity features	Entity components	List or prebuilt entities added as features to an entity will be transferred as added components to that entity. Entity features will not be transferred for intents.
Intents and utterances	Intents and utterances	All intents and utterances will be transferred. Utterances will be labeled with their transferred entities.
Application GUIDs	Project names	A project will be created for each migrating application with the application name. Any special characters in the application names will be removed in CLU.
Versioning	Every time you train, a model is created and acts as a version of your project .	A project will be created for the selected application version.
Evaluation using batch	Evaluation using	Adding your testing dataset will be required.

LUIS features	CLU features	Post migration
testing	testing sets	
Role-Based Access Control (RBAC) for LUIS resources	Role-Based Access Control (RBAC) available for Language resources	Language resource RBAC must be manually added after migration .
Single training mode	Standard and advanced training modes	Training will be required after application migration.
Two publishing slots and version publishing	Ten deployment slots with custom naming	Deployment will be required after the application's migration and training.
LUIS authoring APIs and SDK support in .NET, Python, Java, and Node.js	CLU Authoring REST APIs	For more information, see the quickstart article for information on the CLU authoring APIs. Refactoring will be necessary to use the CLU authoring APIs.
LUIS Runtime APIs and SDK support in .NET, Python, Java, and Node.js	CLU Runtime APIs . CLU Runtime SDK support for .NET and Python .	See how to call the API for more information. Refactoring will be necessary to use the CLU runtime API response.

Migrate your LUIS applications

Use the following steps to migrate your LUIS application using either the LUIS portal or REST API.

LUIS portal

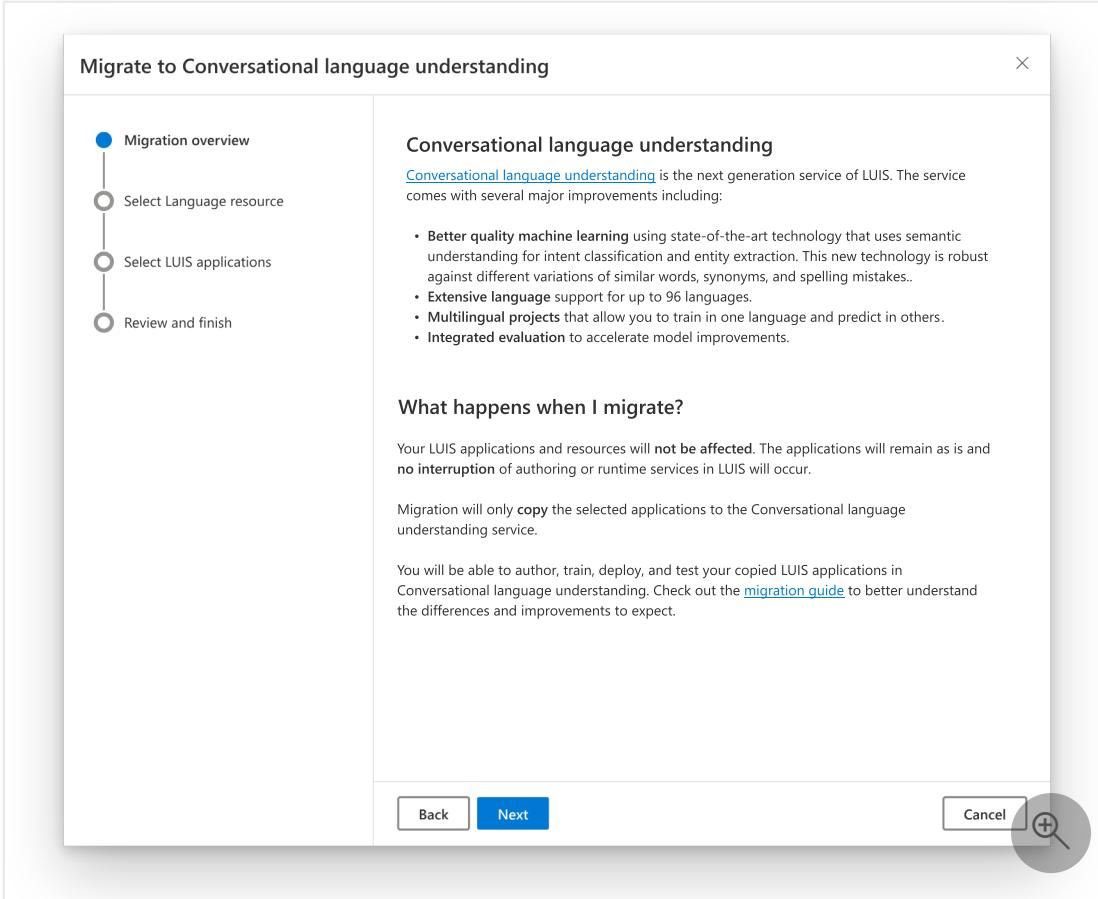
Migrate your LUIS applications using the LUIS portal

Follow these steps to begin migration using the [LUIS Portal](#):

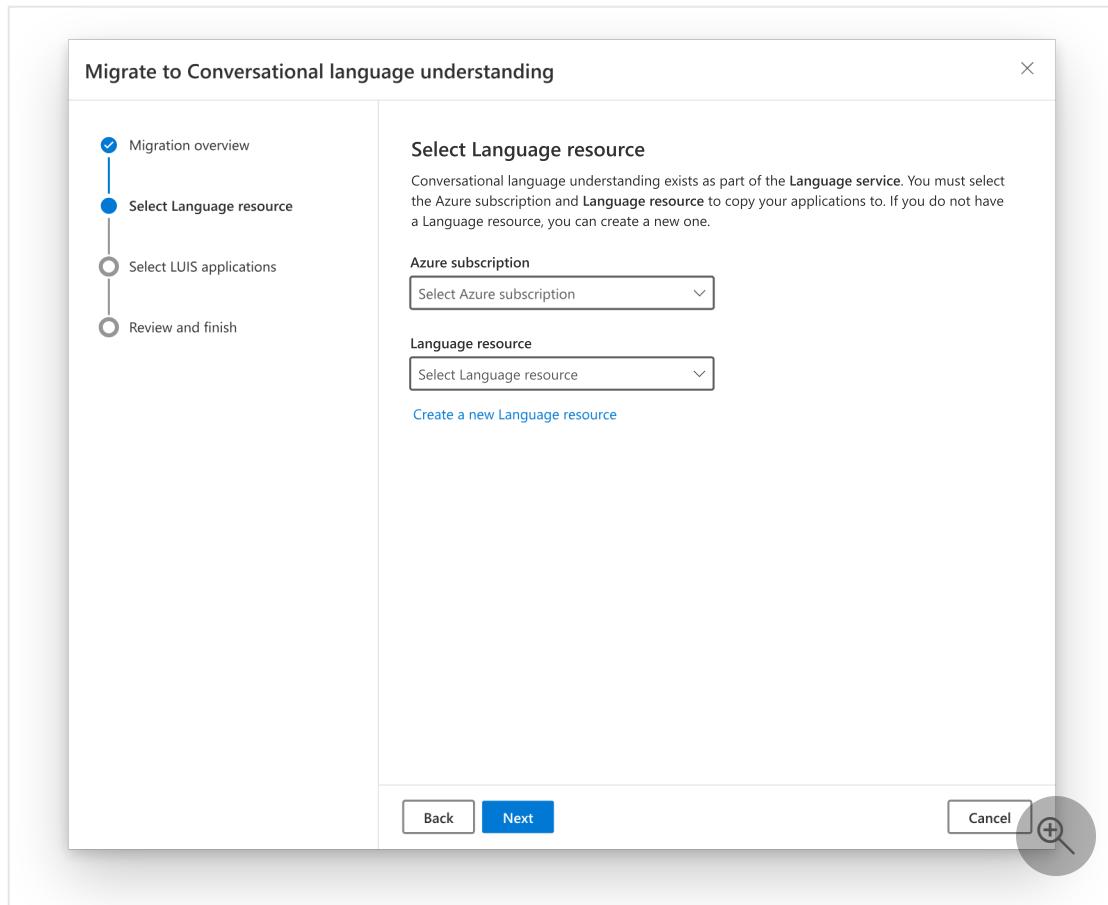
1. After logging into the LUIS portal, click the button on the banner at the top of the screen to launch the migration wizard. The migration will only copy your selected LUIS applications to CLU.



The migration overview tab provides a brief explanation of conversational language understanding and its benefits. Press Next to proceed.



2. Determine the Language resource that you wish to migrate your LUIS application to. If you have already created your Language resource, select your Azure subscription followed by your Language resource, and then select **Next**. If you don't have a Language resource, click the link to create a new Language resource. Afterwards, select the resource and select **Next**.

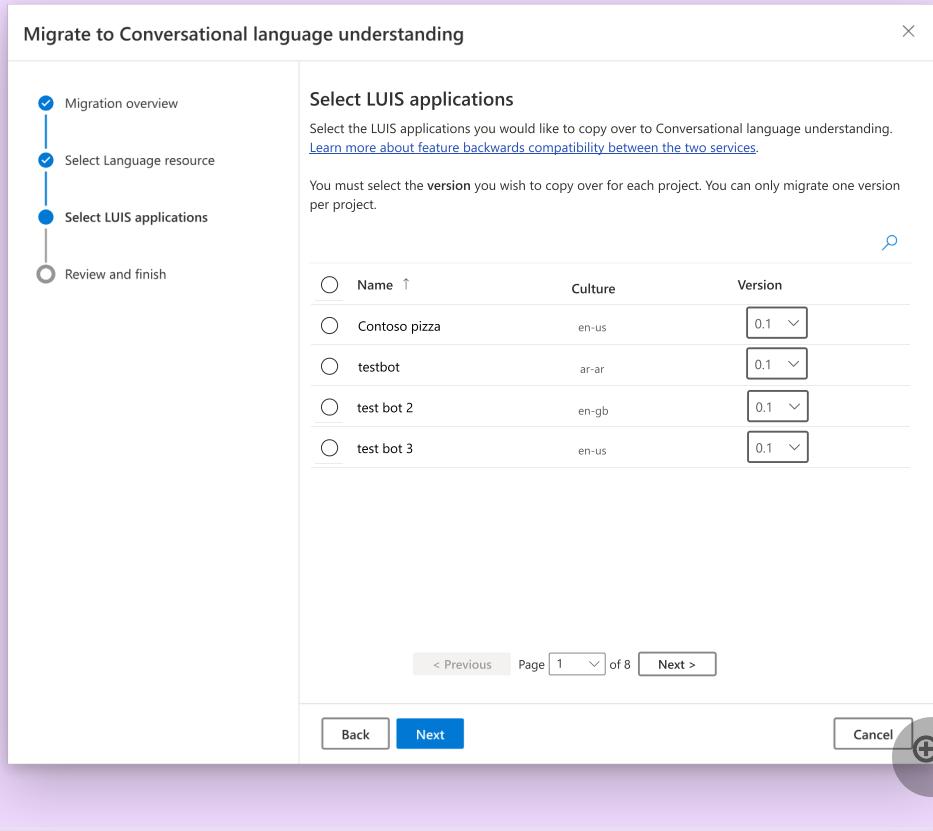


3. Select all your LUIS applications that you want to migrate, and specify each of their versions. Select **Next**. After selecting your application and version, you will be prompted with a message informing you of any features that won't be carried over from your LUIS application.

! Note

Special characters are not supported by conversational language understanding. Any special characters in your selected LUIS application

names will be removed in your new migrated applications.



4. Review your Language resource and LUIS applications selections. Select **Finish** to migrate your applications.
5. A popup window will let you track the migration status of your applications. Applications that have not started migrating will have a status of **Not started**. Applications that have begun migrating will have a status of **In progress**, and once they have finished migrating their status will be **Succeeded**. A **Failed** application means that you must repeat the migration process. Once the migration has completed for all applications, select **Done**.

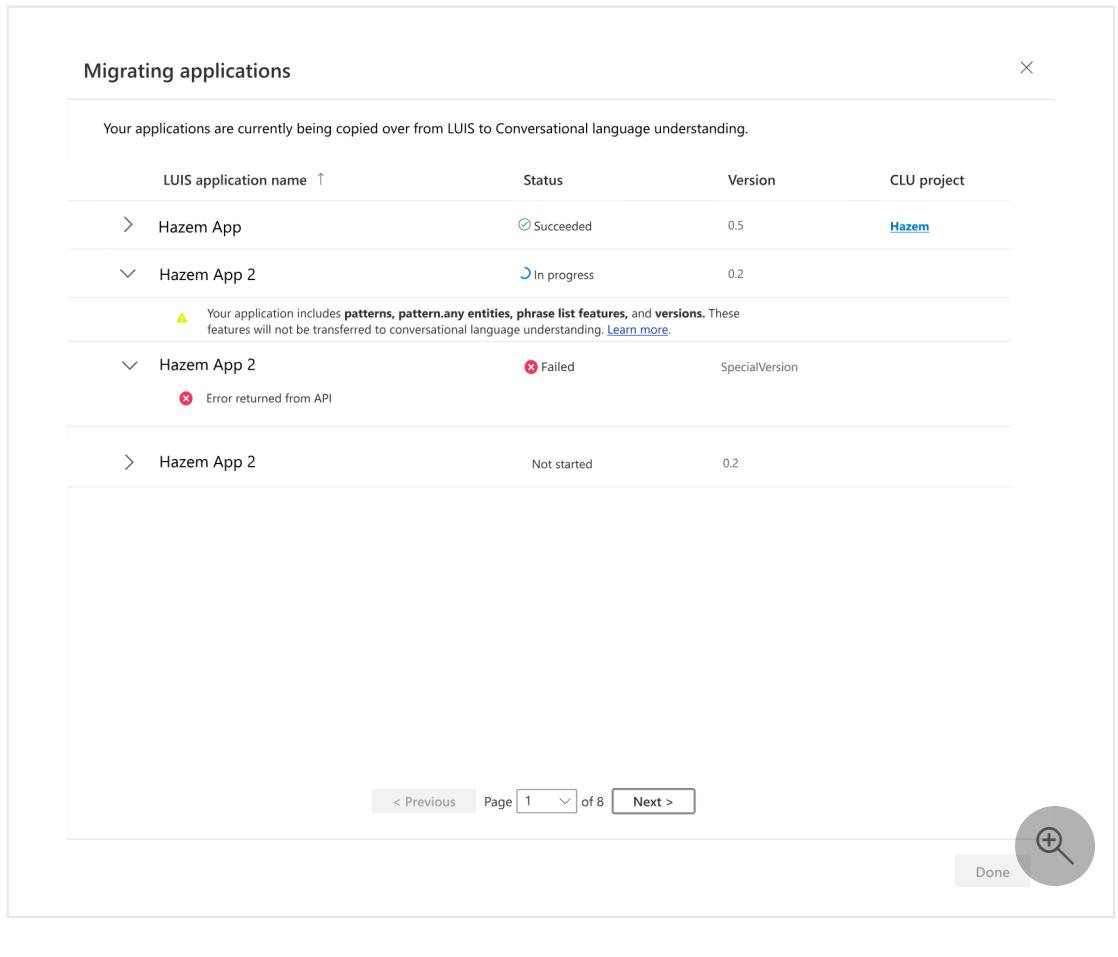
Migrating applications X

Your applications are currently being copied over from LUIS to Conversational language understanding.

LUIS application name ↑	Status	Version	CLU project
> Hazem App	✓ Succeeded	0.5	Hazem
▽ Hazem App 2	⌚ In progress	0.2	
<small>⚠ Your application includes patterns, pattern.any entities, phrase list features, and versions. These features will not be transferred to conversational language understanding. Learn more.</small>			
▽ Hazem App 2	✗ Failed	SpecialVersion	
<small>✗ Error returned from API</small>			
> Hazem App 2	Not started	0.2	

< Previous Page 1 ▼ of 8 Next >

+
🔍 Done



6. After your applications have migrated, you can perform the following steps:

- [Train your model](#)
- [Deploy your model](#)
- [Call your deployed model](#)

Frequently asked questions

Which LUIS JSON version is supported by CLU?

CLU supports the model JSON version 7.0.0. If the JSON format is older, it would need to be imported into LUIS first, then exported from LUIS with the most recent version.

How are entities different in CLU?

In CLU, a single entity can have multiple entity components, which are different methods for extraction. Those components are then combined together using rules you can define. The available components are:

- Learned: Equivalent to ML entities in LUIS, labels are used to train a machine-learned model to predict an entity based on the content and context of the provided labels.
- List: Just like list entities in LUIS, list components exact match a set of synonyms and maps them back to a normalized value called a **list key**.
- Prebuilt: Prebuilt components allow you to define an entity with the prebuilt extractors for common types available in both LUIS and CLU.
- Regex: Regex components use regular expressions to capture custom defined patterns, exactly like regex entities in LUIS.

Entities in LUIS will be transferred over as entities of the same name in CLU with the equivalent components transferred.

After migrating, your structured machine-learned leaf nodes and bottom-level subentities will be transferred to the new CLU model while all the parent entities and higher-level entities will be ignored. The name of the entity will be the bottom-level entity's name concatenated with its parent entity.

Example:

LUIS entity:

- Pizza Order
 - Topping
 - Size

Migrated LUIS entity in CLU:

- Pizza Order.Topping
- Pizza Order.Size

You also cannot label 2 different entities in CLU for the same span of characters. Learned components in CLU are mutually exclusive and do not provide overlapping predictions for learned components only. When migrating your LUIS application, entity labels that overlapped preserved the longest label and ignored any others.

For more information on entity components, see [Entity components](#).

How are entity roles transferred to CLU?

Your roles will be transferred as distinct entities along with their labeled utterances. Each role's entity type will determine which entity component will be populated. For example,

a list entity role will be transferred as an entity with the same name as the role, with a populated list component.

How do entity features get transferred in CLU?

Entities used as features for intents will not be transferred. Entities used as features for other entities will populate the relevant component of the entity. For example, if a list entity named *SizeList* was used as a feature to a machine-learned entity named *Size*, then the *Size* entity will be transferred to CLU with the list values from *SizeList* added to its list component. The same is applied for prebuilt and regex entities.

How are entity confidence scores different in CLU?

Any extracted entity has a 100% confidence score and therefore entity confidence scores should not be used to make decisions between entities.

How is conversational language understanding multilingual?

Conversational language understanding projects accept utterances in different languages. Furthermore, you can train your model in one language and extend it to predict in other languages.

Example:

Training utterance (English): *How are you?*

Labeled intent: Greeting

Runtime utterance (French): *Comment ça va?*

Predicted intent: Greeting

How is the accuracy of CLU better than LUIS?

CLU uses state-of-the-art models to enhance machine learning performance of different models of intent classification and entity extraction.

These models are insensitive to minor variations, removing the need for the following settings: *Normalize punctuation*, *normalize diacritics*, *normalize word form*, and *use all training data*.

Additionally, the new models do not support phrase list features as they no longer require supplementary information from the user to provide semantically similar words for better accuracy. Patterns were also used to provide improved intent classification using rule-based matching techniques that are not necessary in the new model paradigm. The question below explains this in more detail.

What do I do if the features I am using in LUIS are no longer present?

There are several features that were present in LUIS that will no longer be available in CLU. This includes the ability to do feature engineering, having patterns and pattern.any entities, and structured entities. If you had dependencies on these features in LUIS, use the following guidance:

- **Patterns:** Patterns were added in LUIS to assist the intent classification through defining regular expression template utterances. This included the ability to define Pattern only intents (without utterance examples). CLU is capable of generalizing by leveraging the state-of-the-art models. You can provide a few utterances to that matched a specific pattern to the intent in CLU, and it will likely classify the different patterns as the top intent without the need of the pattern template utterance. This simplifies the requirement to formulate these patterns, which was limited in LUIS, and provides a better intent classification experience.
- **Phrase list features:** The ability to associate features mainly occurred to assist the classification of intents by highlighting the key elements/features to use. This is no longer required since the deep models used in CLU already possess the ability to identify the elements that are inherent in the language. In turn removing these features will have no effect on the classification ability of the model.
- **Structured entities:** The ability to define structured entities was mainly to enable multilevel parsing of utterances. With the different possibilities of the sub-entities, LUIS needed all the different combinations of entities to be defined and presented to the model as examples. In CLU, these structured entities are no longer supported, since overlapping learned components are not supported. There are a few possible approaches to handling these structured extractions:
 - **Non-ambiguous extractions:** In most cases the detection of the leaf entities is enough to understand the required items within a full span. For example, structured entity such as *Trip* that fully spanned a source and destination (*London to New York* or *Home to work*) can be identified with the individual spans predicted for source and destination. Their presence as individual predictions would inform you of the *Trip* entity.

- **Ambiguous extractions:** When the boundaries of different sub-entities are not very clear. To illustrate, take the example “I want to order a pepperoni pizza and an extra cheese vegetarian pizza”. While the different pizza types as well as the topping modifications can be extracted, having them extracted without context would have a degree of ambiguity of where the extra cheese is added. In this case the extent of the span is context based and would require ML to determine this. For ambiguous extractions you can use one of the following approaches:

1. Combine sub-entities into different entity components within the same entity.

Example:

LUIS Implementation:

- Pizza Order (entity)
 - Size (subentity)
 - Quantity (subentity)

CLU Implementation:

- Pizza Order (entity)
 - Size (list entity component: small, medium, large)
 - Quantity (prebuilt entity component: number)

In CLU, you would label the entire span for *Pizza Order* inclusive of the size and quantity, which would return the pizza order with a list key for size, and a number value for quantity in the same entity object.

2. For more complex problems where entities contain several levels of depth, you can create a project for each level of depth in the entity structure. This gives you the option to:

- Pass the utterance to each project.
- Combine the analyses of each project in the stage proceeding CLU.

For a detailed example on this concept, check out the pizza sample projects available on [GitHub](#).

How do I manage versions in CLU?

CLU saves the data assets used to train your model. You can export a model's assets or load them back into the project at any point. So models act as different versions of your project.

You can export your CLU projects using [Language Studio](#) or [programmatically](#) and store different versions of the assets locally.

Why is CLU classification different from LUIS? How does None classification work?

CLU presents a different approach to training models by using multi-classification as opposed to binary classification. As a result, the interpretation of scores is different and also differs across training options. While you are likely to achieve better results, you have to observe the difference in scores and determine a new threshold for accepting intent predictions. You can easily add a confidence score threshold for the [None intent](#) in your project settings. This will return *None* as the top intent if the top intent did not exceed the confidence score threshold provided.

Do I need more data for CLU models than LUIS?

The new CLU models have better semantic understanding of language than in LUIS, and in turn help make models generalize with a significant reduction of data. While you shouldn't aim to reduce the amount of data that you have, you should expect better performance and resilience to variations and synonyms in CLU compared to LUIS.

If I don't migrate my LUIS apps, will they be deleted?

Your existing LUIS applications will be available until October 1, 2025. After that time you will no longer be able to use those applications, the service endpoints will no longer function, and the applications will be permanently deleted.

Are .LU files supported on CLU?

Only JSON format is supported by CLU. You can import your .LU files to LUIS and export them in JSON format, or you can follow the migration steps above for your application.

What are the service limits of CLU?

See the [service limits](#) article for more information.

Do I have to refactor my code if I migrate my applications from LUIS to CLU?

The API objects of CLU applications are different from LUIS and therefore code refactoring will be necessary.

If you are using the LUIS [programmatic](#) and [runtime](#) APIs, you can replace them with their equivalent APIs.

[CLU authoring APIs](#): Instead of LUIS's specific CRUD APIs for individual actions such as *add utterance*, *delete entity*, and *rename intent*, CLU offers an [import API](#) that replaces the full content of a project using the same name. If your service used LUIS programmatic APIs to provide a platform for other customers, you must consider this new design paradigm. All other APIs such as: *listing projects*, *training*, *deploying*, and *deleting* are available. APIs for actions such as *importing* and *deploying* are asynchronous operations instead of synchronous as they were in LUIS.

[CLU runtime APIs](#): The new API request and response includes many of the same parameters such as: *query*, *prediction*, *top intent*, *intents*, *entities*, and their values. The CLU response object offers a more straightforward approach. Entity predictions are provided as they are within the utterance text, and any additional information such as resolution or list keys are provided in extra parameters called `extraInformation` and `resolution`.

You can use the [.NET](#) or [Python](#) CLU runtime SDK to replace the LUIS runtime SDK. There is currently no authoring SDK available for CLU.

How are the training times different in CLU? How is standard training different from advanced training?

CLU offers standard training, which trains and learns in English and is comparable to the training time of LUIS. It also offers advanced training, which takes a considerably longer duration as it extends the training to all other [supported languages](#). The train API will continue to be an asynchronous process, and you will need to assess the change in the DevOps process you employ for your solution.

How has the experience changed in CLU compared to LUIS? How is the development lifecycle different?

In LUIS you would Build-Train-Test-Publish, whereas in CLU you Build-Train-Evaluate-Deploy-Test.

1. **Build:** In CLU, you can define your intents, entities, and utterances before you train. CLU additionally offers you the ability to specify *test data* as you build your application to be used for model evaluation. Evaluation assesses how well your

model is performing on your test data and provides you with precision, recall, and F1 metrics.

2. **Train:** You create a model with a name each time you train. You can overwrite an already trained model. You can specify either *standard* or *advanced* training, and determine if you would like to use your test data for evaluation, or a percentage of your training data to be left out from training and used as testing data. After training is complete, you can evaluate how well your model is doing on the outside.
3. **Deploy:** After training is complete and you have a model with a name, it can be deployed for predictions. A deployment is also named and has an assigned model. You could have multiple deployments for the same model. A deployment can be overwritten with a different model, or you can swap models with other deployments in the project.
4. **Test:** Once deployment is complete, you can use it for predictions through the deployment endpoint. You can also test it in the studio in the Test deployment page.

This process is in contrast to LUIS, where the application ID was attached to everything, and you deployed a version of the application in either the staging or production slots.

This will influence the DevOps processes you use.

Does CLU have container support?

No, you cannot export CLU to containers.

How will my LUIS applications be named in CLU after migration?

Any special characters in the LUIS application name will be removed. If the cleared name length is greater than 50 characters, the extra characters will be removed. If the name after removing special characters is empty (for example, if the LUIS application name was @@), the new name will be *untitled*. If there is already a conversational language understanding project with the same name, the migrated LUIS application will be appended with _1 for the first duplicate and increase by 1 for each additional duplicate. In case the new name's length is 50 characters and it needs to be renamed, the last 1 or 2 characters will be removed to be able to concatenate the number and still be within the 50 characters limit.

Migration from LUIS Q&A

If you have any questions that were unanswered in this article, consider leaving your questions at our [Microsoft Q&A thread ↗](#).

Next steps

- [Quickstart: create a CLU project](#)
- [CLU language support](#)
- [CLU FAQ](#)

Migrate to V3 Authoring entity

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

The V3 authoring provides one new entity type, the machine-learning entity, along with the ability to add relationships to the machine-learning entity and other entities or features of the application. There is currently no date by which migration needs to be completed.

Entities are decomposable in V3

Entities created with the V3 authoring APIs, either using the [APIs](#) or with the portal, allow you to build a layered entity model with a parent and children. The parent is known to as the **machine-learning entity** and the children are known as **subentities** of the machine learned entity.

Each subentity is also a machine-learning entity but with the added configuration options of features.

- **Required features** are rules that guarantee an entity is extracted when it matches a feature. The rule is defined by required feature to the model:
 - [Prebuilt entity](#)
 - [Regular expression entity](#)
 - [List entity](#).

How do these new relationships compare to V2 authoring

V2 authoring provided hierarchical and composite entities along with roles and features to accomplish this same task. Because the entities, features, and roles were not explicitly related to each other, it was difficult to understand how LUIS implied the relationships during prediction.

With V3, the relationship is explicit and designed by the app authors. This allows you, as the app author, to:

- Visually see how LUIS is predicting these relationships, in the example utterances
- Test for these relationships either with the [interactive test pane](#) or at the endpoint
- Use these relationships in the client application, via a well-structured, named, nested [.json object](#)

Planning

When you migrate, consider the following in your migration plan:

- Back up your LUIS app, and perform the migration on a separate app. Having a V2 and V3 app available at the same time allows you to validate the changes required and the impact on the prediction results.
- Capture current prediction success metrics
- Capture current dashboard information as a snapshot of app status
- Review existing intents, entities, phrase lists, patterns, and batch tests
- The following elements can be migrated **without change**:
 - Intents
 - Entities
 - Regular expression entity
 - List entity
 - Features
 - Phrase list
- The following elements need to be migrated **with changes**:
 - Entities
 - Hierarchical entity
 - Composite entity
 - Roles - roles can only be applied to a machine-learning (parent) entity. Roles can't be applied to subentities
 - Batch tests and patterns that use the hierarchical and composite entities

When you design your migration plan, leave time to review the final machine-learning entities, after all hierarchical and composite entities have been migrated. While a straight migration will work, after you make the change and review your batch test results, and prediction JSON, the more unified JSON may lead you to make changes so the final information delivered to the client-side app is organized differently. This is similar to code refactoring and should be treated with the same review process your organization has in place.

If you don't have batch tests in place for your V2 model, and migrate the batch tests to the V3 model as part of the migration, you won't be able to validate how the migration will impact the endpoint prediction results.

Migrating from V2 entities

As you begin to move to the V3 authoring model, you should consider how to move to the machine-learning entity, and its subentities and features.

The following table notes which entities need to migrate from a V2 to a V3 entity design.

V2 authoring entity type	V3 authoring entity type	Example
Composite entity	Machine learned entity	learn more
Hierarchical entity	machine-learning entity's role	learn more

Migrate V2 Composite entity

Each child of the V2 composite should be represented with a subentity of the V3 machine-learning entity. If the composite child is a prebuilt, regular expression, or a list entity, this should be applied as a required feature on the subentity.

Considerations when planning to migrate a composite entity to a machine-learning entity:

- Child entities can't be used in patterns
- Child entities are no longer shared
- Child entities need to be labeled if they used to be non-machine-learned

Existing features

Any phrase list used to boost words in the composite entity should be applied as a feature to either the machine-learning (parent) entity, the subentity (child) entity, or the intent (if the phrase list only applies to one intent). Plan to add the feature to the entity where it should boost most significantly. Do not add the feature generically to the machine-learning (parent) entity, if it will most significantly boost the prediction of a subentity (child).

New features

In V3 authoring, add a planning step to evaluate entities as possible features for all the entities and intents.

Example entity

This entity is an example only. Your own entity migration may require other considerations.

Consider a V2 composite for modifying a pizza `order` that uses:

- prebuilt datetimeV2 for delivery time
- phrase list to boost certain words such as pizza, pie, crust, and topping
- list entity to detect toppings such as mushrooms, olives, pepperoni.

An example utterance for this entity is:

```
Change the toppings on my pie to mushrooms and delivery it 30 minutes later
```

The following table demonstrates the migration:

V2 models	V3 models
Parent - Component entity named <code>Order</code>	Parent - machine-learning entity named <code>Order</code>
Child - Prebuilt datetimeV2	* Migrate prebuilt entity to new app. * Add required feature on parent for prebuilt datetimeV2.
Child - list entity for toppings	* Migrate list entity to new app. * Then add a required feature on the parent for the list entity.

Migrate V2 Hierarchical entity

In V2 authoring, a hierarchical entity was provided before roles existing in LUIS. Both served the same purpose of extracting entities based on context usage. If you have hierarchical entities, you can think of them as simple entities with roles.

In V3 authoring:

- A role can be applied on the machine-learning (parent) entity.
- A role can't be applied to any subentities.

This entity is an example only. Your own entity migration may require other considerations.

Consider a V2 hierarchical entity for modifying a pizza `order`:

- where each child determines either an original topping or the final topping

An example utterance for this entity is:

`Change the topping from mushrooms to olives`

The following table demonstrates the migration:

V2 models	V3 models
Parent - Component entity named <code>Order</code>	Parent - machine-learning entity named <code>Order</code>
Child - Hierarchical entity with original and final pizza topping	* Add role to <code>Order</code> for each topping.

API change constraint replaced with required feature

This change was made in May 2020 at the //Build conference and only applies to the v3 authoring APIs where an app is using a constrained feature. If you are migrating from v2 authoring to v3 authoring, or have not used v3 constrained features, skip this section.

Functionality - ability to require an existing entity as a feature to another model and only extract that model if the entity is detected. The functionality has not changed but the API and terminology have changed.

Previous terminology	New terminology
<code>constrained feature</code>	<code>required feature</code>
<code>constraint</code>	<code>isRequired</code>
<code>instanceOf</code>	

Automatic migration

Starting June 19 2020, you won't be allowed to create constraints programmatically using the previous authoring API that exposed this functionality.

All existing constraint features will be automatically migrated to the required feature flag. No programmatic changes are required to your prediction API and no resulting change on the quality of the prediction accuracy.

LUIS portal changes

The LUIS preview portal referenced this functionality as a **constraint**. The current LUIS portal designates this functionality as a **required feature**.

Previous authoring API

This functionality was applied in the preview authoring [Create Entity Child API](#) as the part of an entity's definition, using the `instanceOf` property of an entity's child:

JSON

```
{  
  "name" : "dayOfWeek",  
  "instanceOf": "datetimeV2",  
  "children": [  
    {  
      "name": "dayNumber",  
      "instanceOf": "number",  
      "children": []  
    }  
  ]  
}
```

New authoring API

This functionality is now applied with the [Add entity feature relation API](#) using the `featureName` and `isRequired` properties. The value of the `featureName` property is the name of the model.

JSON

```
{  
  "featureName": "YOUR-MODEL-NAME-HERE",  
  "isRequired" : true  
}
```

Next steps

- Developer resources

Prediction endpoint changes for V3

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

The query prediction endpoint V3 APIs have changed. Use this guide to understand how to migrate to version 3 endpoint APIs. There is currently no date by which migration needs to be completed.

Generally available status - this V3 API include significant JSON request and response changes from V2 API.

The V3 API provides the following new features:

- [External entities](#)
- [Dynamic lists](#)
- [Prebuilt entity JSON changes](#)

The 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](#)

[Reference documentation](#) is available for V3.

V3 changes from preview to GA

V3 made the following changes as part of the move to GA:

- The following prebuilt entities have different JSON responses:
 - [OrdinalV1](#)
 - [GeographyV2](#)
 - [DatetimeV2](#)
 - Measurable unit key name from `units` to `unit`

- Request body JSON change:
 - from `preferExternalEntities` to `preferExternalEntities`
 - optional `score` parameter for external entities
- Response body JSON changes:
 - `normalizedQuery` removed

Suggested adoption strategy

If you use Bot Framework, Bing Spell Check V7, or want to migrate your LUIS app authoring only, continue to use the V2 endpoint.

If you know none of your client application or integrations (Bot Framework, and Bing Spell Check V7) are impacted and you are comfortable migrating your LUIS app authoring and your prediction endpoint at the same time, begin using the V3 prediction endpoint. The V2 prediction endpoint will still be available and is a good fall-back strategy.

For information on using the Bing Spell Check API, see [How to correct misspelled words](#).

Not supported

Bot Framework and Azure AI Bot Service client applications

Continue to use the V2 API prediction endpoint until the V4.7 of the Bot Framework is released.

Endpoint URL changes

Changes by slot name and version name

The [format of the V3 endpoint HTTP call](#) has changed.

If you want to query by version, you first need to [publish via API](#) with `"directVersionPublish":true`. Query the endpoint referencing the version ID instead of the slot name.

Valid values for `SLOT-NAME`

`production`

`staging`

Request changes

Query string changes

V3 API query string parameters include:

Query parameter	LUIS portal name	Type	Version	Default	Purpose
<code>log</code>	Save logs	boolean	V2 & V3	false	Store query in log file. Default value is false.
<code>query</code>	-	string	V3 only	No default - it is required in the GET request	In V2 , the utterance to be predicted is in the <code>q</code> parameter. In V3 , the functionality is passed in the <code>query</code> parameter.
<code>show-all-intents</code>	Include scores for all intents	boolean	V3 only	false	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: <code>prediction.intents.give</code> . In V2, these were returned in an array.
<code>verbose</code>	Include more entities details	boolean	V2 & V3	false	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> . In V3 , this parameter only provides entity metadata details of entity prediction.

Query parameter	Luis portal name	Type	Version	Default	Purpose
<code>timezoneOffset</code>	-	string	V2	-	Timezone applied to datetimeV2 entities.
<code>datetimeReference</code>	-	string	V3	-	Timezone applied to datetimeV2 entities. Replaces <code>timezoneOffset</code> from V2.

V3 POST body

JSON

```
{
  "query": "your utterance here",
  "options": {
    "datetimeReference": "2019-05-05T12:00:00",
    "preferExternalEntities": 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.preferExternalEntities</code>	boolean	V3 only	false	Specifies if user's external entity (with same name as existing entity) is used or the

Property	Type	Version	Default	Purpose
				existing entity in the model is used for prediction.
query	string	V3 only	Required.	In V2, the utterance to be predicted is in the <code>q</code> parameter. In V3, the functionality is passed in the <code>query</code> parameter.

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:

```
JSON
{
  "query": "this is your utterance you want predicted",
  "topScoringIntent": {},
  "intents": [],
  "entities": [],
  "compositeEntities": []
}
```

The top JSON properties for V3 are:

```
JSON
{
  "query": "this is your utterance you want predicted",
  "prediction": {
    "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:

Node.js

```
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 `name` 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.

Entity response changes

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`.

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.

Prebuilt entity changes

The V3 response object includes changes to prebuilt entities. Review [specific prebuilt entities](#) to learn more.

List entity prediction changes

The JSON for a list entity prediction has changed to be an array of arrays:

JSON

```
"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.

Node.js

```
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:

JSON

```
"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:

JSON

```
"entities": {
  "Destination": [
    "Yellow Bird Lane"
  ]
}
```

In V3, the same result with the `verbose` flag to return entity metadata:

JSON

```
"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"
      }
    ]
  }
}
```

Extend the app at prediction time

Learn [concepts](#) about how to extend the app at prediction runtime.

Next steps

Use the V3 API documentation to update existing REST calls to LUIS [endpoint ↗](#) APIs.

Upgrade composite entity to machine-learning entity

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Upgrade composite entity to machine-learning entity to build an entity that receives more complete predictions with better decomposability for debugging the entity.

Current version model restrictions

The upgrade process creates machine-learning entities, based on the existing composite entities found in your app, into a new version of your app. This includes composite entity children and roles. The process also switches the labels in example utterances to use the new entity.

Upgrade process

The upgrade process:

- Creates new machine-learning entity for each composite entity.
- Child entities:
 - If child entity is only used in composite entity, it will only be added to machine-learning entity.
 - If child entity is used in composite *and* as a separate entity (labeled in example utterances), it will be added to the version as an entity and as a subentity to the new machine-learning entity.
 - If the child entity uses a role, each role will be converted into a subentity of the same name.
 - If the child entity is a non-machine-learning entity (regular expression, list entity, or prebuilt entity), a new subentity is created with the same name, and the new subentity has a feature using the non-machine-learning entity with the required feature added.

- Names are retained but must be unique at same subentity/sibling level. Refer to [unique naming limits](#).
- Labels in example utterances are switched to new machine-learning entity with subentities.

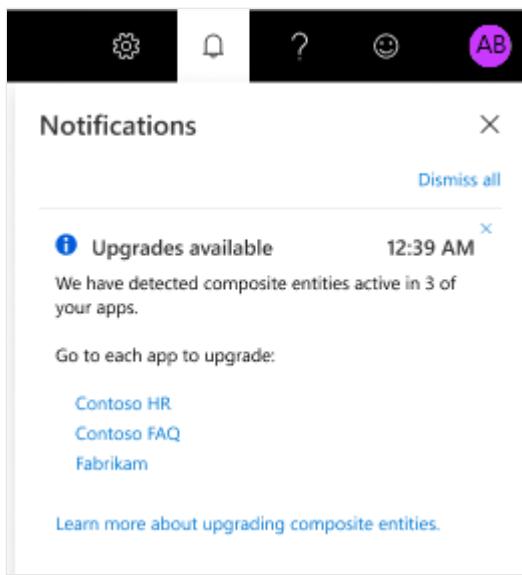
Use the following chart to understand how your model changes:

Old object	New object	Notes
Composite entity	machine-learning entity with structure	Both objects are parent objects.
Composite's child entity is simple entity	subentity	Both objects are child objects.
Composite's child entity is Prebuilt entity such as Number	subentity with name of Prebuilt entity such as Number, and subentity has <i>feature</i> of Prebuilt Number entity with constraint option set to <i>true</i> .	subentity contains feature with constraint at subentity level.
Composite's child entity is Prebuilt entity such as Number, and prebuilt entity has a role	subentity with name of role, and subentity has feature of Prebuilt Number entity with constraint option set to true.	subentity contains feature with constraint at subentity level.
Role	subentity	The role name becomes the subentity name. The subentity is a direct descendant of the machine-learning entity.

Begin upgrade process

Before updating, make sure to:

- Change versions if you are not on the correct version to upgrade
1. Begin the upgrade process from the notification or you can wait until the next scheduled prompt.



2. On the pop-up, select **Upgrade now**.
3. Review the **What happens when you upgrade** information then select **Continue**.
4. Select the composite entities from the list to upgrade then select **Continue**.
5. You can move any untrained changes from the current version into the upgraded version by selecting the checkbox.
6. Select **Continue** to begin the upgrade process.
7. The progress bar indicates the status of the upgrade process.
8. When the process is done, you are on a new trained version with the new machine-learning entities. Select **Try your new entities** to see the new entity.
If the upgrade or training failed for the new entity, a notification provides more information.
9. On the Entity list page, the new entities are marked with **NEW** next to the type name.

Next steps

- [Authors and collaborators](#)

Migrate to an Azure resource authoring key

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

ⓘ Important

As of December 3rd 2020, existing LUIS users must have completed the migration process to continue authoring LUIS applications.

Language Understanding (LUIS) authoring authentication has changed from an email account to an Azure resource. Use this article to learn how to migrate your account, if you haven't migrated yet.

What is migration?

Migration is the process of changing authoring authentication from an email account to an Azure resource. Your account will be linked to an Azure subscription and an Azure authoring resource after you migrate.

Migration has to be done from the [LUIS portal](#). If you create the authoring keys by using the LUIS CLI, for example, you'll need to complete the migration process in the LUIS portal. You can still have co-authors on your applications after migration, but these will be added on the Azure resource level instead of the application level. Migrating your account can't be reversed.

ⓘ Note

- If you need to create a prediction runtime resource, there's a [separate process](#) to create it.

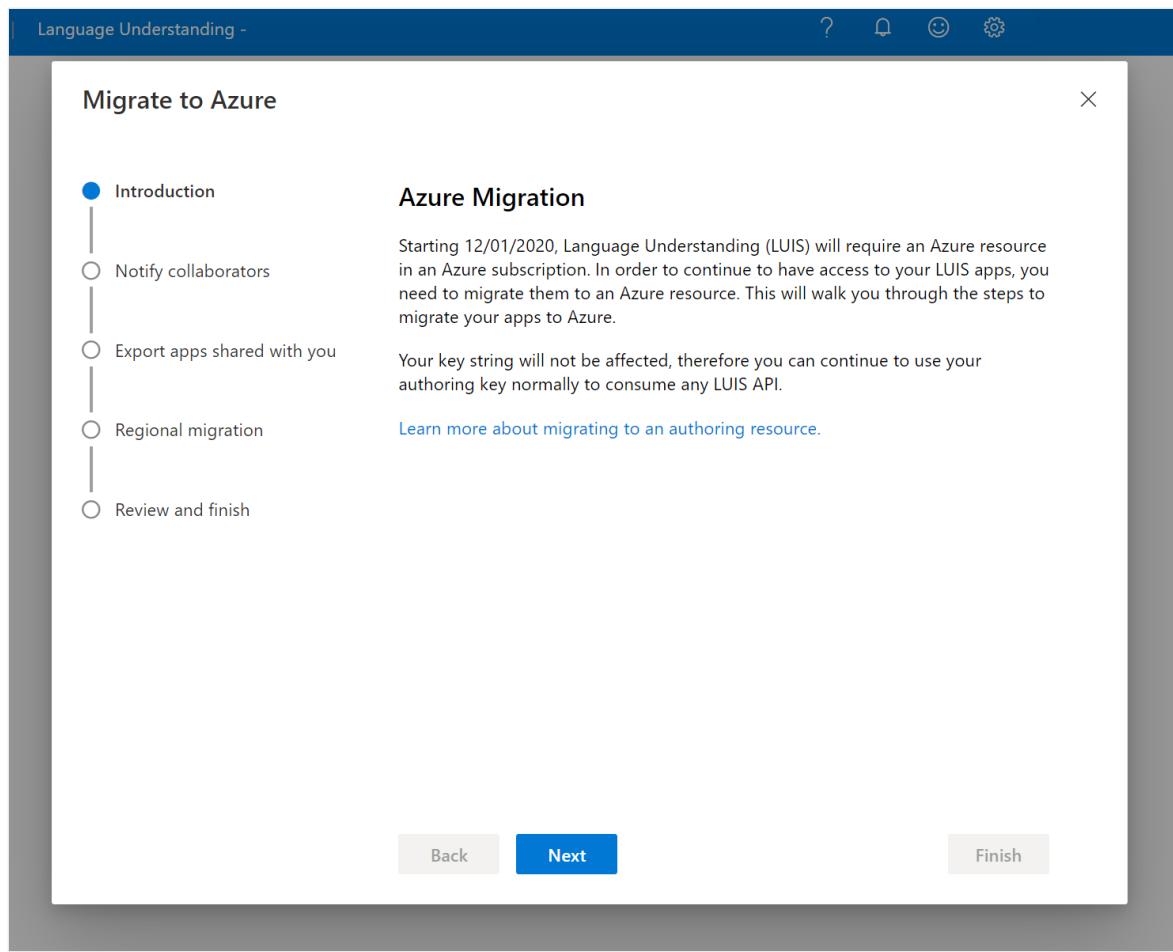
- See the [migration notes](#) section below for information on how your applications and contributors will be affected.
- Authoring your LUIS app is free, as indicated by the F0 tier. Learn [more about pricing tiers](#).

Migration prerequisites

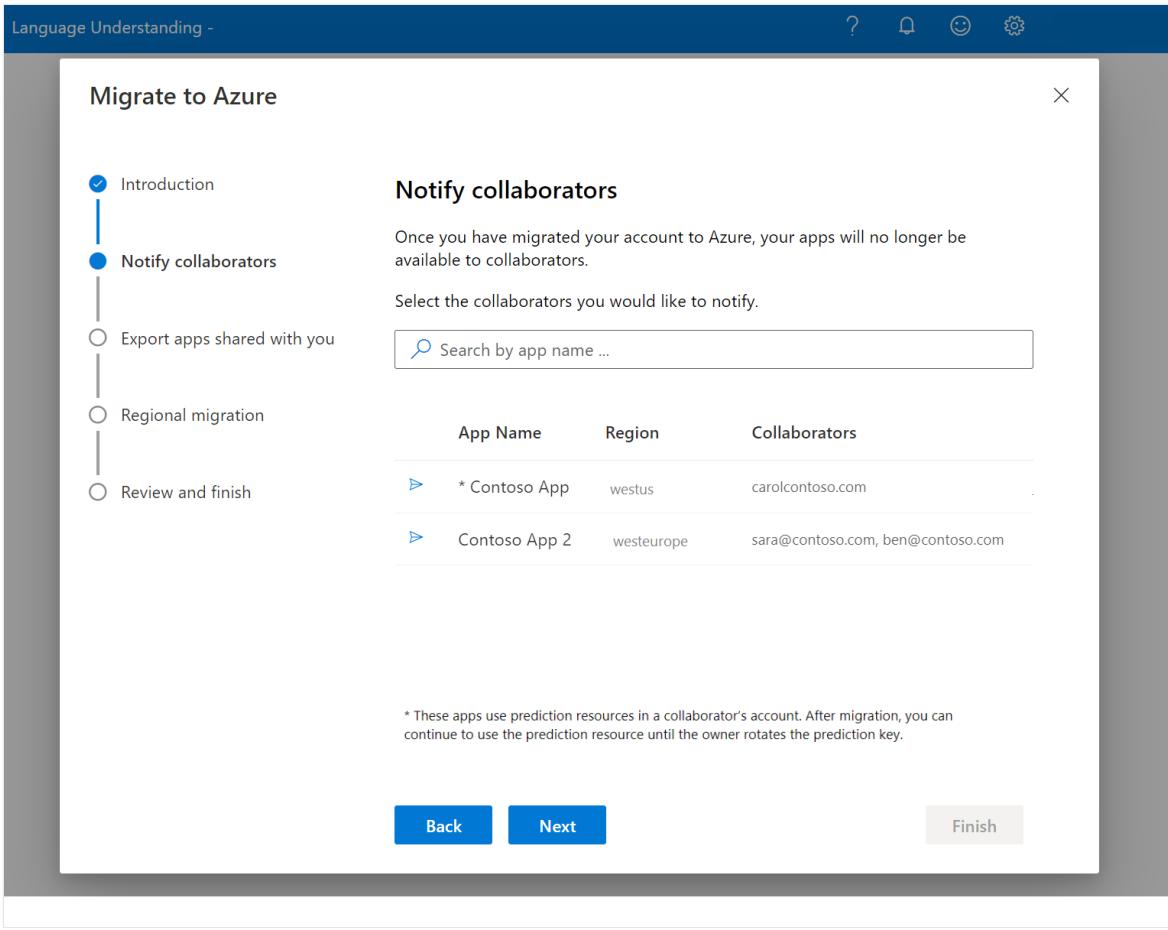
- A valid Azure subscription. Ask your tenant admin to add you on the subscription, or [sign up for a free one](#).
- A LUIS Azure authoring resource from the LUIS portal or from the [Azure portal](#).
 - Creating an authoring resource from the LUIS portal is part of the migration process described in the next section.
- If you're a collaborator on applications, applications won't automatically migrate. You will be prompted to export these apps while going through the migration flow. You can also use the [export API](#). You can import the app back into LUIS after migration. The import process creates a new app with a new app ID, for which you're the owner.
- If you're the owner of the application, you won't need to export your apps because they'll migrate automatically. An email template with a list of all collaborators for each application is provided, so they can be notified of the migration process.

Migration steps

1. When you sign-in to the [LUIS portal](#), an Azure migration window will open with the steps for migration. If you dismiss it, you won't be able to proceed with authoring your LUIS applications, and the only action displayed will be to continue with the migration.



2. If you have collaborators on any of your apps, you will see a list of application names owned by you, along with the authoring region and collaborator emails on each application. We recommend sending your collaborators an email notifying them about the migration by clicking on the **send** symbol button on the left of the application name. A ***** symbol will appear next to the application name if a collaborator has a prediction resource assigned to your application. After migration, these apps will still have these prediction resources assigned to them even though the collaborators will not have access to author your applications. However, this assignment will be broken if the owner of the prediction resource **regenerated the keys** from the Azure portal.



For each collaborator and app, the default email application opens with a lightly formatted email. You can edit the email before sending it. The email template includes the exact app ID and app name.

HTML

Dear Sir/Madam,

I will be migrating my LUIS account to Azure. Consequently, you will no longer have access to the following app:

App Id: <app-ID-omitted>
App name: Human Resources

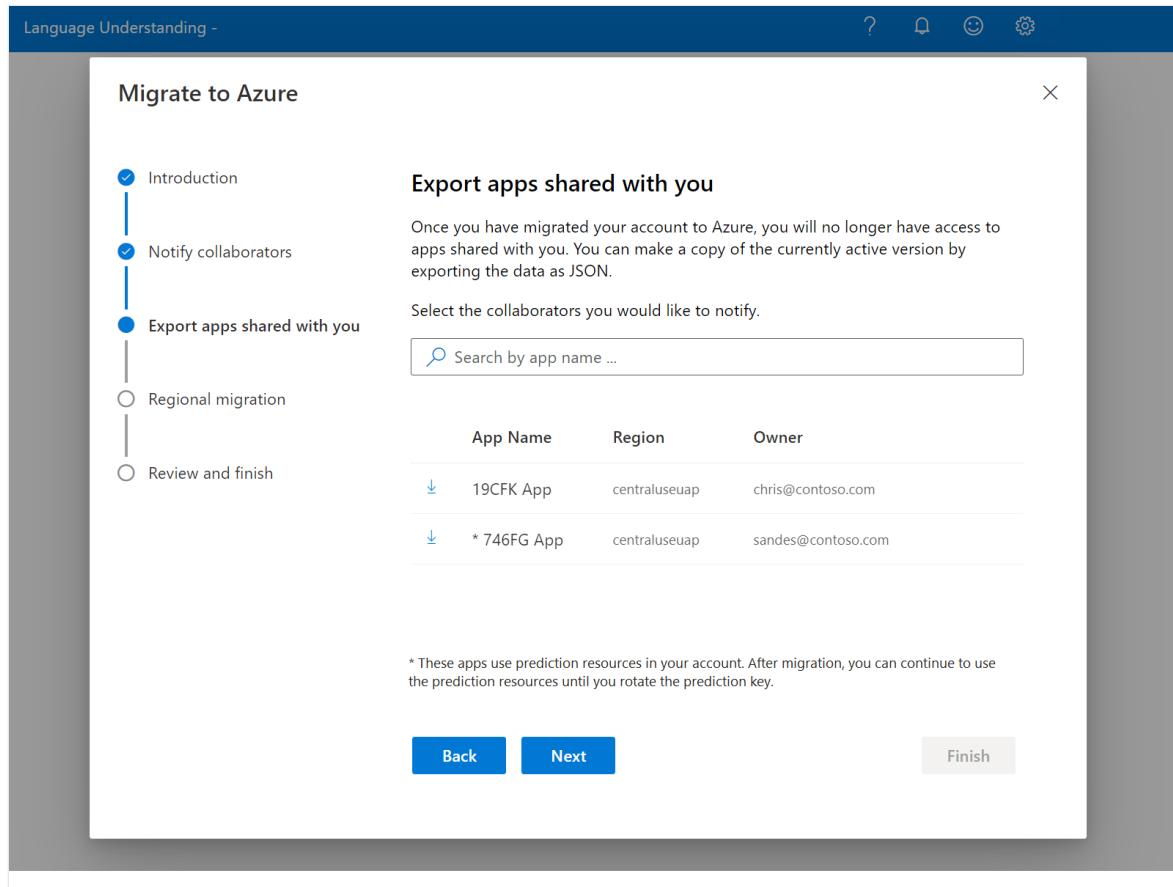
Thank you

(!) Note

After you migrate your account to Azure, your apps will no longer be available to collaborators.

3. If you're a collaborator on any apps, a list of application names shared with you is shown along with the authoring region and owner emails on each application. It is

recommend to export a copy of the apps by clicking on the export button on the left of the application name. You can import these apps back after you migrate, because they won't be automatically migrated with you. A * symbol will appear next to the application name if you have a prediction resource assigned to an application. After migration, your prediction resource will still be assigned to these applications even though you will no longer have access to author these apps. If you want to break the assignment between your prediction resource and the application, you will need to go to Azure portal and [regenerate the keys](#).



4. In the window for migrating regions, you will be asked to migrate your applications to an Azure resource in the same region they were authored in. LUIS has three authoring regions [and portals](#). The window will show the regions where your owned applications were authored. The displayed migration regions may be different depending on the regional portal you use, and apps you've authored.

Language Understanding -

Migrate to Azure

X

- Introduction
- Notify collaborators
- Export apps shared with you
- Regional migration
 - East Australia migration
 - West Europe migration
 - West US migration
- Review and finish

Migrate in "australiaeast"

Note: By default, your apps will be shared with other members in this subscription.

Use existing authoring resource

Azure directory
Luis E2e Test

Note: To switch to another directory, use the top right user avatar.

Azure subscription *
Select azure subscription ...

Authoring resource* ?
Select an authoring resource ...

Create new authoring resource

Back Migrate and Next Finish

5. For each region, choose to create a new LUIS authoring resource, or to migrate to an existing one using the buttons.

Language Understanding -

Migrate to Azure

X

- Introduction
- Regional migration
 - East Australia migration
 - West Europe migration
 - West US migration
- Review and finish

Migrate in "westeurope"

Note: By default, your apps will be shared with other members in this subscription.

Use existing authoring resource

Create new authoring resource

Azure directory *

Note: To switch to another directory, use the top right user avatar.

Azure subscription *
Select azure subscription ...

Azure resource group *
Search a resource group ...

Azure resource name *
Type resource name ...

Back Migrate and Next Finish

Provide the following information:

- **Tenant Name:** The tenant that your Azure subscription is associated with. By default this is set to the tenant you're currently using. You can switch tenants by closing this window and selecting the avatar in the top right of the screen, containing your initials. Select **Migrate to Azure** to re-open the window.
- **Azure Subscription Name:** The subscription that will be associated with the resource. If you have more than one subscription that belongs to your tenant, select the one you want from the drop-down list.
- **Authoring Resource Name:** A custom name that you choose. It's used as part of the URL for your authoring and prediction endpoint queries. If you are creating a new authoring resource, note that the resource name can only include alphanumeric characters, `-`, and can't start or end with `-`. If any other symbols are included in the name, resource creation and migration will fail.
- **Azure Resource Group Name:** A custom resource group name that you choose from the drop-down list. Resource groups allow you to group Azure resources for access and management. If you currently do not have a resource group in your subscription, you will not be allowed to create one in the LUIS portal. Go to [Azure portal](#) to create one then go to LUIS to continue the sign-in process.

6. After you have successfully migrated in all regions, select finish. You will now have access to your applications. You can continue authoring and maintaining all your applications in all regions within the portal.

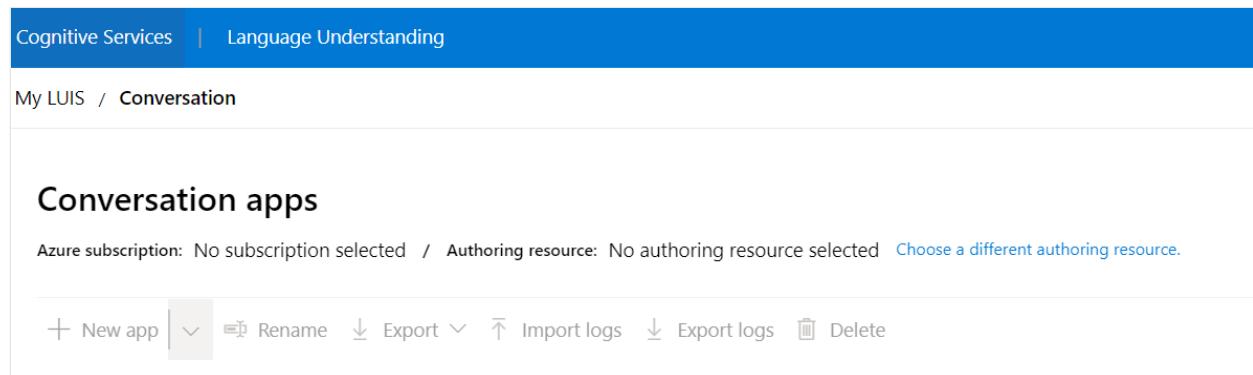
Migration notes

- Before migration, coauthors are known as *collaborators* on the LUIS app level. After migration, the Azure role of *contributor* is used for the same functionality on the Azure resource level.
- If you have signed-in to more than one [LUIS regional portal](#), you will be asked to migrate in multiple regions at once.
- Applications will automatically migrate with you if you're the owner of the application. Applications will not migrate with you if you're a collaborator on the application. However, collaborators will be prompted to export the apps they need.
- Application owners can't choose a subset of apps to migrate and there is no way for an owner to know if collaborators have migrated.
- Migration does not automatically move or add collaborators to the Azure authoring resource. The app owner is the one who needs to complete this step after migration. This step requires [permissions to the Azure authoring resource](#).

- After contributors are assigned to the Azure resource, they will need to migrate before they can access applications. Otherwise, they won't have access to author the applications.

Using apps after migration

After the migration process, all your LUIS apps for which you're the owner will now be assigned to a single LUIS authoring resource. The **My Apps** list shows the apps migrated to the new authoring resource. Before you access your apps, select **Choose a different authoring resource** to select the subscription and authoring resource to view the apps that can be authored.



The screenshot shows the Microsoft LUIS portal interface. At the top, there's a blue header bar with 'Cognitive Services' and 'Language Understanding' tabs. Below the header, the URL 'My LUIS / Conversation' is visible. The main content area is titled 'Conversation apps'. It displays a list of apps with a 'New app' button and various management icons like 'Rename', 'Export', 'Import logs', 'Export logs', and 'Delete'. A message at the top states 'Azure subscription: No subscription selected / Authoring resource: No authoring resource selected' with a link to 'Choose a different authoring resource'.

If you plan to edit your apps programmatically, you'll need the authoring key values. These values are displayed by clicking **Manage** at the top of the screen in the LUIS portal, and then selecting **Azure Resources**. They're also available in the Azure portal on the resource's **Key and endpoints** page. You can also create more authoring resources and assign them from the same page.

Adding contributors to authoring resources

If your apps need to be authored by other people, you need to add the associated email addresses in the Azure portal's authoring resource.

If you're the owner or administrator of your Azure subscription, you can add a contributor to the resource.

If you're not the owner or administrator of your Azure subscription, your Azure account needs to have `Microsoft.Authorization/roleAssignments/write` permissions.

If you have trouble with this role assignment, review:

- [Assign Azure roles](#)
- [Azure access control troubleshooting](#)

Learn [how to add contributors](#) on your authoring resource. Contributors will have access to all applications under that resource.

You can add contributors to the authoring resource from the Azure portal, on the **Access Control (IAM)** page for that resource. For more information, see [Add contributors to your app](#).

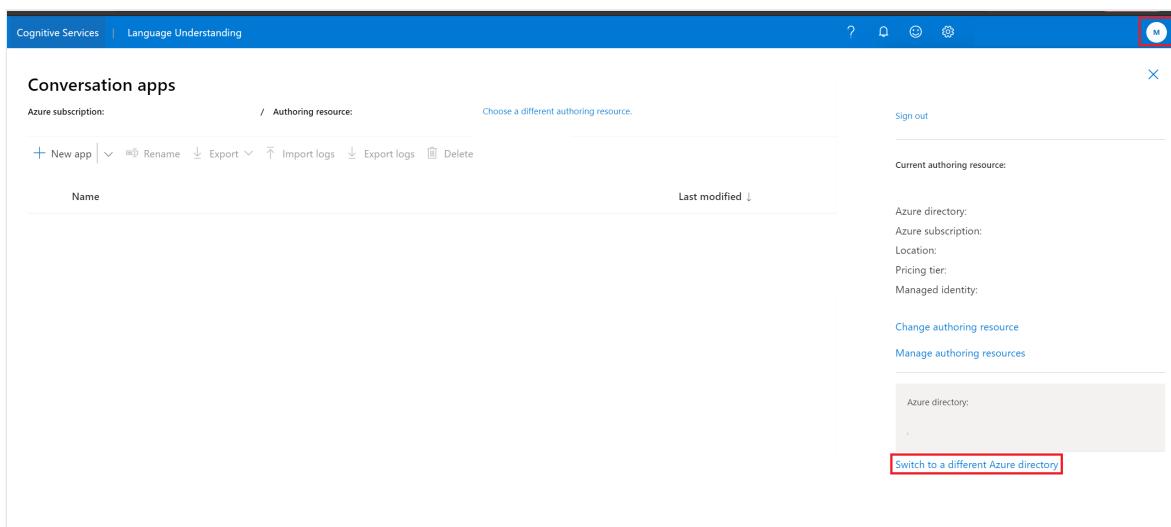
 **Note**

If the owner of the LUIS app migrated and added the collaborator as a contributor on the Azure resource, the collaborator will still have no access to the app unless they also migrate.

Troubleshooting the migration process

If you cannot find your Azure subscription in the drop-down list:

- Ensure that you have a valid Azure subscription that's authorized to create Azure AI services resources. Go to the [Azure portal](#) and check the status of the subscription. If you don't have one, [create a free Azure account](#).
- Ensure that you're in the proper tenant associated with your valid subscription. You can switch tenants selecting the avatar in the top right of the screen, containing your initials.



If you have an existing authoring resource but can't find it when you select the **Use Existing Authoring Resource** option:

- Your resource was probably created in a different region than the one you are trying to migrate in.

- Create a new resource from the LUIS portal instead.

If you select the **Create New Authoring Resource** option and migration fails with the error message "Failed retrieving user's Azure information, retry again later":

- Your subscription might have 10 or more authoring resources per region, per subscription. If that's the case, you won't be able to create a new authoring resource.
- Migrate by selecting the **Use Existing Authoring Resource** option and selecting one of the existing resources under your subscription.

Create new support request

If you are having any issues with the migration that are not addressed in the troubleshooting section, please [create a support topic](#) and provide the information below with the following fields:

- **Issue Type:** Technical
- **Subscription:** Choose a subscription from the dropdown list
- **Service:** Search and select "Azure AI services"
- **Resource:** Choose a LUIS resource if there is an existing one. If not, select General question.

Next steps

- Review [concepts about authoring and runtime keys](#)
- Review how to [assign keys](#) and [add contributors](#)

How to create and manage LUIS resources

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Use this article to learn about the types of Azure resources you can use with LUIS, and how to manage them.

Authoring Resource

An authoring resource lets you create, manage, train, test, and publish your applications. One [pricing tier](#) is available for the LUIS authoring resource - the free (F0) tier, which gives you:

- 1 million authoring transactions
- 1,000 testing prediction endpoint requests per month.

You can use the [v3.0-preview LUIS Programmatic APIs](#) to manage authoring resources.

Prediction resource

A prediction resource lets you query your prediction endpoint beyond the 1,000 requests provided by the authoring resource. Two [pricing tiers](#) are available for the prediction resource:

- The free (F0) prediction resource, which gives you 10,000 prediction endpoint requests monthly.
- Standard (S0) prediction resource, which is the paid tier.

You can use the [v3.0-preview LUIS Endpoint API](#) to manage prediction resources.

ⓘ Note

- You can also use a **multi-service resource** to get a single endpoint you can use for multiple Azure AI services.
- LUIS provides two types of F0 (free tier) resources: one for authoring transactions and one for prediction transactions. If you're running out of free quota for prediction transactions, make sure you're using the F0 prediction resource, which gives you a 10,000 free transactions monthly, and not the authoring resource, which gives you 1,000 prediction transactions monthly.
- You should author LUIS apps in the **regions** where you want to publish and query.

Create LUIS resources

To create LUIS resources, you can use the LUIS portal, [Azure portal](#), or Azure CLI. After you've created your resources, you'll need to assign them to your apps to be used by them.

LUIS portal

Create a LUIS authoring resource using the LUIS portal

1. Sign in to the [LUIS portal](#), select your country/region and agree to the terms of use. If you see the **My Apps** section in the portal, a LUIS resource already exists and you can skip the next step.
2. In the **Choose an authoring** window that appears, find your Azure subscription, and LUIS authoring resource. If you don't have a resource, you can create a new one.

Choose an authoring resource

X

Switching your authoring resource will also switch to your LUIS apps. You can switch back at any time. [Learn more about LUIS resources in Azure.](#)

Azure directory ?

Microsoft

Azure Subscription *

Select azure subscription ...

LUIS authoring resource* ?

Select a LUIS authoring resource ...

[Create a new LUIS authoring resource](#)

Done

Cancel

When you create a new authoring resource, provide the following information:

- **Tenant name:** the tenant your Azure subscription is associated with.
- **Azure subscription name:** the subscription that will be billed for the resource.
- **Azure resource group name:** a custom resource group name you choose or create. Resource groups allow you to group Azure resources for access and management.
- **Azure resource name:** a custom name you choose, used as part of the URL for your authoring and prediction endpoint queries.
- **Pricing tier:** the pricing tier determines the maximum transaction per second and month.

Create a LUIS Prediction resource using the LUIS portal

1. Go to the **Manage** tab in your application.
2. In the left navigation, select **Azure Resources**.
3. select **Add Prediction resource** button.

The screenshot shows the LUIS Azure Resources page. At the top, there are tabs for DASHBOARD, BUILD, and MANAGE, with MANAGE being the active tab and highlighted with a red box. Below the tabs, there's a sidebar with options: Settings, Publish Settings, Azure Resources (which is also highlighted with a red box), and Versions. The main content area is titled "Azure Resources" and contains a sub-section "Prediction Resources". It says, "There are no prediction resources assigned. Click on the 'Add prediction resource' button to assign one to the application." A blue button labeled "Add prediction resource" is visible. On the right side, there's a search icon.

If you already have a prediction resource, add it. Otherwise, select **Create a new prediction resource**.

The screenshot shows the "Add a prediction resource" dialog box. At the top, it says "Add a prediction resource" and has a close button. The main text reads: "Your application needs a Language Understanding resource for you to access your language model. [Learn more about resources in Azure.](#)" Below this, there's a section for "Azure directory" with a dropdown menu showing "Microsoft" (which is highlighted with a gray background). A note says, "Note: To switch to another directory, use the top right user avatar." Under "Azure subscription *", there's a dropdown menu showing "Loading ...". At the bottom, there's a red button labeled "Create a new prediction resource". To the right of the button are "Done" and "Cancel" buttons, with the "Cancel" button having a magnifying glass icon.

Assign LUIS resources

Creating a resource doesn't necessarily mean that it is put to use, you need to assign it to your apps. You can assign an authoring resource for a single app or for all apps in LUIS.

The screenshot shows a section titled "Assign resources using the LUIS portal". At the top left, there's a button labeled "LUIS portal". The main content area is currently empty, indicated by a large white space.

Assign an authoring resource to all your apps

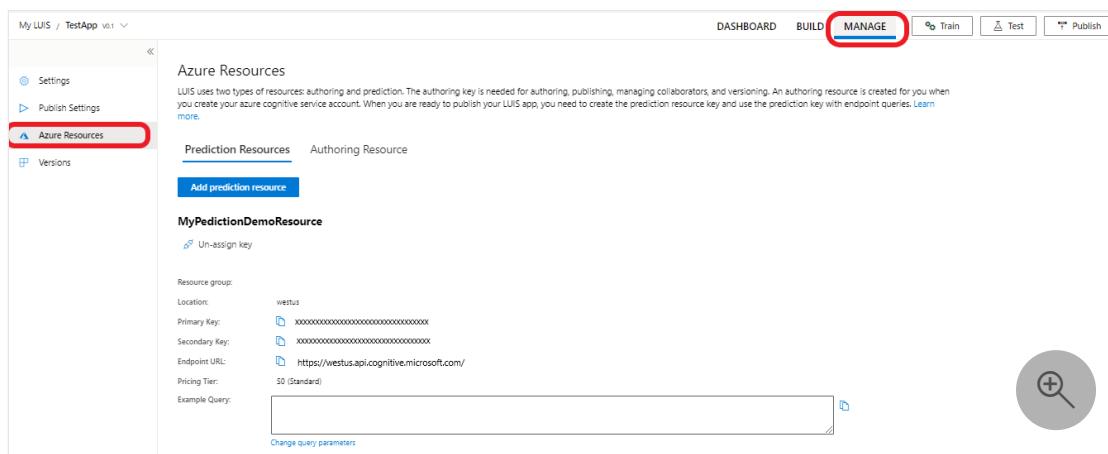
The following procedure assigns the authoring resource to all apps.

1. Sign in to the [LUIS portal](#).
2. In the upper-right corner, select your user account, and then select **Settings**.
3. On the **User Settings** page, select **Add authoring resource**, and then select an existing authoring resource. Select **Save**.

Assign a resource to a specific app

The following procedure assigns a resource to a specific app.

1. Sign in to the [LUIS portal](#). Select an app from the **My apps** list.
2. Go to **Manage > Azure Resources**:



3. On the **Prediction resource** or **Authoring resource** tab, select the **Add prediction resource** or **Add authoring resource** button.
4. Use the fields in the form to find the correct resource, and then select **Save**.

Unassign a resource

When you unassign a resource, it's not deleted from Azure. It's only unlinked from LUIS.

LUIS portal

Unassign resources using LUIS portal

1. Sign in to the [LUIS portal](#), and then select an app from the **My apps** list.

2. Go to Manage > Azure Resources.
3. Select the **Unassign resource** button for the resource.

Resource ownership

An Azure resource, like a LUIS resource, is owned by the subscription that contains the resource.

To change the ownership of a resource, you can take one of these actions:

- Transfer the [ownership](#) of your subscription.
- Export the LUIS app as a file, and then import the app on a different subscription.
Export is available on the [My apps](#) page in the LUIS portal.

Resource limits

Authoring key creation limits

You can create as many as 10 authoring keys per region, per subscription. Publishing regions are different from authoring regions. Make sure you create an app in the authoring region that corresponds to the publishing region where you want your client application to be located. For information on how authoring regions map to publishing regions, see [Authoring and publishing regions](#).

See [resource limits](#) for more information.

Errors for key usage limits

Usage limits are based on the pricing tier.

If you exceed your transactions-per-second (TPS) quota, you receive an HTTP 429 error. If you exceed your transaction-per-month (TPM) quota, you receive an HTTP 403 error.

Change the pricing tier

1. In [the Azure portal](#), Go to All resources and select your resource

The screenshot shows the Azure portal's 'All resources' page. At the top, there are filters for 'Subscription == 3 of 14 selected', 'Resource group == all', 'Type == all', and 'Location == all'. Below the filters, it says 'Showing 1 to 1 of 1 records.' The results table has columns for Name, Type, Resource group, Location, and Subscription. A single row is selected, highlighted with a red border. The row contains 'MyDemoResource' under Name, 'Cognitive Services' under Type, 'Maged' under Resource group, 'West US' under Location, and 'Luis Internal' under Subscription.

2. From the left side menu, select **Pricing tier** to see the available pricing tiers
3. Select the pricing tier you want, and click **Select** to save your change. When the pricing change is complete, a notification will appear in the top right with the pricing tier update.

View Azure resource metrics

View a summary of Azure resource usage

You can view LUIS usage information in the Azure portal. The **Overview** page shows a summary, including recent calls and errors. If you make a LUIS endpoint request, allow up to five minutes for the change to appear.



Customizing Azure resource usage charts

The **Metrics** page provides a more detailed view of the data. You can configure your metrics charts for a specific **time period** and **metric**.

Home > All resources > cahann-authoring-wus

Cognitive Services

Search (Ctrl+ /)

Identity

Billing By Subscription

Properties

Locks

Monitoring

Alerts

Metrics

Diagnostic settings

Logs

Automation

Tasks (preview)

Export template

Support + troubleshooting

Resource health

Metrics

Line chart

Drill into Logs

New alert rule

Pin to dashboard

Local Time: Last 24 hours (Automatic)

Scope Metric Namespace Metric Aggregation

100

90

80

70

60

50

40

30

20

10

n

Select a metric above to see data appear on this chart or learn more below:

Filter + Split

Plot multiple metrics

Build custom dashboards

Total transactions threshold alert

If you want to know when you reach a certain transaction threshold, for example 10,000 transactions, you can create an alert:

1. From the left side menu, select Alerts
2. From the top menu select New alert rule

Home > MyDemoResource

MyDemoResource | Alerts

Search (Ctrl+ /)

+ New alert rule

Manage alert rules

Manage actions

View classic alerts

Refresh

Feedback

Subscription : Resource group : Time range : Past 24 hours Resource : mydemoresource

Pay attention to what matters.

You have not configured any alert rules.

Configure alert rules and attend to fired alerts to efficiently monitor your Azure resources. [Learn more](#)

mydemoresource

3. Select Add condition

Home > MyDemoResource >

Create alert rule

Create an alert rule to identify and address issues when important conditions are found in your monitoring data. [View tutorial + read more](#)

When defining the alert rule, check that your inputs do not contain any sensitive content.

Scope

Select the target resource you wish to monitor.

Resource

mydemoresource

Hierarchy

Edit resource

Condition

Configure when the alert rule should trigger by selecting a signal and defining its logic.

Condition name

No condition selected yet

Add condition

4. Select Total calls

The screenshot shows the 'Create alert rule' wizard. On the left, there's a sidebar with 'Resource' set to 'mydemoresource'. Under 'Condition', it says 'No condition selected yet' and 'Add condition'. Under 'Actions', it says 'No action group selected yet' and 'Create alert rule'. On the right, the 'Select a signal' step is shown. It has filters for 'Signal type: All' and 'Monitor service: All'. Below is a table titled 'Displaying 1 - 14 signals out of total 14 signals'. The 'Total Calls' signal is highlighted with a red box and a magnifying glass icon.

Signal name	Signal type	Monitor service
Blocked Calls	Metric	Platform
Client Errors	Metric	Platform
Data In	Metric	Platform
Data Out	Metric	Platform
Latency	Metric	Platform
Server Errors	Metric	Platform
Successful Calls	Metric	Platform
Total Calls	Metric	Platform
Total Errors	Metric	Platform
All Administrative operations	Activity Log	Administrative

5. Scroll down to the **Alert logic** section and set the attributes as you want and click **Done**

The screenshot shows the 'Configure signal logic' page. At the top, it says 'Use dimensions to monitor specific time series. If you select more than one dimension value, each time series that results from the combination will trigger its own alert and will be charged separately. [About monitoring multiple time series](#)'.

Under 'Alert logic', there are sections for 'Threshold' (Static selected), 'Operator' (Greater than or equal to), 'Aggregation type' (Total), 'Threshold value' (10000), and 'Unit' (Count). A red box highlights these four fields. Below this, there's a 'Condition preview' stating 'Whenever the total total calls is greater than or equal to 10000'. There's also a 'Evaluated based on' section with 'Aggregation granularity (Period)' set to '5 minutes' and 'Frequency of evaluation' set to 'Every 1 Minute'.

6. To send notifications or invoke actions when the alert rule triggers go to the **Actions** section and add your action group.

The screenshot shows the 'Create alert rule' interface in the Azure portal. At the top, there's a breadcrumb navigation: Home > MyDemoResource >. Below it, a 'Create alert rule' button and three dots for more options. The main area is divided into two sections: 'Condition' and 'Actions'.
Condition: A table with columns: Condition name, Time series monitored, and Estimated monthly cost (USD). It lists one condition: 'Whenever the total total calls is greater than or equal to 10000' (Time series monitored: 1, Cost: \$1). There are 'Add condition' and 'Delete' buttons.
Actions: A table with columns: Action group name and Contains actions. It shows 'No action group selected yet'. There is an 'Add action groups' button, which is highlighted with a red rectangle. To the right of the table is a circular icon with a magnifying glass and a plus sign.

Reset an authoring key

For [migrated authoring resource](#) apps: If your authoring key is compromised, reset the key in the Azure portal, on the [Keys](#) page for the authoring resource.

For apps that haven't been migrated: 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.

Regenerate an Azure key

You can regenerate an Azure key from the [Keys](#) page in the Azure portal.

App ownership, access, and security

An app is defined by its Azure resources, which are determined by the owner's subscription.

You can move your LUIS app. Use the following resources to help you do so by using the Azure portal or Azure CLI:

- [Move an app between LUIS authoring resources ↗](#)
- [Move a resource to a new resource group or subscription](#)
- [Move a resource within the same subscription or across subscriptions](#)

Next steps

- Learn [how to use versions](#) to control your app life cycle.

Application and version settings

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Configure your application settings in the LUIS portal such as utterance normalization and app privacy.

View application name, description, and ID

You can edit your application name, and description. You can copy your App ID. The culture can't be changed.

1. Sign into the [LUIS portal](#).
2. Select an app from the **My apps** list.
3. Select **Manage** from the top navigation bar, then **Settings** from the left navigation bar.

The screenshot shows the LUIS portal interface. At the top, there are navigation links: Cognitive Services, Language Understanding, and My apps. On the far right are icons for settings, notifications, help, and a user profile. Below the header, the app name "Pizzav6 (V1.2)" is displayed, along with tabs for DASHBOARD, BUILD, MANAGE, Train (highlighted in yellow), Publish, and Test. The left sidebar has a "Settings" section with options for Publish Settings, Azure Resources, and Versions. The main content area is titled "Application Settings". It includes fields for "App name" (Pizzav6), "App description (optional)" (a placeholder text input field), "App ID" (YOUR-APP-ID), "Culture" (en-us), and a toggle switch for "Make endpoints public" which is currently off. Below this is a "Version Settings" section with three toggles: "Use non-deterministic training" (On, recommended), "Normalize punctuation" (Off), and "Normalize word forms" (Off).

Change application settings

To change a setting, select the toggle on the page.

Use [app ↗](#) APIs to update settings or use the LUIS portal's **Manage** section, **Settings** page.

UI setting	API setting	Information
Make endpoints public	Public	Anyone can access your public app' prediction endpoint if they have a prediction key, app ID, and version ID.

Change version settings

To change a setting, select the toggle on the page.

Learn [concepts](#) of normalization and how to use [version ↗](#) APIs to update these settings or use the LUIS portal's **Manage** section, **Settings** page.

UI setting	API setting	Information
Use non-deterministic training	<code>UseAllTrainingData</code>	Training uses a small percentage of negative sampling. If you want to use all data instead of the small negative sampling, set to <code>true</code> .
Normalize diacritics	<code>NormalizeDiacritics</code>	Normalizing diacritics replaces the characters with diacritics in utterances with regular characters. This setting is only available on languages that support diacritics.
Normalize punctuation	<code>NormalizePunctuation</code>	Normalizing punctuation means that before your models get trained and before your endpoint queries get predicted, punctuation will be removed from the utterances.
Normalize word forms	<code>NormalizeWordForm</code>	Ignore word forms beyond root.

Next steps

- How to [collaborate](#) with other authors
- [Publish settings](#)

Add contributors to your app

Article • 10/12/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

An app owner can add contributors to apps. These contributors can modify the model, train, and publish the app. Once you have [migrated](#) your account, *contributors* are managed in the Azure portal for the authoring resource, using the **Access control (IAM)** page. Add a user, using the collaborator's email address and the *contributor* role.

Add contributor to Azure authoring resource

You have migrated if your LUIS authoring experience is tied to an Authoring resource on the [Manage -> Azure resources](#) page in the LUIS portal.

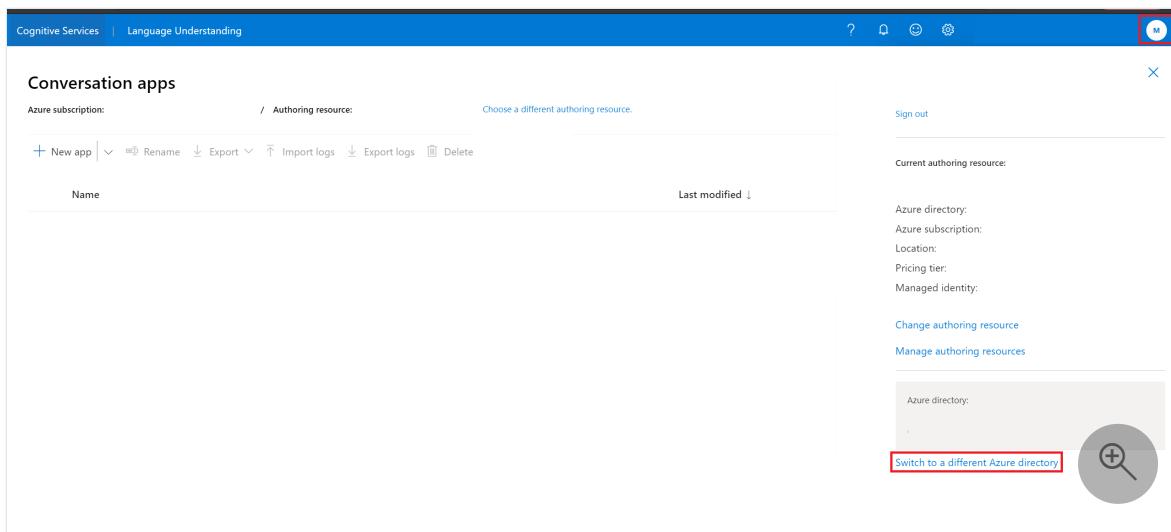
In the Azure portal, find your Language Understanding (LUIS) authoring resource. It has the type `LUIS.Authoring`. In the resource's **Access Control (IAM)** page, add the role of **contributor** for the user that you want to contribute. For detailed steps, see [Assign Azure roles using the Azure portal](#).

View the app as a contributor

After you have been added as a contributor, [sign in to the LUIS portal](#).

If you don't see an app that was created by you or shared with you, you may need to switch to a different Azure directory.

1. Select the avatar in the top right corner of the screen. Then click [Switch to a different Azure directory](#).
2. In the window that appears, be sure to select the Azure directory that contains the LUIS resource that was shared with you.



Users with multiple emails

If you add contributors to a LUIS app, you are specifying the exact email address. While Microsoft Entra ID 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 when adding the contributor.

Microsoft Entra resources

If you use [Microsoft Entra ID](#) (Microsoft Entra ID) 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.

Microsoft Entra tenant user

LUIS uses standard Microsoft Entra consent flow.

The tenant admin should work directly with the user who needs access granted to use LUIS in the Microsoft Entra ID.

- 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. On this page you can filter the list to items that include the name LUIS.

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 Microsoft Entra ID), 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 the [Microsoft Entra identity and access management API in Microsoft Graph](#) to provide consent to each specific user.

Learn more about Microsoft Entra users and consent:

- [Restrict your app](#) to a set of users

Next steps

- Learn [how to use versions](#) to control your app life cycle.
- Understand the about [authoring resources](#) and [adding contributors](#) on that resource.
- Learn [how to create](#) authoring and runtime resources
- Migrate to the new [authoring resource](#)

Use versions to edit and test without impacting staging or production apps

Article • 07/18/2023

Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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.

The active version is the version you are editing in the LUIS portal **Build** section with intents, entities, features, and patterns. When using the authoring APIs, you don't need to set the active version because the version-specific REST API calls include the version in the route.

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.

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".

Clone version

New version name (Required)

Done

Cancel

! 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 **Activate** from the toolbar.

Import version

You can import a `.json` or a `.lu` version of your application.

1. Select **Import** from the toolbar, then select the format.
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 file already exists in the app.

Import new version

Version file (Required)

No file chosen

Version Name (Optional)

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 or LU to export for a backup or to save in source control, choose **Export for container** to [use this app in a LUIS container](#).

See also

See the following links to view the REST APIs for importing and exporting applications:

- [Importing applications ↗](#)
- [Exporting applications ↗](#)

Use Microsoft Azure Traffic Manager to manage endpoint quota across keys

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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.

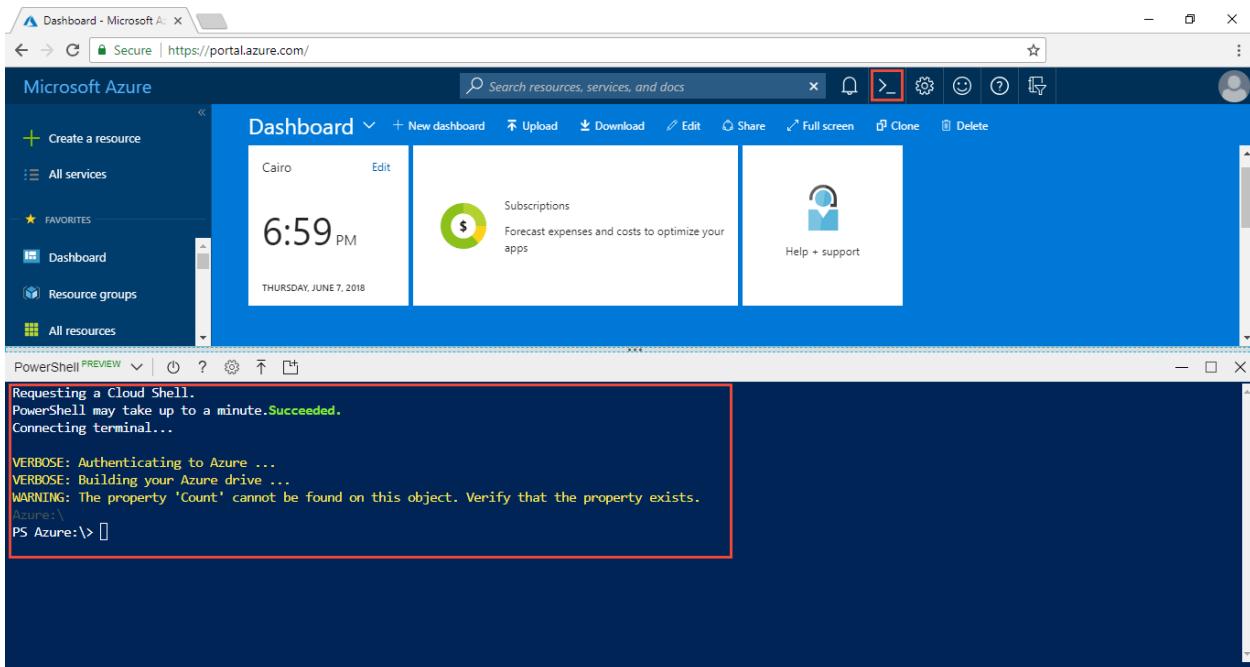
This article explains how to manage the traffic across keys with Azure [Traffic Manager](#). You must already have a trained and published LUIS app. If you do not have one, follow the Prebuilt domain [quickstart](#).

ⓘ Note

We recommend that you use the Azure Az PowerShell module to interact with Azure. See [Install Azure PowerShell](#) to get started. To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

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:

powerShell

```
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
<input type="checkbox"/> luis-tm-east	Cognitive Services	luis-traffic-manager	East US	Pay-As-You-Go
<input type="checkbox"/> 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 **Azure Resources** 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.

powerShell

```
$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

Configuration parameter	Variable name or Value	Purpose
-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	HTTPS	Port and protocol for LUIS is
-MonitorPort	443	HTTPS/443
-MonitorPath	/luis/v2.0/apps/<appIdLuis>?subscription-key=<subscriptionKeyLuis>&q=traffic-manager-east	Replace <appIdLuis> and <subscriptionKeyLuis> with your own values.

A successful request has no response.

2. Add East US endpoint with [Add-AzTrafficManagerEndpointConfig](#) cmdlet

powerShell

```
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:

Console

```

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

powerShell

```
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.

powerShell

```
$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

Configuration parameter	Variable name or Value	Purpose
		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	HTTPS	Port and protocol for LUIS is
-MonitorPort	443	HTTPS/443
-MonitorPath	/luis/v2.0/apps/<appIdLuis>?subscription-key=<subscriptionKeyLuis>&q=traffic-manager-west	Replace <appId> and <subscriptionKey> 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

powerShell

```
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
-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

Configuration parameter	Variable name or Value	Purpose
-EndpointStatus	Enabled	Enable endpoint when it is created

The successful response looks like:

```
Console

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

powerShell

```
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

powerShell

```
$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
-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	HTTPS	Port and protocol for LUIS is HTTPS/443
-MonitorPort	443	
-MonitorPath	/	This path doesn't matter because the child endpoint paths are used instead.

A successful request has no response.

2. Add East US child profile to parent with [Add-AzTrafficManagerEndpointConfig](#) and **NestedEndpoints** type

powerShell

```
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:

Console

```

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

powerShell

```
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:

Console

```
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

```
powerShell
```

```
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.

```
powerShell
```

```
$<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.

The screenshot shows the Microsoft Azure 'All resources' blade. In the center, there's a table listing five resources under the 'Subscriptions: Pay-As-You-Go' category. The columns are NAME, TYPE, RESOURCE GROUP, LOCATION, and SUBSCRIPTION. The first three items in the list ('luis-profile-eastus', 'luis-profile-parent', and 'luis-profile-westus') are highlighted with a red box.

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.

The screenshot shows the 'luis-profile-eastus' profile details in the Azure portal. On the right, the 'Essentials' section displays the DNS name as <http://luis-dns-eastus.trafficmanager.net> and the 'Monitor status' as **Online**, which is highlighted with a red box. On the left, the 'SETTINGS' sidebar lists various configuration options like Configuration, Real user measurements, Traffic view, Endpoints, Properties, Locks, and Automation script.

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-`.

Console

```
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:

JavaScript

```
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:

JSON

```
[  
  {  
    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.

Install and run Docker containers for LUIS

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

ⓘ Note

The container image location has recently changed. Read this article to see the updated location for this container.

Containers enable you to use LUIS in your own environment. Containers are great for specific security and data governance requirements. In this article you'll learn how to download, install, and run a LUIS container.

The Language Understanding (LUIS) container loads your trained or published Language Understanding model. As a [LUIS app](#), the docker container provides access to the query predictions from the container's API endpoints. You can collect query logs from the container and upload them 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

To run the LUIS container, note the following prerequisites:

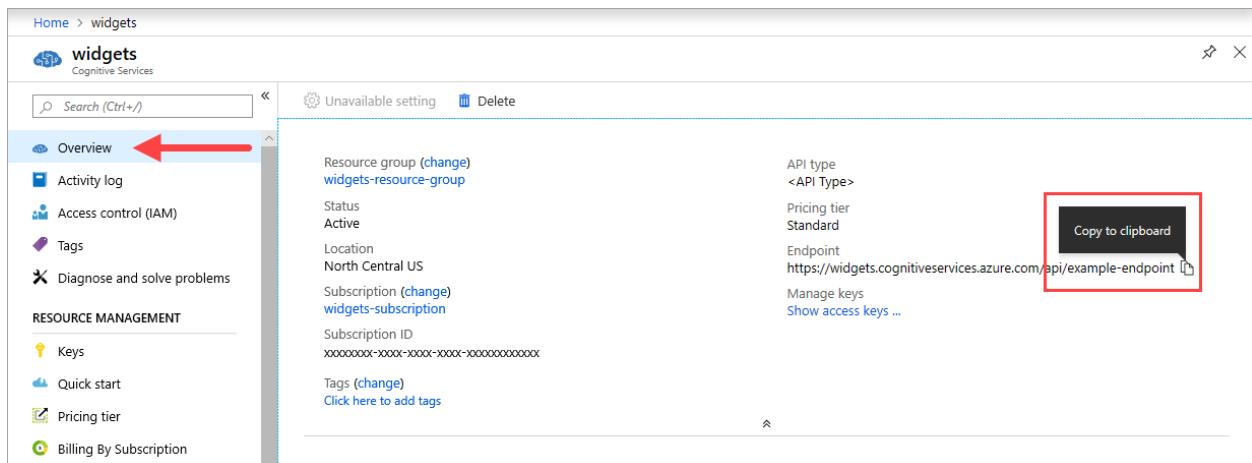
- Docker [installed](#) on a host computer. Docker must be configured to allow the containers to connect with and send billing data to Azure.
 - On Windows, Docker must also be configured to support Linux containers.
 - You should have a basic understanding of [Docker concepts](#).
- A [LUIS resource](#) with the free (F0) or standard (S) [pricing tier](#).
- 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 *Authoring Key*.

Gather required parameters

Three primary parameters for all Azure AI services containers are required. The Microsoft Software License Terms must be present with a value of **accept**. An Endpoint URI and API key are also needed.

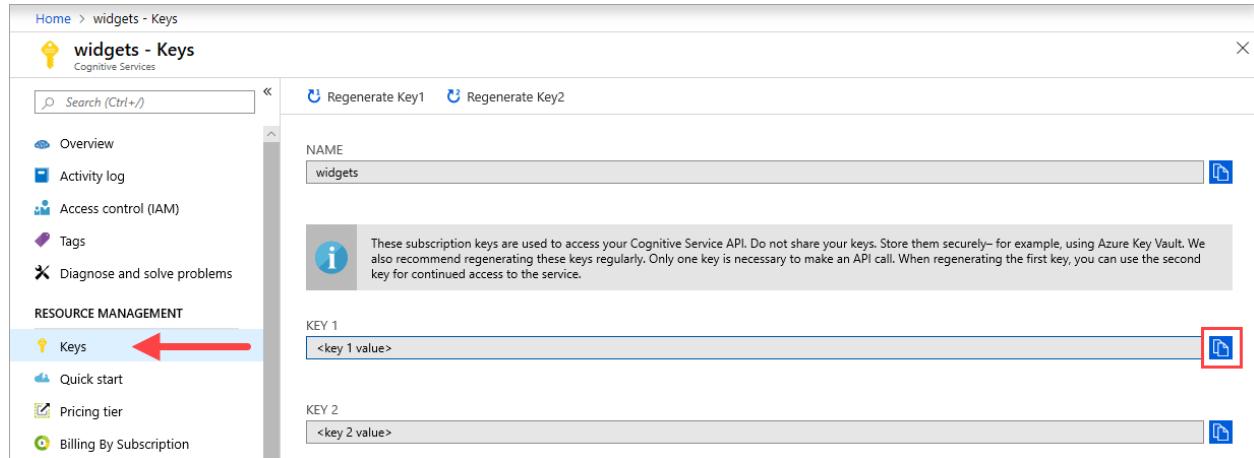
Endpoint URI

The `{ENDPOINT_URI}` value is available on the Azure portal **Overview** page of the corresponding Azure AI services resource. Go to the **Overview** page, hover over the endpoint, and a **Copy to clipboard** icon appears. Copy and use the endpoint where needed.



Keys

The `{API_KEY}` value is used to start the container and is available on the Azure portal's **Keys** page of the corresponding Azure AI services resource. Go to the **Keys** page, and select the **Copy to clipboard**  icon.



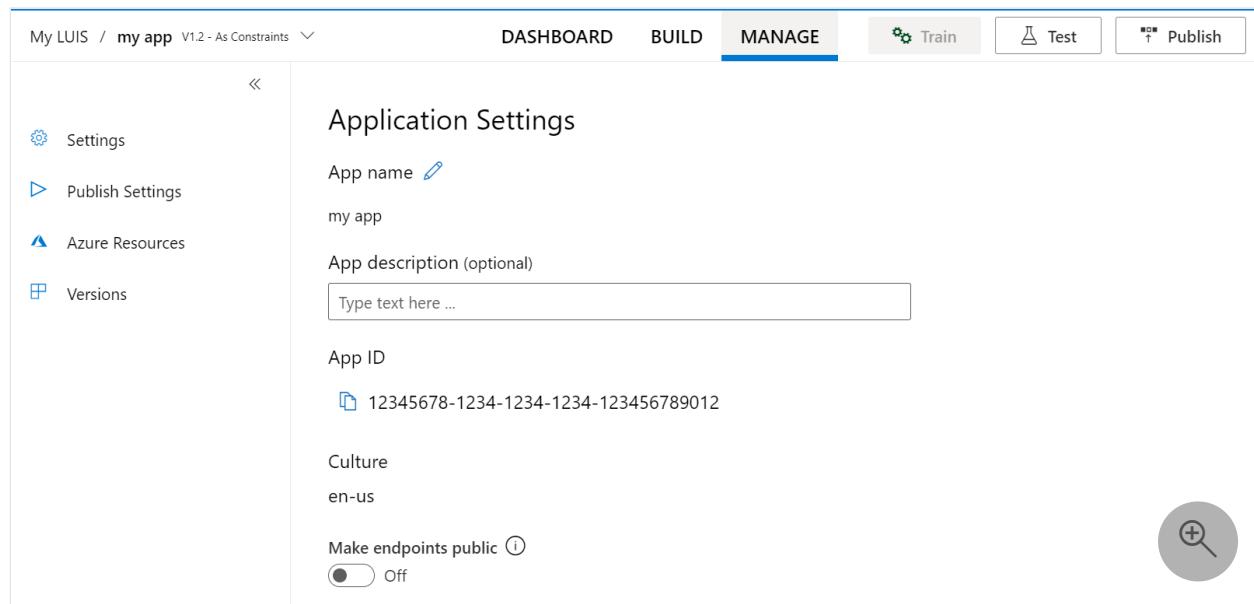
A screenshot of the Azure portal showing the 'Keys' page for a Cognitive Services resource named 'widgets'. The left sidebar shows navigation options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, RESOURCE MANAGEMENT, and Keys. A red arrow points to the 'Keys' option. The main area displays a table with two rows: 'NAME' (widgets) and 'KEY 1' (<key 1 value>). An information box states: 'These subscription keys are used to access your Cognitive Service API. Do not share your keys. Store them securely—for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.' The 'KEY 1' value is selected, and its copy icon is highlighted with a red box.

Important

These subscription keys are used to access your Azure AI services API. Don't share your keys. Store them securely. For example, use Azure Key Vault. We also recommend that you regenerate these keys regularly. Only one key is necessary to make an API call. When you regenerate the first key, you can use the second key for continued access to the service.

App ID `{APP_ID}`

This ID is used to select the app. You can find the app ID in the [LUIS portal](#) by clicking **Manage** at the top of the screen for your app, and then **Settings**.



A screenshot of the LUIS portal showing the Application Settings page for an app named 'my app'. The left sidebar lists settings, publish settings, Azure resources, and versions. The main area shows the 'Application Settings' section with fields for 'App name' (my app), 'App description (optional)' (Type text here ...), 'App ID' (12345678-1234-1234-1234-123456789012), 'Culture' (en-us), and 'Make endpoints public' (Off). A large '+' button is located in the bottom right corner.

Authoring key {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. You will need your authoring key if you [export your app using the REST API](#), described later in the article.

You can get your authoring key from the [LUIS portal](#) by clicking **Manage** at the top of the screen for your app, and then **Azure Resources**.

The screenshot shows the LUIS portal interface. At the top, there's a navigation bar with 'My LUIS / my app V1.2 - As Constraints'. Below it are tabs: DASHBOARD, BUILD, MANAGE (which is highlighted), Train, Test, and Publish. On the left, a sidebar has links for Settings, Publish Settings, Azure Resources (which is expanded), and Versions. The main content area is titled 'Azure Resources' and explains that LUIS uses two types of resources: authoring and prediction. It says the authoring key is needed for publishing, managing collaborators, and versioning. It also mentions creating a prediction resource key. A 'Change authoring resource' button is visible. Below this, a section titled 'aahi-luis-test-resource-Authoring' lists resource details: Resource group: my-resource, Location: westus, Primary Key: (redacted), Secondary Key: (redacted), Endpoint URL: https://my-resource.cognitiveservices.azure.com/, and Pricing Tier: F0 (Free). A magnifying glass icon is in the bottom right corner of the content area.

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 an 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

The below table lists minimum and recommended values for the container host. Your requirements may change depending on traffic volume.

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`

The LUIS container image can be found on the `mcr.microsoft.com` container registry syndicate. It resides within the `azure-cognitive-services/language` repository and is named `luis`. The fully qualified container image name is, `mcr.microsoft.com/azure-cognitive-services/language/luis`.

To use the latest version of the container, you can use the `latest` tag. You can also find a full list of tags on the [MCR](#).

Use the [`docker pull`](#) command to download a container image from the `mcr.microsoft.com/azure-cognitive-services/language/luis` repository:

```
docker pull mcr.microsoft.com/azure-cognitive-services/language/luis:latest
```

For a full description of available tags, such as `latest` used in the preceding command, see [LUIS](#) on Docker Hub.

Tip

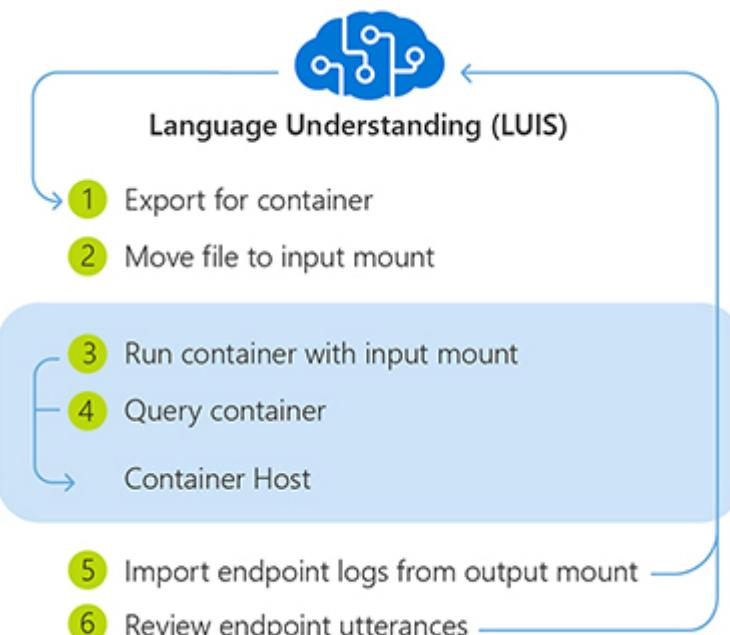
You can use the [`docker images`](#) command to list your downloaded container images. For example, the following command lists the ID, repository, and tag of each downloaded container image, formatted as a table:

```
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 the 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 **Versioned** models of the app simultaneously. All the packages are mounted.

Package Type	Query Endpoint API	Query availability	Package filename format
Versioned	GET, POST	Container only	{APP_ID}_v{APP_VERSION}.gz
Staging	GET, POST	Azure and container	{APP_ID}_STAGING.gz
Production	GET, POST	Azure and container	{APP_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 AI services resource instance	Supported regions include West US (<code>westus</code>)

Packaging Requirements	Details
	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 'My Apps' list page. At the top, there are buttons for 'Rename', 'Export', 'Export endpoint logs', and 'Delete'. Below this, a table lists an app entry. The app name is 'human resources with' and it has a 'Culture' set to 'Production slot'. To the left of the app name, there are checkboxes for 'Name' and 'Human Resources with Key Phrase entity'. A contextual toolbar is open above the table, with 'Export for container (GZIP)' highlighted in blue. Other options in the toolbar include 'Export as JSON' and 'Culture'. The table also includes columns for 'Created date' (5/3/18) and 'Endpoint hits' (0).

Export versioned app's package from LUIS portal

The versioned 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 'Versions' section of the LUIS portal. A table lists a single version: '0.1 (Active & Production)' created on '5/3/18' and last modified on '9/6/18'. The 'Export' menu is open, with 'Export for container (GZIP)' highlighted. Other options like 'Export as JSON' and 'Clone' are also visible.

Version name	Created	Last modified
0.1 (Active & Production)	5/3/18	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.

HTTP

```
GET /luis/api/v2.0/package/{APP_ID}/slot/{SLOT_NAME}/gzip HTTP/1.1
Host: {AZURE_REGION}.api.cognitive.microsoft.com
Ocp-Apim-Subscription-Key: {AUTHORING_KEY}
```

Placeholder	Value
{APP_ID}	The application ID of the published LUIS app.
{SLOT_NAME}	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.
{AZURE_REGION}	The appropriate Azure region: westus - West US westeurope - West Europe australiaeast - Australia East

To download the published package, 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 versioned 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.

HTTP

```
GET /luis/api/v2.0/package/{APP_ID}/versions/{APP_VERSION}/gzip HTTP/1.1
Host: {AZURE_REGION}.api.cognitive.microsoft.com
Ocp-Apim-Subscription-Key: {AUTHORING_KEY}
```

Placeholder	Value
{APP_ID}	The application ID of the trained LUIS app.
{APP_VERSION}	The application version of the trained LUIS app.
{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: <code>westus</code> - West US <code>westeurope</code> - West Europe <code>australiaeast</code> - Australia East

To download the versioned package, 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. Refer to [gather required parameters](#) for details on how to get the `{ENDPOINT_URI}` and `{API_KEY}` values.

[Examples](#) of the `docker run` command are available.

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/language/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 **Azure Resources** page in the LUIS portal and is also available on the Azure [Azure AI 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](#) 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 versioned apps.

Use the host, `http://localhost:5000`, for container APIs.

V3 prediction endpoint				
Package type	HTTP verb	Route	Query parameters	
Published	GET,	/luis/v3.0/apps/{appId}/slots/{slotName}/predict?	query=	{query} [&verbose] [&log] [&show-all-intents]
	POST	/luis/prediction/v3.0/apps/{appId}/slots/{slotName}/predict?		
Versioned	GET,	/luis/v3.0/apps/{appId}/versions/{versionId}/predict?	query=	{query} [&verbose] [&log] [&show-all-intents]
	POST	/luis/prediction/v3.0/apps/{appId}/versions/{versionId}/predict?		

The query parameters configure how and what is returned in the query response:

Query parameter	Type	Purpose
query	string	The user's utterance.
verbose	boolean	A boolean value indicating whether to return all the metadata

Query parameter	Type	Purpose
		for the predicted models. Default is false.
<code>log</code>	boolean	Logs queries, which can be used later for active learning . Default is false.
<code>show-all-intents</code>	boolean	A boolean value indicating whether to return all the intents or the top scoring intent only. Default is false.

Query the LUIS app

An example CURL command for querying the container for a published app is:

V3 prediction endpoint

To query a model in a slot, use the following API:

Bash

```
curl -G \
-d verbose=false \
-d log=true \
--data-urlencode "query=turn the lights on" \
"http://localhost:5000/luis/v3.0/apps/{APP_ID}/slots/production/predict"
```

To make queries to the **Staging** environment, replace `production` in the route with `staging`:

```
http://localhost:5000/luis/v3.0/apps/{APP_ID}/slots/staging/predict
```

To query a versioned model, use the following API:

Bash

```
curl -G \
-d verbose=false \
-d log=false \
--data-urlencode "query=turn the lights on" \
"http://localhost:5000/luis/v3.0/apps/{APP_ID}/versions/{APP_VERSION}/predict"
```

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.



From the LUIS portal, select your app, then select **Import endpoint logs** to upload these logs.

A screenshot of the LUIS portal's "My Apps" page. The page title is "My Apps". Below it is a toolbar with buttons for "Rename", "Export", "Import endpoint logs" (which is highlighted with a red box), "Export endpoint logs", and "Delete". Below the toolbar is a table with columns for "Name", "Culture", and "Created date". A single row is selected, showing "new-prebuilt (v 0.1)" in the Name column, "en-us" in the Culture column, and "9/20/18" in the Created date column.

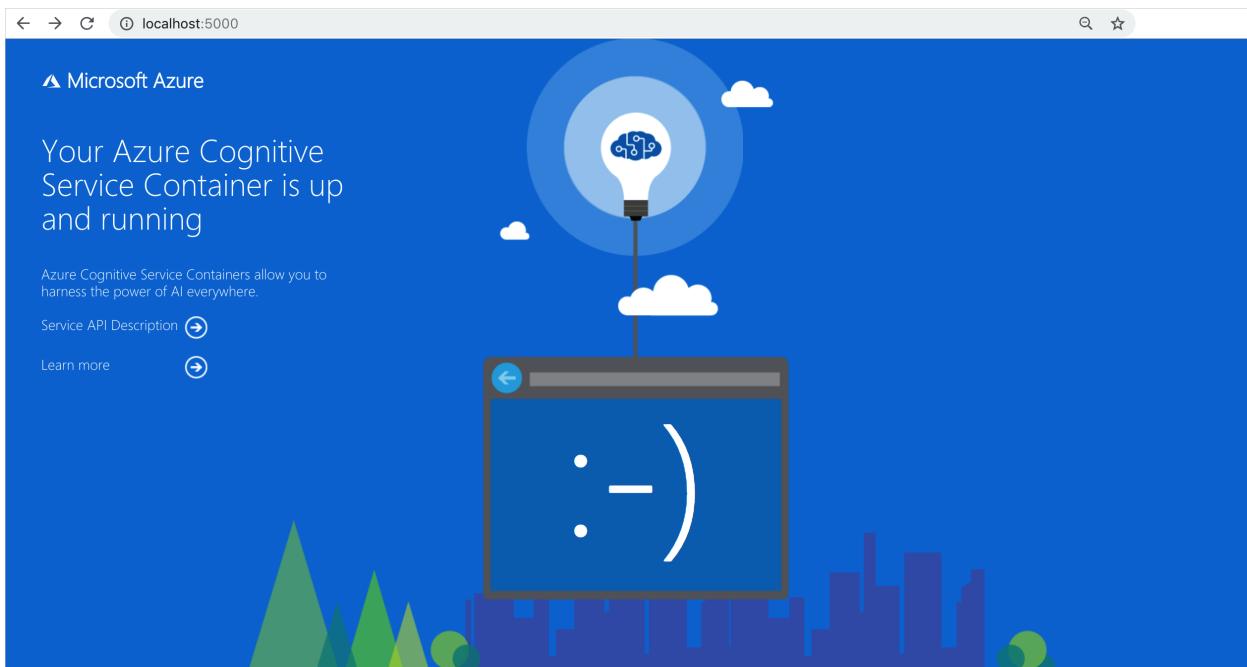
After the log is uploaded, [review the endpoint](#) utterances in the LUIS portal.

Validate that a container is running

There are several ways to validate that the container is running. Locate the *External IP* address and exposed port of the container in question, and open your favorite web browser. Use the various request URLs that follow to validate the container is running. The example request URLs listed here are `http://localhost:5000`, but your specific container might vary. Make sure to rely on your container's *External IP* address and exposed port.

Request URL	Purpose
<code>http://localhost:5000/</code>	The container provides a home page.
<code>http://localhost:5000/ready</code>	Requested with GET, this URL provides a verification that the container is ready to accept a query against the model. This request can be used for Kubernetes liveness and readiness probes .

Request URL	Purpose
<code>http://localhost:5000/status</code>	Also requested with GET, this URL verifies if the api-key used to start the container is valid 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 Try it out 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.



Run the container disconnected from the internet

To use this container disconnected from the internet, you must first request access by filling out an application, and purchasing a commitment plan. See [Use Docker containers in disconnected environments](#) for more information.

If you have been approved to run the container disconnected from the internet, use the following example shows the formatting of the `docker run` command you'll use, with placeholder values. Replace these placeholder values with your own values.

The `DownloadLicense=True` parameter in your `docker run` command will download a license file that will enable your Docker container to run when it isn't connected to the internet. It also contains an expiration date, after which the license file will be invalid to

run the container. You can only use a license file with the appropriate container that you've been approved for. For example, you can't use a license file for a speech to text container with a Document Intelligence container.

Placeholder	Value	Format or example
{IMAGE}	The container image you want to use.	mcr.microsoft.com/azure-cognitive-services/form-recognizer/invoice
{LICENSE_MOUNT}	The path where the license will be downloaded, and mounted.	/host/license:/path/to/license/directory
{ENDPOINT_URI}	The endpoint for authenticating your service request. You can find it on your resource's Key and endpoint page, on the Azure portal.	https://<your-custom-subdomain>.cognitiveservices.azure.com
{API_KEY}	The key for your Text Analytics resource. You can find it on your resource's Key and endpoint page, on the Azure portal.	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
{CONTAINER_LICENSE_DIRECTORY}	Location of the license folder on the container's local filesystem.	/path/to/license/directory

Bash

```
docker run --rm -it -p 5000:5000 \
-v {LICENSE_MOUNT} \
{IMAGE} \
eula=accept \
billing={ENDPOINT_URI} \
apikey={API_KEY} \
DownloadLicense=True \
Mounts:License={CONTAINER_LICENSE_DIRECTORY}
```

Once the license file has been downloaded, you can run the container in a disconnected environment. The following example shows the formatting of the `docker run` command you'll use, with placeholder values. Replace these placeholder values with your own values.

Wherever the container is run, the license file must be mounted to the container and the location of the license folder on the container's local filesystem must be specified with `Mounts:License=`. An output mount must also be specified so that billing usage records can be written.

Placeholder	Value	Format or example
{IMAGE}	The container image you want to use.	<code>mcr.microsoft.com/azure-cognitive-services/form-recognizer/invoice</code>
{MEMORY_SIZE}	The appropriate size of memory to allocate for your container.	<code>4g</code>
{NUMBER_CPUS}	The appropriate number of CPUs to allocate for your container.	<code>4</code>
{LICENSE_MOUNT}	The path where the license will be located and mounted.	<code>/host/license:/path/to/license/directory</code>
{OUTPUT_PATH}	The output path for logging usage records .	<code>/host/output:/path/to/output/directory</code>
{CONTAINER_LICENSE_DIRECTORY}	Location of the license folder on the container's local filesystem.	<code>/path/to/license/directory</code>
{CONTAINER_OUTPUT_DIRECTORY}	Location of the output folder on the container's local filesystem.	<code>/path/to/output/directory</code>

Bash

```
docker run --rm -it -p 5000:5000 --memory {MEMORY_SIZE} --cpus {NUMBER_CPUS}  
\  
-v {LICENSE_MOUNT} \  
-v {OUTPUT_PATH} \  
{IMAGE} \  
eula=accept \  
Mounts:License={CONTAINER_LICENSE_DIRECTORY}  
Mounts:Output={CONTAINER_OUTPUT_DIRECTORY}
```

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.

Tip

For more troubleshooting information and guidance, see [Azure AI services containers frequently asked questions \(FAQ\)](#).

If you're having trouble running an Azure AI services container, you can try using the Microsoft diagnostics container. Use this container to diagnose common errors in your deployment environment that might prevent Azure AI services containers from functioning as expected.

To get the container, use the following `docker pull` command:

Bash

```
docker pull mcr.microsoft.com/azure-cognitive-services/diagnostic
```

Then run the container. Replace `{ENDPOINT_URI}` with your endpoint, and replace `{API_KEY}` with your key to your resource:

Bash

```
docker run --rm mcr.microsoft.com/azure-cognitive-services/diagnostic \
eula=accept \
Billing={ENDPOINT_URI} \
ApiKey={API_KEY}
```

The container will test for network connectivity to the billing endpoint.

Billing

The LUIS container sends billing information to Azure, using a *Azure AI 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` parameter.

Azure Cognitive Services containers aren't licensed to run without being connected to the metering or billing endpoint. 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 serving requests. See the [Cognitive Services container FAQ](#) for an example of the information sent to Microsoft for billing.

Billing arguments

The [docker run ↗](#) command will start the container when all three of the following options are provided 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> .

Option	Description
Billing	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.
Eula	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](#).

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

Azure AI 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. Azure AI 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.
- See [LUIS container limitations](#) for known capability restrictions.
- Refer to [Troubleshooting](#) to resolve issues related to LUIS functionality.
- Use more [Azure AI containers](#)

Configure Language Understanding Docker containers

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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 *Azure AI services* resource specified for the [Billing](#) configuration setting.

This setting can be found in the following places:

- Azure portal: [Azure AI 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	<code>InstrumentationKey</code>	String	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 . Example: <code>InstrumentationKey=123456789</code>

Billing setting

The `Billing` setting specifies the endpoint URI of the *Azure AI 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 *Azure AI 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: **Azure AI services** Overview, labeled `Endpoint`
- LUIS portal: **Keys and Endpoint settings** page, as part of the endpoint URI.

Required	Name	Data type	Description
Yes	<code>Billing</code>	string	Billing endpoint URI. For more information on obtaining the billing URI, see gather required parameters . For more information and a complete list of regional endpoints, see Custom subdomain names for Azure AI services .

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	<code>Eula</code>	String	License acceptance Example: <code>Eula=accept</code>

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
<code>Host</code>	String	The IP address or DNS host name of the Fluentd server.
<code>Port</code>	Integer	The port of the Fluentd server. The default value is 24224.
<code>HeartbeatMs</code>	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).
<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 <proxy-url></code>
<code>HTTP_PROXY_CREDS</code>	string	Any credentials needed to authenticate against the proxy, for

Name	Data type	Description
		example, <code>username:password</code> . This value must be in lower-case.
<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.

Bash

```
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:

Bash

```
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 \
Mounts:Output=/output
```

This container command shows debugging information, prefixed with `dbug`, while the container is running:

Bash

```
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 <code>-1</code> 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 <code>1</code> .

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	<p>The target of the input mount. The default value is <code>/input</code>. This is the location of the LUIS package files.</p> <p>Example:</p> <pre>--mount type=bind,src=c:\input,target=/input</pre>
No	<code>Output</code>	String	<p>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.</p> <p>Example:</p> <pre>--mount type=bind,src=c:\output,target=/output</pre>

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.

Replace `{argument_name}` with your own values:

Placeholder	Value	Format or example
{API_KEY}	The endpoint key of the LUIS resource on the Azure LUIS Keys page.	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
{ENDPOINT_URI}	The billing endpoint value is available on the Azure LUIS Overview page.	See gather required parameters for explicit examples.

(i) Note

New resources created after July 1, 2019, will use custom subdomain names. For more information and a complete list of regional endpoints, see [Custom subdomain names for Cognitive Services](#).

(i) 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 AI services resource keys page.

Basic example

The following example has the fewest arguments possible to run the container:

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: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:

Console

```
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](#).

Console

```
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 [Azure AI containers](#)

Deploy and run container on Azure Container Instance

Article • 07/18/2023

With the following steps, scale Azure AI services applications in the cloud easily with Azure [Container Instances](#). Containerization helps you focus on building your applications instead of managing the infrastructure. For more information on using containers, see [features and benefits](#).

Prerequisites

The recipe works with any Azure AI services container. The Azure AI services resource must be created before using the recipe. Each Azure AI service that supports containers has a "How to install" article for installing and configuring the service for a container. Some services require a file or set of files as input for the container, it is important that you understand and have used the container successfully before using this solution.

- An Azure resource for the Azure AI service you're using.
- Azure AI service resource **endpoint URL** - review your specific service's "How to install" for the container, to find where the endpoint URL is from within the Azure portal, and what a correct example of the URL looks like. The exact format can change from service to service.
- Azure AI service resource **key** - the keys are on the **Keys** page for the Azure resource. You only need one of the two keys. The key is a string of 32 alphanumeric characters.
- A single Azure AI services Container on your local host (your computer). Make sure you can:
 - Pull down the image with a `docker pull` command.
 - Run the local container successfully with all required configuration settings with a `docker run` command.
 - Call the container's endpoint, getting a response of HTTP 2xx and a JSON response back.

All variables in angle brackets, `<>`, need to be replaced with your own values. This replacement includes the angle brackets.

 **Important**

The LUIS container requires a `.gz` model file that is pulled in at runtime. The container must be able to access this model file via a volume mount from the container instance. To upload a model file, follow these steps:

1. [Create an Azure file share](#). Take note of the Azure Storage account name, key, and file share name as you'll need them later.
2. [export your LUIS model \(packaged app\) from the LUIS portal](#).
3. In the Azure portal, navigate to the [Overview](#) page of your storage account resource, and select [File shares](#).
4. Select the file share name that you recently created, then select [Upload](#). Then upload your packaged app.

Azure portal

Create an Azure Container Instance resource using the Azure portal

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	If your container image is stored in a container registry that doesn't require credentials, choose <code>Public</code> . If accessing your container image requires credentials, choose <code>Private</code> . Refer to container repositories and images for details on whether or not the container image is <code>Public</code> or <code>Private</code> ("Public Preview").
Image name	Enter the Azure AI services container location. The location is what's used as an argument to the <code>docker pull</code> command. Refer to the container repositories and images for the available image names and their corresponding repository.

Setting	Value
	<p>The image name must be fully qualified specifying three parts. First, the container registry, then the repository, finally the image name: <code><container-registry>/<repository>/<image-name></code>.</p> <p>Here is an example, <code>mcr.microsoft.com/azure-cognitive-services/keyphrase</code> would represent the Key Phrase Extraction image in the Microsoft Container Registry under the Azure AI services repository. Another example is, <code>containerpreview.azurecr.io/microsoft/cognitive-services-speech-to-text</code> which would represent the Speech to text image in the Microsoft repository of the Container Preview container registry.</p>
OS type	Linux
Size	<p>Change size to the suggested recommendations for your specific Azure AI container:</p> <ul style="list-style-type: none"> 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 Azure Container Instance resource:

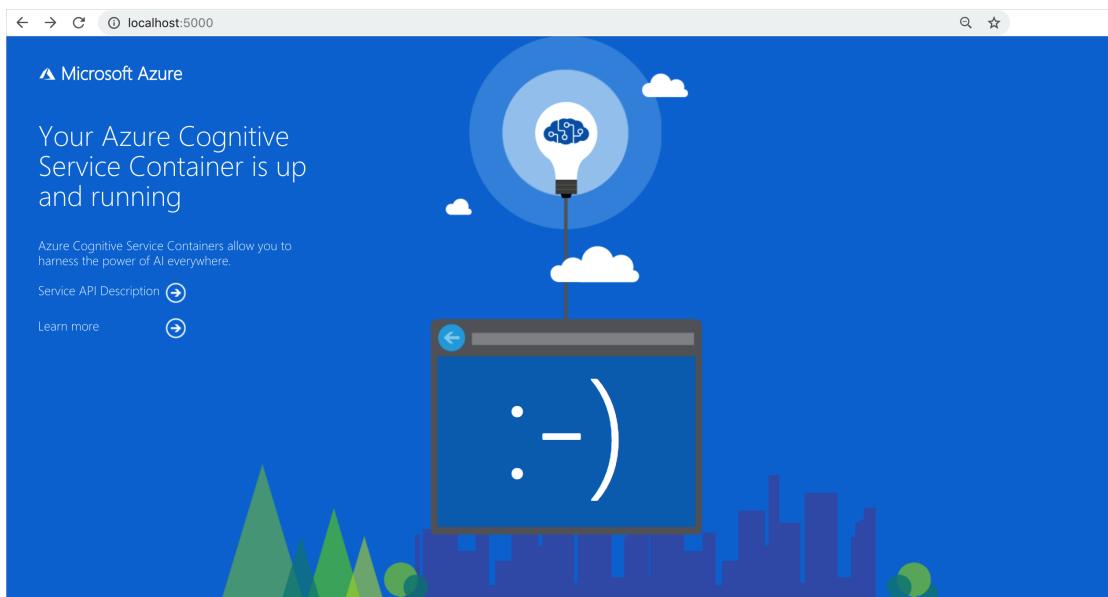
Key	Value
ApiKey	Copied from the Keys and endpoint page of the resource. It is a 32 alphanumeric-character string with no spaces or dashes, <code>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</code> .
Billing	Your endpoint URL copied from the Keys and endpoint page of the resource.
Eula	<code>accept</code>

5. Select **Review and Create**
6. After validation passes, click **Create** to finish the creation process
7. When the resource is successfully deployed, it's ready

Use the Container Instance

Azure portal

1. Select the **Overview** and copy the IP address. It will be a numeric IP address such as 55.55.55.55.
2. Open a new browser tab and use the IP address, for example, `http://<IP-address>:5000` (`http://55.55.55.55:5000`). You will see the container's home page, letting you know the container is running.



3. Select **Service API Description** to view the swagger page for the container.
4. Select any of the **POST** APIs and select **Try it out**. The parameters are displayed including the input. Fill in the parameters.
5. Select **Execute** to send the request to your Container Instance.

You have successfully created and used Azure AI services containers in Azure Container Instance.

Language Understanding (LUIS) container limitations

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

The LUIS containers have a few notable limitations. From unsupported dependencies, to a subset of languages supported, this article details these restrictions.

Supported dependencies for `latest` container

The latest LUIS container supports:

- [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

To [export for container](#), you must remove unsupported dependencies from your LUIS app. When you attempt to export for container, the LUIS portal reports these unsupported features that 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	The Dutch (<code>n1-NL</code>), Japanese (<code>ja-JP</code>) and German (<code>de-DE</code>) languages are only supported with the 1.0.2 tokenizer .
Unsupported entities for all cultures	KeyPhrase prebuilt entity for all cultures

Unsupported app configurations	Details
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.
Bing spell check	External dependencies are not supported in the container.

Languages supported

LUIS containers support a subset of the [languages supported](#) by LUIS proper. The LUIS containers are capable of understanding utterances in the following languages:

Language	Locale	Prebuilt domain	Prebuilt entity	Phrase list recommendations	**Sentiment analysis and key phrase extraction
English (United States)	en-US	✓	✓	✓	✓
Arabic (preview - modern standard Arabic)	ar-AR	✗	✗	✗	✗
*Chinese	zh-CN	✓	✓	✓	✗
French (France)	fr-FR	✓	✓	✓	✓
French (Canada)	fr-CA	✗	✗	✗	✓
German	de-DE	✓	✓	✓	✓
Hindi	hi-IN	✗	✗	✗	✗
Italian	it-IT	✓	✓	✓	✓
Korean	ko-KR	✓	✗	✗	Key phrase only
Marathi	mr-IN	✗	✗	✗	✗
Portuguese (Brazil)	pt-BR	✓	✓	✓	not all sub-cultures
Spanish (Spain)	es-ES	✓	✓	✓	✓
Spanish (Mexico)	es-MX	✗	✗	✓	✓

Language	Locale	Prebuilt domain	Prebuilt entity	Phrase list recommendations	**Sentiment analysis and key phrase extraction
Tamil	ta-IN	✗	✗	✗	
Telugu	te-IN	✗	✗	✗	
Turkish	tr-TR	✓	✗	✗	<i>Sentiment only</i>

*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.

**Language service support notes

The Language service 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.

Use Docker containers in disconnected environments

Article • 11/15/2023

Containers enable you to run Azure AI services APIs in your own environment, and are great for your specific security and data governance requirements. Disconnected containers enable you to use several of these APIs disconnected from the internet.

Currently, the following containers can be run in this manner:

- [Speech to text](#)
- [Custom Speech to text](#)
- [Neural Text to speech](#)
- [Text Translation \(Standard\)](#)
- [Azure AI Language](#)
 - [Sentiment Analysis](#)
 - [Key Phrase Extraction](#)
 - [Language Detection](#)
 - [Summarization](#)
 - [Named Entity Recognition](#)
- [Azure AI Vision - Read](#)
- [Document Intelligence](#)

Before attempting to run a Docker container in an offline environment, make sure you know the steps to successfully download and use the container. For example:

- Host computer requirements and recommendations.
- The Docker `pull` command you'll use to download the container.
- How to validate that a container is running.
- How to send queries to the container's endpoint, once it's running.

Request access to use containers in disconnected environments

Fill out and submit the [request form](#) to request access to the containers disconnected from the internet.

The form requests information about you, your company, and the user scenario for which you'll use the container. After you submit the form, the Azure AI services team reviews it and emails you with a decision within 10 business days.

Important

- On the form, you must use an email address associated with an Azure subscription ID.
- The Azure resource you use to run the container must have been created with the approved Azure subscription ID.
- Check your email (both inbox and junk folders) for updates on the status of your application from Microsoft.

After you're approved, you'll be able to run the container after you download it from the Microsoft Container Registry (MCR), described later in the article.

You won't be able to run the container if your Azure subscription hasn't been approved.

Access is limited to customers that meet the following requirements:

- Your organization should be identified as strategic customer or partner with Microsoft.
- Disconnected containers are expected to run fully offline, hence your use cases must meet one of below or similar requirements:
 - Environment or device(s) with zero connectivity to internet.
 - Remote location that occasionally has internet access.
 - Organization under strict regulation of not sending any kind of data back to cloud.
- Application completed as instructed - Please pay close attention to guidance provided throughout the application to ensure you provide all the necessary information required for approval.

Purchase a commitment tier pricing plan for disconnected containers

Create a new resource

1. Sign in to the [Azure portal](#) and select **Create a new resource** for one of the applicable Azure AI services listed above.
2. Enter the applicable information to create your resource. Be sure to select **Commitment tier disconnected containers** as your pricing tier.

Note

- You will only see the option to purchase a commitment tier if you have been approved by Microsoft.
- Pricing details are for example only.

3. Select **Review + Create** at the bottom of the page. Review the information, and select **Create**.

Configure container for disconnected usage

See the following documentation for steps on downloading and configuring the container for disconnected usage:

- [Vision - Read](#)
- [Language Understanding \(LUIS\)](#)
- [Text Translation \(Standard\)](#)
- [Document Intelligence](#)

Speech service

- [Speech to text](#)
- [Custom Speech to text](#)
- [Neural Text to speech](#)

Language service

- [Sentiment Analysis](#)
- [Key Phrase Extraction](#)
- [Language Detection](#)

Environment variable names in Kubernetes deployments

Some Azure AI Containers, for example Translator, require users to pass environmental variable names that include colons (:) when running the container. This will work fine when using Docker, but Kubernetes does not accept colons in environmental variable names. To resolve this, you can replace colons with double underscore characters (__) when deploying to Kubernetes. See the following example of an acceptable format for environment variable names:

Kubernetes

```
env:  
  - name: Mounts__License  
    value: "/license"  
  - name: Mounts__Output  
    value: "/output"
```

This example replaces the default format for the `Mounts:License` and `Mounts:Output` environment variable names in the docker run command.

Container image and license updates

Container license files are used as keys to decrypt certain files within each container image. If these encrypted files happen to be updated within a new container image, the license file you have may fail to start the container even if it worked with the previous version of the container image. To avoid this issue, we recommend that you download a new license file from the resource endpoint for your container provided in Azure portal after you pull new image versions from mcr.microsoft.com.

To download a new license file, you can add `DownloadLicense=True` to your docker run command along with a license mount, your API Key, and your billing endpoint. Refer to your [container's documentation](#) for detailed instructions.

Usage records

When operating Docker containers in a disconnected environment, the container will write usage records to a volume where they're collected over time. You can also call a REST endpoint to generate a report about service usage.

Arguments for storing logs

When run in a disconnected environment, an output mount must be available to the container to store usage logs. For example, you would include `-v /host/output:{OUTPUT_PATH}` and `Mounts:Output={OUTPUT_PATH}` in the example below, replacing `{OUTPUT_PATH}` with the path where the logs will be stored:

Docker

```
docker run -v /host/output:{OUTPUT_PATH} ... <image> ... Mounts:Output={OUTPUT_PATH}
```

Get records using the container endpoints

The container provides two endpoints for returning records about its usage.

Get all records

The following endpoint will provide a report summarizing all of the usage collected in the mounted billing record directory.

HTTP

```
https://<service>/records/usage-logs/
```

It will return JSON similar to the example below.

JSON

```
{
  "apiType": "noop",
  "serviceName": "noop",
  "meters": [
    {
      "name": "Sample.Meter",
      "quantity": 253
    }
  ]
}
```

Get records for a specific month

The following endpoint will provide a report summarizing usage over a specific month and year.

HTTP

```
https://<service>/records/usage-logs/{MONTH}/{YEAR}
```

it will return a JSON response similar to the example below:

JSON

```
{
  "apiType": "string",
  "serviceName": "string",
  "meters": [
```

```
{  
    "name": "string",  
    "quantity": 253  
}  
]  
}
```

Purchase a commitment plan to use containers in disconnected environments

Commitment plans for disconnected containers have a calendar year commitment period. When you purchase a plan, you'll be charged the full price immediately. During the commitment period, you can't change your commitment plan, however you can purchase additional unit(s) at a pro-rated price for the remaining days in the year. You have until midnight (UTC) on the last day of your commitment, to end a commitment plan.

You can choose a different commitment plan in the **Commitment Tier pricing** settings of your resource.

End a commitment plan

If you decide that you don't want to continue purchasing a commitment plan, you can set your resource's auto-renewal to **Do not auto-renew**. Your commitment plan will expire on the displayed commitment end date. After this date, you won't be charged for the commitment plan. You'll be able to continue using the Azure resource to make API calls, charged at pay-as-you-go pricing. You have until midnight (UTC) on the last day of the year to end a commitment plan for disconnected containers, and not be charged for the following year.

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.

💡 Tip

For more troubleshooting information and guidance, see [Disconnected containers Frequently asked questions \(FAQ\)](#).

Next steps

[Azure AI containers overview](#)

What are Azure AI containers?

Article • 09/01/2023

Azure AI services provides several [Docker containers](#) that let you use the same APIs that are available in Azure, on-premises. Using these containers gives you the flexibility to bring Azure AI services closer to your data for compliance, security or other operational reasons. Container support is currently available for a subset of Azure AI services.

<https://www.youtube-nocookie.com/embed/hdfbn4Q8jbo>

Containerization is an approach to software distribution in which an application or service, including its dependencies & configuration, is packaged together as a container image. With little or no modification, a container image can be deployed on a container host. Containers are isolated from each other and the underlying operating system, with a smaller footprint than a virtual machine. Containers can be instantiated from container images for short-term tasks, and removed when no longer needed.

Features and benefits

- **Immutable infrastructure:** Enable DevOps teams to leverage a consistent and reliable set of known system parameters, while being able to adapt to change. Containers provide the flexibility to pivot within a predictable ecosystem and avoid configuration drift.
- **Control over data:** Choose where your data gets processed by Azure AI services. This can be essential if you can't send data to the cloud but need access to Azure AI services APIs. Support consistency in hybrid environments – across data, management, identity, and security.
- **Control over model updates:** Flexibility in versioning and updating of models deployed in their solutions.
- **Portable architecture:** Enables the creation of a portable application architecture that can be deployed on Azure, on-premises and the edge. Containers can be deployed directly to [Azure Kubernetes Service](#), [Azure Container Instances](#), or to a [Kubernetes](#) cluster deployed to [Azure Stack](#). For more information, see [Deploy Kubernetes to Azure Stack](#).
- **High throughput / low latency:** Provide customers the ability to scale for high throughput and low latency requirements by enabling Azure AI services to run physically close to their application logic and data. Containers don't cap transactions per second (TPS) and can be made to scale both up and out to handle demand if you provide the necessary hardware resources.

- **Scalability:** With the ever growing popularity of containerization and container orchestration software, such as Kubernetes; scalability is at the forefront of technological advancements. Building on a scalable cluster foundation, application development caters to high availability.

Containers in Azure AI services

Azure AI containers provide the following set of Docker containers, each of which contains a subset of functionality from services in Azure AI services. You can find instructions and image locations in the tables below.

 **Note**

See [Install and run Document Intelligence containers for Azure AI Document Intelligence](#) container instructions and image locations.

Decision containers

Service	Container	Description	Availability
Anomaly detector	Anomaly Detector (image ↗)	The Anomaly Detector API enables you to monitor and detect abnormalities in your time series data with machine learning.	Generally available

Language containers

Service	Container	Description	Availability
LUIS	LUIS (image ↗)	Loads a 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 LUIS portal ↗ to improve the app's prediction accuracy.	Generally available. This container can also run in disconnected environments .
Language service	Key Phrase Extraction (image ↗)	Extracts key phrases to identify the main points. For example, for the input text "The food was delicious and there were wonderful staff", the API returns the main talking points: "food" and "wonderful staff".	Generally available. This container can also run in

Service	Container	Description	Availability
			disconnected environments.
Language service	Text Language Detection (image )	For up to 120 languages, detects which language the input text is written in and report a single language code for every document submitted on the request. The language code is paired with a score indicating the strength of the score.	Generally available. This container can also run in disconnected environments.
Language service	Sentiment Analysis (image )	Analyzes raw text for clues about positive or negative sentiment. This version of sentiment analysis returns sentiment labels (for example <i>positive</i> or <i>negative</i>) for each document and sentence within it.	Generally available. This container can also run in disconnected environments.
Language service	Text Analytics for health (image )	Extract and label medical information from unstructured clinical text.	Generally available
Language service	Custom Named Entity Recognition (image )	Extract named entities from text, using a custom model you create using your data.	Preview
Translator	Translator (image )	Translate text in several languages and dialects.	Generally available. Gated - request access  . This container can also run in disconnected environments.

Speech containers

Service	Container	Description	Availability
Speech Service API	Speech to text (image )	Transcribes continuous real-time speech into text.	Generally available. This container can also run in disconnected environments.
Speech Service API	Custom Speech to text (image )	Transcribes continuous real-time speech into text using a custom model.	Generally available This container can also run in disconnected environments.

Service	Container	Description	Availability
Speech Service API	Neural Text to speech (image )	Converts text to natural-sounding speech using deep neural network technology, allowing for more natural synthesized speech.	Generally available. This container can also run in disconnected environments.
Speech Service API	Speech language detection (image )	Determines the language of spoken audio.	Preview

Vision containers

Service	Container	Description	Availability
Azure AI Vision	Read OCR (image )	The Read OCR container allows you to extract printed and handwritten text from images and documents with support for JPEG, PNG, BMP, PDF, and TIFF file formats. For more information, see the Read API documentation .	Generally Available. This container can also run in disconnected environments.
Spatial Analysis	Spatial analysis (image )	Analyzes real-time streaming video to understand spatial relationships between people, their movement, and interactions with objects in physical environments.	Preview

Additionally, some containers are supported in the Azure AI services [multi-service resource](#) offering. You can create one single Azure AI services All-In-One resource and use the same billing key across supported services for the following services:

- Azure AI Vision
- LUIS
- Language service

Prerequisites

You must satisfy the following prerequisites before using Azure AI containers:

Docker Engine: You must have Docker Engine installed locally. Docker provides packages that configure the Docker environment on [macOS](#) , [Linux](#) , and [Windows](#) .

On Windows, Docker must be configured to support Linux containers. Docker containers can also be deployed directly to [Azure Kubernetes Service](#) or [Azure Container Instances](#).

Docker must be configured to allow the containers to connect with and send billing data to Azure.

Familiarity with Microsoft Container Registry and Docker: You should have a basic understanding of both Microsoft Container Registry and Docker concepts, like registries, repositories, containers, and container images, as well as knowledge of basic `docker` commands.

For a primer on Docker and container basics, see the [Docker overview](#).

Individual containers can have their own requirements, as well, including server and memory allocation requirements.

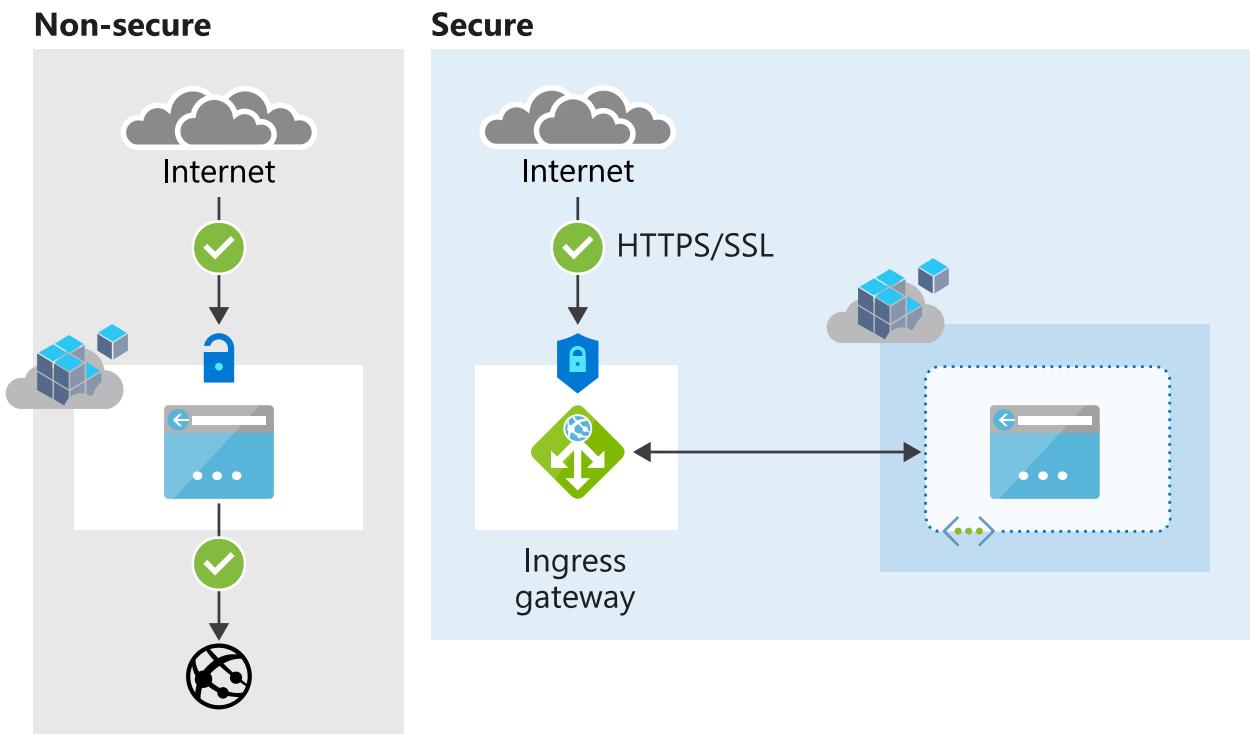
Azure AI services container security

Security should be a primary focus whenever you're developing applications. The importance of security is a metric for success. When you're architecting a software solution that includes Azure AI containers, it's vital to understand the limitations and capabilities available to you. For more information about network security, see [Configure Azure AI services virtual networks](#).

Important

By default there is *no security* on the Azure AI services container API. The reason for this is that most often the container will run as part of a pod which is protected from the outside by a network bridge. However, it is possible for users to construct their own authentication infrastructure to approximate the authentication methods used when accessing the [cloud-based Azure AI services](#).

The following diagram illustrates the default and **non-secure** approach:



As an example of an alternative and *secure* approach, consumers of Azure AI containers could augment a container with a front-facing component, keeping the container endpoint private. Let's consider a scenario where we use [Istio](#) as an ingress gateway. Istio supports HTTPS/TLS and client-certificate authentication. In this scenario, the Istio frontend exposes the container access, presenting the client certificate that is approved beforehand with Istio.

[Nginx](#) is another popular choice in the same category. Both Istio and Nginx act as a service mesh and offer additional features including things like load-balancing, routing, and rate-control.

Container networking

The Azure AI containers are required to submit metering information for billing purposes. Failure to allowlist various network channels that the Azure AI containers rely on will prevent the container from working.

Allowlist Azure AI services domains and ports

The host should allowlist **port 443** and the following domains:

- *.cognitive.microsoft.com
- *.cognitiveservices.azure.com

Disable deep packet inspection

[Deep packet inspection](#) (DPI) is a type of data processing that inspects in detail the data sent over a computer network, and usually takes action by blocking, rerouting, or logging it accordingly.

Disable DPI on the secure channels that the Azure AI containers create to Microsoft servers. Failure to do so will prevent the container from functioning correctly.

Developer samples

Developer samples are available at our [GitHub repository](#).

Next steps

Learn about [container recipes](#) you can use with the Azure AI services.

Install and explore the functionality provided by containers in Azure AI services:

- [Anomaly Detector containers](#)
- [Azure AI Vision containers](#)
- [Language Understanding \(LUIS\) containers](#)
- [Speech Service API containers](#)
- [Language service containers](#)
- [Translator containers](#)

Configure Azure AI services virtual networks

Article • 10/27/2023

Azure AI services provide a layered security model. This model enables you to secure your Azure AI services accounts to a specific subset of networks. When network rules are configured, only applications that request data over the specified set of networks can access the account. You can limit access to your resources with *request filtering*, which allows requests that originate only from specified IP addresses, IP ranges, or from a list of subnets in [Azure Virtual Networks](#).

An application that accesses an Azure AI services resource when network rules are in effect requires authorization. Authorization is supported with [Microsoft Entra ID](#) credentials or with a valid API key.

Important

Turning on firewall rules for your Azure AI services account blocks incoming requests for data by default. To allow requests through, one of the following conditions needs to be met:

- The request originates from a service that operates within an Azure Virtual Network on the allowed subnet list of the target Azure AI services account. The endpoint request that originated from the virtual network needs to be set as the [custom subdomain](#) of your Azure AI services account.
- The request originates from an allowed list of IP addresses.

Requests that are blocked include those from other Azure services, from the Azure portal, and from logging and metrics services.

Note

We recommend that you use the Azure Az PowerShell module to interact with Azure. See [Install Azure PowerShell](#) to get started. To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

Scenarios

To secure your Azure AI services resource, you should first configure a rule to deny access to traffic from all networks, including internet traffic, by default. Then, configure rules that grant access to traffic from specific virtual networks. This configuration enables you to build a secure network boundary for your applications. You can also configure rules to grant access to traffic from select public internet IP address ranges and enable connections from specific internet or on-premises clients.

Network rules are enforced on all network protocols to Azure AI services, including REST and WebSocket. To access data by using tools such as the Azure test consoles, explicit network rules must be configured. You can apply network rules to existing Azure AI services resources, or when you create new Azure AI services resources. After network rules are applied, they're enforced for all requests.

Supported regions and service offerings

Virtual networks are supported in [regions where Azure AI services are available](#). Azure AI services support service tags for network rules configuration. The services listed here are included in the `CognitiveServicesManagement` service tag.

- ✓ Anomaly Detector
- ✓ Azure OpenAI
- ✓ Content Moderator
- ✓ Custom Vision
- ✓ Face
- ✓ Language Understanding (LUIS)
- ✓ Personalizer
- ✓ Speech service
- ✓ Language
- ✓ QnA Maker
- ✓ Translator

Note

If you use Azure OpenAI, LUIS, Speech Services, or Language services, the `CognitiveServicesManagement` tag only enables you to use the service by using the SDK or REST API. To access and use Azure OpenAI Studio, LUIS portal, Speech Studio, or Language Studio from a virtual network, you need to use the following tags:

- `AzureActiveDirectory`
- `AzureFrontDoor.Frontend`

- `AzureResourceManager`
- `CognitiveServicesManagement`
- `CognitiveServicesFrontEnd`

Change the default network access rule

By default, Azure AI services resources accept connections from clients on any network. To limit access to selected networks, you must first change the default action.

Warning

Making changes to network rules can impact your applications' ability to connect to Azure AI services. Setting the default network rule to *deny* blocks all access to the data unless specific network rules that *grant* access are also applied.

Before you change the default rule to deny access, be sure to grant access to any allowed networks by using network rules. If you allow listing for the IP addresses for your on-premises network, be sure to add all possible outgoing public IP addresses from your on-premises network.

Manage default network access rules

You can manage default network access rules for Azure AI services resources through the Azure portal, PowerShell, or the Azure CLI.

Azure portal

1. Go to the Azure AI services resource you want to secure.
2. Select **Resource Management** to expand it, then select **Networking**.

Home > contoso-rg > contoso-custom-vision

contoso-custom-vision | Networking

Custom vision | Directory: Microsoft

Firewalls and virtual networks Private endpoint connections

Save Discard Refresh

Allow access from
 All networks Selected Networks and Private Endpoints Disabled

Configure network security for your Azure AI services account. [Learn more.](#)

Virtual networks
Secure your Azure AI services account with virtual networks. + Add existing virtual network + Add new virtual network

Virtual Network	Subnet	Address range	Endpoint Status	Resource group	Subscription
No network selected.					

Firewall
Add IP ranges to allow access from the internet or your on-premises networks. [Learn more.](#)
 Add your client IP address

Address range
IP address or CIDR

3. To deny access by default, under **Firewalls and virtual networks**, select **Selected Networks and Private Endpoints**.

With this setting alone, unaccompanied by configured virtual networks or address ranges, all access is effectively denied. When all access is denied, requests that attempt to consume the Azure AI services resource aren't permitted. The Azure portal, Azure PowerShell, or the Azure CLI can still be used to configure the Azure AI services resource.

4. To allow traffic from all networks, select **All networks**.

Learn more.' There are tabs for 'Firewalls and virtual networks' and 'Private endpoint connections'. Buttons for 'Save', 'Discard', and 'Refresh' are at the top right. A sidebar on the left shows 'Resource Management' expanded, with 'Networking' highlighted."/>

Home > contoso-rg > contoso-custom-vision

contoso-custom-vision | Networking

Custom vision | Directory: Microsoft

Firewalls and virtual networks Private endpoint connections

Save Discard Refresh

Allow access from
 All networks Selected Networks and Private Endpoints Disabled

All networks, including the internet, can access this resource. [Learn more.](#)

5. Select **Save** to apply your changes.

Grant access from a virtual network

You can configure Azure AI services resources to allow access from specific subnets only. The allowed subnets might belong to a virtual network in the same subscription or in a different subscription. The other subscription can belong to a different Microsoft Entra tenant.

Enable a *service endpoint* for Azure AI services within the virtual network. The service endpoint routes traffic from the virtual network through an optimal path to the Azure AI service. For more information, see [Virtual Network service endpoints](#).

The identities of the subnet and the virtual network are also transmitted with each request. Administrators can then configure network rules for the Azure AI services resource to allow requests from specific subnets in a virtual network. Clients granted access by these network rules must continue to meet the authorization requirements of the Azure AI services resource to access the data.

Each Azure AI services resource supports up to 100 virtual network rules, which can be combined with IP network rules. For more information, see [Grant access from an internet IP range](#) later in this article.

Set required permissions

To apply a virtual network rule to an Azure AI services resource, you need the appropriate permissions for the subnets to add. The required permission is the default *Contributor* role or the *Cognitive Services Contributor* role. Required permissions can also be added to custom role definitions.

The Azure AI services resource and the virtual networks that are granted access might be in different subscriptions, including subscriptions that are part of a different Microsoft Entra tenant.

Note

Configuration of rules that grant access to subnets in virtual networks that are a part of a different Microsoft Entra tenant are currently supported only through PowerShell, the Azure CLI, and the REST APIs. You can view these rules in the Azure portal, but you can't configure them.

Configure virtual network rules

You can manage virtual network rules for Azure AI services resources through the Azure portal, PowerShell, or the Azure CLI.

To grant access to a virtual network with an existing network rule:

1. Go to the Azure AI services resource you want to secure.
2. Select **Resource Management** to expand it, then select **Networking**.
3. Confirm that you selected **Selected Networks and Private Endpoints**.
4. Under **Allow access from**, select **Add existing virtual network**.

The screenshot shows the Azure portal interface for the 'contoso-custom-vision' resource. The left sidebar has 'Resource Management' expanded, with 'Networking' selected. The main content area is titled 'Networking'. It shows the 'Firewalls and virtual networks' tab is active. Under 'Allow access from', the 'Selected Networks and Private Endpoints' radio button is selected. Below it, there's a section for 'Virtual networks' with a 'Secure your Azure AI services account with virtual networks.' link and two buttons: '+ Add existing virtual network' (highlighted with a red box) and '+ Add new virtual network'. A table below lists 'Virtual Network', 'Subnet', 'Address range', 'Endpoint Status', and 'Resource group', showing 'No network selected.'. At the bottom, there's a 'Firewall' section with an 'Add IP ranges to allow access from the internet or your on-premises networks.' link and a checkbox for 'Add your client IP address'. A search bar at the bottom right contains 'IP address or CIDR'.

5. Select the **Virtual networks** and **Subnets** options, and then select **Enable**.

Add networks

X

Subscription *

Contoso Subscription

Virtual networks *

contoso-rg

Subnets *

default (Service endpoint required)



The following networks don't have service endpoints enabled for 'Microsoft.CognitiveServices'. Enabling access will take up to 15 minutes to complete. After starting this operation, it is safe to leave and return later if you do not wish to wait.

Virtual network	Service endpoint status	...
contoso-rg		...
default	Not enabled	...

Enable

Note

If a service endpoint for Azure AI services wasn't previously configured for the selected virtual network and subnets, you can configure it as part of this operation.

Currently, only virtual networks that belong to the same Microsoft Entra tenant are available for selection during rule creation. To grant access to a subnet in a virtual network that belongs to another tenant, use PowerShell, the Azure CLI, or the REST APIs.

6. Select **Save** to apply your changes.

To create a new virtual network and grant it access:

1. On the same page as the previous procedure, select **Add new virtual network**.

The screenshot shows the Azure portal interface for managing networking. The left sidebar lists various service settings like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Resource Management (Keys and Endpoint, Encryption, Pricing tier), and Networking (Identity, Cost analysis, Properties, Locks). The Networking blade is open, and the Firewalls and virtual networks tab is active. In the 'Allow access from' section, the 'Selected Networks and Private Endpoints' radio button is selected. Below this, there's a table for managing virtual networks, with a '+ Add new virtual network' button highlighted by a red box. On the right side, there's a 'Firewall' configuration section with an 'Address range' input field and a circular button with a plus sign and a magnifying glass icon.

2. Provide the information necessary to create the new virtual network, and then select **Create**.

Create virtual network

* Name
widgets-vnet ✓

* Address space ⓘ
10.1.0.0/16
10.1.0.0 - 10.1.255.255 (65536 addresses)

* Subscription
widgets-subscription ▼

* Resource group
widgets-resource-group ▼
[Create new](#)

* Location
(US) West US 2 ▼

Subnet

* Name
default

* Address range ⓘ
10.1.0.0/24 ✓
10.1.0.0 - 10.1.0.255 (256 addresses)

DDoS protection ⓘ
 Basic Standard

Service endpoint ⓘ
Microsoft.CognitiveServices

Firewall ⓘ

Create

3. Select **Save** to apply your changes.

To remove a virtual network or subnet rule:

1. On the same page as the previous procedures, select ...(**More options**) to open the context menu for the virtual network or subnet, and select **Remove**.

The screenshot shows the 'Firewalls and virtual networks' section of the Azure portal. At the top, there are buttons for 'Save', 'Discard', and 'Refresh'. Below that, it says 'Allow access from' with three options: 'All networks' (radio button), 'Selected Networks and Private Endpoints' (selected radio button), and 'Disabled'. A note says 'Configure network security for your Azure AI services account. [Learn more](#)'. Under 'Virtual networks', there's a table with columns 'Virtual Network', 'Subnet', 'Address range', 'Endpoint Status', 'Resource group', and 'Subscription'. One row shows 'contoso-01-vnet', '1', and 'contoso-rg'. To the right of this row are a 'Remove' button (highlighted with a red box) and a '...' button. Below the table is a 'Firewall' section with a note about adding IP ranges. There's also a 'Address range' input field and a search icon.

2. Select **Save** to apply your changes.

ⓘ Important

Be sure to **set the default rule** to *deny*, or network rules have no effect.

Grant access from an internet IP range

You can configure Azure AI services resources to allow access from specific public internet IP address ranges. This configuration grants access to specific services and on-premises networks, which effectively block general internet traffic.

You can specify the allowed internet address ranges by using [CIDR format \(RFC 4632\)](#) in the form `192.168.0.0/16` or as individual IP addresses like `192.168.0.1`.

ⓘ Tip

Small address ranges that use `/31` or `/32` prefix sizes aren't supported. Configure these ranges by using individual IP address rules.

IP network rules are only allowed for *public internet* IP addresses. IP address ranges reserved for private networks aren't allowed in IP rules. Private networks include addresses that start with `10.*`, `172.16.* - 172.31.*`, and `192.168.*`. For more information, see [Private Address Space \(RFC 1918\)](#).

Currently, only IPv4 addresses are supported. Each Azure AI services resource supports up to 100 IP network rules, which can be combined with [virtual network rules](#).

Configure access from on-premises networks

To grant access from your on-premises networks to your Azure AI services resource with an IP network rule, identify the internet-facing IP addresses used by your network. Contact your network administrator for help.

If you use Azure ExpressRoute on-premises for public peering or Microsoft peering, you need to identify the NAT IP addresses. For more information, see [What is Azure ExpressRoute](#).

For public peering, each ExpressRoute circuit by default uses two NAT IP addresses. Each is applied to Azure service traffic when the traffic enters the Microsoft Azure network backbone. For Microsoft peering, the NAT IP addresses that are used are either customer provided or supplied by the service provider. To allow access to your service resources, you must allow these public IP addresses in the resource IP firewall setting.

To find your public peering ExpressRoute circuit IP addresses, [open a support ticket with ExpressRoute](#) use the Azure portal. For more information, see [NAT requirements for Azure public peering](#).

Managing IP network rules

You can manage IP network rules for Azure AI services resources through the Azure portal, PowerShell, or the Azure CLI.

Azure portal

1. Go to the Azure AI services resource you want to secure.
2. Select **Resource Management** to expand it, then select **Networking**.
3. Confirm that you selected **Selected Networks and Private Endpoints**.
4. Under **Firewalls and virtual networks**, locate the **Address range** option. To grant access to an internet IP range, enter the IP address or address range (in [CIDR format](#)). Only valid public IP (nonreserved) addresses are accepted.

Allow access from

All networks Selected Networks and Private Endpoints Disabled

Configure network security for your Azure AI services account. [Learn more](#).

Virtual networks

Secure your Azure AI services account with virtual networks. [+ Add existing virtual network](#) [+ Add new virtual network](#)

Virtual Network	Subnet	Address range	Endpoint Status	Resource group
No network selected.				

Firewall

Add IP ranges to allow access from the internet or your on-premises networks. [Learn more](#).

Add your client IP address (i)

Address range

IP address or CIDR

To remove an IP network rule, select the trash can  icon next to the address range.

5. Select **Save** to apply your changes.

Important

Be sure to **set the default rule** to *deny*, or network rules have no effect.

Use private endpoints

You can use [private endpoints](#) for your Azure AI services resources to allow clients on a virtual network to securely access data over [Azure Private Link](#). The private endpoint uses an IP address from the virtual network address space for your Azure AI services resource. Network traffic between the clients on the virtual network and the resource traverses the virtual network and a private link on the Microsoft Azure backbone network, which eliminates exposure from the public internet.

Private endpoints for Azure AI services resources let you:

- Secure your Azure AI services resource by configuring the firewall to block all connections on the public endpoint for the Azure AI service.
- Increase security for the virtual network, by enabling you to block exfiltration of data from the virtual network.

- Securely connect to Azure AI services resources from on-premises networks that connect to the virtual network by using [Azure VPN Gateway](#) or [ExpressRoutes](#) with private-peering.

Understand private endpoints

A private endpoint is a special network interface for an Azure resource in your [virtual network](#). Creating a private endpoint for your Azure AI services resource provides secure connectivity between clients in your virtual network and your resource. The private endpoint is assigned an IP address from the IP address range of your virtual network. The connection between the private endpoint and the Azure AI service uses a secure private link.

Applications in the virtual network can connect to the service over the private endpoint seamlessly. Connections use the same connection strings and authorization mechanisms that they would use otherwise. The exception is Speech Services, which require a separate endpoint. For more information, see [Private endpoints with the Speech Services](#) in this article. Private endpoints can be used with all protocols supported by the Azure AI services resource, including REST.

Private endpoints can be created in subnets that use service endpoints. Clients in a subnet can connect to one Azure AI services resource using private endpoint, while using service endpoints to access others. For more information, see [Virtual Network service endpoints](#).

When you create a private endpoint for an Azure AI services resource in your virtual network, Azure sends a consent request for approval to the Azure AI services resource owner. If the user who requests the creation of the private endpoint is also an owner of the resource, this consent request is automatically approved.

Azure AI services resource owners can manage consent requests and the private endpoints through the **Private endpoint connection** tab for the Azure AI services resource in the [Azure portal](#).

Specify private endpoints

When you create a private endpoint, specify the Azure AI services resource that it connects to. For more information on creating a private endpoint, see:

- [Create a private endpoint by using the Azure portal](#)
- [Create a private endpoint by using Azure PowerShell](#)
- [Create a private endpoint by using the Azure CLI](#)

Connect to private endpoints

ⓘ Note

Azure OpenAI Service uses a different private DNS zone and public DNS zone forwarder than other Azure AI services. For the correct zone and forwarder names, see [Azure services DNS zone configuration](#).

Clients on a virtual network that use the private endpoint use the same connection string for the Azure AI services resource as clients connecting to the public endpoint. The exception is the Speech service, which requires a separate endpoint. For more information, see [Use private endpoints with the Speech service](#) in this article. DNS resolution automatically routes the connections from the virtual network to the Azure AI services resource over a private link.

By default, Azure creates a [private DNS zone](#) attached to the virtual network with the necessary updates for the private endpoints. If you use your own DNS server, you might need to make more changes to your DNS configuration. For updates that might be required for private endpoints, see [Apply DNS changes for private endpoints](#) in this article.

Use private endpoints with the Speech service

See [Use Speech service through a private endpoint](#).

Apply DNS changes for private endpoints

When you create a private endpoint, the DNS `CNAME` resource record for the Azure AI services resource is updated to an alias in a subdomain with the prefix `privatelink`. By default, Azure also creates a private DNS zone that corresponds to the `privatelink` subdomain, with the DNS A resource records for the private endpoints. For more information, see [What is Azure Private DNS](#).

When you resolve the endpoint URL from outside the virtual network with the private endpoint, it resolves to the public endpoint of the Azure AI services resource. When it's resolved from the virtual network hosting the private endpoint, the endpoint URL resolves to the private endpoint's IP address.

This approach enables access to the Azure AI services resource using the same connection string for clients in the virtual network that hosts the private endpoints and clients outside the virtual network.

If you use a custom DNS server on your network, clients must be able to resolve the fully qualified domain name (FQDN) for the Azure AI services resource endpoint to the private endpoint IP address. Configure your DNS server to delegate your private link subdomain to the private DNS zone for the virtual network.

💡 Tip

When you use a custom or on-premises DNS server, you should configure your DNS server to resolve the Azure AI services resource name in the `privatelink` subdomain to the private endpoint IP address. Delegate the `privatelink` subdomain to the private DNS zone of the virtual network. Alternatively, configure the DNS zone on your DNS server and add the DNS A records.

For more information on configuring your own DNS server to support private endpoints, see the following resources:

- [Name resolution that uses your own DNS server](#)
- [DNS configuration](#)

Grant access to trusted Azure services for Azure OpenAI

You can grant a subset of trusted Azure services access to Azure OpenAI, while maintaining network rules for other apps. These trusted services will then use managed identity to authenticate your Azure OpenAI service. The following table lists the services that can access Azure OpenAI if the managed identity of those services have the appropriate role assignment.

Service	Resource provider name
Azure AI Services	<code>Microsoft.CognitiveServices</code>
Azure Machine Learning	<code>Microsoft.MachineLearningServices</code>
Azure Cognitive Search	<code>Microsoft.Search</code>

You can grant networking access to trusted Azure services by creating a network rule exception using the REST API:

```
Bash
```

```
accessToken=$(az account get-access-token --resource  
https://management.azure.com --query "accessToken" --output tsv)  
rid="/subscriptions/<your subscription id>/resourceGroups/<your resource  
group>/providers/Microsoft.CognitiveServices/accounts/<your Azure AI  
resource name>"  
  
curl -i -X PATCH https://management.azure.com$rid?api-version=2023-10-01-  
preview \  
-H "Content-Type: application/json" \  
-H "Authorization: Bearer $accessToken" \  
-d \  
' \  
{  
    "properties":  
    {  
        "networkAcls": {  
            "bypass": "AzureServices"  
        }  
    }  
}'
```

To revoke the exception, set `networkAcls.bypass` to `None`.

Pricing

For pricing details, see [Azure Private Link pricing](#).

Next steps

- Explore the various [Azure AI services](#)
- Learn more about [Virtual Network service endpoints](#)

Language Understanding service encryption of data at rest

Article • 11/15/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

The Language Understanding service automatically encrypts your data when it is persisted to the cloud. The Language Understanding service encryption protects your data and helps you meet your organizational security and compliance commitments.

About Azure AI services encryption

Data is encrypted and decrypted using [FIPS 140-2](#) compliant [256-bit AES](#) encryption. Encryption and decryption are transparent, meaning encryption and access are managed for you. Your data is secure by default and you don't need to modify your code or applications to take advantage of encryption.

About encryption key management

By default, your subscription uses Microsoft-managed encryption keys. There is also the option to manage your subscription with your own keys called customer-managed keys (CMK). CMK offer greater flexibility to create, rotate, disable, and revoke access controls. You can also audit the encryption keys used to protect your data.

Customer-managed keys with Azure Key Vault

There is also an option to manage your subscription with your own keys. Customer-managed keys (CMK), also known as Bring your own key (BYOK), offer greater flexibility to create, rotate, disable, and revoke access controls. You can also audit the encryption keys used to protect your data.

You must use Azure Key Vault to store your customer-managed keys. You can either create your own keys and store them in a key vault, or you can use the Azure Key Vault

APIs to generate keys. The Azure AI services resource and the key vault must be in the same region and in the same Microsoft Entra tenant, but they can be in different subscriptions. For more information about Azure Key Vault, see [What is Azure Key Vault?](#).

The screenshot shows the 'Create options' page for a LUIS project. At the top, there are tabs for 'Basics' (which is selected), 'Tags', and 'Review + create'. Below the tabs is a descriptive text about LUIS. Underneath, there are several input fields: 'Subscription' (set to 'CS-Platform-Dev-01'), 'Resource group' (with a dropdown menu showing '(US) Central US EUAP' and a 'Create new' button), 'Name' (input field 'Enter a name'), 'Authoring location' (set to '(US) Central US EUAP'), and 'Authoring pricing tier' (dropdown menu showing 'F0 (5 Calls per second, 1M Calls per month)', 'S1', and 'E0').

Limitations

There are some limitations when using the E0 tier with existing/Previously created applications:

- Migration to an E0 resource will be blocked. Users will only be able to migrate their apps to F0 resources. After you've migrated an existing resource to F0, you can create a new resource in the E0 tier. Learn more about [migration here](#).
- Moving applications to or from an E0 resource will be blocked. A work around for this limitation is to export your existing application, and import it as an E0 resource.
- The Bing Spell check feature isn't supported.
- Logging end-user traffic is disabled if your application is E0.
- The Speech priming capability from the Azure AI Bot Service isn't supported for applications in the E0 tier. This feature is available via the Azure AI Bot Service, which doesn't support CMK.

- The speech priming capability from the portal requires Azure Blob Storage. For more information, see [bring your own storage](#).

Enable customer-managed keys

A new Azure AI services resource is always encrypted using Microsoft-managed keys. It's not possible to enable customer-managed keys at the time that the resource is created. Customer-managed keys are stored in Azure Key Vault, and the key vault must be provisioned with access policies that grant key permissions to the managed identity that is associated with the Azure AI services resource. The managed identity is available only after the resource is created using the Pricing Tier for CMK.

To learn how to use customer-managed keys with Azure Key Vault for Azure AI services encryption, see:

- [Configure customer-managed keys with Key Vault for Azure AI services encryption from the Azure portal](#)

Enabling customer managed keys will also enable a system assigned managed identity, a feature of Microsoft Entra ID. Once the system assigned managed identity is enabled, this resource will be registered with Microsoft Entra ID. After being registered, the managed identity will be given access to the Key Vault selected during customer managed key setup. You can learn more about [Managed Identities](#).

Important

If you disable system assigned managed identities, access to the key vault will be removed and any data encrypted with the customer keys will no longer be accessible. Any features depended on this data will stop working.

Important

Managed identities do not currently support cross-directory scenarios. When you configure customer-managed keys in the Azure portal, a managed identity is automatically assigned under the covers. If you subsequently move the subscription, resource group, or resource from one Microsoft Entra directory to another, the managed identity associated with the resource is not transferred to the new tenant, so customer-managed keys may no longer work. For more information, see [Transferring a subscription between Microsoft Entra directories in FAQs and known issues with managed identities for Azure resources](#).

Store customer-managed keys in Azure Key Vault

To enable customer-managed keys, you must use an Azure Key Vault to store your keys. You must enable both the **Soft Delete** and **Do Not Purge** properties on the key vault.

Only RSA keys of size 2048 are supported with Azure AI services encryption. For more information about keys, see **Key Vault keys** in [About Azure Key Vault keys, secrets and certificates](#).

Rotate customer-managed keys

You can rotate a customer-managed key in Azure Key Vault according to your compliance policies. When the key is rotated, you must update the Azure AI services resource to use the new key URI. To learn how to update the resource to use a new version of the key in the Azure portal, see the section titled **Update the key version** in [Configure customer-managed keys for Azure AI services by using the Azure portal](#).

Rotating the key does not trigger re-encryption of data in the resource. There is no further action required from the user.

Revoke access to customer-managed keys

To revoke access to customer-managed keys, use PowerShell or Azure CLI. For more information, see [Azure Key Vault PowerShell](#) or [Azure Key Vault CLI](#). Revoking access effectively blocks access to all data in the Azure AI services resource, as the encryption key is inaccessible by Azure AI services.

Next steps

- [Learn more about Azure Key Vault](#)

Continuous Integration and Continuous Delivery workflows for LUIS DevOps

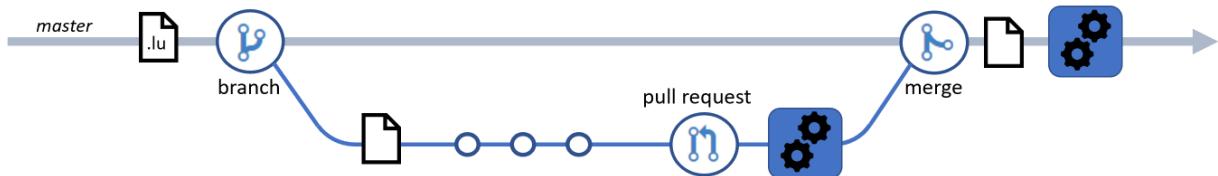
Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Software engineers who are developing a Language Understanding (LUIS) app can apply DevOps practices around [source control](#), [automated builds](#), [testing](#), and [release management](#). This article describes concepts for implementing automated builds for LUIS.

Build automation workflows for LUIS



In your source code management (SCM) system, configure automated build pipelines to run at the following events:

1. **PR workflow** triggered when a [pull request](#) (PR) is raised. This workflow validates the contents of the PR *before* the updates get merged into the main branch.
2. **CI/CD workflow** triggered when updates are pushed to the main branch, for example upon merging the changes from a PR. This workflow ensures the quality of all updates to the main branch.

The **CI/CD workflow** combines two complementary development processes:

- **Continuous Integration** (CI) is the engineering practice of frequently committing code in a shared repository, and performing an automated build on it. Paired with an automated [testing](#) approach, continuous integration allows us to verify that for each update, the LUDown source is still valid and can be imported into a LUIS app,

but also that it passes a group of tests that verify the trained app can recognize the intents and entities required for your solution.

- **Continuous Delivery** (CD) takes the Continuous Integration concept further to automatically deploy the application to an environment where you can do more in-depth testing. CD enables us to learn early about any unforeseen issues that arise from our changes as quickly as possible, and also to learn about gaps in our test coverage.

The goal of continuous integration and continuous delivery is to ensure that "main is always shippable.". For a LUIS app, this means that we could, if we needed to, take any version from the main branch LUIS app and ship it on production.

Tools for building automation workflows for LUIS

Tip

You can find a complete solution for implementing DevOps in the [LUIS DevOps template repo](#).

There are different build automation technologies available to create build automation workflows. All of them require that you can script steps using a command-line interface (CLI) or REST calls so that they can execute on a build server.

Use the following tools for building automation workflows for LUIS:

- [Bot Framework Tools LUIS CLI](#) to work with LUIS apps and versions, train, test, and publish them within the LUIS service.
- [Azure CLI](#) to query Azure subscriptions, fetch LUIS authoring and prediction keys, and to create an Azure [service principal](#) used for automation authentication.
- [NLU.DevOps](#) tool for [testing a LUIS app](#) and to analyze test results.

The PR workflow

As mentioned, you configure this workflow to run when a developer raises a PR to propose changes to be merged from a feature branch into the main branch. Its purpose is to verify the quality of the changes in the PR before they're merged to the main branch.

This workflow should:

- Create a temporary LUIS app by importing the `.luis` source in the PR.
- Train and publish the LUIS app version.
- Run all the [unit tests](#) against it.
- Pass the workflow if all the tests pass, otherwise fail it.
- Clean up and delete the temporary app.

If supported by your SCM, configure branch protection rules so that this workflow must complete successfully before the PR can be completed.

The main branch CI/CD workflow

Configure this workflow to run after the updates in the PR have been merged into the main branch. Its purpose is to keep the quality bar for your main branch high by testing the updates. If the updates meet the quality bar, this workflow deploys the new LUIS app version to an environment where you can do more in-depth testing.

This workflow should:

- Build a new version in your primary LUIS app (the app you maintain for the main branch) using the updated source code.
- Train and publish the LUIS app version.

ⓘ Note

As explained in [Running tests in an automated build workflow](#) you must publish the LUIS app version under test so that tools such as NLU.DevOps can access it. LUIS only supports two named publication slots, *staging* and *production* for a LUIS app, but you can also [publish a version directly](#) and [query by version](#). Use direct version publishing in your automation workflows to avoid being limited to using the named publishing slots.

- Run all the [unit tests](#).
- Optionally run [batch tests](#) to measure the quality and accuracy of the LUIS app version and compare it to some baseline.
- If the tests complete successfully:
 - Tag the source in the repo.
 - Run the Continuous Delivery (CD) job to deploy the LUIS app version to environments for further testing.

Continuous delivery (CD)

The CD job in a CI/CD workflow runs conditionally on success of the build and automated unit tests. Its job is to automatically deploy the LUIS application to an environment where you can do more testing.

There's no one recommended solution on how best to deploy your LUIS app, and you must implement the process that is appropriate for your project. The [LUIS DevOps template](#) ↗ repo implements a simple solution for this which is to [publish the new LUIS app version](#) to the *production* publishing slot. This is fine for a simple setup. However, if you need to support a number of different production environments at the same time, such as *development*, *staging* and *UAT*, then the limit of two named publishing slots per app will prove insufficient.

Other options for deploying an app version include:

- Leave the app version published to the direct version endpoint and implement a process to configure downstream production environments with the direct version endpoint as required.
- Maintain different LUIS apps for each production environments and write automation steps to import the `.1u` into a new version in the LUIS app for the target production environment, to train, and publish it.
- Export the tested LUIS app version into a [LUIS docker container](#) and deploy the LUIS container to Azure [Container instances](#).

Release management

Generally we recommend that you do continuous delivery only to your non-production environments, such as to development and staging. Most teams require a manual review and approval process for deployment to a production environment. For a production deployment, you might want to make sure it happens when key people on the development team are available for support, or during low-traffic periods.

Apply DevOps to LUIS app development using GitHub Actions

Go to the [LUIS DevOps template repo](#) ↗ for a complete solution that implements DevOps and software engineering best practices for LUIS. You can use this template repo to create your own repository with built-in support for CI/CD workflows and

practices that enable [source control](#), automated builds, [testing](#), and release management with LUIS for your own project.

The [LUIS DevOps template repo](#) walks through how to:

- **Clone the template repo** - Copy the template to your own GitHub repository.
- **Configure LUIS resources** - Create the [LUIS authoring and prediction resources in Azure](#) that will be used by the continuous integration workflows.
- **Configure the CI/CD workflows** - Configure parameters for the CI/CD workflows and store them in [GitHub Secrets](#).
- **Walks through the "dev inner loop"** - The developer makes updates to a sample LUIS app while working in a development branch, tests the updates and then raises a pull request to propose changes and to seek review approval.
- **Execute CI/CD workflows** - Execute [continuous integration workflows to build and test a LUIS app](#) using GitHub Actions.
- **Perform automated testing** - Perform automated batch testing for a LUIS app to evaluate the quality of the app.
- **Deploy the LUIS app** - Execute a [continuous delivery \(CD\) job](#) to publish the LUIS app.
- **Use the repo with your own project** - Explains how to use the repo with your own LUIS application.

Next steps

- Learn how to write a [GitHub Actions workflow with NLU.DevOps](#)
- Use the [LUIS DevOps template repo](#) to apply DevOps with your own project.
- [Source control and branch strategies for LUIS](#)
- [Testing for LUIS DevOps](#)

DevOps practices for LUIS

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Software engineers who are developing a Language Understanding (LUIS) app can apply DevOps practices around [source control](#), [automated builds](#), [testing](#), and [release management](#) by following these guidelines.

Source control and branch strategies for LUIS

One of the key factors that the success of DevOps depends upon is [source control](#). A source control system allows developers to collaborate on code and to track changes. The use of branches allows developers to switch between different versions of the code base, and to work independently from other members of the team. When developers raise a [pull request ↗](#) (PR) to propose updates from one branch to another, or when changes are merged, these can be the trigger for [automated builds](#) to build and continuously test code.

By using the concepts and guidance that are described in this document, you can develop a LUIS app while tracking changes in a source control system, and follow these software engineering best practices:

- **Source Control**
 - Source code for your LUIS app is in a human-readable format.
 - The model can be built from source in a repeatable fashion.
 - The source code can be managed by a source code repository.
 - Credentials and secrets such as keys are never stored in source code.
- **Branching and Merging**
 - Developers can work from independent branches.
 - Developers can work in multiple branches concurrently.
 - It's possible to integrate changes to a LUIS app from one branch into another through rebase or merge.
 - Developers can merge a PR to the parent branch.

- **Versioning**
 - Each component in a large application should be versioned independently, allowing developers to detect breaking changes or updates just by looking at the version number.
- **Code Reviews**
 - The changes in the PR are presented as human readable source code that can be reviewed before accepting the PR.

Source control

To maintain the [App schema definition](#) of a LUIS app in a source code management system, use the [LUDown format \(.lu\)](#) representation of the app. `.lu` format is preferred to `.json` format because it's human readable, which makes it easier to make and review changes in PRs.

Save a LUIS app using the LUDown format

To save a LUIS app in `.lu` format and place it under source control:

- EITHER: [Export the app version](#) as `.lu` from the [LUIS portal](#) and add it to your source control repository
- OR: Use a text editor to create a `.lu` file for a LUIS app and add it to your source control repository

Tip

If you are working with the JSON export of a LUIS app, you can [convert it to LUDown](#). Use the `--sort` option to ensure that intents and utterances are sorted alphabetically.

Note that the `.LU` export capability built into the LUIS portal already sorts the output.

Build the LUIS app from source

For a LUIS app, to *build from source* means to [create a new LUIS app version](#) by [importing the .lu source](#), to [train the version](#) and to [publish it](#). You can do this in the LUIS portal, or at the command line:

- Use the LUIS portal to [import the .lu version](#) of the app from source control, and [train](#) and [publish](#) the app.
- Use the [Bot Framework Command Line Interface for LUIS](#) at the command line or in a CI/CD workflow to [import](#) the `.lu` version of the app from source control into a LUIS application, and [train](#) and [publish](#) the app.

Files to maintain under source control

The following types of files for your LUIS application should be maintained under source control:

- `.lu` file for the LUIS application
- [Unit Test definition files](#) (utterances and expected results)
- [Batch test files](#) (utterances and expected results) used for performance testing

Credentials and keys are not checked in

Do not include keys or similar confidential values in files that you check in to your repo where they might be visible to unauthorized personnel. The keys and other values that you should prevent from check-in include:

- LUIS Authoring and Prediction keys
- LUIS Authoring and Prediction endpoints
- Azure resource keys
- Access tokens, such as the token for an Azure [service principal](#) used for automation authentication

Strategies for securely managing secrets

Strategies for securely managing secrets include:

- If you're using Git version control, you can store runtime secrets in a local file and prevent check in of the file by adding a pattern to match the filename to a [.gitignore](#) file
- In an automation workflow, you can store secrets securely in the parameters configuration offered by that automation technology. For example, if you're using [GitHub Actions](#), you can store secrets securely in [GitHub secrets](#).

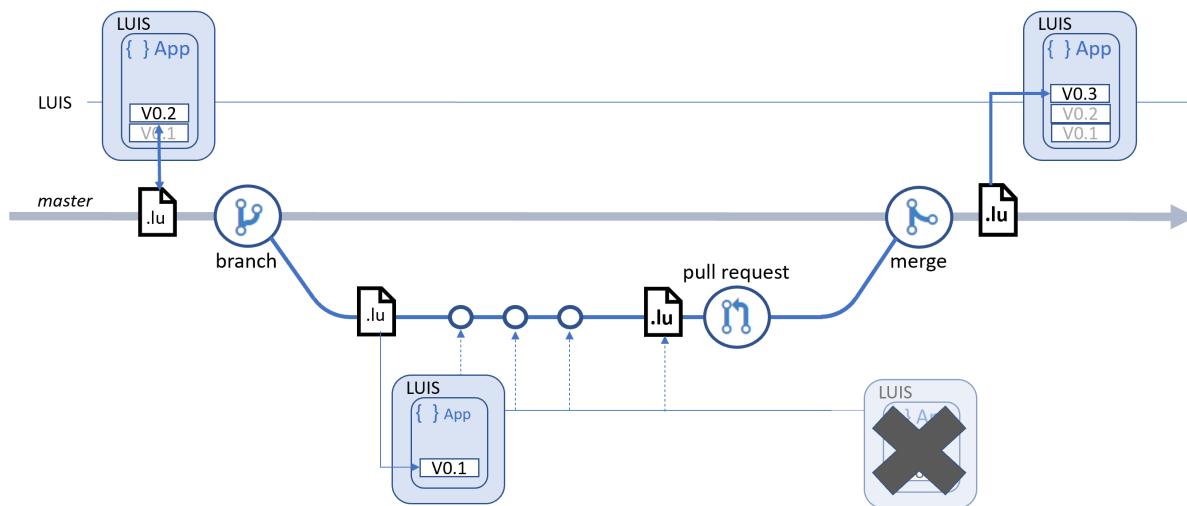
Branching and merging

Distributed version control systems like Git give flexibility in how team members publish, share, review, and iterate on code changes through development branches shared with others. Adopt a [Git branching strategy](#) that is appropriate for your team.

Whichever branching strategy you adopt, a key principle of all of them is that team members can work on the solution within a *feature branch* independently from the work that is going on in other branches.

To support independent working in branches with a LUIS project:

- **The main branch has its own LUIS app.** This app represents the current state of your solution for your project and its current active version should always map to the `.lu` source that is in the main branch. All updates to the `.lu` source for this app should be reviewed and tested so that this app could be deployed to build environments such as Production at any time. When updates to the `.lu` are merged into main from a feature branch, you should create a new version in the LUIS app and [bump the version number](#).
- **Each feature branch must use its own instance of a LUIS app.** Developers work with this app in a feature branch without risk of affecting developers who are working in other branches. This 'dev branch' app is a working copy that should be deleted when the feature branch is deleted.



Developers can work from independent branches

Developers can work on updates on a LUIS app independently from other branches by:

1. Creating a feature branch from the main branch (depending on your branch strategy, usually main or develop).

2. Create a new LUIS app in the LUIS portal (the "*dev branch app*") solely to support the work in the feature branch.

- If the `.lu` source for your solution already exists in your branch, because it was saved after work done in another branch earlier in the project, create your dev branch LUIS app by importing the `.lu` file.
- If you are starting work on a new project, you will not yet have the `.lu` source for your main LUIS app in the repo. You will create the `.lu` file by exporting your dev branch app from the portal when you have completed your feature branch work, and submit it as a part of your PR.

3. Work on the active version of your dev branch app to implement the required changes. We recommend that you work only in a single version of your dev branch app for all the feature branch work. If you create more than one version in your dev branch app, be careful to track which version contains the changes you want to check in when you raise your PR.

4. Test the updates - see [Testing for LUIS DevOps](#) for details on testing your dev branch app.

5. Export the active version of your dev branch app as `.lu` from the [versions list](#).

6. Check in your updates and invite peer review of your updates. If you're using GitHub, you'll raise a [pull request ↗](#).

7. When the changes are approved, merge the updates into the main branch. At this point, you will create a new [version](#) of the *main* LUIS app, using the updated `.lu` in main. See [Versioning](#) for considerations on setting the version name.

8. When the feature branch is deleted, it's a good idea to delete the dev branch LUIS app you created for the feature branch work.

Developers can work in multiple branches concurrently

If you follow the pattern described above in [Developers can work from independent branches](#), then you will use a unique LUIS application in each feature branch. A single developer can work on multiple branches concurrently, as long as they switch to the correct dev branch LUIS app for the branch they're currently working on.

We recommend that you use the same name for both the feature branch and for the dev branch LUIS app that you create for the feature branch work, to make it less likely that you'll accidentally work on the wrong app.

As noted above, we recommend that for simplicity, you work in a single version in each dev branch app. If you are using multiple versions, take care to activate the correct version as you switch between dev branch apps.

Multiple developers can work on the same branch concurrently

You can support multiple developers working on the same feature branch at the same time:

- Developers check out the same feature branch and push and pull changes submitted by themselves and other developers while work proceeds, as normal.
- If you follow the pattern described above in [Developers can work from independent branches](#), then this branch will use a unique LUIS application to support development. That 'dev branch' LUIS app will be created by the first member of the development team who begins work in the feature branch.
- [Add team members as contributors](#) to the dev branch LUIS app.
- When the feature branch work is complete, export the active version of the dev branch LUIS app as `.lu` from the [versions list](#), save the updated `.lu` file in the repo, and check in and PR the changes.

Incorporating changes from one branch to another with rebase or merge

Some other developers on your team working in another branch may have made updates to the `.lu` source and merged them to the main branch after you created your feature branch. You may want to incorporate their changes into your working version before you continue to make own changes within your feature branch. You can do this by [rebase or merge to main ↗](#) in the same way as any other code asset. Since the LUIS app in LUDown format is human readable, it supports merging using standard merge tools.

Follow these tips if you're rebasing your LUIS app in a feature branch:

- Before you rebase or merge, make sure your local copy of the `.lu` source for your app has all your latest changes that you've applied using the LUIS portal, by re-exporting your app from the portal first. That way, you can make sure that any changes you've made in the portal and not yet exported don't get lost.

- During the merge, use standard tools to resolve any merge conflicts.
- Don't forget after rebase or merge is complete to re-import the app back into the portal, so that you're working with the updated app as you continue to apply your own changes.

Merge PRs

After your PR is approved, you can merge your changes to your main branch. No special considerations apply to the LUDown source for a LUIS app: it's human readable and so supports merging using standard Merge tools. Any merge conflicts may be resolved in the same way as with other source files.

After your PR has been merged, it's recommended to cleanup:

- Delete the branch in your repo
- Delete the 'dev branch' LUIS app you created for the feature branch work.

In the same way as with application code assets, you should write unit tests to accompany LUIS app updates. You should employ continuous integration workflows to test:

- Updates in a PR before the PR is merged
- The main branch LUIS app after a PR has been approved and the changes have been merged into main.

For more information on testing for LUIS DevOps, see [Testing for DevOps for LUIS](#). For more details on implementing workflows, see [Automation workflows for LUIS DevOps](#).

Code reviews

A LUIS app in LUDown format is human readable, which supports the communication of changes in a PR suitable for review. Unit test files are also written in LUDown format and also easily reviewable in a PR.

Versioning

An application consists of multiple components that might include things such as a bot running in [Azure AI Bot Service](#), [QnA Maker](#), [Azure AI Speech service](#), and more. To achieve the goal of loosely coupled applications, use [version control](#) so that each component of an application is versioned independently, allowing developers to detect

breaking changes or updates just by looking at the version number. It's easier to version your LUIS app independently from other components if you maintain it in its own repo.

The LUIS app for the main branch should have a versioning scheme applied. When you merge updates to the `.1u` for a LUIS app into main, you'll then import that updated source into a new version in the LUIS app for the main branch.

It is recommended that you use a numeric versioning scheme for the main LUIS app version, for example:

```
major.minor[.build[.revision]]
```

Each update the version number is incremented at the last digit.

The major / minor version can be used to indicate the scope of the changes to the LUIS app functionality:

- Major Version: A significant change, such as support for a new [Intent](#) or [Entity](#)
- Minor Version: A backwards-compatible minor change, such as after significant new training
- Build: No functionality change, just a different build.

Once you've determined the version number for the latest revision of your main LUIS app, you need to build and test the new app version, and publish it to an endpoint where it can be used in different build environments, such as Quality Assurance or Production. It's highly recommended that you automate all these steps in a continuous integration (CI) workflow.

See:

- [Automation workflows](#) for details on how to implement a CI workflow to test and release a LUIS app.
- [Release Management](#) for information on how to deploy your LUIS app.

Versioning the 'feature branch' LUIS app

When you are working with a 'dev branch' LUIS app that you've created to support work in a feature branch, you will be exporting your app when your work is complete and you will include the updated `'1u` in your PR. The branch in your repo, and the 'dev branch' LUIS app should be deleted after the PR is merged into main. Since this app exists solely to support the work in the feature branch, there's no particular versioning scheme you need to apply within this app.

When your changes in your PR are merged into main, that is when the versioning should be applied, so that all updates to main are versioned independently.

Next steps

- Learn about [testing for LUIS DevOps](#)
- Learn how to [implement DevOps for LUIS with GitHub](#)

Testing for LUIS DevOps

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Software engineers who are developing a Language Understanding (LUIS) app can apply DevOps practices around [source control](#), [automated builds](#), [testing](#), and [release management](#) by following these guidelines.

In agile software development methodologies, testing plays an integral role in building quality software. Every significant change to a LUIS app should be accompanied by tests designed to test the new functionality the developer is building into the app. These tests are checked into your source code repository along with the `.luis` source of your LUIS app. The implementation of the change is finished when the app satisfies the tests.

Tests are a critical part of [CI/CD workflows](#). When changes to a LUIS app are proposed in a pull request (PR) or after changes are merged into your main branch, then CI workflows should run the tests to verify that the updates haven't caused any regressions.

How to do Unit testing and Batch testing

There are two different kinds of testing for a LUIS app that you need to perform in continuous integration workflows:

- **Unit tests** - Relatively simple tests that verify the key functionality of your LUIS app. A unit test passes when the expected intent and the expected entities are returned for a given test utterance. All unit tests must pass for the test run to complete successfully.
This kind of testing is similar to [Interactive testing](#) that you can do in the [LUIS portal](#).
- **Batch tests** - Batch testing is a comprehensive test on your current trained model to measure its performance. Unlike unit tests, batch testing isn't pass/fail testing. The expectation with batch testing is not that every test will return the expected

intent and expected entities. Instead, a batch test helps you view the accuracy of each intent and entity in your app and helps you to compare over time as you make improvements.

This kind of testing is the same as the [Batch testing](#) that you can perform interactively in the LUIS portal.

You can employ unit testing from the beginning of your project. Batch testing is only really of value once you've developed the schema of your LUIS app and you're working on improving its accuracy.

For both unit tests and batch tests, make sure that your test utterances are kept separate from your training utterances. If you test on the same data you train on, you'll get the false impression your app is performing well when it's just overfitting to the testing data. Tests must be unseen by the model to test how well it is generalizing.

Writing tests

When you write a set of tests, for each test you need to define:

- Test utterance
- Expected intent
- Expected entities.

Use the LUIS [batch file syntax](#) to define a group of tests in a JSON-formatted file. For example:

```
JSON

[
  {
    "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
      }
    ]
  }
]
```

Some test tools, such as [NLU.DevOps](#) also support LUDown-formatted test files.

Designing unit tests

Unit tests should be designed to test the core functionality of your LUIS app. In each iteration, or sprint, of your app development, you should write a sufficient number of tests to verify that the key functionality you are implementing in that iteration is working correctly.

In each unit test, for a given test utterance, you can:

- Test that the correct intent is returned
- Test that the 'key' entities - those that are critical to your solution - are being returned.
- Test that the [prediction score](#) for intent and entities exceeds a threshold that you define. For example, you could decide that you will only consider that a test has passed if the prediction score for the intent and for your key entities exceeds 0.75.

In unit tests, it's a good idea to test that your key entities have been returned in the prediction response, but to ignore any false positives. *False positives* are entities that are found in the prediction response but which are not defined in the expected results for your test. By ignoring false positives, it makes it less onerous to author unit tests while still allowing you to focus on testing that the data that is key to your solution is being returned in a prediction response.

Tip

The [NLU.DevOps](#) tool supports all your LUIS testing needs. The `compare` command when used in [unit test mode](#) will assert that all tests pass, and will ignore false positive results for entities that are not labeled in the expected results.

Designing Batch tests

Batch test sets should contain a large number of test cases, designed to test across all intents and all entities in your LUIS app. See [Batch testing in the LUIS portal](#) for information on defining a batch test set.

Running tests

The LUIS portal offers features to help with interactive testing:

- **Interactive testing** allows you to submit a sample utterance and get a response of LUIS-recognized intents and entities. You verify the success of the test by visual inspection.
- **Batch testing** uses a batch test file as input to validate your active trained version to measure its prediction accuracy. A batch test helps you view the accuracy of each intent and entity in your active version, displaying results with a chart.

Running tests in an automated build workflow

The interactive testing features in the LUIS portal are useful, but for DevOps, automated testing performed in a CI/CD workflow brings certain requirements:

- Test tools must run in a workflow step on a build server. This means the tools must be able to run on the command line.
- The test tools must be able to execute a group of tests against an endpoint and automatically verify the expected results against the actual results.
- If the tests fail, the test tools must return a status code to halt the workflow and "fail the build".

LUIS does not offer a command-line tool or a high-level API that offers these features. We recommend that you use the [NLU.DevOps](#) tool to run tests and verify results, both at the command line and during automated testing within a CI/CD workflow.

The testing capabilities that are available in the LUIS portal don't require a published endpoint and are a part of the LUIS authoring capabilities. When you're implementing testing in an automated build workflow, you must publish the LUIS app version to be tested to an endpoint so that test tools such as NLU.DevOps can send prediction requests as part of testing.

Tip

- If you're implementing your own testing solution and writing code to send test utterances to an endpoint, remember that if you are using the LUIS authoring key, the allowed transaction rate is limited to 5TPS. Either throttle the sending rate or use a prediction key instead.
- When sending test queries to an endpoint, remember to use `log=false` in the query string of your prediction request. This ensures that your test utterances do not get logged by LUIS and end up in the endpoint utterances review list.

presented by the LUIS **active learning** feature and, as a result, accidentally get added to the training utterances of your app.

Running Unit tests at the command line and in CI/CD workflows

You can use the [NLU.DevOps](#) package to run tests at the command line:

- Use the NLU.DevOps [test command](#) to submit tests from a test file to an endpoint and to capture the actual prediction results in a file.
- Use the NLU.DevOps [compare command](#) to compare the actual results with the expected results defined in the input test file. The `compare` command generates NUnit test output, and when used in [unit test mode](#) by use of the `--unit-test` flag, will assert that all tests pass.

Running Batch tests at the command line and in CI/CD workflows

You can also use the NLU.DevOps package to run batch tests at the command line.

- Use the NLU.DevOps [test command](#) to submit tests from a test file to an endpoint and to capture the actual prediction results in a file, same as with unit tests.
- Use the NLU.DevOps [compare command](#) in [Performance test mode](#) to measure the performance of your app. You can also compare the performance of your app against a baseline performance benchmark, for example, the results from the latest commit to main or the current release. In Performance test mode, the `compare` command generates NUnit test output and [batch test results](#) in JSON format.

Luis non-deterministic training and the effect on testing

When LUIS is training a model, such as an intent, it needs both positive data - the labeled training utterances that you've supplied to train the app for the model - and negative data - data that is *not* valid examples of the usage of that model. During training, LUIS builds the negative data of one model from all the positive data you've supplied for the other models, but in some cases that can produce a data imbalance. To avoid this imbalance, LUIS samples a subset of the negative data in a non-deterministic

fashion to optimize for a better balanced training set, improved model performance, and faster training time.

The result of this non-deterministic training is that you may get a slightly [different prediction response between different training sessions](#), usually for intents and/or entities where the [prediction score](#) is not high.

If you want to disable non-deterministic training for those LUIS app versions that you're building for the purpose of testing, use the [Version settings API](#) with the `UseAllTrainingData` setting set to `true`.

Next steps

- Learn about [implementing CI/CD workflows](#)
- Learn how to [implement DevOps for LUIS with GitHub](#)

Plan your LUIS app

Article • 07/18/2023

Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

A Language Understanding (LUIS) app schema contains [intents](#) and [entities](#) relevant to your subject [domain](#). The intents classify user [utterances](#), and the entities extract data from the user utterances. Intents and entities relevant to your subject domain. The intents classify user utterances.

A LUIS app learns and performs most efficiently when you iteratively develop it. Here's a typical iteration cycle:

1. Create a new version
2. Edit the LUIS app schema. This includes:
 - Intents with example utterances
 - Entities
 - Features
3. Train, test, and publish
4. Test for active learning by reviewing utterances sent to the prediction endpoint
5. Gather data from endpoint queries



Identify your domain

A LUIS app is centered around a subject domain. For example, you may have a travel app that handles 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 relevant 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 two intents, BookFlight and GetWeather for these actions.

In a more complex app with more functions, you likely would have more intents, and you should define them carefully so they aren't 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 that they overlap.

If you don't need to identify overall user intention, add all the example user utterances to the `None` intent. If your app grows into needing more intents, you can create them later.

Create example utterances for each intent

To start, avoid creating too many utterances for each intent. Once you have determined the intents you need for your app, create 15 to 30 example utterances per intent. Each utterance should be different from the previously provided utterances. Include a variety of word counts, word choices, verb tenses, and [punctuation](#).

For more information, see [understanding good utterances for LUIS apps](#).

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. Entities are important for accomplishing an intent.

When determining which entities to use in your app, remember that there are different types of entities for capturing relationships between object types. See [Entities in LUIS](#) for more information about the different types.

Tip

Luis offers [prebuilt entities](#) for common, conversational user scenarios. Consider using prebuilt entities as a starting point for your application development.

Intents versus entities

An intent is the desired outcome of the *whole* utterance while entities are pieces of data extracted from the utterance. Usually intents are tied to actions, which the client application should take. Entities are information needed to perform this action. From a programming perspective, an intent would trigger a method call and the entities would be used as parameters to that method call.

This utterance *must* have an intent and *may* have entities:

"Buy an airline ticket from Seattle to Cairo"

This utterance has a single intention:

- Buying a plane ticket

This utterance may have several entities:

- Locations of Seattle (origin) and Cairo (destination)
- The quantity of a single ticket

Resolution in utterances with more than one function or intent

In many cases, especially when working with natural conversation, users provide an utterance that can contain more than one function or intent. To address this, a general strategy is to understand that output can be represented by both intents and entities. This representation should be mappable to your client application's actions, and doesn't need to be limited to intents.

Int-ent-ties is the concept that actions (usually understood as intents) might also be captured as entities in the app's output, and mapped to specific actions. *Negation, for example, commonly* relies on intent and entity for full extraction. Consider the following two utterances, which are similar in word choice, but have different results:

- "Please schedule my flight from Cairo to Seattle"
- "Cancel my flight from Cairo to Seattle"

Instead of having two separate intents, you should create a single intent with a FlightAction machine learning entity. This machine learning entity should extract the details of the action for both scheduling and canceling requests, and either an origin or destination location.

This FlightAction entity would be structured with the following top-level machine learning entity, and subentities:

- FlightAction
 - Action
 - Origin
 - Destination

To help with extraction, you would add features to the subentities. You would choose features based on the vocabulary you expect to see in user utterances, and the values you want returned in the prediction response.

Best practices

Plan Your schema

Before you start building your app's schema, you should identify how and where you plan to use this app. The more thorough and specific your planning, the better your app becomes.

- Research targeted users
- Define end-to-end personas to represent your app - voice, avatar, issue handling (proactive, reactive)
- Identify channels of user interactions (such as text or speech), handing off to existing solutions or creating a new solution for this app
- End-to-end user journey
 - What do you expect this app to do and not do? What are the priorities of what it should do?
 - What are the main use cases?
- Collecting data - [learn about collecting and preparing data](#)

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.

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.

Build your app iteratively with versions

Each authoring cycle should be contained within a new [version](#), cloned from an existing version.

Don't publish too quickly

Publishing your app too quickly and without proper planning may lead to several issues such as:

- Your app will not work in your actual scenario at an acceptable level of performance.
- The schema (intents and entities) might not be appropriate, and if you have developed client app logic following the schema, you may need to redo it. This might cause unexpected delays and extra costs to the project you are working on.
- Utterances you add to the model might cause biases towards example utterances that are hard to debug and identify. It will also make removing ambiguity difficult after you have committed to a certain schema.

Do monitor the performance of your app

Monitor the prediction accuracy using a [batch test set](#).

Keep a separate set of utterances that aren'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.

Don't create phrase lists with all possible values

Provide a few examples in the [phrase lists](#) but not every word or phrase. LUIS generalizes and takes context into account.

Next steps

[Intents](#)

Intents

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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 would have 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 intents

LUIS provides prebuilt intents and their utterances for each of its prebuilt domains. Intents can be added without adding the whole domain. Adding an intent is the process of adding an intent and its utterances to your app. Both the intent name and the utterance list can be modified.

Return all intents' scores

You assign an utterance to a single intent. When LUIS receives an utterance, by default it returns the top intent for that utterance.

If you want the scores for all intents for the utterance, you can provide a flag in the query string of the prediction API.

Prediction API version	Flag
V2	<code>verbose=true</code>
V3	<code>show-all-intents=true</code>

Intent compared to entity

The intent represents the action the application should take for the user, based on the entire 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 from the table above. Then create entities to represent parameters required to execute the action.

Intent	Entity	Example utterance
CheckWeather	{ "type": "location", "entity": "Seattle" } { "type": "builtin.datetimeV2.date", "entity": "tomorrow", "resolution": "2018-05-23" }	What's the weather like in <code>Seattle</code> <code>tomorrow</code> ?
CheckWeather	{ "type": "date_range", "entity": "this weekend" }	Show me the forecast for <code>this weekend</code>

None intent

The **None** intent is created but left empty on purpose. The **None** intent is a required intent and can't be deleted or renamed. Fill it with utterances that are outside of your domain.

The **None** intent is the fallback intent, and should have 10% of the total utterances. It is important in every app, because it's used to teach LUIS utterances that are not important in the app domain (subject area). 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.

When an utterance is predicted as the **None** intent, the client application can ask more questions or provide a menu to direct the user to valid choices.

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. For example, do not have most of your intents with 10 utterances and another intent with 500 utterances. This is not balanced. In this situation, you would want to 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 the [limits](#) to understand how many intents you can add to a model.

Tip

If you need more than the maximum number of intents, consider whether your system is using too many intents and determine if 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, two intents: 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.

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](#).

Best Practices for Intents:

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:

1. Book a flight
2. Book a hotel

"Book a flight" and "book a hotel" use the same vocabulary of "book a <noun>". This format is the same so it should be the same intent with the different words of flight and hotel as extracted entities.

Do add features to intents

Features describe concepts for an intent. A feature can be a phrase list of words that are significant to that intent or an entity that is significant to that intent.

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 variance doesn't happen. Good distinction for intents should result in the expected top intent every time.

Balance utterances across intents

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 500 example utterances and all your other intents with 10 example utterances, the 500-utterance intent will have a higher rate of prediction.

Add example utterances to none intent

This intent is the fallback intent, indicating everything outside your application. Add one example utterance to the None intent for every 10 example utterances in the rest of your LUIS app.

Don't add many example utterances to intents

After the app is published, only add utterances from active learning in the development lifecycle process. If utterances are too similar, add a pattern.

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 example, for a bot 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.

Next steps

[How to use intents](#)

Entity types

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

An entity is an item or an element that is relevant to the user's intent. Entities define data that can be extracted from the utterance and is essential to complete a user's required action. For example:

Utterance	Intent predicted	Entities extracted	Explanation
Hello, how are you?	Greeting	-	Nothing to extract.
I want to order a small pizza	orderPizza	'small'	'Size' entity is extracted as 'small'.
Turn off bedroom light	turnOff	'bedroom'	'Room' entity is extracted as 'bedroom'.
Check balance in my savings account ending in 4406	checkBalance	'savings', '4406'	'accountType' entity is extracted as 'savings' and 'accountNumber' entity is extracted as '4406'.
Buy 3 tickets to New York	buyTickets	'3', 'New York'	'ticketsCount' entity is extracted as '3' and 'Destination' entity is extracted as 'New York'.

Entities are optional but recommended. You don't need to create entities for every concept in your app, only when:

- The client application needs the data, or
- The entity acts as a hint or signal to another entity or intent. To learn more about entities as Features go to [Entities as features](#).

Entity types

To create an entity, you have to give it a name and a type. There are several types of entities in LUIS.

List entity

A list entity represents a fixed, closed set of related words along with their synonyms. You can use list entities to recognize multiple synonyms or variations and extract a normalized output for them. Use the *recommend* option to see suggestions for new words based on the current list.

A list entity isn't machine-learned, meaning that LUIS doesn't discover more values for list entities. LUIS marks any match to an item in any list as an entity in the response.

Matching list entities is both case sensitive and it has to be an exact match. Normalized values are also used when matching the list entity. For example:

Normalized value	Synonyms
Small	<code>sm</code> , <code>sml</code> , <code>tiny</code> , <code>smallest</code>
Medium	<code>md</code> , <code>mdm</code> , <code>regular</code> , <code>average</code> , <code>middle</code>
Large	<code>lg</code> , <code>lrg</code> , <code>big</code>

See the [list entities reference article](#) for more information.

Regex entity

A regular expression entity extracts an entity based on a regular expression pattern you provide. It ignores case and ignores cultural variant. Regular expression entities are best for structured text or a predefined sequence of alphanumeric values that are expected in a certain format. For example:

Entity	Regular expression	Example
Flight Number	<code>flight [A-Z]{2} [0-9]{4}</code>	<code>flight AS 1234</code>
Credit Card Number	<code>[0-9]{16}</code>	<code>5478789865437632</code>

See the [regex entities reference article](#) for more information.

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 can't be modified.

Prebuilt entity	Example value
PersonName	James, Bill, Tom
DatetimeV2	2019-05-02, May 2nd, 8am on May 2nd 2019

See the [prebuilt entities reference article](#) for more information.

Pattern.Any entity

A pattern.Any entity is a variable-length placeholder used only in a pattern's template utterance to mark where the entity begins and ends. It follows a specific rule or pattern and best used for sentences with fixed lexical structure. For example:

Example utterance	Pattern	Entity
Can I have a burger please?	Can I have a {meal} [please] [?]	burger
Can I have a pizza?	Can I have a {meal} [please] [?]	pizza
Where can I find The Great Gatsby?	Where can I find {bookName}?	The Great Gatsby

See the [Pattern.Any entities reference article](#) for more information.

Machine learned (ML) entity

Machine learned entity uses context to extract entities based on labeled examples. It is the preferred entity for building LUIS applications. It relies on machine-learning algorithms and requires labeling to be tailored to your application successfully. Use an ML entity to identify data that isn't always well formatted but have the same meaning.

Example utterance	Extracted product entity
I want to buy a book.	'book'
Can I get these shoes please?	'shoes'

Example utterance	Extracted product entity
Add those shorts to my basket.	'shorts'

See [Machine learned entities](#) for more information.

ML Entity with Structure

An ML entity can be composed of smaller sub-entities, each of which can have its own properties. For example, an *Address* entity could have the following structure:

- Address: 4567 Main Street, NY, 98052, USA
 - Building Number: 4567
 - Street Name: Main Street
 - State: NY
 - Zip Code: 98052
 - Country: USA

Building effective ML entities

To build machine learned entities effectively, follow these best practices:

- If you have a machine learned entity with sub-entities, make sure that the different orders and variants of the entity and sub-entities are presented in the labeled utterances. Labeled example utterances should include all valid forms, and include entities that appear and are absent and also reordered within the utterance.
- Avoid overfitting the entities to a fixed set. Overfitting happens when the model doesn't generalize well, and is a common problem in machine-learning models. This implies the app wouldn't work on new types of examples adequately. In turn, you should vary the labeled example utterances so the app can generalize beyond the limited examples you provide.
- Your labeling should be consistent across the intents. This includes even utterances you provide in the *None* intent that includes this entity. Otherwise the model won't be able to determine the sequences effectively.

Entities as features

Another important function of entities is to use them as features or distinguishing traits for another intents or entities so that your system observes and learns through them.

Entities as features for intents

You can use entities as a signal for an intent. For example, the presence of a certain entity in the utterance can distinguish which intent does it fall under.

Example utterance	Entity	Intent
Book me a <i>flight</i> to New York.	City	Book Flight
Book me the <i>main conference room</i> .	Room	Reserve Room

Entities as Feature for entities

You can also use entities as an indicator of the presence of other entities. A common example of this is using a prebuilt entity as a feature for another ML entity. If you're building a flight booking system and your utterance looks like 'Book me a flight from Cairo to Seattle', you likely will have *Origin City* and *Destination City* as ML entities. A good practice would be to use the prebuilt GeographyV2 entity as a feature for both entities.

For more information, see the [GeographyV2 entities reference article](#).

You can also use entities as required features for other entities. This helps in the resolution of extracted entities. For example, if you're creating a pizza-ordering application and you have a Size ML entity, you can create SizeList list entity and use it as a required feature for the Size entity. Your application will return the normalized value as the extracted entity from the utterance.

See [features](#) for more information, and [prebuilt entities](#) to learn more about prebuilt entities resolution available in your culture.

Data from entities

Most chat bots 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

Best practices for entities

Use machine-learning entities

Machine learned entities are tailored to your app and require labeling to be successful. If you aren't using machine learned entities, you might be using the wrong entities.

Machine learned entities can use other entities as features. These other entities can be custom entities such as regular expression entities or list entities, or you can use prebuilt entities as features.

Learn about [effective machine learned entities](#).

Next steps

- [How to use entities in your LUIS app](#)
- [Utterances concepts](#)

Utterances

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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

Collect utterances that you think users will enter. Include utterances, which mean the same thing but are constructed in various 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](#) - using both correct and incorrect grammar

Choose varied utterances

When you start [adding example utterances](#) to your LUIS model, there are several principles to keep in mind:

Utterances aren't always well formed

Your app may need to process sentences, like "Book a ticket to Paris for me", or a fragment of a sentence, like "Booking" or "Paris flight". Users also often make spelling mistakes. When planning your app, consider whether or not you want to 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 are common terms or phrases might not be common for the typical user of your client application. They may not have domain experience or use different terminology. Be careful when using terms or phrases that a user would only say if they were an expert.

Choose varied terminology and phrasing

You will find that even if you make efforts to create varied sentence patterns, you will still repeat some vocabulary. For example, the following utterances have similar meaning, but different terminology and phrasing:

- "*How do I get a computer?*"
- "*Where do I get a computer?*"
- "*I want to get a computer, how do I go about it?*"
- "*When can I have a computer?*"

The core term here, *computer*, isn't varied. Use alternatives such as desktop computer, laptop, workstation, or even just machine. LUIS can intelligently infer synonyms from context, but when you create utterances for training, it's always 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 few example utterances, LUIS may not accurately predict the intent.

Add small groups of utterances

Each time you iterate on your model to improve it, don't add large quantities of utterances. Consider adding utterances in quantities of 15. Then [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 the endpoint utterances](#) for correct intent prediction and entity extraction.

Utterance normalization

Utterance normalization is the process of ignoring the effects of types of text, such as punctuation and diacritics, during training and prediction.

Utterance normalization settings are turned off by default. These settings include:

- Word forms
- Diacritics
- Punctuation

If you turn on a normalization setting, scores in the **Test** pane, batch tests, and endpoint queries will change for all utterances for that normalization setting.

When you clone a version in the LUIS portal, the version settings are kept in the new cloned version.

Set your app's version settings using the LUIS portal by selecting **Manage** from the top navigation menu, in the **Application Settings** page. You can also use the [Update Version Settings API](#). See the [Reference](#) documentation for more information.

Word forms

Normalizing **word forms** ignores the differences in words that expand beyond the root.

Diacritics

Diacritics are marks or signs within the text, such as:

İ ı Ş ş Ğ ğ ö ü

Punctuation marks

Normalizing **punctuation** means that before your models get trained and before your endpoint queries get predicted, punctuation will be removed from the utterances.

Punctuation is a separate token in LUIS. An utterance that contains a period at the end is a separate utterance than one that does not contain a period at the end, 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 to include example utterances that use punctuation, and ones that don't, for both styles to return the same relative scores.

Make sure the model handles punctuation either in the example utterances (both having and not having punctuation) or in [patterns](#) where it is easier to ignore punctuation. For example: 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 with all utterances

Training is generally non-deterministic: utterance prediction can 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 data 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 using the testing pane in the LUIS portal are not sent through the endpoint, and don't 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

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.

In the following utterances, the word fair is a homograph, which means it's spelled the same but has a different meaning:

- "*What kind of county fairs are happening in the Seattle area this summer?*"
- "*Is the current 2-star rating for the restaurant fair?*"

If you want an event entity to find all event data, label the word fair in the first utterance, but not in the second.

Don't ignore possible utterance variations

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 the same format	Do use varying formats
Buy a ticket to Seattle	Buy 1 ticket to Seattle
Buy a ticket to Paris	Reserve two seats on the red eye to Paris next Monday
Buy a ticket to Orlando	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 add too many example utterances to intents

After the app is published, only add utterances from active learning in the development lifecycle process. If utterances are too similar, add a pattern.

Next steps

- [Intents](#)
- [Patterns and features concepts](#)

Patterns in LUIS apps

Article • 07/18/2023

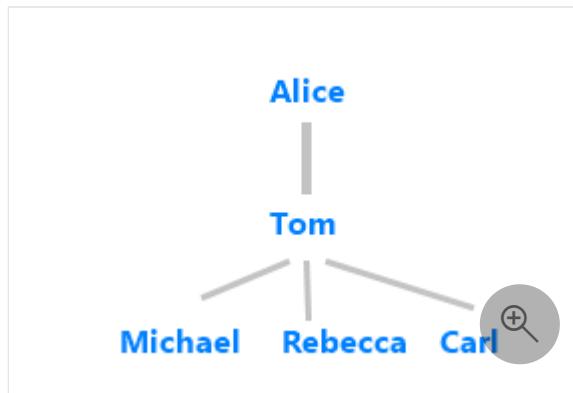
ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Patterns are designed to improve accuracy when multiple utterances are very similar. A pattern allows you to gain more accuracy for an intent without providing several 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 named Alice, and a team of subordinates named: Michael, Rebecca, and Carl.



Utterances	Intent predicted	Intent score
Who is Tom's subordinate?	GetOrgChart	0.30
Who is the subordinate of Tom?	GetOrgChart	0.30

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 techniques. 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-learning 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-learning entity.

Do not expect to see improved entity prediction if you collapse multiple utterances into a single pattern. For simple entities to be utilized by your app, you need to add utterances or use list entities.

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 can 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 by 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.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.

For example, consider a Human Resources app that helps employees find company documents. This app might need to understand the following example utterances.

- "Where is **HRF-123456**?"
- "Who authored **HRF-123234**?"
- "Is **HRF-456098** published in French?"

However, each document has both a formatted name (used in the above list), and a human-readable name, such as Request relocation from employee new to the company 2018 version 5.

Utterances with the human-readable name might look like:

- "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**?"
- "Is **Request relocation from employee new to the company 2018 version 5** is published in French?"

The utterances include 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 document name, so LUIS correctly extracts the form name. For example, the following template utterances:

- Where is {FormName}[]?
- Who authored {FormName}[]?
- Is {FormName} is published in French[]?

Best practices for Patterns:

Do add patterns in later iterations

You should understand how the app behaves before adding patterns because patterns are weighted more heavily than example utterances and will skew confidence.

Once you understand how your app behaves, add patterns as they apply to your app. You do not need to add them each time you iterate on the app's design.

There is no harm in adding them in the beginning of your model design, but it is easier to see how each pattern changes the model after the model is tested with utterances.

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.

Features

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

Machine-learning features give LUIS important cues for where to look for things that distinguish a concept. They're hints that LUIS can use, but they aren't hard rules. LUIS uses these hints with the labels to find the data.

A feature can be described as a function, like `f(x) = y`. In the example utterance, the feature tells you where to look for the distinguishing trait. Use this information to help create your schema.

Types of features

Features are a necessary part of your schema design. LUIS supports both phrase lists and models as features:

- Phrase list feature
- Model (intent or entity) as a feature

Find features in your example utterances

Because LUIS is a language-based application, the features are text-based. Choose text that indicates the trait you want to distinguish. For LUIS, the smallest unit is the *token*. For the English language, a token is a contiguous span of letters and numbers that has no spaces or punctuation.

Because spaces and punctuation aren't tokens, focus on the text clues that you can use as features. Remember to include variations of words, such as:

- Plural forms
- Verb tenses

- Abbreviations
- Spellings and misspellings

Determine if the text needs the following because it distinguishes a trait:

- Match an exact word or phrase: Consider adding a regular expression entity or a list entity as a feature to the entity or intent.
- Match a well-known concept like dates, times, or people's names: Use a prebuilt entity as a feature to the entity or intent.
- Learn new examples over time: Use a phrase list of some examples of the concept as a feature to the entity or intent.

Create a phrase list for a concept

A phrase list is a list of words or phrases that describe a concept. A phrase list is applied as a case-insensitive match at the token level.

When adding a phrase list, you can set the feature to [global](#). A global feature applies to the entire app.

When to use a phrase list

Use a phrase list when you need your LUIS app to generalize and identify new items for the concept. Phrase lists are like domain-specific vocabulary. They enhance the quality of understanding for intents and entities.

How to use a phrase list

With a phrase list, LUIS considers context and generalizes to identify items that are similar to, but aren't, an exact text match. Follow these steps to use a phrase list:

1. Start with a machine-learning entity:
2. Add example utterances.
3. Label with a machine-learning entity.
4. Add a phrase list:
5. Add words with similar meaning. Don't add every possible word or phrase. Instead, add a few words or phrases at a time. Then retrain and publish.
6. Review and add suggested words.

A typical scenario for a phrase list

A typical scenario for a phrase list is to boost words related to a specific idea.

Medical terms are a good example of words that might need a phrase list to boost their significance. These terms can have specific physical, chemical, therapeutic, or abstract meanings. LUIS won't know the terms are important to your subject domain without a phrase list.

For example, to extract the medical terms:

1. Create example utterances and label medical terms within those utterances.
2. Create a phrase list with examples of the terms within the subject domain. This phrase list should include the actual term you labeled and other terms that describe the same concept.
3. Add the phrase list to the entity or subentity that extracts the concept used in the phrase list. The most common scenario is a component (child) of a machine-learning entity. If the phrase list should be applied across all intents or entities, mark the phrase list as a global phrase list. The `enabledForAllModels` flag controls this model scope in the API.

Token matches for a phrase list

A phrase list always applies at the token level. The following table shows how a phrase list that has the word **Ann** applies to variations of the same characters in that order.

Token variation of "Ann"	Phrase list match when the token is found
ANN	Yes - token is Ann
aNN	
Ann's	Yes - token is Ann
Anne	No - token is Anne

A model as a feature helps another model

You can add a model (intent or entity) as a feature to another model (intent or entity). By adding an existing intent or entity as a feature, you're adding a well-defined concept that has labeled examples.

When adding a model as a feature, you can set the feature as:

- **Required**. A required feature must be found for the model to be returned from the prediction endpoint.

- **Global**. A global feature applies to the entire app.

When to use an entity as a feature to an intent

Add an entity as a feature to an intent when the detection of that entity is significant for the intent.

For example, if the intent is for booking a flight, like **BookFlight**, and the entity is ticket information (such as the number of seats, origin, and destination), then finding the ticket-information entity should add significant weight to the prediction of the **BookFlight** intent.

When to use an entity as a feature to another entity

An entity (A) should be added as a feature to another entity (B) when the detection of that entity (A) is significant for the prediction of entity (B).

For example, if a shipping-address entity is contained in a street-address subentity, then finding the street-address subentity adds significant weight to the prediction for the shipping address entity.

- Shipping address (machine-learning entity):
 - Street number (subentity)
 - Street address (subentity)
 - City (subentity)
 - State or Province (subentity)
 - Country/Region (subentity)
 - Postal code (subentity)

Nested subentities with features

A machine-learning subentity indicates a concept is present to the parent entity. The parent can be another subentity or the top entity. The value of the subentity acts as a feature to its parent.

A subentity can have both a phrase list and a model (another entity) as a feature.

When the subentity has a phrase list, it boosts the vocabulary of the concept but won't add any information to the JSON response of the prediction.

When the subentity has a feature of another entity, the JSON response includes the extracted data of that other entity.

Required features

A required feature has to be found in order for the model to be returned from the prediction endpoint. Use a required feature when you know your incoming data must match the feature.

If the utterance text doesn't match the required feature, it won't be extracted.

A required feature uses a non-machine-learning entity:

- Regular-expression entity
- List entity
- Prebuilt entity

If you're confident that your model will be found in the data, set the feature as required. A required feature doesn't return anything if it isn't found.

Continuing with the example of the shipping address:

Shipping address (machine learned entity)

- Street number (subentity)
- Street address (subentity)
- Street name (subentity)
- City (subentity)
- State or Province (subentity)
- Country/Region (subentity)
- Postal code (subentity)

Required feature using prebuilt entities

Prebuilt entities such as city, state, and country/region are generally a closed set of lists, meaning they don't change much over time. These entities could have the relevant recommended features and those features could be marked as required. However, the `isRequired` flag is only related to the entity it is assigned to and doesn't affect the hierarchy. If the prebuilt sub-entity feature is not found, this will not affect the detection and return of the parent entity.

As an example of a required feature, consider you want to detect addresses. You might consider making a street number a requirement. This would allow a user to enter "1

"Microsoft Way" or "One Microsoft Way", and both would resolve to the numeral "1" for the street number sub-entity. See the [prebuilt entity](#) article for more information.

Required feature using list entities

A [list entity](#) is used as a list of canonical names along with their synonyms. As a required feature, if the utterance doesn't include either the canonical name or a synonym, then the entity isn't returned as part of the prediction endpoint.

Suppose that your company only ships to a limited set of countries/regions. You can create a list entity that includes several ways for your customer to reference the country/region. If LUIS doesn't find an exact match within the text of the utterance, then the entity (that has the required feature of the list entity) isn't returned in the prediction.

Canonical name**	Synonyms
United States	U.S. U.S.A US USA 0

A client application, such as a chat bot, can ask a follow-up question to help. This helps the customer understand that the country/region selection is limited and *required*.

Required feature using regular expression entities

A [regular expression entity](#) that's used as a required feature provides rich text-matching capabilities.

In the shipping address example, you can create a regular expression that captures syntax rules of the country/region postal codes.

Global features

While the most common use is to apply a feature to a specific model, you can configure the feature as a [global feature](#) to apply it to your entire application.

The most common use for a global feature is to add an additional vocabulary to the app. For example, if your customers use a primary language, but expect to be able to

use another language within the same utterance, you can add a feature that includes words from the secondary language.

Because the user expects to use the secondary language across any intent or entity, add words from the secondary language to the phrase list. Configure the phrase list as a global feature.

Combine features for added benefit

You can use more than one feature to describe a trait or concept. A common pairing is to use:

- A phrase list feature: You can use multiple phrase lists as features to the same model.
- A model as a feature: [prebuilt entity](#), [regular expression entity](#), [list entity](#).

Example: ticket-booking entity features for a travel app

As a basic example, consider an app for booking a flight with a flight-reservation *intent* and a ticket-booking *entity*. The ticket-booking entity captures the information to book an airplane ticket in a reservation system.

The machine-learning entity for ticket-book has two subentities to capture origin and destination. The features need to be added to each subentity, not the top-level entity.

Name	Machine learning features
ticket-booking	+ Add feature
Origin	OriginPhraselist @ geographyV2
Destination	DestinationPhraselist @ geographyV2

The ticket-booking entity is a machine-learning entity, with subentities including *Origin* and *Destination*. These subentities both indicate a geographical location. To help extract the locations, and distinguish between *Origin* and *Destination*, each subentity should have features.

Type	Origin subentity	Destination subentity
Model as a feature	geographyV2 prebuilt entity	geographyV2 prebuilt entity
Phrase list	Origin words : start at, begin from, leave	Destination words : to, arrive, land at, go, going, stay, heading
Phrase list	Airport codes - same list for both origin and destination	Airport codes - same list for both origin and destination
Phrase list	Airport names - same list for both origin and destination	Airport codes - same list for both origin and destination

If you anticipate that people use airport codes and airport names, then LUIS should have phrase lists that use both types of phrases. Airport codes may be more common with text entered in a chatbot while airport names may be more common with spoken conversation such as a speech-enabled chatbot.

The matching details of the features are returned only for models, not for phrase lists because only models are returned in prediction JSON.

Ticket-booking labeling in the intent

After you create the machine-learning entity, you need to add example utterances to an intent, and label the parent entity and all subentities.

For the ticket booking example, Label the example utterances in the intent with the TicketBooking entity and any subentities in the text.

book me 2 adult business tickets to paris tomorrow on air france
 book a flight from seattle to hong kong on april 1 at 10 am
 leave san diego tomorrow heading to salt lake city

Entity Types:

- ticket-booking
- geography
- geographyV2

Scope Indicators:

- Destina... (red arrow under 'paris')
- Origin (red arrow under 'seattle')
- Destination (red arrow under 'hong kong')
- Origin (red arrow under 'san diego')
- Destination (red arrow under 'salt lake city')

Example: pizza ordering app

For a second example, consider an app for a pizza restaurant, which receives pizza orders including the details of the type of pizza someone is ordering. Each detail of the pizza should be extracted, if possible, in order to complete the order processing.

The machine-learning entity in this example is more complex with nested subentities, phrase lists, prebuilt entities, and custom entities.

Name	Machine learning features
Order	+ Add feature
FullPizzaWithModifiers	<input type="checkbox"/> CrustPhraseList <input type="checkbox"/> ModifierPhraseList <input type="checkbox"/> QuantityPhraseList <input type="checkbox"/> ScopePhraseList <input type="checkbox"/> SizePhraseList <input type="checkbox"/> ToppingPhraseList
PizzaType	+ Add feature
Size	<input checked="" type="radio"/> SizeList * <input type="checkbox"/> SizePhraseList + Add feature
Quantity	<input checked="" type="radio"/> number * <input type="checkbox"/> QuantityPhraseList + Add feature
Crust	<input checked="" type="radio"/> CrustList * <input type="checkbox"/> CrustPhraseList + Add feature
ToppingModifiers	<input type="checkbox"/> ModifierPhraseList <input type="checkbox"/> ScopePhraseList <input type="checkbox"/> ToppingPhraseList + Add feature
Topping	<input checked="" type="radio"/> ToppingList * <input type="checkbox"/> ToppingPhraseList + Add feature
Modifier	<input checked="" type="radio"/> ModifierList * <input type="checkbox"/> AddPhraseList <input type="checkbox"/> ModifierPhraseList <input type="checkbox"/> RemovePhraseList + Add feature
Scope	<input checked="" type="radio"/> ScopeList * <input type="checkbox"/> ScopePhraseList + Add feature
SideOrder	<input type="checkbox"/> SideOrderPhraseList + Add feature
SideProduct	+ Add feature

This example uses features at the subentity level and child of subentity level. Which level gets what kind of phrase list or model as a feature is an important part of your entity

design.

While subentities can have many phrase lists as features that help detect the entity, each subentity has only one model as a feature. In this [pizza app](#), those models are primarily lists.



The correctly labeled example utterances display in a way to show how the entities are nested.

Next steps

- LUIS application design
 - prebuilt models
 - intents
 - entities.

Prebuilt models

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Prebuilt models provide domains, intents, utterances, and entities. You can start your app with a prebuilt model or add a relevant model to your app later.

Types of prebuilt models

LUIS provides three types of prebuilt models. 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 pre-trained models 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 of a prebuilt domain are fully customizable once you've added them to your app.

ⓘ 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 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 a `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 adding example utterances to the intent about making hotel reservations and then retrain the app.

You can find a full listing of the prebuilt domains in the [Prebuilt domains reference](#).

Prebuilt intents

LUIS provides prebuilt intents and their utterances for each of its prebuilt domains. Intents can be added without adding the whole domain. Adding an intent is the process of adding an intent and its utterances to your app. 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.

Next steps

Learn how to [add prebuilt entities](#) to your app.

Extend app at prediction runtime

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

The app's schema (models and features) is trained and published to the prediction endpoint. This published model is used on the prediction runtime. You can pass new information, along with the user's utterance, to the prediction runtime to augment the prediction.

Two prediction runtime schema changes include:

- [External entities](#)
- [Dynamic lists](#)

External entities

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.

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.

External entities are part of the V3 authoring API. Learn more about [migrating](#) to this version.

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.

JSON

```
"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 this turn of the conversation, 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`.

JSON

```
"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 `preferExternalEntities` 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:

JSON

```
"datetimeV2": [
  {
    "type": "date",
    "values": [
      {
        "timex": "2019-06-21",
        "value": "2019-06-21"
      }
    ]
  }
]
```

If the user sends the external entity:

JSON

```
{
  "entityName": "datetimeV2",
  "startIndex": 0,
  "entityLength": 5,
  "resolution": {
```

```
        "date": "2019-06-21"
    }
}
```

If the `preferExternalEntities` is set to `false`, LUIS returns a response as if the external entity were not sent.

JSON

```
"datetimeV2": [
{
    "type": "date",
    "values": [
        {
            "timex": "2019-06-21",
            "value": "2019-06-21"
        }
    ]
}]
```

If the `preferExternalEntities` is set to `true`, LUIS returns a response including:

JSON

```
"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

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:

```
JSON

{
    "query": "Send Hazem a message to add an item to the meeting agenda
about LUIS.",
    "options": {
        "timezoneOffset": "-8:00"
    },
    "dynamicLists": [
        {
            "listEntity*": "ProductList",
            "requestLists": [
                {
                    "name": "Azure AI 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.

Next steps

- [Prediction score](#)
- [Authoring API V3 changes](#)

Prediction scores indicate prediction accuracy for intent and entities

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

A prediction score indicates the degree of confidence LUIS has for prediction results of 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

Top-scoring intent

Every utterance prediction returns a top-scoring intent. This prediction is a numerical comparison of prediction scores.

Proximity of scores to each other

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 using the correct querystring name/value pair.

Prediction API	Querystring name
V3	show-all-intents=true
V2	verbose=true

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 and consistently lower 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.

You can turn off the non-deterministic training by training with all data.

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

Application settings

Use [application settings](#) to control how diacritics and punctuation impact prediction scores.

Next steps

See [Add entities](#) to learn more about how to add entities to your LUIS app.

Design with intent and entity models

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Language understanding provides two types of models for you to define your app schema. Your app schema determines what information you receive from the prediction of a new user utterance.

The app schema is built from models you create using [machine teaching](#):

- [Intents](#) classify user utterances
- [Entities](#) extract data from utterance

Authoring uses machine teaching

LUIS's machine teaching methodology allows you to easily teach concepts to a machine. Understanding *machine learning* is not necessary to use LUIS. Instead, you as the teacher, communicates a concept to LUIS by providing examples of the concept and explaining how a concept should be modeled using other related concepts. You, as the teacher, can also improve LUIS's model interactively by identifying and fixing prediction mistakes.

Intents classify utterances

An intent classifies example utterances to teach LUIS about the intent. Example utterances within an intent are used as positive examples of the utterance. These same utterances are used as negative examples in all other intents.

Consider an app that needs to determine a user's intention to order a book and an app that needs the shipping address for the customer. This app has two intents: `OrderBook` and `ShippingLocation`.

The following utterance is a **positive example** for the `OrderBook` intent and a **negative example** for the `ShippingLocation` and `None` intents:

Buy the top-rated book on bot architecture.

Entities extract data

An entity represents a unit of data you want extracted from the utterance. A machine-learning entity is a top-level entity containing subentities, which are also machine-learning entities.

An example of a machine-learning entity is an order for a plane ticket. Conceptually this is a single transaction with many smaller units of data such as date, time, quantity of seats, type of seat such as first class or coach, origin location, destination location, and meal choice.

Intents versus entities

An intent is the desired outcome of the *whole* utterance while entities are pieces of data extracted from the utterance. Usually intents are tied to actions, which the client application should take. Entities are information needed to perform this action. From a programming perspective, an intent would trigger a method call and the entities would be used as parameters to that method call.

This utterance *must* have an intent and *may* have entities:

Buy an airline ticket from Seattle to Cairo

This utterance has a single intention:

- Buying a plane ticket

This utterance *may* have several entities:

- Locations of Seattle (origin) and Cairo (destination)
- The quantity of a single ticket

Entity model decomposition

LUIS supports *model decomposition* with the authoring APIs, breaking down a concept into smaller parts. This allows you to build your models with confidence in how the various parts are constructed and predicted.

Model decomposition has the following parts:

- [intents](#)
 - [features](#)
- [machine-learning entities](#)
 - subentities (also machine-learning entities)
 - [features](#)
 - [phrase list](#)
 - [non-machine-learning entities](#) such as [regular expressions](#), [lists](#), and [prebuilt entities](#)

Features

A [feature](#) is a distinguishing trait or attribute of data that your system observes. Machine learning features give LUIS important cues for where to look for things that will distinguish a concept. They are hints that LUIS can use, but not hard rules. These hints are used in conjunction with the labels to find the data.

Patterns

[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.

Extending the app at runtime

The app's schema (models and features) is trained and published to the prediction endpoint. You can [pass new information](#), along with the user's utterance, to the prediction endpoint to augment the prediction.

Next steps

- Understand [intents](#) and [entities](#).
- Learn more about [features](#)

Data collection for your app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

A Language Understanding (LUIS) app needs data as part of app development.

Data used in LUIS

Luis uses text as data to train and test your LUIS app for classification for [intents](#) and for extraction of [entities](#). You need a large enough data set that you have sufficient data to create separate data sets for both training and test that have the diversity and distribution called out specifically below. The data in each of these sets should not overlap.

Training data selection for example utterances

Select utterances for your training set based on the following criteria:

- **Real data is best:**
 - **Real data from client application:** Select utterances that are real data from your client application. If the customer sends a web form with their inquiry today, and you're building a bot, you can start by using the web form data.
 - **Crowd-sourced data:** If you don't have any existing data, consider crowd sourcing utterances. Try to crowd-source utterances from your actual user population for your scenario to get the best approximation of the real data your application will see. Crowd-sourced human utterances are better than computer-generated utterances. When you build a data set of synthetic utterances generated on specific patterns, it will lack much of the natural variation you'll see with people creating the utterances and won't end up generalizing well in production.
- **Data diversity:**

- **Region diversity:** Make sure the data for each intent is as diverse as possible including *phrasing* (word choice), and *grammar*. If you are teaching an intent about HR policies about vacation days, make sure you have utterances that represent the terms that are used for all regions you're serving. For example, in Europe people might ask about `taking a holiday` and in the US people might ask about `taking vacation days`.
 - **Language diversity:** If you have users with various native languages that are communicating in a second language, make sure to have utterances that represent non-native speakers.
 - **Input diversity:** Consider your data input path. If you are collecting data from one person, department or input device (microphone) you are likely missing diversity that will be important for your app to learn about all input paths.
 - **Punctuation diversity:** Consider that people use varying levels of punctuation in text applications and make sure you have a diversity of how punctuation is used. If you're using data that comes from speech, it won't have any punctuation, so your data shouldn't either.
- **Data distribution:** Make sure the data spread across intents represents the same spread of data your client application receives. If your LUIS app will classify utterances that are requests to schedule a leave (50%), but it will also see utterances about inquiring about leave days left (20%), approving leaves (20%) and some out of scope and chit chat (10%) then your data set should have the sample percentages of each type of utterance.
 - **Use all data forms:** If your LUIS app will take data in multiple forms, make sure to include those forms in your training utterances. For example, if your client application takes both speech and typed text input, you need to have speech to text generated utterances as well as typed utterances. You will see different variations in how people speak from how they type as well as different errors in speech recognition and typos. All of this variation should be represented in your training data.
 - **Positive and negative examples:** To teach a LUIS app, it must learn about what the intent is (positive) and what it is not (negative). In LUIS, utterances can only be positive for a single intent. When an utterance is added to an intent, LUIS automatically makes that same example utterance a negative example for all the other intents.
 - **Data outside of application scope:** If your application will see utterances that fall outside of your defined intents, make sure to provide those. The examples that aren't assigned to a particular defined intent will be labeled with the **None** intent.

It's important to have realistic examples for the **None** intent to properly predict utterances that are outside the scope of the defined intents.

For example, if you are creating an HR bot focused on leave time and you have three intents:

- schedule or edit a leave
- inquire about available leave days
- approve/disapprove leave

You want to make sure you have utterances that cover both of those intents, but also that cover potential utterances outside that scope that the application should serve like these:

- `What are my medical benefits?`
- `Who is my HR rep?`
- `tell me a joke`

- **Rare examples:** Your app will need to have rare examples as well as common examples. If your app has never seen rare examples, it won't be able to identify them in production. If you're using real data, you will be able to more accurately predict how your LUIS app will work in production.

Quality instead of quantity

Consider the quality of your existing data before you add more data. With LUIS, you're using Machine Teaching. The combination of your labels and the machine learning features you define is what your LUIS app uses. It doesn't simply rely on the quantity of labels to make the best prediction. The diversity of examples and their representation of what your LUIS app will see in production is the most important part.

Preprocessing data

The following preprocessing steps will help build a better LUIS app:

- **Remove duplicates:** Duplicate utterances won't hurt, but they don't help either, so removing them will save labeling time.
- **Apply same client-app preprocess:** If your client application, which calls the LUIS prediction endpoint, applies data processing at runtime before sending the text to LUIS, you should train the LUIS app on data that is processed in the same way.
- **Don't apply new cleanup processes that the client app doesn't use:** If your client app accepts speech-generated text directly without any cleanup such as grammar or punctuation, your utterances need to reflect the same including any missing punctuation and any other misrecognition you'll need to account for.

- **Don't clean up data:** Don't get rid of malformed input that you might get from garbled speech recognition, accidental keypresses, or mistyped/misspelled text. If your app will see inputs like these, it's important for it to be trained and tested on them. Add a *malformed input* intent if you wouldn't expect your app to understand it. Label this data to help your LUIS app predict the correct response at runtime. Your client application can choose an appropriate response to unintelligible utterances such as `Please try again.`

Labeling data

- **Label text as if it was correct:** The example utterances should have all forms of an entity labeled. This includes text that is misspelled, mistyped, and mistranslated.

Data review after LUIS app is in production

Review [endpoint utterances](#) to monitor real utterance traffic once you have deployed an app to production. This allows you to update your training utterances with real data, which will improve your app. Any app built with crowd-sourced or non-real scenario data will need to be improved based on its real use.

Test data selection for batch testing

All of the principles listed above for training utterances apply to utterances you should use for your [test set](#). Ensure the distribution across intents and entities mirror the real distribution as closely as possible.

Don't reuse utterances from your training set in your test set. This improperly biases your results and won't give you the right indication of how your LUIS app will perform in production.

Once the first version of your app is published, you should update your test set with utterances from real traffic to ensure your test set reflects your production distribution and you can monitor realistic performance over time.

Next steps

[Learn how LUIS alters your data before prediction](#)

Alter utterance data before or during prediction

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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

V3 runtime

Preprocess text for spelling corrections before you send the utterance to LUIS. Use example utterances with the correct spelling to ensure you get the correct predictions.

Use [Bing Spell Check](#) to correct text before sending it to LUIS.

Prior to V3 runtime

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](#).

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.

V2 prediction endpoint response

JSON

```
{  
  "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 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.

V3 prediction API to alter timezone

In V3, the `datetimeReference` determines the timezone offset. Learn more about [V3 predictions](#).

V2 prediction API to alter timezone

The timezone is corrected by adding the user's timezone to the endpoint using the `timezoneOffset` parameter based on the API version. The value of the parameter should be the positive or negative number, in minutes, to alter the time.

V2 prediction daylight savings example

If you need the returned prebuilt datetimeV2 to adjust for daylight savings time, you should use the 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}}
```

V2 prediction C# code determines correct value of parameter

The following C# code uses the [TimeZoneInfo](#) class's [FindSystemTimeZoneById](#) method to determine the correct offset value based on system time:

C#

```
// 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/datetimeReference  
int offset = (int)((cstDatetime - utcDatetime).TotalMinutes);
```

Next steps

[Correct spelling mistakes with this tutorial](#)

Data storage and removal in Language Understanding (LUIS) Azure AI services

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

LUIS stores data encrypted in an Azure data store corresponding to [the region](#) specified by the key.

- Data used to train the model such as entities, intents, and utterances will be saved in LUIS for the lifetime of the application. If an owner or contributor deletes the app, this data will be deleted with it. If an application hasn't been used in 90 days, it will be deleted.
- Application authors can choose to [enable logging](#) on the utterances that are sent to a published application. If enabled, utterances will be saved for 30 days, and can be viewed by the application author. If logging isn't enabled when the application is published, this data is not stored.

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 are not migrated, you can delete your account and all your apps will be deleted along with their example utterances and logs. The data is retained for 90 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.

Delete an authoring resource

If you have [migrated to an authoring resource](#), deleting the resource itself from the Azure portal will delete all your applications associated with that resource, along with their example utterances and logs. The data is retained for 90 days before it is deleted permanently.

To delete your resource, go to the [Azure portal](#) and select your LUIS authoring resource. Go to the [Overview](#) tab and select the [Delete](#) button on the top of the page.

Then confirm your resource was deleted.

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](#)

Convert data format of utterances

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

LUIS provides the following conversions of a user utterance before prediction.

- Speech to text using [Azure AI 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

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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-learning data because it isn't an exact text match. Data extraction of the machine-learning [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 extracts data from the user's utterance at the published [endpoint](#). The [HTTPS request](#) (POST or GET) contains the utterance as well as some optional configurations such as staging or production environments.

V2 prediction endpoint request

```
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
```

V3 prediction endpoint request

```
https://westus.api.cognitive.microsoft.com/luis/v3.0-preview/apps/<appID>/slots/<slot-type>/predict?subscription-key=<subscription-key>&verbose=true&timezoneOffset=0&query=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**. The endpoint response is:

V2 prediction endpoint response

JSON

```
{  
  "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.

V2 prediction endpoint response

Set the querystring parameter, `verbose=true`. The endpoint response is:

JSON

```
{  
  "query": "when do you open next?",  
  "topScoringIntent": {  
    "intent": "GetStoreInfo",  
    "score": 0.984749258  
  },  
  "entities": []  
}
```

```

"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:

V2 prediction endpoint response

JSON

```

{
  "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.StartOver"
	Intent	String	intents[2].intent	"None"

Data from entities

Most chat bots 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

Tokenized entity returned

Review the [token support](#) in LUIS.

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::`.

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.

Regular expression entity data

A [regular expression entity](#) extracts an entity based on a regular expression 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.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. The entity used in the pattern must be found in order for the pattern to be applied.

Sentiment analysis

If sentiment analysis is configured while [publishing](#), the LUIS json response includes sentiment analysis. Learn more about sentiment analysis in the [Language service](#) documentation.

Key phrase extraction entity data

The [key phrase extraction entity](#) returns key phrases in the utterance, provided by the [Language service](#).

Data matching multiple entities

LUIS returns all entities discovered in the utterance. As a result, your chat bot may need to make a decision based on the results.

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.

Next steps

See [Add entities](#) to learn more about how to add entities to your LUIS app.

Luis role-based access control

Article • 10/12/2023

ⓘ Important

Luis will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new Luis resources. We recommend [migrating your Luis applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Luis supports Azure role-based access control (Azure RBAC), an authorization system for managing individual access to Azure resources. Using Azure RBAC, you assign different team members different levels of permissions for your Luis authoring resources. See the [Azure RBAC documentation](#) for more information.

Enable Microsoft Entra authentication

To use Azure RBAC, you must enable Microsoft Entra authentication. You can [create a new resource with a custom subdomain](#) or [create a custom subdomain for your existing resource](#).

Add role assignment to Language Understanding Authoring resource

Azure RBAC can be assigned to a Language Understanding Authoring resource. To grant access to an Azure resource, you add a role assignment.

1. In the [Azure portal](#), select All services.
2. Select **Azure AI services**, and navigate to your specific Language Understanding Authoring resource.

ⓘ Note

You can also set up Azure RBAC for whole resource groups, subscriptions, or management groups. Do this by selecting the desired scope level and then navigating to the desired item. For example, selecting **Resource groups** and then navigating to a specific resource group.

3. Select **Access control (IAM)** on the left navigation pane.
4. Select **Add**, then select **Add role assignment**.
5. On the **Role** tab on the next screen, select a role you want to add.
6. On the **Members** tab, select a user, group, service principal, or managed identity.
7. On the **Review + assign** tab, select **Review + assign** to assign the role.

Within a few minutes, the target will be assigned the selected role at the selected scope. For help with these steps, see [Assign Azure roles using the Azure portal](#).

LUIS role types

Use the following table to determine access needs for your LUIS application.

These custom roles only apply to authoring (Language Understanding Authoring) and not prediction resources (Language Understanding).

ⓘ Note

- *Owner* and *Contributor* roles take priority over the custom LUIS roles.
- Microsoft Entra ID (Azure Microsoft Entra ID) is only used with custom LUIS roles.
- If you are assigned as a *Contributor* on Azure, your role will be shown as *Owner* in LUIS portal.

Cognitive Services LUIS Reader

A user that should only be validating and reviewing LUIS applications, typically a tester to ensure the application is performing well before deploying the project. They may want to review the application's assets (utterances, intents, entities) to notify the app developers of any changes that need to be made, but do not have direct access to make them.

Capabilities

API Access

- Read Utterances
- Intents

- Entities
- Test Application

All GET APIs under:

- [LUIS Programmatic v3.0-preview](#)
- [LUIS Programmatic v2.0 APIs](#)

All the APIs under:

- [LUIS Endpoint APIs v2.0](#)
- [LUIS Endpoint APIs v3.0](#)
- [LUIS Endpoint APIs v3.0-preview](#)

All the Batch Testing Web APIs

Cognitive Services LUIS Writer

A user that is responsible for building and modifying LUIS application, as a collaborator in a larger team. The collaborator can modify the LUIS application in any way, train those changes, and validate/test those changes in the portal. However, this user wouldn't have access to deploying this application to the runtime, as they may accidentally reflect their changes in a production environment. They also wouldn't be able to delete the application or alter its prediction resources and endpoint settings (assigning or unassigning prediction resources, making the endpoint public). This restricts this role from altering an application currently being used in a production environment. They may also create new applications under this resource, but with the restrictions mentioned.

Capabilities

API Access

- All functionalities under Cognitive Services LUIS Reader.

The ability to add:

- Utterances
- Intents
- Entities
- All APIs under LUIS reader

All POST, PUT and DELETE APIs under:

- [LUIS Programmatic v3.0-preview](#)
- [LUIS Programmatic v2.0 APIs](#)

Except for

- Delete application
- Move app to another LUIS authoring Azure resource
- Publish an application
- Update application settings
- Assign a LUIS azure accounts to an application
- Remove an assigned LUIS azure accounts from an application

Cognitive Services LUIS Owner

ⓘ Note

- If you are assigned as an *Owner* and *Luis Owner* you will be shown as *Luis Owner* in LUIS portal.

These users are the gatekeepers for LUIS applications in a production environment. They should have full access to any of the underlying functions and thus can view everything in the application and have direct access to edit any changes for both authoring and runtime environments.

Functionality

API Access

- All functionalities under Cognitive Services LUIS Writer
- Deploy a model
- Delete an application
- All APIs available for LUIS

Next steps

- [Managing Azure resources](#)

Authenticate requests to Azure AI services

Article • 10/12/2023

Each request to an Azure AI service must include an authentication header. This header passes along a resource key or authentication token, which is used to validate your subscription for a service or group of services. In this article, you'll learn about three ways to authenticate a request and the requirements for each.

- Authenticate with a [single-service](#) or [multi-service](#) resource key
- Authenticate with a [token](#)
- Authenticate with [Microsoft Entra ID](#)

Prerequisites

Before you make a request, you need an Azure account and an Azure AI services subscription. If you already have an account, go ahead and skip to the next section. If you don't have an account, we have a guide to get you set up in minutes: [Create a multi-service resource](#).

You can get your resource key from the [Azure portal](#) after [creating your account](#)↗.

Authentication headers

Let's quickly review the authentication headers available for use with Azure AI services.

Header	Description
Ocp-Apim-Subscription-Key	Use this header to authenticate with a resource key for a specific service or a multi-service resource key.
Ocp-Apim-Subscription-Region	This header is only required when using a multi-service resource key with the Translator service . Use this header to specify the resource region.
Authorization	Use this header if you are using an access token. The steps to perform a token exchange are detailed in the following sections. The value provided follows this format: <code>Bearer <TOKEN></code> .

Authenticate with a single-service resource key

The first option is to authenticate a request with a resource key for a specific service, like Translator. The keys are available in the Azure portal for each resource that you've created. To use a resource key to authenticate a request, it must be passed along as the `Ocp-Apim-Subscription-Key` header.

These sample requests demonstrate how to use the `Ocp-Apim-Subscription-Key` header. Keep in mind, when using this sample you'll need to include a valid resource key.

This is a sample call to the Translator service:

CURL

```
curl -X POST 'https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&from=en&to=de' \
-H 'Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY' \
-H 'Content-Type: application/json' \
--data-raw '[{"text": "How much for the cup of coffee?" }]' | json_pp
```

The following video demonstrates using an Azure AI services key.

Authenticate with a multi-service resource key

You can use a [multi-service](#) resource key to authenticate requests. The main difference is that the multi-service resource key isn't tied to a specific service, rather, a single key can be used to authenticate requests for multiple Azure AI services. See [Azure AI services pricing](#) ↗ for information about regional availability, supported features, and pricing.

The resource key is provided in each request as the `Ocp-Apim-Subscription-Key` header.



Supported regions

When using the [multi-service](#) resource key to make a request to `api.cognitive.microsoft.com`, you must include the region in the URL. For example: `westus.api.cognitive.microsoft.com`.

When using a multi-service resource key with [Azure AI Translator](#), you must specify the resource region with the `Ocp-Apim-Subscription-Region` header.

Multi-service authentication is supported in these regions:

- `australiaeast`
- `brazilsouth`
- `canadacentral`
- `centralindia`
- `eastasia`
- `eastus`
- `japaneast`
- `northeurope`
- `southcentralus`
- `southeastasia`
- `uksouth`
- `westcentralus`
- `westeurope`
- `westus`
- `westus2`
- `francecentral`
- `koreacentral`
- `northcentralus`
- `southafricanorth`
- `uaenorth`
- `switzerlandnorth`

Sample requests

This is a sample call to the Translator service:

cURL

```
curl -X POST 'https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&from=en&to=de' \
```

```
-H 'Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY' \
-H 'Ocp-Apim-Subscription-Region: YOUR_SUBSCRIPTION_REGION' \
-H 'Content-Type: application/json' \
--data-binary '[{"text": "How much for the cup of coffee?" }]' | json_pp
```

Authenticate with an access token

Some Azure AI services accept, and in some cases require, an access token. Currently, these services support access tokens:

- Text Translation API
- Speech Services: Speech to text API
- Speech Services: Text to speech API

ⓘ Note

QnA Maker also uses the Authorization header, but requires an endpoint key. For more information, see [QnA Maker: Get answer from knowledge base](#).

⚠ Warning

The services that support access tokens may change over time, please check the API reference for a service before using this authentication method.

Both single service and multi-service resource keys can be exchanged for authentication tokens. Authentication tokens are valid for 10 minutes. They're stored in JSON Web Token (JWT) format and can be queried programmatically using the [JWT libraries](#).

Access tokens are included in a request as the `Authorization` header. The token value provided must be preceded by `Bearer`, for example: `Bearer YOUR_AUTH_TOKEN`.

Sample requests

Use this URL to exchange a resource key for an access token: `https://YOUR-REGION.api.cognitive.microsoft.com/sts/v1.0/issueToken`.

cURL

```
curl -v -X POST \
"https://YOUR-REGION.api.cognitive.microsoft.com/sts/v1.0/issueToken" \
-H "Content-type: application/x-www-form-urlencoded" \
```

```
-H "Content-length: 0" \
-H "Ocp-Apim-Subscription-Key: YOUR_SUBSCRIPTION_KEY"
```

These multi-service regions support token exchange:

- `australiaeast`
- `brazilsouth`
- `canadacentral`
- `centralindia`
- `eastasia`
- `eastus`
- `japaneast`
- `northeurope`
- `southcentralus`
- `southeastasia`
- `uksouth`
- `westcentralus`
- `westeurope`
- `westus`
- `westus2`

After you get an access token, you'll need to pass it in each request as the `Authorization` header. This is a sample call to the Translator service:

cURL

```
curl -X POST 'https://api.cognitive.microsofttranslator.com/translate?api-
version=3.0&from=en&to=de' \
-H 'Authorization: Bearer YOUR_AUTH_TOKEN' \
-H 'Content-Type: application/json' \
--data-binary '[{"text": "How much for the cup of coffee?"}]' | json_pp
```

Authenticate with Microsoft Entra ID

ⓘ Important

Microsoft Entra authentication always needs to be used together with custom subdomain name of your Azure resource. **Regional endpoints** do not support Microsoft Entra authentication.

In the previous sections, we showed you how to authenticate against Azure AI services using a single-service or multi-service subscription key. While these keys provide a quick and easy path to start development, they fall short in more complex scenarios that require Azure [role-based access control \(Azure RBAC\)](#). Let's take a look at what's required to authenticate using Microsoft Entra ID.

In the following sections, you'll use either the Azure Cloud Shell environment or the Azure CLI to create a subdomain, assign roles, and obtain a bearer token to call the Azure AI services. If you get stuck, links are provided in each section with all available options for each command in Azure Cloud Shell/Azure CLI.

Important

If your organization is doing authentication through Microsoft Entra ID, you should [disable local authentication](#) (authentication with keys) so that users in the organization must always use Microsoft Entra ID.

Create a resource with a custom subdomain

The first step is to create a custom subdomain. If you want to use an existing Azure AI services resource which does not have custom subdomain name, follow the instructions in [Azure AI services custom subdomains](#) to enable custom subdomain for your resource.

1. Start by opening the Azure Cloud Shell. Then [select a subscription](#):

```
PowerShell
```

```
Set-AzContext -SubscriptionName <SubscriptionName>
```

2. Next, [create an Azure AI services resource](#) with a custom subdomain. The subdomain name needs to be globally unique and cannot include special characters, such as: ".", "!", ",".

```
PowerShell
```

```
$account = New-AzCognitiveServicesAccount -ResourceGroupName  
<RESOURCE_GROUP_NAME> -name <ACCOUNT_NAME> -Type <ACCOUNT_TYPE> -  
SkuName <SUBSCRIPTION_TYPE> -Location <REGION> -CustomSubdomainName  
<UNIQUE_SUBDOMAIN>
```

3. If successful, the **Endpoint** should show the subdomain name unique to your resource.

Assign a role to a service principal

Now that you have a custom subdomain associated with your resource, you're going to need to assign a role to a service principal.

ⓘ Note

Keep in mind that Azure role assignments may take up to five minutes to propagate.

1. First, let's register an [Microsoft Entra application](#).

PowerShell

```
$SecureStringPassword = ConvertTo-SecureString -String <YOUR_PASSWORD>  
-AsPlainText -Force  
  
$app = New-AzureADApplication -DisplayName <APP_DISPLAY_NAME> -  
IdentifierUris <APP_URIS> -PasswordCredentials $SecureStringPassword
```

You're going to need the **ApplicationId** in the next step.

2. Next, you need to [create a service principal](#) for the Microsoft Entra application.

PowerShell

```
New-AzADServicePrincipal -ApplicationId <APPLICATION_ID>
```

ⓘ Note

If you register an application in the Azure portal, this step is completed for you.

3. The last step is to [assign the "Cognitive Services User" role](#) to the service principal (scoped to the resource). By assigning a role, you're granting service principal access to this resource. You can grant the same service principal access to multiple resources in your subscription.

ⓘ Note

The **ObjectId** of the service principal is used, not the **ObjectId** for the application. The **ACCOUNT_ID** will be the Azure resource Id of the Azure AI

services account you created. You can find Azure resource Id from "properties" of the resource in Azure portal.

Azure CLI

```
New-AzRoleAssignment -ObjectId <SERVICE_PRINCIPAL_OBJECTID> -Scope <ACCOUNT_ID> -RoleDefinitionName "Cognitive Services User"
```

Sample request

In this sample, a password is used to authenticate the service principal. The token provided is then used to call the Computer Vision API.

1. Get your TenantId:

PowerShell

```
$context=Get-AzContext  
$context.Tenant.Id
```

2. Get a token:

⚠ Note

If you're using Azure Cloud Shell, the `SecureClientSecret` class isn't available.

PowerShell

```
$authContext = New-Object  
"Microsoft.IdentityModel.Clients.ActiveDirectory.AuthenticationCont  
ext" -ArgumentList "https://login.windows.net/<TENANT_ID>"  
$secureSecretObject = New-Object  
"Microsoft.IdentityModel.Clients.ActiveDirectory.SecureClientSecret  
" -ArgumentList $SecureStringPassword  
$clientCredential = New-Object  
"Microsoft.IdentityModel.Clients.ActiveDirectory.ClientCredential"  
-ArgumentList $app.ApplicationId, $secureSecretObject  
$token=$authContext.AcquireTokenAsync("https://cognitiveservices.az  
ure.com/", $clientCredential).Result  
$token
```

3. Call the Computer Vision API:

PowerShell

```
$url = $account.Endpoint+"vision/v1.0/models"
$result = Invoke-RestMethod -Uri $url -Method Get -Headers
@{"Authorization"=$token.CreateAuthorizationHeader()} -Verbose
$result | ConvertTo-Json
```

Alternatively, the service principal can be authenticated with a certificate. Besides service principal, user principal is also supported by having permissions delegated through another Microsoft Entra application. In this case, instead of passwords or certificates, users would be prompted for two-factor authentication when acquiring token.

Authorize access to managed identities

Azure AI services support Microsoft Entra authentication with [managed identities for Azure resources](#). Managed identities for Azure resources can authorize access to Azure AI services resources using Microsoft Entra credentials from applications running in Azure virtual machines (VMs), function apps, virtual machine scale sets, and other services. By using managed identities for Azure resources together with Microsoft Entra authentication, you can avoid storing credentials with your applications that run in the cloud.

Enable managed identities on a VM

Before you can use managed identities for Azure resources to authorize access to Azure AI services resources from your VM, you must enable managed identities for Azure resources on the VM. To learn how to enable managed identities for Azure Resources, see:

- [Azure portal](#)
- [Azure PowerShell](#)
- [Azure CLI](#)
- [Azure Resource Manager template](#)
- [Azure Resource Manager client libraries](#)

For more information about managed identities, see [Managed identities for Azure resources](#).

Use Azure key vault to securely access credentials

You can [use Azure Key Vault](#) to securely develop Azure AI services applications. Key Vault enables you to store your authentication credentials in the cloud, and reduces the chances that secrets may be accidentally leaked, because you won't store security information in your application.

Authentication is done via Microsoft Entra ID. Authorization may be done via Azure role-based access control (Azure RBAC) or Key Vault access policy. Azure RBAC can be used for both management of the vaults and access data stored in a vault, while key vault access policy can only be used when attempting to access data stored in a vault.

See also

- [What are Azure AI services?](#)
- [Azure AI services pricing ↗](#)
- [Custom subdomains](#)

Build a decomposable LUIS application

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

In this tutorial, you will be able to create a telecom LUIS application that can predict different user intentions. By the end of the tutorial, we should have a telecom application that can predict user intentions based on text provided by users.

We will be handling different user scenarios (intents) such as:

- Signing up for a new telecom line
- Updating an existing tier
- Paying a bill

In this tutorial you will learn how to:

1. Create a LUIS application
2. Create intents
3. Add entities
4. Add utterances
5. Label example utterances
6. Train an app
7. Publish an app
8. Get predictions from the published endpoint

Create a LUIS application

1. Sign in to the [LUIS portal](#)
2. Create a new application by selecting **+New app**.

Conversation apps

Azure subscription: MySubscription / Authoring resource: my-resource [Choose a different authoring resource.](#)

Name	Last modified	Culture
You don't have any apps yet.		

3. In the window that appears, enter the name "Telecom Tutorial", keeping the default culture, **English**. The other fields are optional, do not set them. Select **Done**.

Create new app

Name *

Culture * ⓘ

Description

Prediction resource ⓘ

Done **Cancel**

User intentions as intents

The first thing you will see in the **Build** section are the app's intents. **Intents** represent a task or an action a user want to perform.

Imagine a telecom LUIS application, what would a user need?

They would probably need to perform some type of user action or ask for help. Another user might want to update their tier or **pay a bill**

The resulting schema is as follows. For more information, see [best practices about planning the schema](#).

Intent	Purpose
UserActions	Determine user actions
Help	Request help
UpdateTier	Update current tier
PayBill	Pay outstanding bill
None	Determine if user is asking something the LUIS app is not designed to answer. This intent is provided as part of app creation and can't be deleted.

Create a new Intent

An intent is used to classify user utterances based on the user's intention, determined from the natural language text.

To classify an utterance, the intent needs examples of user utterances that should be classified with this intent.

1. Select **Build** from the top navigation menu, then select **Intents** on the left side of the screen. Select **+ Create** to create a new intent. Enter the new intent name, "UserAction", then select **Done**

UserAction could be one of many intents. For example, some users might want to sign up for a new line, while others might ask to retrieve information.

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

- Hi! I want to sign up for a new line
- Can I Sign up for a new line?
- Hello, I want a new line
- I forgot my line number!
- I would like a new line number

For the **PayBill** intent, some utterances could be:

- I want to pay my bill
- Settle my bill
- Pay bill
- I want to close my current balance
- Hey! I want to pay the current bill

By providing *example utterances*, you are teaching LUIS about what kinds of utterances should be predicted for this intent. These are positive examples. The utterances in all the other intents are treated as negative examples for this intent. Ideally, the more example utterances you add, the better your app's predictions will be.

These few utterances are for demonstration purposes only. A real-world app should have at least 15-30 [utterances](#) of varying length, word order, tense, grammatical correctness, punctuation, and word count.

Creating the remaining intents

Perform the above steps to add the following intents to the app:

"Help"

- "I need help"
- "I need assistance"
- "Help please"
- "Can someone support me?"
- "I'm stuck, can you help me"
- "Can I get help?"

"UpdateTier"

- "I want to update my tier"
- "Update my tier"
- "I want to change to VIP tier"
- "Change my subscription to standard tier"

Example utterances for 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 shouldn'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.

If you leave the "None" intent empty, an utterance that should be predicted outside the subject domain will be predicted in one of the existing subject domain intents. The result is that the client application, such as a chat bot, will perform incorrect operations based on an incorrect prediction.

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 Telecom app. These examples shouldn't use words you expect in your subject domain such as Tier, upgrade, signup, bill.
 - "When is my flight?"
 - "I need to change my pizza order please"
 - "What is the weather like for today?"

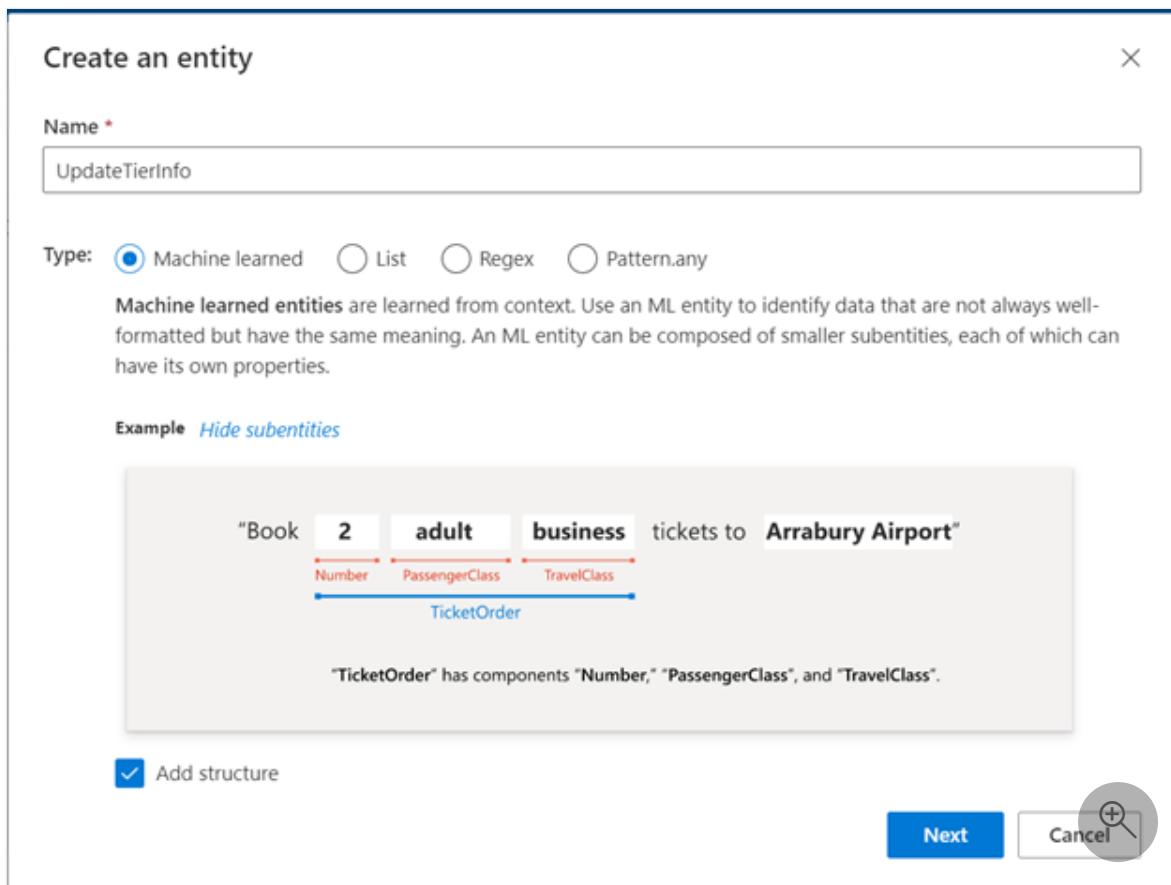
Add entities

An entity is an item or element that is relevant to the user's intent. Entities define data that can be extracted from the utterance and is essential to complete a user's required action.

1. In the build section, select **Entities**.
2. To add a new entity, select **+Create**

In this example, we will be creating two entities, "**UpdateTierInfo**" as a machine-learned entity type, and "Tier" as a list entity type. Luis also lets you create [different entity types](#).

3. In the window that appears, enter "**UpdateTierInfo**", and select Machine learned from the available types. Select the **Add structure** box to be able to add a structure to this entity.



4. Select **Next**.

5. To add a child subentity, select the "+" symbol and start adding the child. For our entity example, "**UpdateTierInfo**", we require three things:

- **OriginalTier**
- **NewTier**
- **PhoneNumber**

Add subentities (optional)

X

An ML entity can be composed of smaller subentities, each of which can have its own properties. You can add subentities when you create your ML entity, and you can add subentities to existing ML entities.

Note: Once you have created your entity you will not be able to rearrange subentities.

UpdateTierInfo

- OriginalTier
- NewTier
- PhoneNumber

Back Create Cancel

6. Select **Create** after adding all THE subentities.

We will create another entity named "Tier", but this time it will be a list entity, and it will include all the tiers that we might provide: Standard tier, Premium tier, and VIP tier.

7. To do this, go to the entities tab, and press on **+create** and select **list** from the types in the screen that appears.

8. Add the items to your list, and optionally, you can add synonyms to make sure that all cases of that mention will be understood.

tier edit

List

List items Examples Roles

Normalized values ↑ Synonyms

	Normalized values	Synonyms
Premium	Premium tier × Premium-tier ×	Type in value ...
Standard	standard tier × standard-tier ×	Type in value ...
VIP	vip × vip tier × vip-tier ×	Type in value ...

9. Now go back to the "UpdateTierInfo" entity and add the "tier" entity as a feature for the "OriginalTier" and "newTier" entities we created earlier. It should look something like this:

The screenshot shows the 'Schema and features' tab for the 'updateTierInfo' entity. The 'Name' column lists 'updateTierInfo', 'originalTier', 'newTier', and 'phoneNumber'. The 'Machine learning features' column shows three rows: 'tier * X' (with a delete button 'X'), 'tier * X' (with a delete button 'X'), and 'phoneNumber * X' (with a delete button 'X'). Each row has a '+ Add feature' button to its right. A magnifying glass icon with a plus sign is located in the bottom right corner of the interface.

We added tier as a feature for both "originalTier" and "newTier", and we added the "Phonenumber" entity, which is a Regex type. It can be created the same way we created an ML and a list entity.

Now we have successfully created intents, added example utterances, and added entities. We created four intents (other than the "none" intent), and three entities.

Label example utterances

The machine learned entity is created and the subentities have features. To complete the extraction improvement, the example utterances need to be labeled with the subentities.

There are two ways to label utterances:

1. Using the labeling tool
 - a. Open the **Entity Palette**, and select the "@" symbol in the contextual toolbar.
 - b. Select each entity row in the palette, then use the palette cursor to select the entity in each example utterance.
2. Highlight the text by dragging your cursor. Using the cursor, highlight over the text you want to label. In the following image, we highlighted "vip - tier" and select the "NewTier" entity.

The screenshot shows the LUIS Examples page. On the left, a sidebar lists categories like App Assets, Intents, Entities, Prebuilt Domains, Improve app performance, Review endpoint utterances, Features, and Patterns. The main area displays an example user input: "my standard tier is not enough for my usage , i want to upgrade to [vip - tier]". Below this, a list of entities is shown with their scores: tier (List) (.863), OriginalTier (.892), and PhoneNumber (.941). A tooltip for 'tier' indicates it's a list type. The right side shows a legend for label entities: tier (LIST), UpdateTierInfo (blue dot), OriginalTier (red dot), NewTier (red dot), and PhoneNumber (red dot). A search icon is also present.

Train the app

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

Before training, make sure there is at least one utterance for each intent.



Publish the app

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

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



2. Select the **Production** slot, then select **Done**.

Choose your publishing slot and settings

X

Staging slot

Production Slot

Last Published: 08/04/2021

Sentiment Analysis: Off

Speech Priming: Off

[Change settings](#)

[Done](#)

[Cancel](#)



3. Select **Access your endpoint URLs** in the notification to go to the **Azure Resources** page. You will only be able to see the URLs if you have a prediction resource associated with the app. You can also find the **Azure Resources** page by clicking **Manage** on the left of the screen.



Publish app ‘Telecom App’ completed

11:46 AM



[Access your endpoint Urls](#)



Get intent prediction

1. Select **Manage** in the top-right menu, then select **Azure Resources** on the left.
2. Copy the **Example Query URL** and paste it into a new web browser tab.

The endpoint URL will have the following format.

HTTP

```
https://YOUR-CUSTOM-
SUBDOMAIN.api.cognitive.microsoft.com/luis/prediction/v3.0/apps/YOUR-
APP-ID/slots/production/predict?subscription-key=YOUR-KEY-
ID&verbose=true&show-all-
intents=true&log=true&query=YOUR\_QUERY\_HERE
```

3. Go to the end of the URL in the address bar and replace the `query=` string parameter with:

"Hello! I am looking for a new number please."

The utterance `query` is passed in the URI. This utterance is not the same as any of the example utterances, and should be a good test to check if LUIS predicts the `UserAction` intent as the top scoring intent.

```
JSON

{
  "query": "hello! i am looking for a new number please",
  "prediction": {
    "topIntent": "UserAction",
    "intents": {
      "UserAction": {
        "score": 0.8607431},
      "Help": {
        "score": 0.031376917},
      "PayBill": {
        "score": 0.01989629},
      "None": {
        "score": 0.013738701},
      "UpdateTier": {
        "score": 0.012313577}
    },
    "entities": {}
  }
}
```

The JSON result identifies the top scoring intent as `prediction.topIntent` property. All scores are between 1 and 0, with the better score being closer to 1.

Client-application next steps

This tutorial created a LUIS app, created intents, entities, 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.

Luis does not provide answers to user utterances, it only identifies what type of information is being asked for in natural language. The conversation 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**. In the pop-up dialog named **Delete app?**, select **Ok**.

Add language understanding

Article • 10/19/2022

APPLIES TO: Composer v1.x and v2.x

ⓘ Note

Language Understanding (LUIS) will be retired on 1 October 2025 [↗](#). Beginning 1 April 2023, you won't be able to create new LUIS resources. A newer version of language understanding is now available as part of Azure Cognitive Service for Language.

Conversational language understanding (CLU), a feature of Azure Cognitive Service for Language, is the updated version of LUIS. For more information about question-and-answer support in Composer, see [Natural language processing](#).

This article shows how to integrate language understanding in your bot using the cloud-based service [LUIS ↗](#). LUIS lets your bots identify valuable information from user input by interpreting user needs ([intents](#)) and extracting key information ([entities](#)). Understanding user intent makes it possible for your bot to know how to respond with helpful information using [language generation](#).

Prerequisites

- [A basic bot built using Composer](#)
- Knowledge of [language understanding](#) and [events and triggers](#)

Update the recognizer type to LUIS

Composer uses [recognizers](#) to interpret user input. A dialog can use only one type of recognizer, each of which are set independently from the other dialogs in your bot.

Composer v2.x

1. Select the dialog that needs language understanding capabilities in the bot explorer.
2. In the properties pane, go to **Recognizer/Dispatch type** and select **Change**. The **Choose a recognizer type** menu will appear.

3. Select **Default** and then select **Done** to set the recognizer type to LUIS.

Add language understanding data and conditions

Composer v2.x

1. Select the desired dialog in the bot explorer. Select the three dots next to the dialog and then + **Add new trigger**.
2. On the **Create a trigger** screen:
 - a. Select **Intent recognized** for the trigger type.
 - b. Enter a name for the trigger, like **Goodbye**.
 - c. Enter example phrases in the **Trigger phrases** field using the [.lu file format](#).

Language understanding

- Bye.
- Goodbye.
- See you later.
- Take it easy.

Here's an example of what your screen should look like:

Create a trigger

What is the type of this trigger?

Intent recognized

What is the name of this trigger?

Goodbye

Trigger phrases

+ Add entity ▾ ➔ Insert entity ▾

- Bye.
- Goodbye.
- See you later.
- Take it easy.

Cancel

Submit

3. After you select **Submit**, you'll see the trigger node created in the authoring canvas.
4. If you want to specify a threshold for a specific intent, set a condition for this intent in the **Condition** field on the properties pane on the right. An example condition can be `=#goodbye.score >=0.8`.
5. You can add actions to execute when the trigger fires, for example, the **Send a response** action. Create a new action by selecting the plus (+) icon in the authoring canvas, then **Send a response** from the drop-down list.
6. Enter **This is a greeting intent!** in the **response editor** of the properties pane and press Enter.

Update LUIS keys

After you're done with all previous steps, you need to provide Composer with information from LUIS.

Composer v2.x

1. Select the error icon in the upper right corner of Composer. The number 2 is next to it, indicating that there are two errors.

These errors occur because you've yet to add the LUIS information Composer needs. After selecting the error icon, the **Problems** pane will appear on the bottom of the screen:

2. Select the `settings` link in blue in either of the errors. Composer will switch to the **Azure Language Understanding** of the **Configure your bot** page.

Azure Language Understanding

Language Understanding (LUIS) is an Azure Cognitive Service that uses machine learning to understand natural language input and direct the conversation flow. [Learn more](#). Use an existing Language Understanding (LUIS) key from Azure or create a new key. [Learn more](#)

Application name [?](#)

empty_test

Language Understanding authoring key * [?](#)

Type Language Understanding authoring key



LUIS key is required with the current recognizer setting to start your bot locally, and publish

Language Understanding region * [?](#)

Select region

Set up Language Understanding

3. Select the **Set up Language Understanding** button. The **Set up Language Understanding** window will appear.

Set up Language Understanding

X

To understand natural language input and direct the conversation flow, your bot needs a language understanding service. [Learn more](#)

- Use existing resources
- Create and configure new Azure resources
- Generate instructions for Azure administrator

Next

Cancel

The following sections explain the different options for setting up language understanding. Below is a link to each setup option and when to choose each:

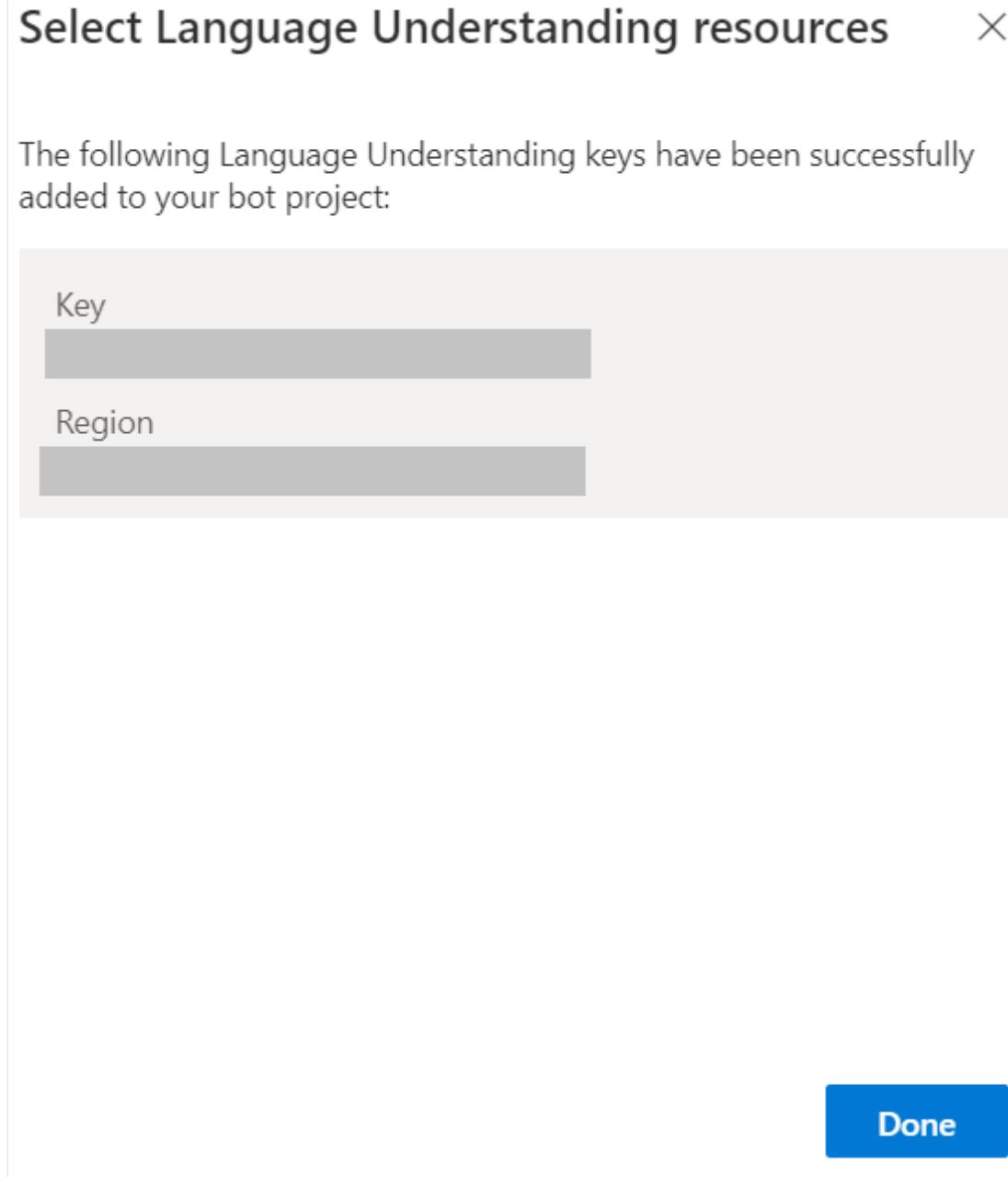
- [Use existing resources](#): You already have existing Azure resources.
- [Create and configure new Azure resources](#): You don't have existing Azure resources and need to create and configure new ones.
- [Generate instructions for Azure administrator](#): You need an administrator to create Azure resources for you.

Use existing resources

Choose this option if you already have existing language understanding Azure resources for your bot.

1. Select **Use existing resources** and then select **Next** at the bottom of the window.

2. Now select the Azure directory, Azure subscription, and Language Understanding resource name.
3. Select **Next** on the bottom. Your **Key** and **Region** will appear on the next window.



4. Select **Done**.

Your bot is now ready to test.

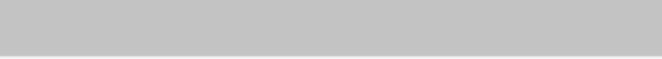
Create and configure new Azure resources

Choose this option if you need to create and configure new Azure resources for language understanding.

1. Select **Create and configure new Azure resources** and then select **Next** at the bottom of the window.
2. Select the **Azure directory** and **Azure subscription** you want to use to create your resources.
3. Fill in the values for the following on the next window:
 - a. **Azure resource group**: An existing Azure resource group, or you can create a new one (seen in the screenshot below).
 - b. **Region**: A region from the drop-down list.
 - c. **Language Understanding resource name**: The name of your language understanding resource.
4. Select the **Next** button. Blue text will appear on the bottom left of the window indicating that Azure is **Creating resources**.
5. Once your resource has been successfully created, Composer will display the **Subscription**, **Resource Group**, **Region**, and **Resource name**, as shown in the screenshot below:

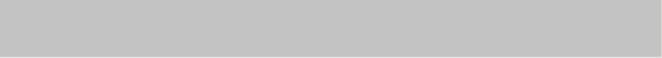
Create Language Understanding resources X

The following Language Understanding resource was successfully created and added to your bot project:

Subscription


Resource Group


Region


Resource name


Done

6. Select **Done**.

Your bot is now ready to test.

Generate instructions for Azure administrator

Choose this option if you need an administrator to create Azure resources for you.

1. Select **Generate instructions for Azure administrator** and then select **Next** at the bottom of the window.
2. Copy the instructions on the next window and send them to your Azure administrator. They'll create the resources you need to LUIS in your bot.

Generate instructions for Azure administrator

X

If Azure resources and subscription are managed by others, use the following information to request creation of the resources that you need to build and run your bot.

Instructions



I am creating a conversational experience using Microsoft Bot Framework project. For my project to work, it needs Azure resources including Language Understanding. Below are the steps to create these resources.

1. Using the Azure portal, please create a Language Understanding resource.
2. Once created, securely share the resulting credentials with me as described in the link below.

Detailed instructions:

<https://aka.ms/bfcomposerhandoffluis>

Back

Okay

3. Select **Okay**.

4. Your administrator will send you a **JSON** with your bot's configuration information. On the **Configure your bot** page, toggle on the **Advanced Settings View (json)**. Replace the existing JSON with the one supplied by your administrator.

Your bot is now ready to test.

Test

It's always a good idea to verify that your bot works correctly when you add new functionality. You can test your bot's new language understanding capabilities using the Emulator.

1. Select **Start bot** on the top right to start your bot's local runtime.
2. Once the bot is running, select **Open Web Chat** in the **Local bot runtime manager**.
3. When Web Chat pops open, send it messages using the various utterances you created to see if it executes your **Intent recognized** triggers.

Next

- Learn how to add a QnA Maker knowledge base to your bot.

Add natural language understanding to your bot

Article • 12/13/2022

APPLIES TO: SDK v4

ⓘ Note

Language Understanding (LUIS) will be retired on 1 October 2025 [↗](#). Beginning 1 April 2023, you won't be able to create new LUIS resources. A newer version of language understanding is now available as part of Azure AI Language.

Conversational language understanding (CLU), a feature of Azure AI Language, is the updated version of LUIS. For more information about language understanding support in the Bot Framework SDK, see [Natural language understanding](#).

The ability to understand what your user means conversationally and contextually can be a difficult task, but can provide your bot a more natural conversation feel. *Language Understanding (LUIS)* is a cloud-based API service that enables you to do just that so that your bot can recognize the intent of user messages, allow for more natural language from your user, and better direct the conversation flow.

This topic walks you through adding LUIS to a flight booking application to recognize different intents and entities contained within user input.

ⓘ Note

The Bot Framework JavaScript and C# SDKs will continue to be supported, however, the Python and Java SDKs are being retired with final long-term support ending in November 2023. Only critical security and bug fixes within this repository will be undertaken.

Existing bots built with these SDKs will continue to function.

For new bot building, consider using [Power Virtual Agents](#) and read about [choosing the right chatbot solution](#).

For more information, see [The future of bot building](#) [↗](#).

Prerequisites

- A [LUIS](#) account.
- A copy of the **Core Bot** sample in [C#](#), [JavaScript](#), [Java](#), or [Python](#).
- Knowledge of [bot basics](#) and [natural language processing](#).

About this sample

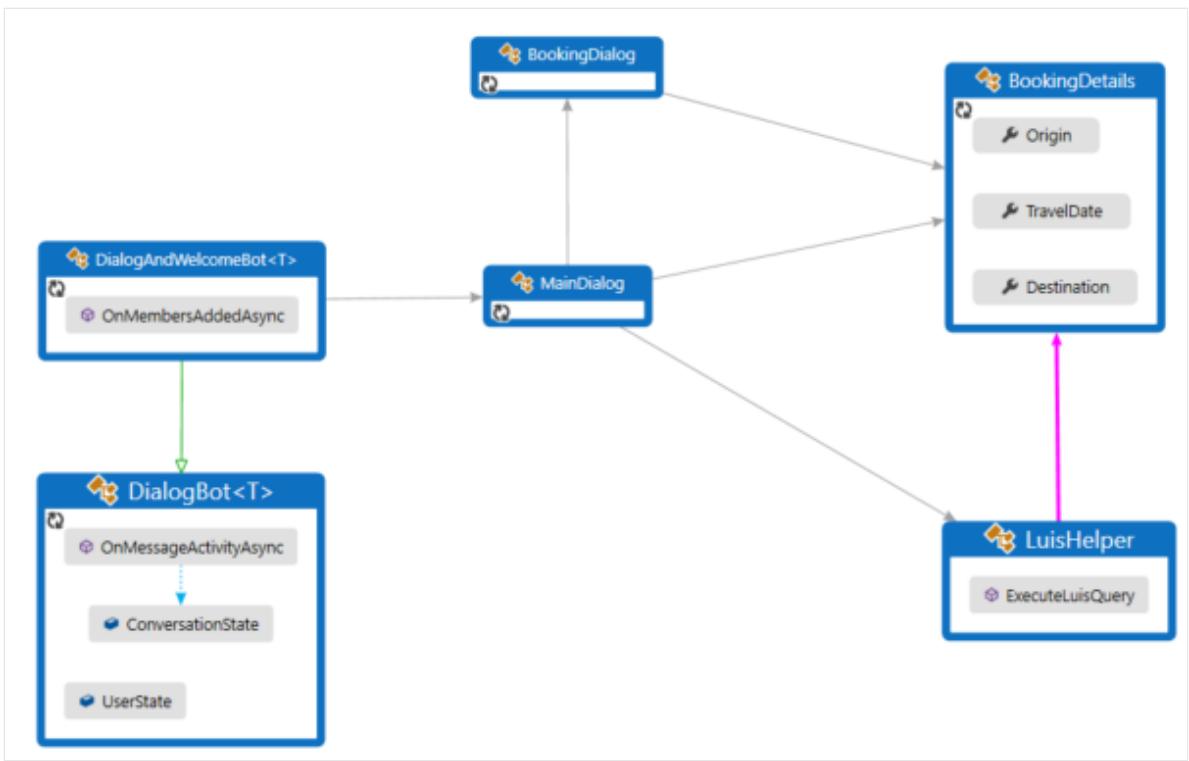
This core bot sample shows an example of an airport flight booking application. It uses a LUIS service to recognize the user input and return the top recognized LUIS intent.

The language model contains three intents: `Book Flight`, `Cancel`, and `None`. LUIS will use these intents to understand what the user meant when they send a message to the bot. The language model also defines entities that LUIS can extract from the user's input, such as the origin or destination airport.

C#

After each processing of user input, `DialogBot` saves the current state of both `UserState` and `ConversationState`. Once all the required information has been gathered, the coding sample creates a demo flight booking reservation. In this article, we'll be covering the LUIS aspects of this sample. However, the general flow of the sample is:

- `OnMembersAddedAsync` is called when a new user is connected and displays a welcome card.
- `OnMessageActivityAsync` is called for each user input received.



The `OnMessageActivityAsync` module runs the appropriate dialog through the `Run` dialog extension method. Then the main dialog calls the LUIS helper to find the top scoring user intent. If the top intent for the user input returns "BookFlight", the helper fills out information from the user that LUIS returned. After that, the main dialog starts the `BookingDialog`, which acquires additional information as needed from the user such as:

- `Origin` the originating city
- `TravelDate` the date to book the flight
- `Destination` the destination city

This article covers how to add LUIS to a bot. For information about using dialogs or state, see how to [gather user input using a dialog prompt](#) or [save user and conversation data](#), respectively.

Create a LUIS app in the LUIS portal

1. Sign in to the LUIS portal and if needed [create an account](#) and [authoring resource](#).
2. On the **Conversation apps** page in [LUIS](#), select **Import**, then **Import as JSON**.
3. In the **Import new app** dialog:
 - a. Choose the `FlightBooking.json` file in the **CognitiveModels** folder of the sample.
 - b. Enter `FlightBooking` as the optional name of the app, and select **Done**.

4. The site may display **How to create an effective LUIS app** and **Upgrade your composite entities** dialogs. You can dismiss these dialogs and continue.
5. Train your app, then publish your app to the *production* environment. For more information, see the LUIS documentation on how to [train](#) and [publish](#) an app.

Why use entities

LUIS entities enable your bot to understand events beyond standard intents. This enables you to gather from users additional information, so your bot can ask questions and respond more intelligently. Along with definitions for the three LUIS intents 'Book Flight', 'Cancel', and 'None', the FlightBooking.json file also contains a set of entities such as 'From.Airport' and 'To.Airport'. These entities allow LUIS to detect and return additional information contained within the user's original input when they request a new travel booking.

Obtain values to connect to your LUIS app

Once your LUIS app is published, you can access it from your bot. You'll need to record several values to access your LUIS app from within your bot. You can retrieve that information using the LUIS portal.

Retrieve application information from the LUIS.ai portal

The settings file (`appsettings.json`, `.env` or `config.py`) acts as the place to bring all service references together in one place. The information you retrieve will be added to this file in the next section.

1. Select your published LUIS app from [luis.ai](#).
2. With your published LUIS app open, select the **MANAGE** tab.
3. Select the **Settings** tab on the left side and record the value shown for *Application ID* as <YOUR_APP_ID>.

My LUIS / FlightBooking v0.1

DASHBOARD BUILD MANAGE Train Test Publish

Application Settings

App name FlightBooking

App description (optional) Luis Model for CoreBot

App ID (redacted)

Culture en-us

Make endpoints public Off

4. Select **Azure Resources**, then **Prediction Resource**. Record the value shown for *Location* as <YOUR_REGION> and *Primary Key* as <YOUR_AUTHORING_KEY>.

My LUIS / v0.1

DASHBOARD BUILD MANAGE Train Test Publish

Azure Resources

LUIS uses two types of resources: authoring and prediction. The authoring key is needed for authoring, publishing, managing collaborators, and versioning. An authoring resource is created for you when you create your azure cognitive service account. When you are ready to publish your LUIS app, you need to create the prediction resource key and use the prediction key with endpoint queries. [Learn more](#)

Prediction Resources Authoring Resource

Add prediction resource

FormSkillDemo-luis

Un-assign key

Resource group: (redacted)

Location: westus

Primary Key: (redacted)

Secondary Key: (redacted)

Alternatively, you can use the region and primary key for your authoring resource.

Update the settings file

C#

Add the information required to access your LUIS app including application ID, authoring key, and region into the `appsettings.json` file. In the previous step, you retrieved these values from your published LUIS app. The API host name should be in the format `<your region>.api.cognitive.microsoft.com`.

`appsetting.json`

JSON

```
{  
    "MicrosoftAppType": "",  
    "MicrosoftAppId": "",  
    "MicrosoftAppPassword": "",  
    "MicrosoftAppTenantId": "",  
    "LuisAppId": "",  
    "LuisAPIKey": "",  
    "LuisAPIHostName": ""  
}
```

Configure your bot to use your LUIS app

C#

Be sure that the **Microsoft.Bot.Builder.AI.Luis** NuGet package is installed for your project.

To connect to the LUIS service, the bot pulls the information you added to the `appsetting.json` file. The `FlightBookingRecognizer` class contains code with your settings from the `appsetting.json` file and queries the LUIS service by calling `RecognizeAsync` method.

FlightBookingRecognizer.cs

C#

```
public class FlightBookingRecognizer : IRecognizer  
{  
    private readonly LuisRecognizer _recognizer;  
  
    public FlightBookingRecognizer(IConfiguration configuration)  
    {  
        var luisIsConfigured =  
!string.IsNullOrEmpty(configuration["LuisAppId"]) &&  
!string.IsNullOrEmpty(configuration["LuisAPIKey"]) &&  
!string.IsNullOrEmpty(configuration["LuisAPIHostName"]);  
        if (luisIsConfigured)  
        {  
            var luisApplication = new LuisApplication(  
                configuration["LuisAppId"],  
                configuration["LuisAPIKey"],  
                "https://" + configuration["LuisAPIHostName"]);  
            // Set the recognizer options depending on which endpoint  
            // version you want to use.  
            // More details can be found in
```

```

https://docs.microsoft.com/en-gb/azure/cognitive-services/luis/luis-migration-api-v3
    var recognizerOptions = new LuisRecognizerOptionsV3(luisApplication)
    {
        PredictionOptions = new Bot.Builder.AI.LuisV3.LuisPredictionOptions
        {
            IncludeInstanceData = true,
        }
    };
}

_recognizer = new LuisRecognizer(recognizerOptions);
}

// Returns true if luis is configured in the appsettings.json and initialized.
public virtual bool IsConfigured => _recognizer != null;

public virtual async Task<RecognizerResult>
RecognizeAsync(ITurnContext turnContext, CancellationToken cancellationToken)
=> await _recognizer.RecognizeAsync(turnContext, cancellationToken);

public virtual async Task<T> RecognizeAsync<T>(ITurnContext turnContext, CancellationToken cancellationToken)
    where T : IRecognizerConvert, new()
    => await _recognizer.RecognizeAsync<T>(turnContext, cancellationToken);
}

```

The `FlightBookingEx.cs` contains the logic to extract *From*, *To* and *TravelDate*; it extends the partial class `FlightBooking.cs` used to store LUIS results when calling `FlightBookingRecognizer.RecognizeAsync<FlightBooking>` from the `MainDialog.cs`.

CognitiveModels\FlightBookingEx.cs

```

C#
// Extends the partial FlightBooking class with methods and properties
// that simplify accessing entities in the luis results
public partial class FlightBooking
{
    public (string From, string Airport) FromEntities
    {
        get
        {
            var fromValue =
Entities?._instance?.From?.FirstOrDefault()?.Text;
            var fromAirportValue =

```

```

        Entities?.From?.FirstOrDefault()?.Airport?.FirstOrDefault()?.FirstOrDefa
        ult();
                return (fromValue, fromAirportValue);
            }
        }

    public (string To, string Airport) ToEntities
    {
        get
        {
            var toValue =
        Entities?._instance?.To?.FirstOrDefault()?.Text;
            var toAirportValue =
        Entities?.To?.FirstOrDefault()?.Airport?.FirstOrDefault()?.FirstOrDefau
        lt();
            return (toValue, toAirportValue);
        }
    }

    // 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.
    public string TravelDate
        =>
    Entities.datetime?.FirstOrDefault()?.Expressions.FirstOrDefault()?.Split
    ('T')[0];
}

```

LUIS is now configured and connected for your bot.

Test the bot

Download and install the latest [Bot Framework Emulator](#)

1. Run the sample locally on your machine. If you need instructions, refer to the `README` file for the [C# Sample](#), [JS Sample](#) or [Python Sample](#).
2. In the Emulator, type a message such as "travel to paris" or "going from paris to berlin". Use any utterance found in the file `FlightBooking.json` for training the intent "Book flight".

If the top intent returned from LUIS resolves to "Book flight", your bot will ask more questions until it has enough information stored to create a travel booking. At that point it will return this booking information back to your user.

At this point, the code bot logic will reset and you can continue to create more bookings.

Additional information

For more about LUIS, see the LUIS documentation:

- [What is Language Understanding \(LUIS\)?](#)
- [Create a new LUIS app in the LUIS portal](#)
- [Design with intent and entity models](#)
- [Migrate to V3 Authoring APIs](#)
- [Migrate to V3 Prediction APIs](#)

 **Tip**

Different parts of the SDK define separate *entity* classes or elements. For message entities, see [Entities and activity types](#).

Build a LUIS app programmatically using Node.js

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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.

✖ Caution

This document has not been updated with text and screenshots for the latest LUIS portal.

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 article starts with a CSV for a hypothetical company's log files of user requests. Download it [here](#).
- Install the latest Node.js with NPM. Download it from [here](#).
- [Recommended] Visual Studio Code for IntelliSense and debugging, download it from [here](#) for free.

All of the code in this article 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.

1	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`.

ⓘ Important

Remember to remove the key from your code when you're done, and never post it publicly. For production, use a secure way of storing and accessing your credentials like [Azure Key Vault](#). See the Azure AI services [security](#) article for more information.

JavaScript

```
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}/versi
ons/{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:

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`.

JavaScript

```

// 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`.

JavaScript

```

// 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`.

JavaScript

```
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`.

JavaScript

```

// 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`.

JavaScript

```

// 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}/versi
ons/{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.

Console

```
> 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.

JavaScript

```
// 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.

Console

```
> node index.js
```

or

Console

```
> 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.

Console

```
> node index.js
intents: ["TurnOn","TurnOff","Dim","Other"]
entities: ["Operation","Device","Room"]
```

```

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

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

```

Open the LUIS app

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 Microsoft LUIS portal's 'Intents' section for the 'HomeAutomation.TurnOn' intent. The left sidebar has sections for 'App Assets' (Intents, Entities) and 'Improve app performance' (Review endpoint utterances, Phrase lists, Patterns). The main area displays the intent name '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' (dropdown), 'Show All' (radio button), 'Entities View' (checkbox), and a search icon. A table lists utterances with their labeled intent and three-dot actions:

Utterance	Labeled intent	Actions
change HomeAutomation.Operation to seventy two degrees	HomeAuto... ▾	...
HomeAutomation.Device HomeAutomation.Operation please	HomeAuto... ▾	...
play HomeAutomation.Device	HomeAuto... ▾	...
turn HomeAutomation.Device on 70 .	HomeAuto... ▾	...
HomeAutomation.Device please	HomeAuto... ▾	...
make some HomeAutomation.Device	HomeAuto... ▾	...

A 'PREVIEW' button is visible at the bottom left.

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 ↗](#)

SDK, REST, and CLI developer resources for Language Understanding (LUIS)

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

SDKs, REST APIs, CLI, help you develop Language Understanding (LUIS) apps in your programming language. Manage your Azure resources and LUIS predictions.

Azure resource management

Use the Azure AI services Management layer to create, edit, list, and delete the Language Understanding or Azure AI services resource.

Find reference documentation based on the tool:

- [Azure CLI](#)
- [Azure RM PowerShell](#)

Language Understanding authoring and prediction requests

The Language Understanding service is accessed from an Azure resource you need to create. There are two resources:

- Use the **authoring** resource for training to create, edit, train, and publish.
- Use the **prediction** for runtime to send user's text and receive a prediction.

Learn about the [V3 prediction endpoint](#).

Use [Azure AI services sample code](#) to learn and use the most common tasks.

REST specifications

The [LUIS REST specifications](#), along with all [Azure REST specifications](#), are publicly available on GitHub.

REST APIs

Both authoring and prediction endpoint APIs are available from REST APIs:

Type	Version
Authoring	V2 ↗ preview V3 ↗
Prediction	V2 ↗ V3 ↗

REST Endpoints

LUIS currently has 2 types of endpoints:

- **authoring** on the training endpoint
- query **prediction** on the runtime endpoint.

Purpose	URL
V2 Authoring on training endpoint	https://{{your-resource-name}}.api.cognitive.microsoft.com/luis/api/v2.0/apps/{{appID}}/
V3 Authoring on training endpoint	https://{{your-resource-name}}.api.cognitive.microsoft.com/luis/authoring/v3.0-preview/apps/{{appID}}/
V2 Prediction - all predictions on runtime endpoint	https://{{your-resource-name}}.api.cognitive.microsoft.com/luis/v2.0/apps/{{appId}}?q={{q}} [&timezoneOffset][&verbose][&spellCheck][&staging][&bing-spell-check-subscription-key][&log]
V3 Prediction - versions prediction on runtime endpoint	https://{{your-resource-name}}.api.cognitive.microsoft.com/luis/prediction/v3.0/apps/{{appId}}/versions/{{versionId}}/predict?query={{query}}[&verbose][&log][&show-all-intents]
V3 Prediction - slot prediction on runtime endpoint	https://{{your-resource-name}}.api.cognitive.microsoft.com/luis/prediction/v3.0/apps/{{appId}}/slots/{{slotName}}/predict?query={{query}}[&verbose][&log][&show-all-intents]

The following table explains the parameters, denoted with curly braces {}, in the previous table.

Parameter	Purpose
<code>your-resource-name</code>	Azure resource name
<code>q</code> or <code>query</code>	utterance text sent from client application such as chat bot
<code>version</code>	10 character version name
<code>slot</code>	<code>production</code> or <code>staging</code>

REST query string parameters

V3 API query string parameters include:

Query parameter	LUIS portal name	Type	Version	Default	Purpose
<code>log</code>	Save logs	boolean	V2 & V3	false	Store query in log file. Default value is false.
<code>query</code>	-	string	V3 only	No default - it is required in the GET request	In V2, the utterance to be predicted is in the <code>q</code> parameter. In V3, the functionality is passed in the <code>query</code> parameter.
<code>show-all-intents</code>	Include scores for all intents	boolean	V3 only	false	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: <code>prediction.intents.give</code> . In V2, these were returned in an array.
<code>verbose</code>	Include more entities details	boolean	V2 & V3	false	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> . In V3, this parameter only provides entity metadata details of entity prediction.
<code>timezoneOffset</code>	-	string	V2	-	Timezone applied to datetimeV2 entities.
<code>datetimeReference</code>	-	string	V3	-	<code>Timezone</code> applied to datetimeV2 entities. Replaces <code>timezoneOffset</code> from V2.

App schema

The [app schema](#) is imported and exported in a `.json` or `.lu` format.

Language-based SDKs

Language	Reference documentation	Package	Quickstarts
C#	Authoring Prediction	NuGet authoring ↗ NuGet prediction ↗	Authoring Query prediction
Go	Authoring and prediction ↗	SDK ↗	
Java	Authoring and prediction	Maven authoring ↗ Maven prediction ↗	
JavaScript	Authoring Prediction	NPM authoring ↗ NPM prediction ↗	Authoring Prediction
Python	Authoring and prediction	Pip ↗	Authoring Prediction

Containers

Language Understanding (LUIS) provides a [container](#) to provide on-premises and contained versions of your app.

Export and import formats

Language Understanding provides the ability to manage your app and its models in a JSON format, the `.lu` ([LUDown](#) ↗) format, and a compressed package for the Language Understanding container.

Importing and exporting these formats is available from the APIs and from the LUIS portal. The portal provides import and export as part of the Apps list and Versions list.

Workshops

- GitHub: (Workshop) [Conversational-AI : NLU using LUIS](#) ↗

Continuous integration tools

- GitHub: (Preview) [Developing a LUIS app using DevOps practices](#) ↗
- GitHub: [NLU.DevOps](#) ↗ - Tools supporting continuous integration and deployment for NLU services.

Bot Framework tools

The bot framework is available as [an SDK](#) in a variety of languages and as a service using [Azure AI Bot Service](#).

Bot framework provides [several tools](#) to help with Language Understanding, including:

- [Bot Framework emulator](#) - a desktop application that allows bot developers to test and debug bots built using the Bot Framework SDK
- [Bot Framework Composer](#) - an integrated development tool for developers and multi-disciplinary teams to build bots and conversational experiences with the Microsoft Bot Framework
- [Bot Framework Samples](#) - in #C, JavaScript, TypeScript, and Python

Next steps

- Learn about the common [HTTP error codes](#)
- [Reference documentation](#) for all APIs and SDKs
- [Bot framework](#) and [Azure AI Bot Service](#)
- [LUDown](#)
- [Cognitive Containers](#)

App schema definition

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

The LUIS app is represented in either the `.json` or `.lu` and includes all intents, entities, example utterances, features, and settings.

Format

When you import and export the app, choose either `.json` or `.lu`.

Format	Information
<code>.json</code>	Standard programming format
<code>.lu</code>	Supported by the Bot Framework's Bot Builder tools .

Version 7.x

- Moving to version 7.x, the entities are represented as nested machine-learning entities.
- Support for authoring nested machine-learning entities with `enableNestedChildren` property on the following authoring APIs:
 - [Add label](#)
 - [Add batch label](#)
 - [Review labels](#)
 - [Suggest endpoint queries for entities](#)
 - [Suggest endpoint queries for intents](#)

JSON

```
{  
  "luis_schema_version": "7.0.0",  
  "intents": [  
    {  
      "name": "Greet",  
      "utterances": [  
        "Hello",  
        "Hi",  
        "Hey"  
      ],  
      "entities": [  
        {"entity": "GREET",  
         "text": "Hello",  
         "start": 0,  
         "end": 4}  
      ]  
    }  
  ]  
}
```

```

{
  "name": "None",
  "features": []
}
],
"entities": [],
"hierarchicals": [],
"composites": [],
"closedLists": [],
"prebuiltEntities": [],
"utterances": [],
"versionId": "0.1",
"name": "example-app",
"desc": "",
"culture": "en-us",
"tokenizerVersion": "1.0.0",
"patternAnyEntities": [],
"regex_entities": [],
"phraselists": [
],
"regex_features": [],
"patterns": [],
"settings": []
}

```

element	Comment
"hierarchicals": [],	Deprecated, use machine-learning entities .
"composites": [],	Deprecated, use machine-learning entities . Composite entity reference.
"closedLists": [],	List entities reference, primarily used as features to entities.
"versionId": "0.1",	Version of a LUIS app.
"name": "example-app",	Name of the LUIS app.
"desc": "",	Optional description of the LUIS app.
"culture": "en-us",	Language of the app, impacts underlying features such as prebuilt entities, machine-learning, and tokenizer.
"tokenizerVersion": "1.0.0",	Tokenizer
"patternAnyEntities": [],	Pattern.any entity
"regex_entities": [],	Regular expression entity
"phraselists": [],	Phrase lists (feature)
"regex_features": [],	Deprecated, use machine-learning entities .

element	Comment
"patterns": [],	Patterns improve prediction accuracy with pattern syntax
"settings": []	App settings

Version 6.x

- Moving to version 6.x, use the new [machine-learning entity](#) to represent your entities.

JSON

```
{
  "luis_schema_version": "6.0.0",
  "intents": [
    {
      "name": "None",
      "features": []
    }
  ],
  "entities": [],
  "hierarchicals": [],
  "composites": [],
  "closedLists": [],
  "prebuiltEntities": [],
  "utterances": [],
  "versionId": "0.1",
  "name": "example-app",
  "desc": "",
  "culture": "en-us",
  "tokenizerVersion": "1.0.0",
  "patternAnyEntities": [],
  "regex_entities": [],
  "phraselists": [],
  "regex_features": [],
  "patterns": [],
  "settings": []
}
```

Version 4.x

JSON

```
{
  "luis_schema_version": "4.0.0",
  "versionId": "0.1",
  "name": "example-app",
```

```
"desc": "",  
"culture": "en-us",  
"tokenizerVersion": "1.0.0",  
"intents": [  
    {  
        "name": "None"  
    }  
,  
    "entities": [],  
    "composites": [],  
    "closedLists": [],  
    "patternAnyEntities": [],  
    "regex_entities": [],  
    "prebuiltEntities": [],  
    "model_features": [],  
    "regex_features": [],  
    "patterns": [],  
    "utterances": [],  
    "settings": []  
}
```

Next steps

- Migrate to the [V3 authoring APIs](#)

az cognitiveservices

Reference

Manage Azure Cognitive Services accounts.

This article lists the Azure CLI commands for Azure Cognitive Services account and subscription management only. Refer to the documentation at <https://docs.microsoft.com/azure/cognitive-services/> for individual services to learn how to use the APIs and supported SDKs.

Commands

Name	Description	Type	Status
az cognitiveservices account	Manage Azure Cognitive Services accounts.	Core	GA
az cognitiveservices account commitment-plan	Manage commitment plans for Azure Cognitive Services accounts.	Core	GA
az cognitiveservices account commitment-plan create	Create a commitment plan for Azure Cognitive Services account.	Core	GA
az cognitiveservices account commitment-plan delete	Delete a commitment plan from Azure Cognitive Services account.	Core	GA
az cognitiveservices account commitment-plan list	Show all commitment plans from Azure Cognitive Services account.	Core	GA
az cognitiveservices account commitment-plan show	Show a commitment plan from Azure Cognitive Services account.	Core	GA
az cognitiveservices account create	Manage Azure Cognitive Services accounts.	Core	GA
az cognitiveservices account delete	Manage Azure Cognitive Services accounts.	Core	GA
az cognitiveservices account deployment	Manage deployments for Azure Cognitive Services accounts.	Core	GA
az cognitiveservices account deployment create	Create a deployment for Azure Cognitive Services account.	Core	GA
az cognitiveservices account deployment delete	Delete a deployment from Azure Cognitive Services account.	Core	GA

Name	Description	Type	Status
az cognitiveservices account deployment list	Show all deployments for Azure Cognitive Services account.	Core	GA
az cognitiveservices account deployment show	Show a deployment for Azure Cognitive Services account.	Core	GA
az cognitiveservices account identity	Manage identity of Cognitive Services accounts.	Core	GA
az cognitiveservices account identity assign	Assign an identity of a Cognitive Services account.	Core	GA
az cognitiveservices account identity remove	Remove the identity from a Cognitive Services account.	Core	GA
az cognitiveservices account identity show	Show the identity of a Cognitive Services account.	Core	GA
az cognitiveservices account keys	Manage Azure Cognitive Services accounts.	Core	GA
az cognitiveservices account keys list	Manage Azure Cognitive Services accounts.	Core	GA
az cognitiveservices account keys regenerate	Manage Azure Cognitive Services accounts.	Core	GA
az cognitiveservices account list	Manage Azure Cognitive Services accounts.	Core	GA
az cognitiveservices account list-deleted	List soft-deleted Azure Cognitive Services accounts.	Core	GA
az cognitiveservices account list-kinds	List all valid kinds for Azure Cognitive Services account.	Core	GA
az cognitiveservices account list-models	Manage Azure Cognitive Services accounts.	Core	GA
az cognitiveservices account list-skus	Manage Azure Cognitive Services accounts.	Core	GA
az cognitiveservices account list-usage	List usages for Azure Cognitive Services account.	Core	GA
az cognitiveservices account network-rule	Manage network rules.	Core	GA
az cognitiveservices account network-rule add	Add a network rule.	Core	GA

Name	Description	Type	Status
az cognitiveservices account network-rule list	List network rules.	Core	GA
az cognitiveservices account network-rule remove	Remove a network rule.	Core	GA
az cognitiveservices account purge	Purge a soft-deleted Azure Cognitive Services account.	Core	GA
az cognitiveservices account recover	Recover a soft-deleted Azure Cognitive Services account.	Core	GA
az cognitiveservices account show	Manage Azure Cognitive Services accounts.	Core	GA
az cognitiveservices account show-deleted	Show a soft-deleted Azure Cognitive Services account.	Core	GA
az cognitiveservices account update	Manage Azure Cognitive Services accounts.	Core	GA
az cognitiveservices commitment-tier	Manage commitment tiers for Azure Cognitive Services.	Core	GA
az cognitiveservices commitment-tier list	Show all commitment tiers for Azure Cognitive Services.	Core	GA
az cognitiveservices list	Manage Azure Cognitive Services accounts.	Core	Deprecated
az cognitiveservices model	Manage model for Azure Cognitive Services.	Core	GA
az cognitiveservices model list	Show all models for Azure Cognitive Services.	Core	GA
az cognitiveservices usage	Manage usage for Azure Cognitive Services.	Core	GA
az cognitiveservices usage list	Show all usages for Azure Cognitive Services.	Core	GA

az cognitiveservices list

 Edit

Deprecated

This command has been deprecated and will be removed in a future release. Use 'az cognitiveservices account list' instead.

Manage Azure Cognitive Services accounts.

This article lists the Azure CLI commands for Azure Cognitive Services account and subscription management only. Refer to the documentation at <https://docs.microsoft.com/azure/cognitive-services/> for individual services to learn how to use the APIs and supported SDKs.

Azure CLI

```
az cognitiveservices list [--resource-group]
```

Examples

List all the Cognitive Services accounts in a resource group.

Azure CLI

```
az cognitiveservices list -g MyResourceGroup
```

Optional Parameters

--resource-group -g

Name of resource group. You can configure the default group using `az configure --defaults group=<name>`.

▼ Global Parameters

--debug

Increase logging verbosity to show all debug logs.

--help -h

Show this help message and exit.

--only-show-errors

Only show errors, suppressing warnings.

--output -o

Output format.

accepted values: json, jsonc, none, table, tsv, yaml, yamlc

default value: json

--query

JMESPath query string. See <http://jmespath.org/> for more information and examples.

--subscription

Name or ID of subscription. You can configure the default subscription using `az`

`account set -s NAME_OR_ID`.

--verbose

Increase logging verbosity. Use --debug for full debug logs.

AzureRM.CognitiveServices

Reference

This topic displays help topics for the Azure Cognitive Services cmdlets.

Cognitive Services

[] Expand table

[Get-AzureRmCognitiveServicesAccount](#)

Gets an account.

⊗ Caution

Because Az PowerShell modules now have all the capabilities of AzureRM PowerShell modules and more, we'll retire AzureRM PowerShell modules on 29 February 2024.

To avoid service interruptions, [update your scripts](#) that use AzureRM PowerShell modules to use Az PowerShell modules by 29 February 2024. To automatically update your scripts, follow the [quickstart guide](#).

[Get-AzureRmCognitiveServicesAccountKey](#)

Gets the API keys for an account.

⊗ Caution

Because Az PowerShell modules now have all the capabilities of AzureRM PowerShell modules and more, we'll retire AzureRM PowerShell modules on 29 February 2024.

To avoid service interruptions, [update your scripts](#) that use AzureRM PowerShell modules to use Az PowerShell modules by 29 February 2024. To automatically update your scripts, follow the [quickstart guide](#).

Get-
AzureRmCognitiveServicesAccountSkus

Gets the available SKUs for an account.

 **Caution**

Because Az PowerShell modules now have all the capabilities of AzureRM PowerShell modules and more, we'll retire AzureRM PowerShell modules on 29 February 2024.

To avoid service interruptions, [update your scripts](#) that use AzureRM PowerShell modules to use Az PowerShell modules by 29 February 2024. To automatically update your scripts, follow the [quickstart guide](#).

Get-
AzureRmCognitiveServicesAccountType

Gets the available Cognitive Services Account Types.

 **Caution**

Because Az PowerShell modules now have all the capabilities of AzureRM PowerShell modules and more, we'll retire AzureRM PowerShell modules on 29 February 2024.

To avoid service interruptions, [update your scripts](#) that use AzureRM PowerShell modules to use Az PowerShell modules by 29 February 2024. To automatically update your scripts, follow the [quickstart guide](#).

Get-
AzureRmCognitiveServicesAccountUsage

Get current usages for a Cognitive Services account.

 **Caution**

Because Az PowerShell modules now have all the capabilities of AzureRM PowerShell modules and more, we'll retire AzureRM PowerShell modules on 29 February 2024.

To avoid service interruptions, [update your scripts](#) that use AzureRM

PowerShell modules to use Az PowerShell modules by 29 February 2024. To automatically update your scripts, follow the [quickstart guide](#).

New-AzureRmCognitiveServicesAccount

Creates a Cognitive Services account.

⊗ Caution

Because Az PowerShell modules now have all the capabilities of AzureRM PowerShell modules and more, we'll retire AzureRM PowerShell modules on 29 February 2024.

To avoid service interruptions, [update your scripts](#) that use AzureRM PowerShell modules to use Az PowerShell modules by 29 February 2024. To automatically update your scripts, follow the [quickstart guide](#).

New-AzureRmCognitiveServicesAccountKey

Regenerates an account key.

⊗ Caution

Because Az PowerShell modules now have all the capabilities of AzureRM PowerShell modules and more, we'll retire AzureRM PowerShell modules on 29 February 2024.

To avoid service interruptions, [update your scripts](#) that use AzureRM PowerShell modules to use Az PowerShell modules by 29 February 2024. To automatically update your scripts, follow the [quickstart guide](#).

Remove-AzureRmCognitiveServicesAccount

Deletes a Cognitive Services account.

⊗ Caution

Because Az PowerShell modules now have all the capabilities of AzureRM PowerShell

modules and more, we'll retire AzureRM PowerShell modules on 29 February 2024.

To avoid service interruptions, [update your scripts](#) that use AzureRM PowerShell modules to use Az PowerShell modules by 29 February 2024. To automatically update your scripts, follow the [quickstart guide](#).

[Set-AzureRmCognitiveServicesAccount](#)

Modifies an account.

Caution

Because Az PowerShell modules now have all the capabilities of AzureRM PowerShell modules and more, we'll retire AzureRM PowerShell modules on 29 February 2024.

To avoid service interruptions, [update your scripts](#) that use AzureRM PowerShell modules to use Az PowerShell modules by 29 February 2024. To automatically update your scripts, follow the [quickstart guide](#).

Common API response codes and their meaning

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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 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

Code	API	Explanation
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

Azure Language Understanding SDK for .NET - latest

Article • 02/14/2023

Packages - latest

Reference	Package	Source
Conversation Analysis	Azure.AI.Language.Conversations	GitHub
LUIS Authoring	Microsoft.Azure.CognitiveServices.Language.LUIS.Authoring	GitHub
LUIS Runtime	Microsoft.Azure.CognitiveServices.Language.LUIS.Runtime	GitHub

Language Understanding

Reference

Packages

 Expand table

com.microsoft.azure.cognitiveservices.language.luis.authoring
com.microsoft.azure.cognitiveservices.language.luis.authoring.models
com.microsoft.azure.cognitiveservices.language.luis.runtime
com.microsoft.azure.cognitiveservices.language.luis.runtime.models

azure-cognitiveservices-language-luis Package

Reference

Packages

 Expand table

luis

Modules

 Expand table

version
version
version

Export and delete your customer data in Language Understanding (LUIS) in Azure AI services

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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 about how to delete personal data from the device or service and can be used to support your obligations under the GDPR. For general information about GDPR, see the [GDPR section of the Microsoft Trust Center](#) and 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](#).

Note

As of January 20th 2021, The luis.au and luis.eu portals have been consolidated into a single LUIS portal. If you were using one of these portals, instead access LUIS at [luis.ai](#).

Europe

Europe Authoring (also known as Programmatic APIs) resources are hosted in Azure's Europe geography, and 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.

Australia

Australia Authoring (also known as Programmatic APIs) resources are hosted in Azure's Australia geography, and 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.

United States

United States Authoring (also known as Programmatic APIs) resources are hosted in Azure's United States geography, and 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.

Switzerland North

Switzerland North Authoring (also known as Programmatic APIs) resources are hosted in Azure's Switzerland geography, and support deployment of endpoints to the following Azure geographies:

- Switzerland

When deploying to these Azure geographies, the utterances received by the endpoint from end users of your app will be stored in Azure's Switzerland geography for active learning.

Disable active learning

To disable active learning, see [Disable active learning](#).

Next steps

[LUIS regions reference](#)

Authoring and publishing regions and the associated keys

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

LUIS authoring regions are supported by the LUIS portal. To publish a LUIS app to more than one region, you need at least one prediction key per region.

LUIS Authoring regions

Authoring regions are the regions where the application gets created and the training take place.

LUIS has the following authoring regions available with [paired fail-over regions](#):

- Australia east
- West Europe
- West US
- Switzerland north

LUIS has one portal you can use regardless of region, www.luis.ai.

Publishing regions and Azure resources

Publishing regions are the regions where the application will be used in runtime. To use the application in a publishing region, you must create a resource in this region and assign your application to it. For example, if you create an app with the *westus* authoring region and publish it to the *eastus* and *brazilsouth* regions, the app will run in those two regions.

Public apps

A public app is published in all regions so that a user with a supported prediction resource can access the app in all regions.

Publishing regions are tied to authoring regions

When you first create our LUIS application, you are required to choose an [authoring region](#). To use the application in runtime, you are required to create a resource in a publishing region.

Every authoring region has corresponding prediction regions that you can publish your application to, which are listed in the tables below. If your app is currently in the wrong authoring region, export the

app, and import it into the correct authoring region to match the required publishing region.

Single data residency

Single data residency means that the data does not leave the boundaries of the region.

! Note

- Make sure to set `log=false` for [V3 APIs](#) to disable active learning. By default this value is `false`, to ensure that data does not leave the boundaries of the runtime region.
- If `log=true`, data is returned to the authoring region for active learning.

Publishing to Europe

Global region	Authoring API region	Publishing & querying region	Endpoint URL format
API region name			
Europe	westeurope	France Central francecentral	<code>https://francecentral.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
Europe	westeurope	North Europe northeurope	<code>https://northeurope.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
Europe	westeurope	West Europe westeurope	<code>https://westeurope.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
Europe	westeurope	UK South uksouth	<code>https://uksouth.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
Europe	westeurope	Switzerland North switzerlandnorth	<code>https://switzerlandnorth.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
Europe	westeurope	Norway East norwayeast	<code>https://norwayeast.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>

Publishing to Australia

Global region	Authoring API region	Publishing & querying region	Endpoint URL format
API region name			
Australia	australiaeast	Australia East australiaeast	<code>https://australiaeast.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>

Other publishing regions

Global region	Authoring API region	Publishing & querying region API region name	Endpoint URL format
Africa	westus www.luis.ai ↗	South Africa North southafricanorth	<code>https://southafricanorth.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
Asia	westus www.luis.ai ↗	Central India centralindia	<code>https://centralindia.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
Asia	westus www.luis.ai ↗	East Asia eastasia	<code>https://eastasia.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
Asia	westus www.luis.ai ↗	Japan East japaneast	<code>https://japaneast.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
Asia	westus www.luis.ai ↗	Japan West japanwest	<code>https://japanwest.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
Asia	westus www.luis.ai ↗	Jio India West jioindiawest	<code>https://jioindiawest.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
Asia	westus www.luis.ai ↗	Korea Central koreacentral	<code>https://koreacentral.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
Asia	westus www.luis.ai ↗	Southeast Asia southeastasia	<code>https://southeastasia.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
Asia	westus www.luis.ai ↗	North UAE northuae	<code>https://northuae.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
North America	westus www.luis.ai ↗	Canada Central canadacentral	<code>https://canadacentral.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
North America	westus www.luis.ai ↗	Central US centralus	<code>https://centralus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
North America	westus www.luis.ai ↗	East US eastus	<code>https://eastus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
North America	westus www.luis.ai ↗	East US 2 eastus2	<code>https://eastus2.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
North America	westus www.luis.ai ↗	North Central US northcentralus	<code>https://northcentralus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
North America	westus www.luis.ai ↗	South Central US southcentralus	<code>https://southcentralus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
North America	westus www.luis.ai ↗	West Central US westcentralus	<code>https://westcentralus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
North America	westus www.luis.ai ↗	West US westus	<code>https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
North America	westus www.luis.ai ↗	West US 2 westus2	<code>https://westus2.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>

Global region	Authoring API region	Publishing & querying region	Endpoint URL format
API region name			
North America	westus www.luis.ai	West US 3 westus3	<code>https://westus3.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>
South America	westus www.luis.ai	Brazil South brazilsouth	<code>https://brazilsouth.api.cognitive.microsoft.com/luis/v2.0/apps/YOUR-APP-ID?subscription-key=YOUR-SUBSCRIPTION-KEY</code>

Endpoints

Learn more about the [authoring and prediction endpoints](#).

Failover regions

Each region has a secondary region to fail over to. Failover will only happen in the same geographical region.

Authoring regions have [paired fail-over regions](#).

The following publishing regions do not have a failover region:

- Brazil South
- Southeast Asia

Next steps

[Prebuilt entities reference](#)

Limits for your LUIS model and keys

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

LUIS has several limit areas. The first is the [model limit](#), which controls intents, entities, and features in LUIS. The second area is [quota limits](#) based on resource type. A third area of limits 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 limits

If your app exceeds the LUIS model limits, consider using a [LUIS dispatch](#) app or using a [LUIS container](#).

Area	Limit
App name	*Default character max
Applications	500 applications per Azure authoring resource
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. Dispatch-based ↗ 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-learning entities + roles: composite,	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 limit. 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.

Area	Limit
simple, entity role	Subentities can be nested up to 5 levels, with a maximum of 20 children per level.
Model as a feature	Maximum number of models that can be used as a feature to a specific model to be 10 models. The maximum number of phrase lists used as a feature for a specific model to be 10 phrase lists.
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	500 phrase lists. 10 global phrase lists due to the model as a feature limit. Non-interchangeable phrase list has max of 5,000 phrases. Interchangeable phrase list has max of 50,000 phrases. Maximum number of total phrases per application of 500,000 phrases.
Prebuilt entities	no 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 If you have text longer than this character limit, you need to segment the utterance prior to input to LUIS and you will receive individual intent responses per segment. There are obvious breaks you can work with, such as punctuation marks and long pauses in speech.
Utterance examples	15,000 per application - there is no limit on the number of utterances per intent If you need to train the application with more examples, use a dispatch model approach. You train individual LUIS apps (known as child apps to the parent dispatch app) with one or more intents and then train a dispatch app that samples from each child LUIS app's utterances to direct the prediction request to the correct child app.
Versions	100 versions per application
Version name	128 characters

*Default character max is 50 characters.

Name uniqueness

Object names must be unique when compared to other objects of the same level.

Objects	Restrictions
Intent, entity	All intent and entity names must be unique in a version of an app.
ML entity components	All machine-learning entity components (child entities) must be unique, within that entity for components at the same level.
Features	All named features, such as phrase lists, must be unique within a version of an app.
Entity roles	All roles on an entity or entity component must be unique when they are at the same entity level (parent, child, grandchild, etc.).

Object naming

Do not use the following characters in the following names.

Object	Exclude characters
Intent, entity, and role names	: \$ & % * () + ? ~
Version name	\ / : ? & = + () % @ \$ ~ ! #

Resource usage and limits

Language Understand has separate resources, 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](#).

Authoring resource limits

Use the *kind*, `LUIS.Authoring`, when filtering resources in the Azure portal. LUIS limits 500 applications per Azure authoring resource.

Authoring resource	Authoring TPS
F0 - Free tier	1 million/month, 5/second

- TPS = Transactions per second

[Learn more about pricing.](#) ↗

Query prediction resource limits

Use the *kind*, LUIS, when filtering resources in the Azure portal. The LUIS query prediction endpoint resource, used on the runtime, is only valid for endpoint queries.

Query Prediction resource	Query TPS
F0 - Free tier	10 thousand/month, 5/second
S0 - Standard tier	50/second

Sentiment analysis

[Sentiment analysis integration](#), which provides sentiment information, is provided without requiring another Azure resource.

Speech integration

Speech integration provides 1 thousand endpoint requests per unit cost.

[Learn more about pricing.](#) ↗

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.

Prebuilt domain reference for your LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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.

Prebuilt domains per language

The table below summarizes the currently supported domains. Support for English is usually more complete than others.

Entity Type	EN-US	ZH-CN	DE	FR	ES	IT	PT-BR	KO	NL	TR
Web	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Prebuilt domains are **not supported** in:

- French Canadian
- Hindi
- Spanish Mexican
- Japanese

Next steps

Learn the [simple entity](#).

Entities per culture in your LUIS model

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

Language Understanding (LUIS) provides prebuilt entities.

Entity resolution

When a prebuilt entity is included in your application, LUIS includes the corresponding entity resolution in the endpoint response. All example utterances are also labeled with the entity.

The behavior of prebuilt entities can't be modified but you can improve resolution by [adding the prebuilt entity as a feature to a machine-learning entity or sub-entity](#).

Availability

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)	
English	en-GB (British)	
French	fr-CA (Canada), fr-FR (France),	
German	de-DE	
Italian	it-IT	

Culture	Subcultures	Notes
Japanese	ja-JP	
Korean	ko-KR	
Portuguese	pt-BR (Brazil)	
Spanish	es-ES (Spain), es-MX (Mexico)	
Turkish	turkish	

Prediction endpoint runtime

The availability of a prebuilt entity in a specific language is determined by the prediction endpoint runtime version.

Chinese entity support

The following entities are supported:

Prebuilt entity	zh-CN
Age :	V2, V3
year	
month	
week	
day	
Currency (money) :	V2, V3
dollar	
fractional unit (ex: penny)	
DatetimeV2 :	V2, V3
date	
daterange	
time	
timerange	
Dimension :	V2, V3
volume	
area	
weight	
information (ex: bit/byte)	
length (ex: meter)	
speed (ex: mile per hour)	

Prebuilt entity	zh-CN
Email	V2, V3
Number	V2, V3
Ordinal	V2, V3
Percentage	V2, V3
PersonName	V2, V3
Phonenumber	V2, V3
Temperature: fahrenheit kelvin rankine delisle celsius	V2, V3
URL	V2, V3

Dutch entity support

The following entities are supported:

Prebuilt entity	nl-NL
Age: year month week day	V2, V3
Currency (money): dollar fractional unit (ex: penny)	V2, V3
Dimension: volume area weight information (ex: bit/byte) length (ex: meter) speed (ex: mile per hour)	V2, V3
Email	V2, V3

Prebuilt entity	nl-NL
KeyPhrase	V2, V3
Number	V2, V3
Ordinal	V2, V3
Percentage	V2, V3
Phonenumber	V2, V3
Temperature:	V2, V3
fahrenheit	
kelvin	
rankine	
delisle	
celsius	
URL	V2, V3

English (American) entity support

The following entities are supported:

Prebuilt entity	en-US
Age:	V2, V3
year	
month	
week	
day	
Currency (money):	V2, V3
dollar	
fractional unit (ex: penny)	
DatetimeV2:	V2, V3
date	
daterange	
time	
timerange	
Dimension:	V2, V3
volume	
area	
weight	
information (ex: bit/byte)	

Prebuilt entity	en-US
length (ex: meter) speed (ex: mile per hour)	
Email	V2, V3
GeographyV2	V2, V3
KeyPhrase	V2, V3
Number	V2, V3
Ordinal	V2, V3
OrdinalV2	V2, V3
Percentage	V2, V3
PersonName	V2, V3
Phonenumber	V2, V3
Temperature: fahrenheit kelvin rankine delisle celsius	V2, V3
URL	V2, V3

English (British) entity support

The following entities are supported:

Prebuilt entity	en-GB
Age: year month week day	V2, V3
Currency (money): dollar fractional unit (ex: penny)	V2, V3
DatetimeV2: date	V2, V3

Prebuilt entity	en-GB
daterange time timerange	
Dimension: volume area weight information (ex: bit/byte) length (ex: meter) speed (ex: mile per hour)	V2, V3
Email	V2, V3
GeographyV2	V2, V3
KeyPhrase	V2, V3
Number	V2, V3
Ordinal	V2, V3
OrdinalV2	V2, V3
Percentage	V2, V3
PersonName	V2, V3
Phonenumber	V2, V3
Temperature: fahrenheit kelvin rankine delisle celsius	V2, V3
URL	V2, V3

French (France) entity support

The following entities are supported:

Prebuilt entity	fr-FR
Age: year month	V2, V3

Prebuilt entity	fr-FR
week	
day	
Currency (money):	V2, V3
dollar	
fractional unit (ex: penny)	
DatetimeV2:	V2, V3
date	
daterange	
time	
timerange	
Dimension:	V2, V3
volume	
area	
weight	
information (ex: bit/byte)	
length (ex: meter)	
speed (ex: mile per hour)	
Email	V2, V3
GeographyV2	-
KeyPhrase	V2, V3
Number	V2, V3
Ordinal	V2, V3
Percentage	V2, V3
Phonenumber	V2, V3
Temperature:	V2, V3
fahrenheit	
kelvin	
rankine	
delisle	
celsius	
URL	V2, V3

French (Canadian) entity support

The following entities are supported:

Prebuilt entity	fr-CA
Age: year month week day	V2, V3
Currency (money): dollar fractional unit (ex: penny)	V2, V3
DatetimeV2: date daterange time timerange	V2, V3
Dimension: volume area weight information (ex: bit/byte) length (ex: meter) speed (ex: mile per hour)	V2, V3
Email	V2, V3
KeyPhrase	V2, V3
Number	V2, V3
Ordinal	V2, V3
Percentage	V2, V3
Phonenumber	V2, V3
Temperature: fahrenheit kelvin rankine delisle celsius	V2, V3
URL	V2, V3

German entity support

The following entities are supported:

Prebuilt entity	de-DE
Age: year month week day	V2, V3
Currency (money): dollar fractional unit (ex: penny)	V2, V3
DatetimeV2: date daterange time timerange	V2, V3
Dimension: volume area weight information (ex: bit/byte) length (ex: meter) speed (ex: mile per hour)	V2, V3
Email	V2, V3
KeyPhrase	V2, V3
Number	V2, V3
Ordinal	V2, V3
Percentage	V2, V3
Phonenumber	V2, V3
Temperature: fahrenheit kelvin rankine delisle celsius	V2, V3
URL	V2, V3

Italian entity support

Italian prebuilt age, currency, dimension, number, percentage *resolution* changed from V2 and V3 preview.

The following entities are supported:

Prebuilt entity	it-IT
Age :	V2, V3
year	
month	
week	
day	
Currency (money) :	V2, V3
dollar	
fractional unit (ex: penny)	
DatetimeV2 :	V2, V3
date	
daterange	
time	
timerange	
Dimension :	V2, V3
volume	
area	
weight	
information (ex: bit/byte)	
length (ex: meter)	
speed (ex: mile per hour)	
Email	V2, V3
KeyPhrase	V2, V3
Number	V2, V3
Ordinal	V2, V3
Percentage	V2, V3
Phonenumber	V2, V3
Temperature :	V2, V3
fahrenheit	
kelvin	
rankine	
delisle	
celsius	
URL	V2, V3

Japanese entity support

The following entities are supported:

Prebuilt entity	ja-JP
Age :	V2, -
year	
month	
week	
day	
Currency (money) :	V2, -
dollar	
fractional unit (ex: penny)	
Dimension :	V2, -
volume	
area	
weight	
information (ex: bit/byte)	
length (ex: meter)	
speed (ex: mile per hour)	
Email	V2, V3
KeyPhrase	V2, V3
Number	V2, -
Ordinal	V2, -
Percentage	V2, -
Phonenumber	V2, V3
Temperature :	V2, -
fahrenheit	
kelvin	
rankine	
delisle	
celsius	
URL	V2, V3

Korean entity support

The following entities are supported:

Prebuilt entity	ko-KR
Email	V2, V3
KeyPhrase	V2, V3
Phonenumber	V2, V3
URL	V2, V3

Portuguese (Brazil) entity support

The following entities are supported:

Prebuilt entity	pt-BR
Age :	V2, V3
year	
month	
week	
day	
Currency (money) :	V2, V3
dollar	
fractional unit (ex: penny)	
DatetimeV2 :	V2, V3
date	
daterange	
time	
timerange	
Dimension :	V2, V3
volume	
area	
weight	
information (ex: bit/byte)	
length (ex: meter)	
speed (ex: mile per hour)	
Email	V2, V3
KeyPhrase	V2, V3
Number	V2, V3
Ordinal	V2, V3
Percentage	V2, V3

Prebuilt entity	pt-BR
Phonenumber	V2, V3
Temperature: fahrenheit kelvin rankine delisle celsius	V2, V3
URL	V2, V3

KeyPhrase is not available in all subcultures of Portuguese (Brazil) - pt-BR.

Spanish (Spain) entity support

The following entities are supported:

Prebuilt entity	es-ES
Age: year month week day	V2, V3
Currency (money): dollar fractional unit (ex: penny)	V2, V3
DatetimeV2: date daterange time timerange	V2, V3
Dimension: volume area weight information (ex: bit/byte) length (ex: meter) speed (ex: mile per hour)	V2, V3
Email	V2, V3
KeyPhrase	V2, V3

Prebuilt entity	es-ES
Number	V2, V3
Ordinal	V2, V3
Percentage	V2, V3
Phonenumber	V2, V3
Temperature: fahrenheit kelvin rankine delisle celsius	V2, V3
URL	V2, V3

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)	-

Prebuilt entity	es-MX
Email	V2, V3
KeyPhrase	V2, V3
Number	V2, V3
Ordinal	-
Percentage	-
Phonenumber	V2, V3
Temperature: fahrenheit kelvin rankine delisle celsius	-
URL	V2, V3

Turkish entity support

Prebuilt entity	tr-tr
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)	-

Prebuilt entity	tr-tr
length (ex: meter)	-
speed (ex: mile per hour)	-
Email	-
Number	-
Ordinal	-
Percentage	-
Temperature:	-
fahrenheit	-
kelvin	-
rankine	-
delisle	-
celsius	-
URL	-

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

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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

V3 response

The following JSON is with the `verbose` parameter set to `false`:

JSON

```
"entities": {  
    "age": [  
        {  
            "number": 90,  
            "unit": "Day"  
        }  
    ]  
}
```

Next steps

Learn more about the [V3 prediction endpoint](#).

Learn about the [currency](#), [datetimeV2](#), and [dimension](#) entities.

Currency prebuilt entity for a LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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

V3 response

The following JSON is with the `verbose` parameter set to `false`:

JSON

```
"entities": {  
    "money": [  
        {  
            "number": 10.99,  
            "units": "Dollar"  
        }  
    ]  
}
```

Next steps

Learn more about the [V3 prediction endpoint](#).

Learn about the [datetimeV2](#), [dimension](#), and [email](#) entities.

DatetimeV2 prebuilt entity for a LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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 utterance and its partial JSON response is shown below.

8am on may 2nd 2019

V3 response

JSON

```
"entities": {  
    "datetimeV2": [  
        {  
            "type": "datetime",  
            "values": [  
                {  
                    "timex": "2019-05-02T08",  
                    "resolution": [  
                        {  
                            "value": "2019-05-02 08:00:00"  
                        }  
                    ]  
                }  
            ]  
        }  
    ]  
}
```

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

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	Property description
name	
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.
mod	term used to describe how to use the value such as <code>before</code> , <code>after</code> .

Property	Property description
<code>name</code>	
<code>type</code>	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> .
<code>value</code>	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 `duration`.

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 following 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.

Date resolution example

The following utterance and its partial JSON response is shown below.

`May 2nd`

V3 response

JSON

```
"entities": {  
    "datetimeV2": [  
        {  
            "type": "date",  
            "values": [  
                {  
                    "timex": "XXXX-05-02",  
                    "resolution": [  
                        {  
                            "value": "2019-05-02"  
                        },  
                        {  
                            "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.

The following utterance and its partial JSON response is shown below.

May 2nd to May 5th

V3 response

JSON

```
"entities": {  
    "datetimeV2": [
```

```
{  
    "type": "daterange",  
    "values": [  
        {  
            "timex": "(XXXX-05-02,XXXX-05-05,P3D)",  
            "resolution": [  
                {  
                    "start": "2019-05-02",  
                    "end": "2019-05-05"  
                },  
                {  
                    "start": "2020-05-02",  
                    "end": "2020-05-05"  
                }  
            ]  
        }  
    ]  
}
```

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.

The following utterance and its partial JSON response is shown below.

Tuesday to Thursday

V3 response

JSON

```
"entities": {  
    "datetimeV2": [  
        {  
            "type": "daterange",  
            "values": [  
                {  
                    "timex": "(XXXX-WXX-2,XXXX-WXX-4,P2D)",  
                    "resolution": [  
                        {  
                            "start": "2019-10-08",  
                            "end": "2019-10-10"  
                        }  
                    ]  
                }  
            ]  
        }  
    ]  
}
```

```
        },
        {
            "start": "2019-10-15",
            "end": "2019-10-17"
        }
    ]
}
]
```

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

DatetimeV2 JSON response has changed in the API V3. The following example shows how LUIS uses **datetimeV2** to resolve the utterance that has a time range.

Changes from API V2:

- `datetimeV2.timex.type` property is no longer returned because it is returned at the parent level, `datetimev2.type`.
- The `datetimeV2.value` property has been renamed to `datetimeV2.timex`.

The following utterance and its partial JSON response is shown below.

from 6pm to 7pm

V3 response

The following JSON is with the `verbose` parameter set to `false`:

JSON

```
"entities": {
    "datetimeV2": [
        {
            "type": "timerange",
            "values": [
                {

```

```
        "timex": "(T18,T19,PT1H)",  
        "resolution": [  
            {  
                "start": "18:00:00",  
                "end": "19:00:00"  
            }  
        ]  
    }  
}  
]
```

Time resolution example

The following utterance and its partial JSON response is shown below.

8am

V3 response

JSON

```
"entities": {  
    "datetimeV2": [  
        {  
            "type": "time",  
            "values": [  
                {  
                    "timex": "T08",  
                    "resolution": [  
                        {  
                            "value": "08:00:00"  
                        }  
                    ]  
                }  
            ]  
        }  
    }  
}
```

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. Select **Add prebuilt entity**
4. Select `datetimeV2` and click **Save**.

Next steps

Learn more about the [V3 prediction endpoint](#).

Learn about the [dimension](#), [email](#) entities, and [number](#).

Dimension prebuilt entity for a LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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

The following entity objects are returned for the query:

10 1/2 miles of cable

V3 response

The following JSON is with the `verbose` parameter set to `false`:

JSON

```
"entities": {  
    "dimension": [  
        {  
            "number": 10.5,  
            "units": "Mile"  
        }  
    ]  
}
```

Next steps

Learn more about the [V3 prediction endpoint](#).

Learn about the [email](#), [number](#), and [ordinal](#) entities.

Deprecated prebuilt entities in a LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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>	<code>{ "type": "builtin.geography.city", "entity": "seattle" }</code>
<code>builtin.geography.city</code>	<code>paris</code>	<code>{ "type": "builtin.geography.city", "entity": "paris" }</code>
<code>builtin.geography.country</code>	<code>australia</code>	<code>{ "type": "builtin.geography.country", "entity": "australia" }</code>
<code>builtin.geography.country</code>	<code>japan</code>	<code>{ "type": "builtin.geography.country", "entity": "japan" }</code>
<code>builtin.geography.pointOfInterest</code>	<code>amazon river</code>	<code>{ "type": "builtin.geography.pointOfInterest", "entity": "amazon river" }</code>
<code>builtin.geography.pointOfInterest</code>	<code>sahara desert</code>	<code>{ "type": "builtin.geography.pointOfInterest", "entity": "sahara desert" }</code>

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:

JSON

```
{  
    "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.martial_artist	bruce lee
builtin.encyclopedia.people.person	builtin.encyclopedia.architecture.architect	james gallien
builtin.encyclopedia.people.person	builtin.encyclopedia.geography.mountaineer	jean couzy
builtin.encyclopedia.people.person	builtin.encyclopedia.celebrities.celebrity	angelina jolie
builtin.encyclopedia.people.person	builtin.encyclopedia.music.musician	bob dylan
builtin.encyclopedia.people.person	builtin.encyclopedia.soccer.player	diego maradona
builtin.encyclopedia.people.person	builtin.encyclopedia.baseball.player	babe ruth
builtin.encyclopedia.people.person	builtin.encyclopedia.basketball.player	heiko schaffartzik
builtin.encyclopedia.people.person	builtin.encyclopedia.olympics.athlete	andre agassi
builtin.encyclopedia.people.person	builtin.encyclopedia.basketball.coach	bob huggins
builtin.encyclopedia.people.person	builtin.encyclopedia.american_football.coach	james franklin
builtin.encyclopedia.people.person	builtin.encyclopedia.cricket.coach	andy flower
builtin.encyclopedia.people.person	builtin.encyclopedia.ice_hockey.coach	david quinn
builtin.encyclopedia.people.person	builtin.encyclopedia.ice_hockey.player	vincent lecavalier
builtin.encyclopedia.people.person	builtin.encyclopedia.government.politician	harold nicolson
builtin.encyclopedia.people.person	builtin.encyclopedia.government.us_president	barack obama
builtin.encyclopedia.people.person	builtin.encyclopedia.government.us_vice_president	dick cheney
builtin.encyclopedia.organization.organization	builtin.encyclopedia.organization.organization	united nations
builtin.encyclopedia.organization.organization	builtin.encyclopedia.sports.league	american league
builtin.encyclopedia.organization.organization	builtin.encyclopedia.ice_hockey.conference	western hockey league

Prebuilt entity	Prebuilt entity (sub-types)	Example utterance
builtin.encyclopedia.organization.organization	builtin.encyclopedia.baseball.division	american league east
builtin.encyclopedia.organization.organization	builtin.encyclopedia.baseball.league	major league baseball
builtin.encyclopedia.organization.organization	builtin.encyclopedia.basketball.conference	national basketball league
builtin.encyclopedia.organization.organization	builtin.encyclopedia.basketball.division	pacific division
builtin.encyclopedia.organization.organization	builtin.encyclopedia.soccer.league	premier league
builtin.encyclopedia.organization.organization	builtin.encyclopedia.american_football.division	afc north
builtin.encyclopedia.organization.organization	builtin.encyclopedia.broadcast.broadcast	nebraska educational telecommunications
builtin.encyclopedia.organization.organization	builtin.encyclopedia.broadcast.tv_station	abc
builtin.encyclopedia.organization.organization	builtin.encyclopedia.broadcast.tv_channel	cnn world
builtin.encyclopedia.organization.organization	builtin.encyclopedia.broadcast.radio_station	bbc radio 1
builtin.encyclopedia.organization.organization	builtin.encyclopedia.business.operation	bank of china
builtin.encyclopedia.organization.organization	builtin.encyclopedia.music.record_label	pixar
builtin.encyclopedia.organization.organization	builtin.encyclopedia.aviation.airline	air france
builtin.encyclopedia.organization.organization	builtin.encyclopedia.automotive.company	general motors
builtin.encyclopedia.organization.organization	builtin.encyclopedia.music.musical_instrument_company	gibson guitar corporation
builtin.encyclopedia.organization.organization	builtin.encyclopedia.tv.network	cartoon network
builtin.encyclopedia.organization.organization	builtin.encyclopedia.education.educational_institution	cornwall hill college
builtin.encyclopedia.organization.organization	builtin.encyclopedia.education.school	boston arts academy
builtin.encyclopedia.organization.organization	builtin.encyclopedia.education.university	johns hopkins university
builtin.encyclopedia.organization.organization	builtin.encyclopedia.sports.team	united states national handball team
builtin.encyclopedia.organization.organization	builtin.encyclopedia.basketball.team	chicago bulls
builtin.encyclopedia.organization.organization	builtin.encyclopedia.sports.professional_sports_team	boston celtics
builtin.encyclopedia.organization.organization	builtin.encyclopedia.cricket.team	mumbai indians
builtin.encyclopedia.organization.organization	builtin.encyclopedia.baseball.team	houston astros
builtin.encyclopedia.organization.organization	builtin.encyclopedia.american_football.team	green bay packers
builtin.encyclopedia.organization.organization	builtin.encyclopedia.ice_hockey.team	hamilton bulldogs
builtin.encyclopedia.organization.organization	builtin.encyclopedia.soccer.team	fc bayern munich
builtin.encyclopedia.organization.organization builtin.encyclopedia.government.political_party pertubuhan		

Prebuilt entity	Prebuilt entity (sub-types)	Example utterance
kebangsaan melayu singapura		
builtin.encyclopedia.time.event	builtin.encyclopedia.time.event	1740 batavia massacre
builtin.encyclopedia.time.event	builtin.encyclopedia.sports.championship_event	super bowl xxxix
builtin.encyclopedia.time.event	builtin.encyclopedia.award.competition	eurovision song contest 2003
builtin.encyclopedia.tv.series_episode	builtin.encyclopedia.tv.series_episode	the magnificent seven
builtin.encyclopedia.tv.series_episode	builtin.encyclopedia.tv.multipart_tv_episode	the deadly assassin
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.commerce.consumer_product	nokia lumia 620
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.music.album	dance pool
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.automotive.model	pontiac fiero
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.computer.computer	toshiba satellite
builtin.encyclopedia.commerce.consumer_product	builtin.encyclopedia.computer.web_browser	internet explorer
builtin.encyclopedia.commerce.brand	builtin.encyclopedia.commerce.brand	diet coke
builtin.encyclopedia.commerce.brand	builtin.encyclopedia.automotive.make	chrysler
builtin.encyclopedia.music.artist	builtin.encyclopedia.music.artist	michael jackson
builtin.encyclopedia.music.artist	builtin.encyclopedia.music.group	the yardbirds
builtin.encyclopedia.music.music_video	builtin.encyclopedia.music.music_video	the beatles anthology
builtin.encyclopedia.theater.play	builtin.encyclopedia.theater.play	camelot
builtin.encyclopedia.sports.fight_song	builtin.encyclopedia.sports.fight_song	the cougar song
builtin.encyclopedia.film.series	builtin.encyclopedia.film.series	the twilight saga
builtin.encyclopedia.tv.program	builtin.encyclopedia.tv.program	late night with david letterman
builtin.encyclopedia.radio.radio_program	builtin.encyclopedia.radio.radio_program	grand ole opry
builtin.encyclopedia.film.film	builtin.encyclopedia.film.film	alice in wonderland
builtin.encyclopedia.cricket.tournament	builtin.encyclopedia.cricket.tournament	cricket world cup
builtin.encyclopedia.government.government	builtin.encyclopedia.government.government	european commission
builtin.encyclopedia.sports.team_owner	builtin.encyclopedia.sports.team_owner	bob castellini
builtin.encyclopedia.music.genre	builtin.encyclopedia.music.genre	eastern europe
builtin.encyclopedia.ice_hockey.division	builtin.encyclopedia.ice_hockey.division	hockeyallsvenskan
builtin.encyclopedia.architecture.style	builtin.encyclopedia.architecture.style	spanish colonial revival architecture

Prebuilt entity	Prebuilt entity (sub-types)	Example utterance
builtin.encyclopedia.broadcast.producer	builtin.encyclopedia.broadcast.producer	columbia tristar television
builtin.encyclopedia.book.author	builtin.encyclopedia.book.author	adam maxwell
builtin.encyclopedia.religion.founding_figur	builtin.encyclopedia.religion.founding_figur	gautama buddha
builtin.encyclopedia.martial_arts.martial_art	builtin.encyclopedia.martial_arts.martial_art	american kenpo
builtin.encyclopedia.sports.school	builtin.encyclopedia.sports.school	yale university
builtin.encyclopedia.business.product_line	builtin.encyclopedia.business.product_line	canon powershot
builtin.encyclopedia.internet.website	builtin.encyclopedia.internet.website	bing
builtin.encyclopedia.time.holiday	builtin.encyclopedia.time.holiday	easter
builtin.encyclopedia.food.candy_bar	builtin.encyclopedia.food.candy_bar	cadbury dairy milk
builtin.encyclopedia.finance.stock_exchange	builtin.encyclopedia.finance.stock_exchange	tokyo stock exchange
builtin.encyclopedia.film.festival	builtin.encyclopedia.film.festival	berlin international film festival

Next steps

Learn about the [dimension](#), [email](#) entities, and [number](#).

Email prebuilt entity for a LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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

The following entity objects are returned for the query:

```
please send the information to patti@contoso.com
```

V3 response

The following JSON is with the `verbose` parameter set to `false`:

JSON

```
"entities": {  
    "email": [  
        "patti@contoso.com"  
    ]  
}
```

Next steps

Learn more about the [V3 prediction endpoint](#).

Learn about the [number](#), [ordinal](#), and [percentage](#).

GeographyV2 prebuilt entity for a LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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

The following entity objects are returned for the query:

```
Carol is visiting the sphinx in gizah egypt in africa before heading to texas.
```

V3 response

The following JSON is with the `verbose` parameter set to `false`:

JSON

```
"entities": {  
    "geographyV2": [  
        {  
            "value": "the sphinx",  
            "type": "poi"  
        },  
        {  
            "value": "gizah",  
            "type": "city"  
        },  
        {  
            "value": "egypt",  
            "type": "countryRegion"  
        },  
        {  
            "value": "africa",  
            "type": "continent"  
        },  
        {  
            "value": "texas",  
            "type": "state"  
        }  
    ]  
}
```

In the preceding JSON, `poi` is an abbreviation for **Point of Interest**.

Next steps

Learn more about the [V3 prediction endpoint](#).

Learn about the [email](#), [number](#), and [ordinal](#) entities.

keyPhrase prebuilt entity for a LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

The keyPhrase entity extracts a variety of key phrases from an utterance. You don't need to add example utterances containing keyPhrase to the application. The keyPhrase entity is supported in [many cultures](#) as part of the [Language service](#) features.

Resolution for prebuilt keyPhrase entity

The following entity objects are returned for the query:

```
where is the educational requirements form for the development and engineering group
```

V3 response

The following JSON is with the `verbose` parameter set to `false`:

JSON

```
"entities": {  
    "keyPhrase": [  
        "educational requirements",  
        "development"  
    ]  
}
```

Next steps

Learn more about the [V3 prediction endpoint](#).

Learn about the [percentage](#), [number](#), and [age](#) entities.

Number prebuilt entity for a LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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

The following entity objects are returned for the query:

```
order two dozen eggs
```

V3 response

The following JSON is with the `verbose` parameter set to `false`:

JSON

```
"entities": {  
    "number": [  
        24  
    ]  
}
```

Next steps

Learn more about the [V3 prediction endpoint](#).

Learn about the [currency](#), [ordinal](#), and [percentage](#).

Ordinal prebuilt entity for a LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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

The following entity objects are returned for the query:

`Order the second option`

V3 response

The following JSON is with the `verbose` parameter set to `false`:

JSON

```
"entities": {  
    "ordinal": [  
        2  
    ]  
}
```

Next steps

Learn more about the [V3 prediction endpoint](#).

Learn about the [OrdinalV2](#), [phone number](#), and [temperature](#) entities.

Ordinal V2 prebuilt entity for a LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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

The following entity objects are returned for the query:

```
what is the second to last choice in the list
```

V3 response

The following JSON is with the `verbose` parameter set to `false`:

JSON

```
"entities": {  
    "ordinalV2": [  
        {  
            "offset": -1,  
            "relativeTo": "end"  
        }  
    ]  
}
```

Next steps

Learn more about the [V3 prediction endpoint](#).

Learn about the [percentage](#), [phone number](#), and [temperature](#) entities.

Percentage prebuilt entity for a LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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

The following entity objects are returned for the query:

```
set a trigger when my stock goes up 2%
```

V3 response

The following JSON is with the `verbose` parameter set to `false`:

JSON

```
"entities": {  
    "percentage": [  
        2  
    ]  
}
```

Next steps

Learn more about the [V3 prediction endpoint](#).

Learn about the [ordinal](#), [number](#), and [temperature](#) entities.

PersonName prebuilt entity for a LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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

The following entity objects are returned for the query:

Is Jill Jones in Cairo?

V3 response

The following JSON is with the `verbose` parameter set to `false`:

JSON

```
"entities": {  
    "personName": [  
        "Jill Jones"  
    ]  
}
```

Next steps

Learn more about the [V3 prediction endpoint](#).

Learn about the [email](#), [number](#), and [ordinal](#) entities.

Phone number prebuilt entity for a LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

The `phonenumbers` 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 `phonenumbers` 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

The following entity objects are returned for the query:

```
my mobile is 1 (800) 642-7676
```

V3 response

The following JSON is with the `verbose` parameter set to `false`:

JSON

```
"entities": {  
    "phonenumbers": [  
        "1 (800) 642-7676"  
    ]  
}
```

Next steps

Learn more about the [V3 prediction endpoint](#).

Learn about the [percentage](#), [number](#), and [temperature](#) entities.

Temperature prebuilt entity for a LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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

The following entity objects are returned for the query:

```
set the temperature to 30 degrees
```

V3 response

The following JSON is with the `verbose` parameter set to `false`:

JSON

```
"entities": [
    "temperature": [
        {
            "number": 30,
            "units": "Degree"
        }
    ]
}
```

Next steps

Learn more about the [V3 prediction endpoint](#).

Learn about the [percentage](#), [number](#), and [age](#) entities.

URL prebuilt entity for a LUIS app

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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

The following entity objects are returned for the query:

```
https://www.luis.ai is a great Azure AI services example of artificial  
intelligence
```

V3 response

The following JSON is with the `verbose` parameter set to `false`:

JSON

```
"entities": {  
    "url": [  
        "https://www.luis.ai"  
    ]  
}
```

Next steps

Learn more about the [V3 prediction endpoint](#).

Learn about the [ordinal](#), [number](#), and [temperature](#) entities.

Machine-learning entity

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

The machine-learning entity is the preferred entity for building LUIS applications.

Example JSON

Suppose the app takes pizza orders, such as the [decomposable entity tutorial](#). Each order can include several different pizzas, including different sizes.

Example utterances include:

Example utterances for pizza app

```
Can I get a pepperoni pizza and a can of coke please
```

```
can I get a small pizza with onions peppers and olives
```

```
pickup an extra large meat lovers pizza
```

V3 prediction endpoint response

Because a machine-learning entity can have many subentities with required features, this is just an example only. It should be considered a guide for what your entity will return.

Consider the query:

```
deliver 1 large cheese pizza on thin crust and 2 medium pepperoni pizzas on  
deep dish crust
```

This is the JSON if `verbose=false` is set in the query string:

```
JSON
```

```
"entities": {
    "Order": [
        {
            "FullPizzaWithModifiers": [
                {
                    "PizzaType": [
                        "cheese pizza"
                    ],
                    "Size": [
                        [
                            "Large"
                        ]
                    ],
                    "Quantity": [
                        1
                    ]
                },
                {
                    "PizzaType": [
                        "pepperoni pizzas"
                    ],
                    "Size": [
                        [
                            "Medium"
                        ]
                    ],
                    "Quantity": [
                        2
                    ],
                    "Crust": [
                        [
                            "Deep Dish"
                        ]
                    ]
                }
            ]
        }
    ],
    "ToppingList": [
        [
            "Cheese"
        ],
        [
            "Pepperoni"
        ]
    ],
    "CrustList": [
        [
            "Thin"
        ]
    ]
}
```

This is the JSON if `verbose=true` is set in the query string:

JSON

```
"entities": {
  "Order": [
    {
      "FullPizzaWithModifiers": [
        {
          "PizzaType": [
            "cheese pizza"
          ],
          "Size": [
            [
              "Large"
            ]
          ],
          "Quantity": [
            1
          ],
          "$instance": {
            "PizzaType": [
              {
                "type": "PizzaType",
                "text": "cheese pizza",
                "startIndex": 16,
                "length": 12,
                "score": 0.999998868,
                "modelTypeId": 1,
                "modelType": "Entity Extractor",
                "recognitionSources": [
                  "model"
                ]
              }
            ],
            "Size": [
              {
                "type": "SizeList",
                "text": "large",
                "startIndex": 10,
                "length": 5,
                "score": 0.998720646,
                "modelTypeId": 1,
                "modelType": "Entity Extractor",
                "recognitionSources": [
                  "model"
                ]
              }
            ],
            "Quantity": [
              {
                "type": "builtin.number",
                "text": "1",
                "startIndex": 8,
                "length": 1
              }
            ]
          }
        }
      ]
    }
  ]
}
```

```
        "length": 1,
        "score": 0.999878645,
        "modelTypeId": 1,
        "modelType": "Entity Extractor",
        "recognitionSources": [
            "model"
        ]
    }
}
},
{
    "PizzaType": [
        "pepperoni pizzas"
    ],
    "Size": [
        [
            [
                "Medium"
            ]
        ],
        [
            "Quantity": [
                2
            ],
            "Crust": [
                [
                    [
                        "Deep Dish"
                    ]
                ],
                "$instance": {
                    "PizzaType": [
                        {
                            "type": "PizzaType",
                            "text": "pepperoni pizzas",
                            "startIndex": 56,
                            "length": 16,
                            "score": 0.999987066,
                            "modelTypeId": 1,
                            "modelType": "Entity Extractor",
                            "recognitionSources": [
                                "model"
                            ]
                        }
                    ],
                    "Size": [
                        {
                            "type": "SizeList",
                            "text": "medium",
                            "startIndex": 49,
                            "length": 6,
                            "score": 0.999841452,
                            "modelTypeId": 1,
                            "modelType": "Entity Extractor",
                            "recognitionSources": [
                                "model"
                            ]
                        }
                    ]
                }
            ]
        ]
    ]
}
```

```
        }
    ],
    "Quantity": [
        {
            "type": "builtin.number",
            "text": "2",
            "startIndex": 47,
            "length": 1,
            "score": 0.9996054,
            "modelTypeId": 1,
            "modelType": "Entity Extractor",
            "recognitionSources": [
                "model"
            ]
        }
    ],
    "Crust": [
        {
            "type": "CrustList",
            "text": "deep dish crust",
            "startIndex": 76,
            "length": 15,
            "score": 0.761551,
            "modelTypeId": 1,
            "modelType": "Entity Extractor",
            "recognitionSources": [
                "model"
            ]
        }
    ]
},
],
"$instance": {
    "FullPizzaWithModifiers": [
        {
            "type": "FullPizzaWithModifiers",
            "text": "1 large cheese pizza on thin crust",
            "startIndex": 8,
            "length": 34,
            "score": 0.616001546,
            "modelTypeId": 1,
            "modelType": "Entity Extractor",
            "recognitionSources": [
                "model"
            ]
        },
        {
            "type": "FullPizzaWithModifiers",
            "text": "2 medium pepperoni pizzas on deep dish
crust",
            "startIndex": 47,
            "length": 44,
            "score": 0.7395033,
            "modelTypeId": 1,
            "modelType": "Entity Extractor",
            "recognitionSources": [
                "model"
            ]
        }
    ]
}
```

```
        "modelType": "Entity Extractor",
        "recognitionSources": [
            "model"
        ]
    }
}
],
{
    "ToppingList": [
        [
            [
                "Cheese"
            ],
            [
                [
                    "Pepperoni"
                ]
            ],
            [
                "CrustList": [
                    [
                        [
                            "Thin"
                        ]
                    ],
                    "$instance": {
                        "Order": [
                            {
                                "type": "Order",
                                "text": "1 large cheese pizza on thin crust and 2 medium
pepperoni pizzas on deep dish crust",
                                "startIndex": 8,
                                "length": 83,
                                "score": 0.6881274,
                                "modelTypeId": 1,
                                "modelType": "Entity Extractor",
                                "recognitionSources": [
                                    "model"
                                ]
                            }
                        ],
                        "ToppingList": [
                            {
                                "type": "ToppingList",
                                "text": "cheese",
                                "startIndex": 16,
                                "length": 6,
                                "modelTypeId": 5,
                                "modelType": "List Entity Extractor",
                                "recognitionSources": [
                                    "model"
                                ]
                            },
                            {
                                "type": "ToppingList",
                                "text": "pepperoni",
                                "startIndex": 56,
                                "length": 9,
                                "modelTypeId": 5,
                                "modelType": "List Entity Extractor",
                                "recognitionSources": [
                                    "model"
                                ]
                            }
                        ]
                    }
                ]
            ]
        ]
    ]
}
```

```
        "modelType": "List Entity Extractor",
        "recognitionSources": [
            "model"
        ]
    },
    "CrustList": [
        {
            "type": "CrustList",
            "text": "thin crust",
            "startIndex": 32,
            "length": 10,
            "modelTypeId": 5,
            "modelType": "List Entity Extractor",
            "recognitionSources": [
                "model"
            ]
        }
    ]
}
```

Next steps

Learn more about the machine-learning entity including a [tutorial](#), [concepts](#), and [how-to guide](#).

Learn about the [list](#) entity and [regular expression](#) entity.

List entity

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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 a case-insensitive match with a synonym or the canonical name. LUIS doesn't use the list beyond the match. Fuzzy matching, stemming, plurals, and other variations are not resolved with a list entity. To manage variations, consider using a [pattern](#) with the optional text syntax.

Token view: synonym 1	I want to travel to Arrabury Airport
Token view: synonym 2	I want to travel to AAB Airport
Entity view	I want to travel to airport
Airport List entity	Values: AAA = {"Anaa Airport", "AAA"} AAB = {"Arrabury Airport", "AAB"} AAC = {"El Arish International Airport", "AAC"} ...

Example .json to import into list entity

You can import values into an existing list entity using the following .json format:

JSON

```
[  
  {  
    "canonicalForm": "Blue",  
    "list": [  
      "navy",  
      "royal",  
      "baby"  
    ]  
  },  
  {  
    "canonicalForm": "Green",  
    "list": [  
      "kelly",  
      "forest",  
      "avacado"  
    ]  
  }  
]
```

Example JSON response

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.

V2 prediction endpoint response

JSON

```
"entities": [  
  {
```

```
"entity": "paris",
"type": "Cities",
"startIndex": 18,
"endIndex": 22,
"resolution": {
  "values": [
    "Paris"
  ]
}
]
```

Data object	Entity name	Value
List Entity	Cities	paris

Next steps

Learn more about entities:

- [Entity types](#)
- [Create entities](#)

Pattern.any entity

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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?
--

Utterance with entity in bold

```
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

Consider the following query:

```
where is the form Understand your responsibilities as a member of the community and  
who needs to sign it after I read it?
```

With the embedded form name to extract as a Pattern.any:

```
Understand your responsibilities as a member of the community
```

V2 prediction endpoint response

JSON

```
"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

Article • 07/18/2023

ⓘ Important

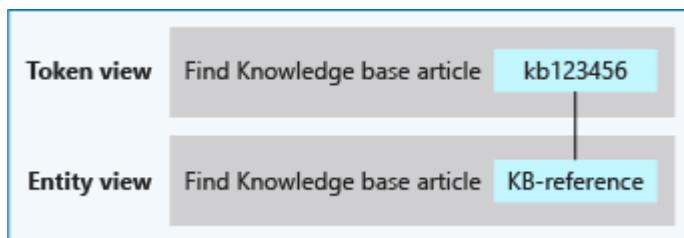
LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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 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.



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?:

V2 prediction endpoint response

JSON

```
"entities": [  
    {  
        "entity": "kb123456",  
        "type": "KB number",  
        "startIndex": 9,  
        "endIndex": 16  
    }  
]
```

Next steps

Learn more about entities:

- [Entity types](#)
- [Create entities](#)

Simple entity

Article • 07/18/2023

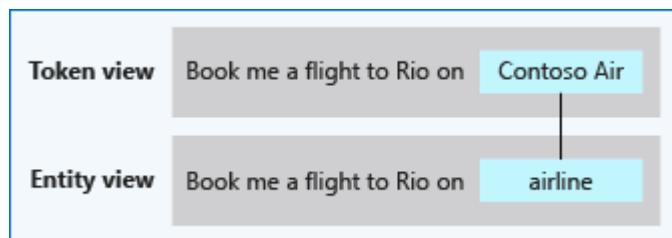
ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

A simple entity is a generic entity that describes a single concept and is learned from the machine-learning 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:

V2 prediction endpoint response

```
JSON
{
  "entities": [
    {
      "entity": "bob jones",
      "type": "Customer",
      "text": "Bob Jones",
      "offset": 0,
      "score": 0.99
    }
  ]
}
```

```
"startIndex": 0,  
"endIndex": 8,  
"score": 0.473899543  
}  
]
```

Data object	Entity name	Value
Simple Entity	Customer	bob jones

Next steps

[Learn pattern syntax](#)

Pattern syntax

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

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.

✖ Caution

Patterns only include machine-learning entity parents, not subentities. 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	<code>{}</code> - curly brackets	2	Where is form {entity-name}?
optional	<code>[]</code> - 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	<code>()</code> - parentheses	2	is (a b)
or	<code> </code> - 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	<code>^</code> - caret	-	^begin the utterance the utterance is done^

Function	Syntax	Nesting level	Example
			^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>

Allowed	Example
No	([([test1] test2)] test3)

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	(test1 test2 (test3 test4))
No	(test1 test2 test3 (test4 test5))

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 `Who does {Employee} manage?`.

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
<code>Book a ticket from {Location:Origin} to {Location:Destination}</code>

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.

Next steps

Learn more about patterns:

- [How to add patterns](#)
- [How to add pattern.any entity](#)
- [Patterns Concepts](#)

Understand how `sentiment` is returned in the .json response.

Sentiment analysis

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications to conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

If Sentiment analysis is configured, the LUIS json response includes sentiment analysis. Learn more about sentiment analysis in the [Language service](#) documentation.

LUIS uses V2 of the API.

Sentiment Analysis is configured when publishing your application. See [how to publish an app](#) for more information.

Resolution for sentiment

Sentiment data is a score between 1 and 0 indicating the positive (closer to 1) or negative (closer to 0) sentiment of the data.

English language

When culture is `en-us`, the response is:

JSON

```
"sentimentAnalysis": {  
    "label": "positive",  
    "score": 0.9163064  
}
```

Next steps

Learn more about the [V3 prediction endpoint](#).

Azure AI services support and help options

Article • 07/18/2023

Are you just starting to explore the functionality of Azure AI services? Perhaps you are implementing a new feature in your application. Or after using the service, do you have suggestions on how to improve it? Here are options for where you can get support, stay up-to-date, give feedback, and report bugs for Azure AI services.

Create an Azure support request

A

Explore the range of [Azure support options and choose the plan](#) that best fits, whether you're a developer just starting your cloud journey or a large organization deploying business-critical, strategic applications. Azure customers can create and manage support requests in the Azure portal.

- [Azure portal](#)
- [Azure portal for the United States government](#)

Post a question on Microsoft Q&A

For quick and reliable answers on your technical product questions from Microsoft Engineers, Azure Most Valuable Professionals (MVPs), or our expert community, engage with us on [Microsoft Q&A](#), Azure's preferred destination for community support.

If you can't find an answer to your problem using search, submit a new question to Microsoft Q&A. Use one of the following tags when you ask your question:

- [Azure AI services](#)

Vision

- [Azure AI Vision](#)
- [Custom Vision](#)
- [Face](#)
- [Document Intelligence](#)
- [Video Indexer](#)

Language

- Immersive Reader
- Language Understanding (LUIS)
- QnA Maker
- Language service
- Translator

Speech

- Speech service

Decision

- Anomaly Detector
- Content Moderator
- Metrics Advisor
- Personalizer

Azure OpenAI

- Azure OpenAI

Post a question to Stack Overflow



For answers on your developer questions from the largest community developer ecosystem, ask your question on Stack Overflow.

If you do submit a new question to Stack Overflow, please use one or more of the following tags when you create the question:

- Azure AI services ↗

Vision

- Azure AI Vision ↗
- Custom Vision ↗
- Face ↗
- Document Intelligence ↗
- Video Indexer ↗

Language

- Immersive Reader ↗
- Language Understanding (LUIS) ↗

- [QnA Maker](#)
- [Language service](#)
- [Translator](#)

Speech

- [Speech service](#)

Decision

- [Anomaly Detector](#)
- [Content Moderator](#)
- [Metrics Advisor](#)
- [Personalizer](#)

Azure OpenAI

- [Azure OpenAI](#)

Submit feedback

To request new features, post them on <https://feedback.azure.com>. Share your ideas for making Azure AI services and its APIs work better for the applications you develop.

- [Azure AI services](#)

Vision

- [Azure AI Vision](#)
- [Custom Vision](#)
- [Face](#)
- [Document Intelligence](#)
- [Video Indexer](#)

Language

- [Immersive Reader](#)
- [Language Understanding \(LUIS\)](#)
- [QnA Maker](#)
- [Language service](#)
- [Translator](#)

Speech

- [Speech service](#)

Decision

- [Anomaly Detector ↗](#)
- [Content Moderator ↗](#)
- [Metrics Advisor ↗](#)
- [Personalizer ↗](#)

Stay informed

Staying informed about features in a new release or news on the Azure blog can help you find the difference between a programming error, a service bug, or a feature not yet available in Azure AI services.

- Learn more about product updates, roadmap, and announcements in [Azure Updates ↗](#).
- News about Azure AI services is shared in the [Azure blog ↗](#).
- [Join the conversation on Reddit ↗](#) about Azure AI services.

Next steps

[What are Azure AI services?](#)

Language understanding glossary of common vocabulary and concepts

Article • 07/18/2023

ⓘ Important

LUIS will be retired on October 1st 2025 and starting April 1st 2023 you will not be able to create new LUIS resources. We recommend [migrating your LUIS applications](#) to [conversational language understanding](#) to benefit from continued product support and multilingual capabilities.

The Language Understanding (LUIS) glossary explains terms that you might encounter as you work with the LUIS service.

Active version

The active version is the [version](#) of your app that is updated when you make changes to the model using the LUIS portal. 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.

Active learning

Active learning is a technique of machine learning in which the machine learned model is used to identify informative new examples to label. In LUIS, active learning refers to adding utterances from the endpoint traffic whose current predictions are unclear to improve your model. Select "review endpoint utterances", to view utterances to label.

See also:

- [Conceptual information](#)
- [Tutorial reviewing endpoint utterances](#)
- How to improve the LUIS app by [reviewing endpoint utterances](#)

Application (App)

In LUIS, your application, or app, is a collection of machine learned models, built on the same data set, that works together to predict intents and entities for a particular scenario. Each application has a separate prediction endpoint.

If you are building an HR bot, you might have a set of intents, such as "Schedule leave time", "inquire about benefits" and "update personal information" and entities for each one of those intents that you group into a single application.

Authoring

Authoring is the ability to create, manage and deploy a LUIS app, either using the LUIS portal or the authoring APIs.

Authoring Key

The [authoring key](#) is used to author the app. Not used for production-level endpoint queries. For more information, see [resource limits](#).

Authoring Resource

Your LUIS [authoring resource](#) is a manageable item that is available through Azure. The resource is your access to the associated authoring, training, and publishing abilities of the Azure service. The resource includes authentication, authorization, and security information you need to access the associated Azure service.

The authoring resource has an Azure "kind" of `LUIS-Authoring`.

Batch test

Batch testing is the ability to validate a current LUIS app's models 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](#) run a batch test
- [Tutorial](#) - create and run a batch test

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.

Precision

In batch testing, precision (also called positive predictive value) is the fraction of relevant utterances among the retrieved utterances.

An example for an animal batch test is the number of sheep that were predicted divided by the total number of animals (sheep and non-sheep alike).

Recall

In batch testing, recall (also known as sensitivity), is the ability for LUIS to generalize.

An example for an animal batch test is the number of sheep that were predicted divided by the total number of sheep available.

True negative (TN)

A true negative is when your app correctly predicts no match. In batch testing, a true negative occurs when your app does predict an intent or entity for an example that has not been labeled with that intent or entity.

True positive (TP)

True positive (TP) A true positive is when your app correctly predicts a match. In batch testing, a true positive occurs when your app predicts an intent or entity for an example that has been labeled with that intent or entity.

Classifier

A classifier is a machine learned model that predicts what category or class an input fits into.

An [intent](#) is an example of a classifier.

Collaborator

A collaborator is conceptually the same thing as a [contributor](#). A collaborator is granted access when an owner adds the collaborator's email address to an app that isn't controlled with Azure role-based access control (Azure RBAC). If you are still using collaborators, you should migrate your LUIS account, and use LUIS authoring resources to manage contributors with Azure RBAC.

Contributor

A contributor is not the [owner](#) of the app, but has the same permissions to add, edit, and delete the intents, entities, utterances. A contributor provides Azure role-based access control (Azure RBAC) to a LUIS app.

See also:

- [How-to add contributors](#)

Descriptor

A descriptor is the term formerly used for a machine learning [feature](#).

Domain

In the LUIS context, a domain is an area of knowledge. Your domain is specific to your scenario. Different domains use specific language and terminology that have meaning in the context of the domain. For example, if you are building an application to play music, your application would have terms and language specific to music – words like "song, track, album, lyrics, b-side, artist". For examples of domains, see [prebuilt domains](#).

Endpoint

Authoring endpoint

The LUIS authoring endpoint URL is where you author, train, and publish your app. The endpoint URL contains the region or custom subdomain of the published app as well as the app ID.

Learn more about authoring your app programmatically from the [Developer reference](#)

Prediction endpoint

The LUIS prediction endpoint URL is where you submit LUIS queries after the [LUIS app](#) is authored and published. The endpoint URL contains the region or custom subdomain of the published app as well as the app ID. You can find the endpoint on the [Azure resources](#) page of your app, or you can get the endpoint URL from the [Get App Info](#) API.

Your access to the prediction endpoint is authorized with the LUIS prediction key.

Entity

[Entities](#) are words in utterances that describe information used to fulfill or identify an intent. If your entity is complex and you would like your model to identify specific parts, you can break your model into subentities. For example, you might want your model to predict an address, but also the subentities of street, city, state, and zipcode. Entities can also be used as features to models. Your response from the LUIS app will include both the predicted intents and all the entities.

Entity extractor

An entity extractor sometimes known only as an extractor is the type of machine learned model that LUIS uses to predict entities.

Entity schema

The entity schema is the structure you define for machine learned entities with subentities. The prediction endpoint returns all of the extracted entities and subentities defined in the schema.

Entity's subentity

A subentity is a child entity of a machine-learning entity.

Non-machine-learning entity

An entity that uses text matching to extract data:

- List entity
- Regular expression entity

List entity

A [list entity](#) represents a fixed, closed set of related words along with their synonyms. List entities are exact matches, unlike machine learned entities.

The entity will be predicted if a word in the list entity is included in the list. For example, if you have a list entity called "size" and you have the words "small, medium, large" in the list, then the size entity will be predicted for all utterances where the words "small", "medium", or "large" are used regardless of the context.

Regular expression

A [regular expression entity](#) represents a regular expression. Regular expression entities are exact matches, unlike machine learned entities.

Prebuilt entity

See Prebuilt model's entry for [prebuilt entity](#)

Features

In machine learning, a feature is a characteristic that helps the model recognize a particular concept. It is a hint that LUIS can use, but not a hard rule.

This term is also referred to as a [machine-learning feature](#).

These hints are used in conjunction with the labels to learn how to predict new data. LUIS supports both phrase lists and using other models as features.

Required feature

A required feature is a way to constrain the output of a LUIS model. When a feature for an entity is marked as required, the feature must be present in the example for the entity to be predicted, regardless of what the machine learned model predicts.

Consider an example where you have a prebuilt-number feature that you have marked as required on the quantity entity for a menu ordering bot. When your bot sees `I want a bajillion large pizzas?`, bajillion will not be predicted as a quantity regardless of the context in which it appears. Bajillion is not a valid number and won't be predicted by the number pre-built entity.

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, or paying a bill. In LUIS, an utterance as a whole is classified as an intent, but parts of the utterance are extracted as entities

Labeling examples

Labeling, or marking, is the process of associating a positive or negative example with a model.

Labeling for intents

In LUIS, intents within an app are mutually exclusive. This means when you add an utterance to an intent, it is considered a *positive* example for that intent and a *negative* example for all other intents. Negative examples should not be confused with the "None" intent, which represents utterances that are outside the scope of the app.

Labeling for entities

In LUIS, you [label](#) a word or phrase in an intent's example utterance with an entity as a *positive* example. Labeling shows the intent what it should predict for that utterance. The labeled utterances are used to train the intent.

LUIS app

See the definition for [application \(app\)](#).

Model

A (machine learned) model is a function that makes a prediction on input data. In LUIS, we refer to intent classifiers and entity extractors generically as "models", and we refer to a collection of models that are trained, published, and queried together as an "app".

Normalized value

You add values to your [list](#) entities. Each of those values can have a list of one or more synonyms. Only the normalized value is returned in the response.

Overfitting

Overfitting happens when the model is fixated on the specific examples and is not able to generalize well.

Owner

Each app has one owner who is the person that created the app. The owner manages permissions to the application in the Azure portal.

Phrase list

A [phrase list](#) is a specific type of machine learning feature that 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).

Prebuilt model

A [prebuilt model](#) is an intent, entity, or collection of both, along with labeled examples. These common prebuilt models can be added to your app to reduce the model development work required for your app.

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. These are created based on public data. You can choose to add a prebuilt entity as a stand-alone entity, or as a feature to an entity

Prebuilt intent

A prebuilt intent is an intent LUIS provides for common types of information and come with their own labeled example utterances.

Prediction

A prediction is a REST request to the Azure LUIS prediction service that takes in new data (user utterance), and applies the trained and published application to that data to determine what intents and entities are found.

Prediction key

The [prediction key](#) is the key associated with the LUIS service you created in Azure that authorizes your usage of the prediction endpoint.

This key is not the authoring key. If you have a prediction endpoint key, it should be used for any endpoint requests instead of the authoring key. You can see your current prediction key inside the endpoint URL at the bottom of Azure resources page in LUIS website. It is the value of the subscription-key name/value pair.

Prediction resource

Your LUIS prediction resource is a manageable item that is available through Azure. The resource is your access to the associated prediction of the Azure service. The resource includes predictions.

The prediction resource has an Azure "kind" of [LUIS](#).

Prediction score

The [score](#) is a number from 0 and 1 that is a measure of how confident the system is that a particular input utterance matches a particular intent. A score closer to 1 means the system is very confident about its output and a score closer to 0 means the system is confident that the input does not match a particular output. Scores in the middle mean the system is very unsure of how to make the decision.

For example, take a model that is used to identify if some customer text includes a food order. It might give a score of 1 for "I'd like to order one coffee" (the system is very confident that this is an order) and a score of 0 for "my team won the game last night" (the system is very confident that this is NOT an order). And it might have a score of 0.5 for "let's have some tea" (isn't sure if this is an order or not).

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).

Schema

Your schema includes your intents and entities along with the subentities. The schema is initially planned for then iterated over time. The schema doesn't include app settings, features, or example utterances.

Sentiment Analysis

Sentiment analysis provides positive or negative values of the utterances provided by the [Language service](#).

Speech priming

Speech priming improves the recognition of spoken words and phrases that are commonly used in your scenario with [Speech Services](#). For speech priming enabled applications, all LUIS labeled examples are used to improve speech recognition accuracy by creating a customized speech model for this specific application. For example, in a chess game you want to make sure that when the user says "Move knight", it isn't interpreted as "Move night". The LUIS app should include examples in which "knight" is labeled as an entity.

Starter key

A free key to use when first starting out using LUIS.

Synonyms

In LUIS [list entities](#), you can create a normalized value, which can each have a list of synonyms. For example, if you create a size entity that has normalized values of small, medium, large, and extra-large. You could create synonyms for each value like this:

Normalized value	Synonyms
Small	the little one, 8 ounce
Medium	regular, 12 ounce
Large	big, 16 ounce
Xtra large	the biggest one, 24 ounce

The model will return the normalized value for the entity when any of synonyms are seen in the input.

Test

[Testing](#) a LUIS app means viewing model predictions.

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 of text that LUIS can recognize. This differs slightly across languages.

For **English**, a token is a continuous span (no spaces or punctuation) of letters and numbers. A space is NOT a token.

Phrase	Token count	Explanation
Dog	1	A single word with no punctuation or spaces.

Phrase	Token count	Explanation
RMT33W	1	A record locator number. It may have numbers and letters, but does not have any punctuation.
425-555-5555	5	A phone number. Each punctuation mark is a single token so 425-555-5555 would be 5 tokens: 425 - 555 - 5555
https://luis.ai	7	https : / / luis .ai

Train

[Training](#) is the process of teaching LUIS about any changes to the active version since the last training.

Training data

Training data is the set of information that is needed to train a model. This includes the schema, labeled utterances, features, and application settings.

Training errors

Training errors are predictions on your training data that do not match their labels.

Utterance

An [utterance](#) is user input that is short text representative of a sentence in a conversation. It is a natural language phrase such as "book 2 tickets to Seattle next Tuesday". Example utterances are added to train the model and the model predicts on new utterance at runtime

Version

A LUIS [version](#) is a specific instance of a LUIS application associated with a LUIS app ID and the published endpoint. Every LUIS app has at least one version.