GenAI Tool: Data Generator

Connected System Plugin for **Appian**

Appian Corporation

Version 1.0.0

Table of Contents

Overview

The GenAI Tool: Data Generator Connected System allows developers to generate sample data of an inputted Record Type with generative AI. This tool generates data that adheres to the structures of 1:1, 1:N, and N:1 relationships by generating the relational data alongside the primary data. This data is primarily for the use of demonstrating the functionality of an application with life-like data that fits into a custom database design.

Developers can generate sample records through Appian with this connected system by entering the credentials retrieved from either OpenAI or Azure OpenAI Studio. This documentation outlines the process of obtaining and leveraging these credentials within the Appian platform. The documentation also gives a step by step tutorial on how to set up the Sample App which can be downloaded from the Appian AppMarket with the connected system for this tool.

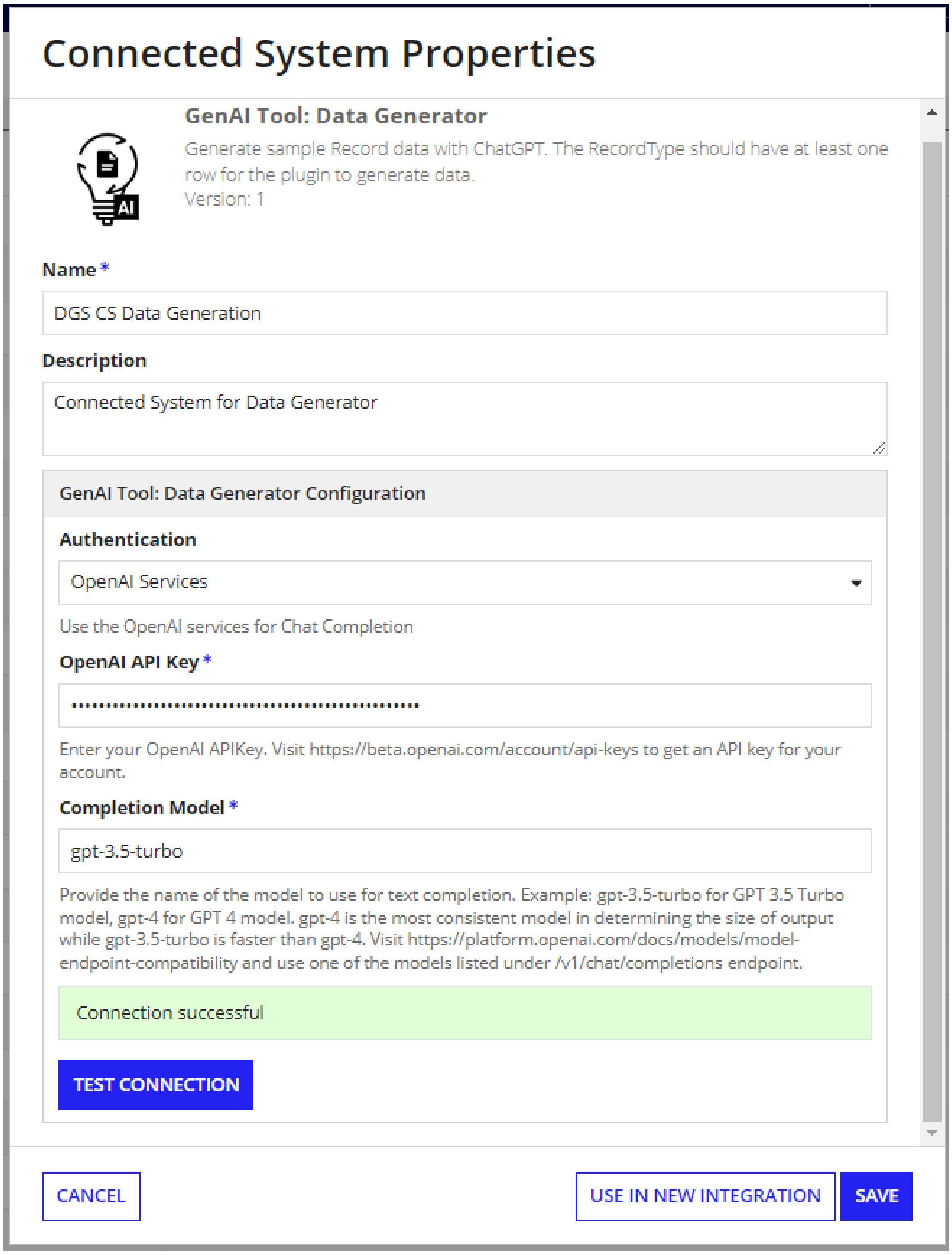
Privacy Policy

All information passed through AI tools will be processed and may remain with the organizations that develop those tools. Please exercise caution with what information is disclosed to the AI tool for this reason.

Features

- Generate sample data for each field of a specified Record Type
- Builds data for 1:1, 1:N, and N:1 Record Type relationships

Chat Completion Model: OpenAI



Connected System Properties

GenAI Tool: Data Generator

Generate sample Record data with ChatGPT. The RecordType should have at least one row for the plugin to generate data.

Version: 1

Name *

DGS CS Data Generation

Description

Connected System for Data Generator

GenAI Tool: Data Generator Configuration

Authentication

OpenAI Services

Use the OpenAI services for Chat Completion

OpenAI API Key *

••

Enter your OpenAI APIKey. Visit https://beta.openai.com/account/api-keys to get an API key for your account.

Completion Model *

gpt-3.5-turbo

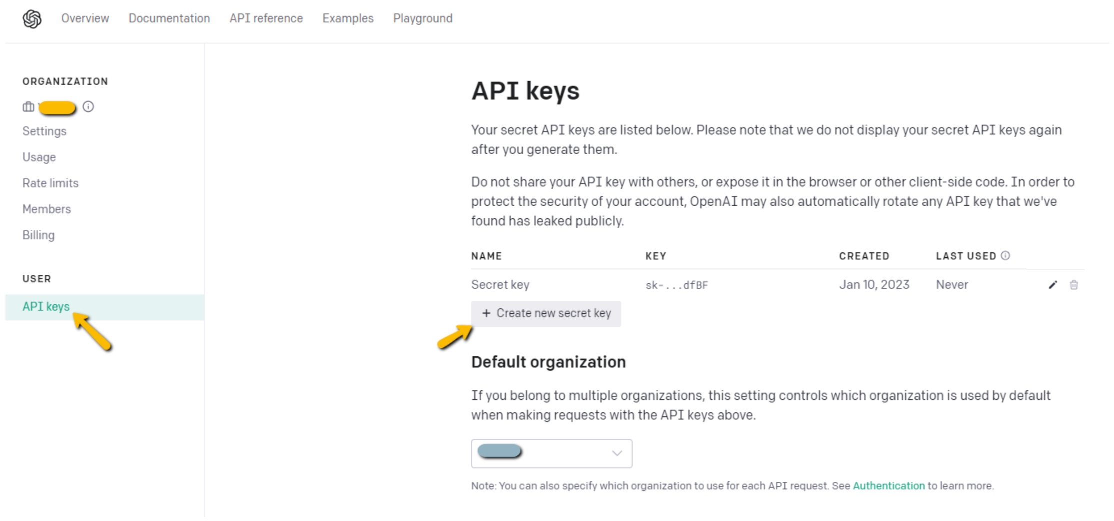
Provide the name of the model to use for text completion. Example: gpt-3.5-turbo for GPT 3.5 Turbo model, gpt-4 for GPT 4 model. gpt-4 is the most consistent model in determining the size of output while gpt-3.5-turbo is faster than gpt-4. Visit https://platform.openai.com/docs/models/model-endpoint-compatibility and use one of the models listed under /v1/chat/completions endpoint.

Connection successful

The Document Summarization Connected system with OpenAI authentication requires the following credentials: **OpenAI API Key** and **Chat Completion model**.

To retrieve your OpenAI API Key:

1. Go to the OpenAI console. Make sure that the **API keys** menu is selected.



[redacted]

API keys

Your secret API keys are listed below. Please note that we do not display your secret API keys again after you generate them.

Do not share your API key with others, or expose it in the browser or other client-side code. In order to protect the security of your account, OpenAI may also automatically rotate any API key that we've found has leaked publicly.

NAME	KEY	CREATED	LAST USED
Secret key	sk-...dfBF	Jan 10, 2023	Never

+ Create new secret key

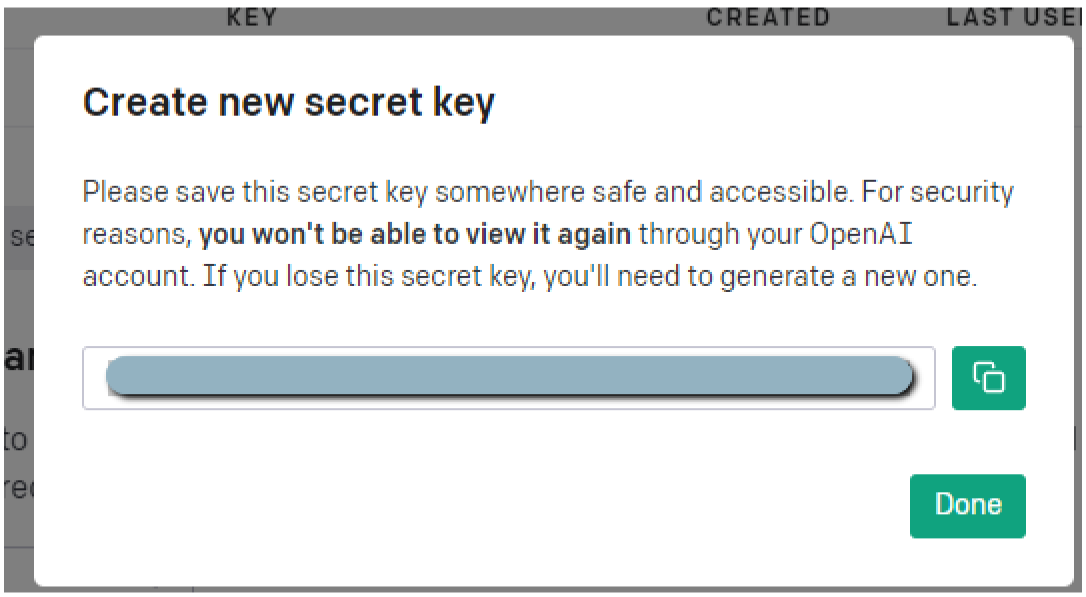
Default organization

If you belong to multiple organizations, this setting controls which organization is used by default when making requests with the API keys above.

[redacted]

Note: You can also specify which organization to use for each API request. See Authentication to learn more.

2. Click on **Create new secret key** to generate a new API key.
3. Copy the value and save it separately as we won't be able to access it again. Paste the API key in the connected system dialog box.



Create new secret key

Please save this secret key somewhere safe and accessible. For security reasons, **you won't be able to view it again** through your OpenAI account. If you lose this secret key, you'll need to generate a new one.

[redacted]

Done

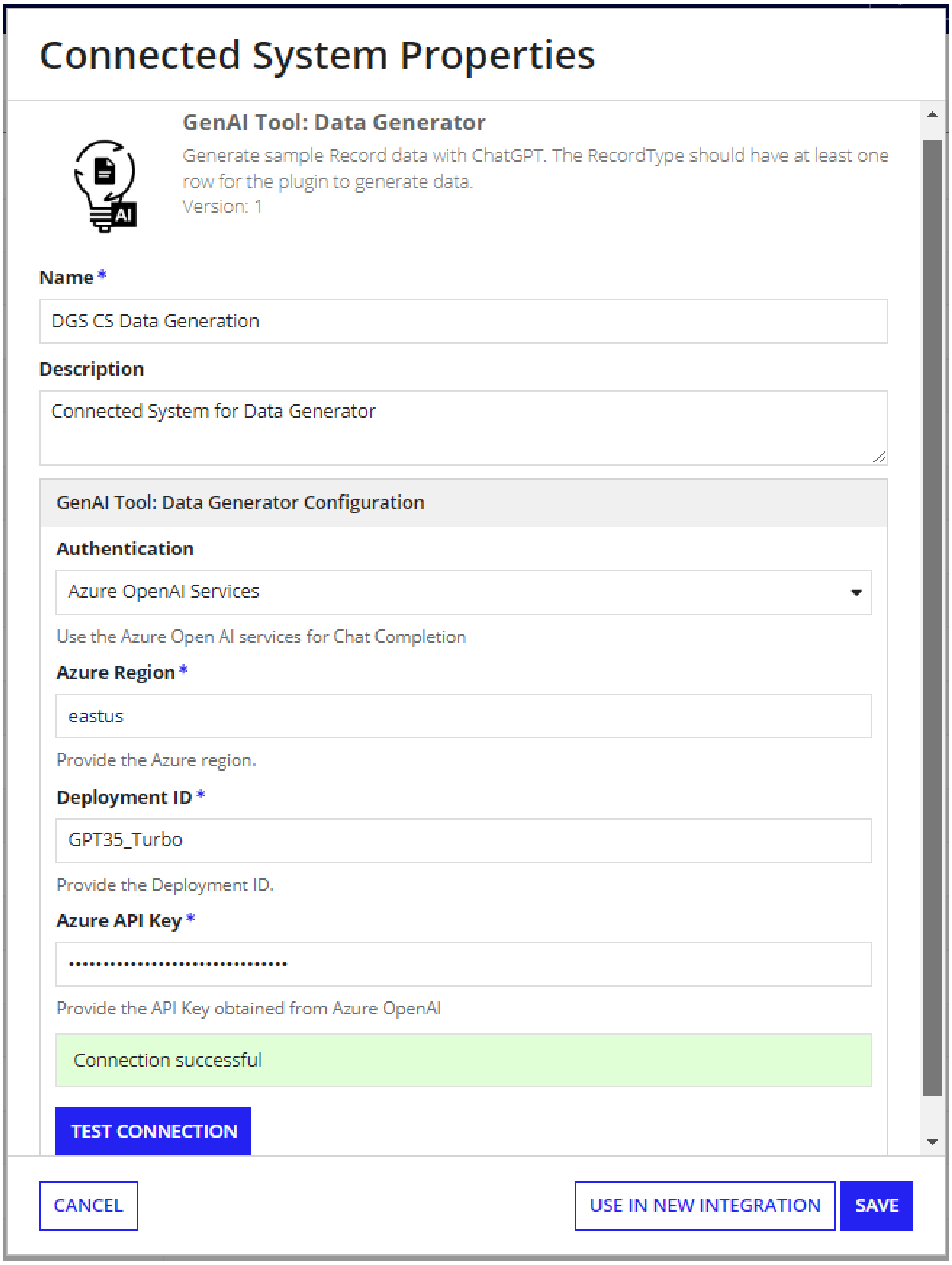
To find the appropriate Chat Completions model:

1. Visit https://platform.openai.com/docs/models/model-endpoint-compatibility and use one of the models listed under /v1/chat/completions endpoint. Example: gpt-3.5-turbo

for GPT 3.5 Turbo model, gpt-4 for GPT 4 model.

a. Each model has unique strengths so try to select the most appropriate for your use. If you would like to prioritize consistency in the size and format of your generated summary, we recommend you use a GPT 4 model. If you need to prioritize speed of generation, GPT 3.5 Turbo might be better suited.

Chat Completion Model: Azure OpenAI



Connected System Properties

GenAI Tool: Data Generator

Generate sample Record data with ChatGPT. The RecordType should have at least one row for the plugin to generate data.

Version: 1

Name *

DGS CS Data Generation

Description

Connected System for Data Generator

GenAI Tool: Data Generator Configuration

Authentication

Azure OpenAI Services

Use the Azure Open AI services for Chat Completion

Azure Region *

eastus

Provide the Azure region.

Deployment ID *

GPT35_Turbo

Provide the Deployment ID.

Azure API Key *

••••••••••••••••••••••••••••••••

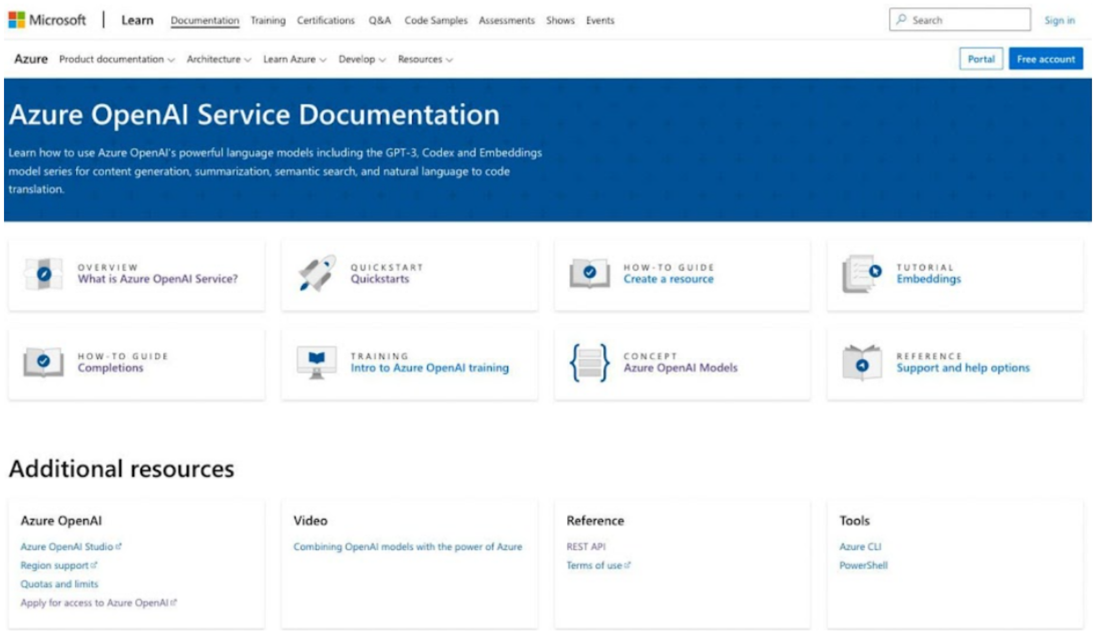
Provide the API Key obtained from Azure OpenAI

Connection successful

This authentication requires the following credentials: Azure Region, Azure Deployment ID and Azure API Key. Follow these steps to get the Azure credentials.

Set up your Azure OpenAI Account

1. Navigate to Azure's OpenAI API docs and ensure you have met the listed prerequisites. View the prerequisites by selecting "Quickstarts." If you have not already done so, create an Azure subscription.


Microsoft | Learn
Documentation Training Certifications Q&A Code Samples Assessments Shows Events
Search
Sign in
Azure Product documentation Architecture Learn Azure Develop Resources
Portal
Free account

Azure OpenAI Service Documentation

Learn how to use Azure OpenAI's powerful language models including the GPT-3, Codex and Embeddings model series for content generation, summarization, semantic search, and natural language to code translation.

OVERVIEW
What is Azure OpenAI Service?

QUICKSTART
Quickstarts

HOW-TO GUIDE
Create a resource

TUTORIAL
Embeddings

HOW-TO GUIDE
Completions

TRAINING
Intro to Azure OpenAI training

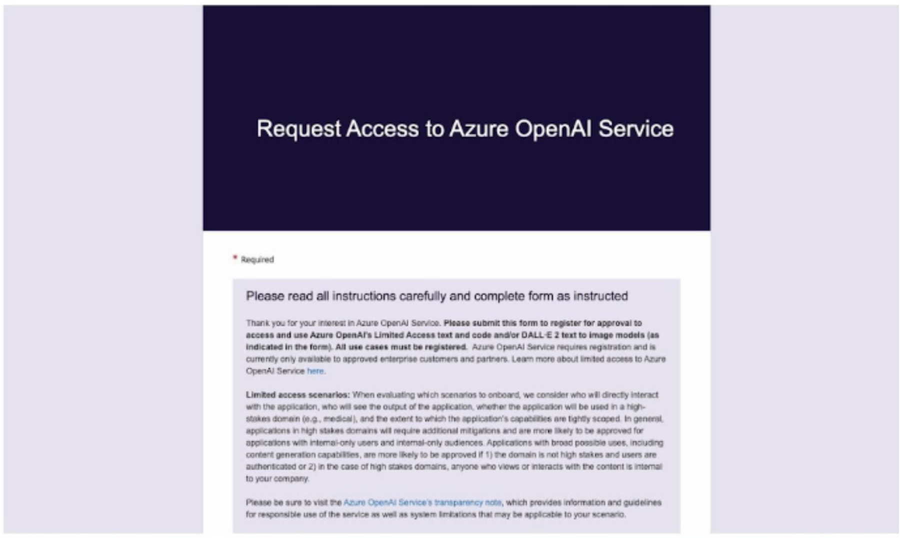
CONCEPT
Azure OpenAI Models

REFERENCE
Support and help options

Additional resources

Azure OpenAI	Video	Reference	Tools
Azure OpenAI Studio	Combining OpenAI models with the power of Azure	REST API	Azure CLI
Region support		Terms of use	PowerShell
Quotas and limits			
Apply for access to Azure OpenAI			

2. Apply for access to Azure OpenAI services by completing the form here. You will need your subscription ID from the previous step.



Request Access to Azure OpenAI Service

* Required

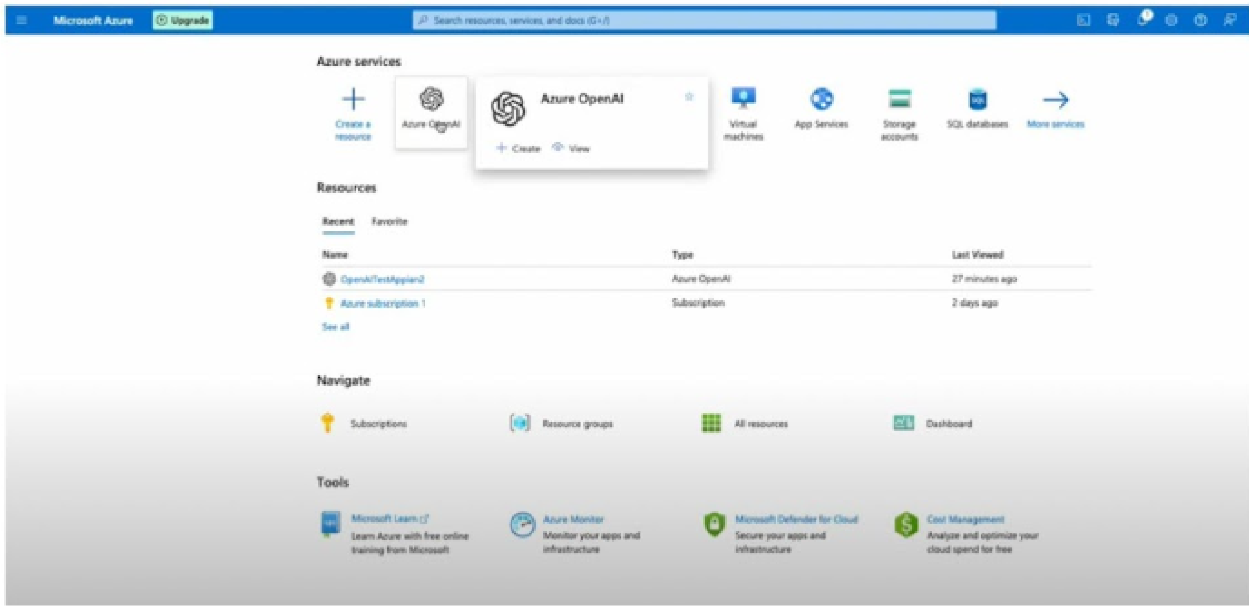
Please read all instructions carefully and complete form as instructed

Thank you for your interest in Azure OpenAI Service. **Please submit this form to register for approval to access and use Azure OpenAI's Limited Access text and code and/or DALL-E 2 text to image models (as indicated in the form). All use cases must be registered.** Azure OpenAI Service requires registration and is currently only available to approved enterprise customers and partners. Learn more about limited access to Azure OpenAI Service here.

Limited access scenarios: When evaluating which scenarios to onboard, we consider who will directly interact with the application, who will see the output of the application, whether the application will be used in a high-stakes domain (e.g., medical), and the extent to which the application's capabilities are tightly scoped. In general, applications in high stakes domains will require additional mitigations and are more likely to be approved for applications with internal-only users and internal-only audiences. Applications with broad possible uses, including content generation capabilities, are more likely to be approved if 1) the domain is not high stakes and users are authenticated or 2) in the case of high stakes domains, anyone who views or interacts with the content is internal to your company.

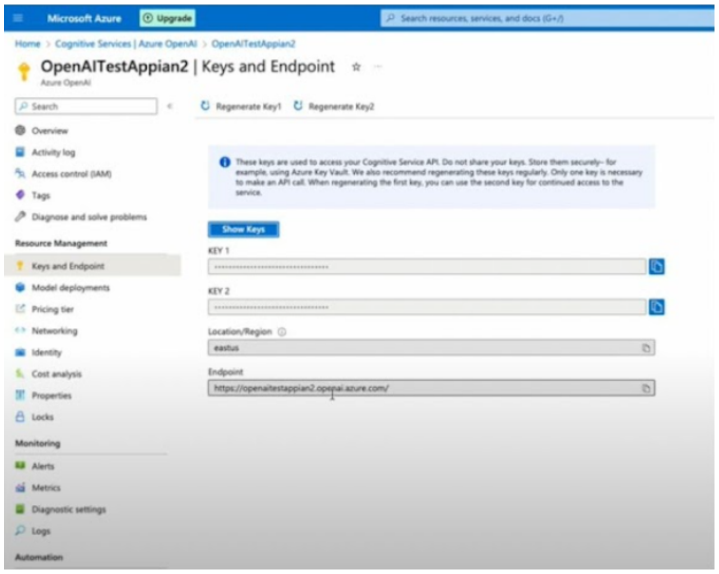
Please be sure to visit the Azure OpenAI Service's transparency note, which provides information and guidelines for responsible use of the service as well as system limitations that may be applicable to your scenario.

3. Create a service and set your domain name.



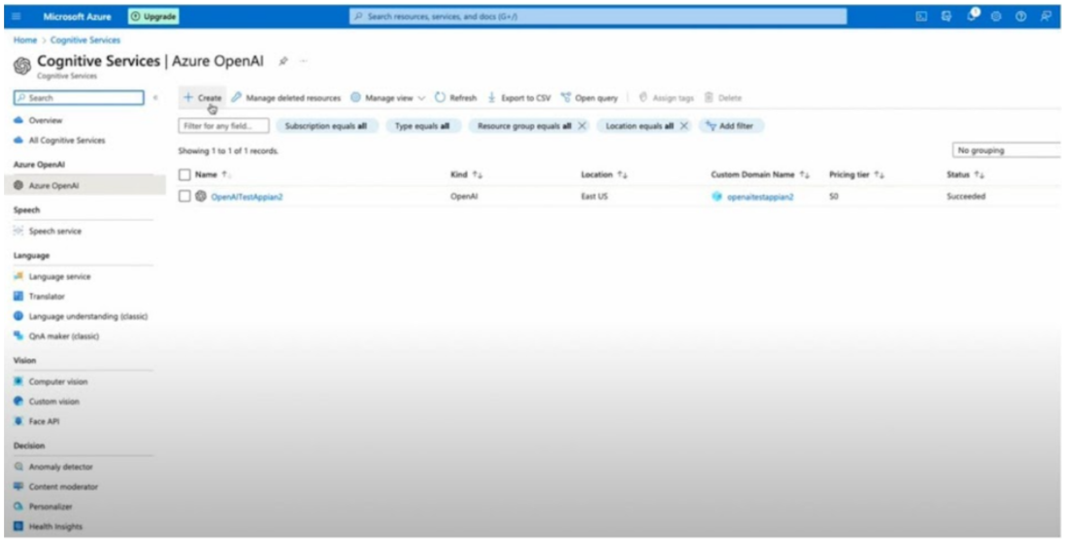
Create and Access API Keys

4. Within your service, create and access API keys through "Keys and Endpoints" under Resource Management. The "Location/Region" listed in this window will be used as your **Azure Region** in the Connected System configuration.



Deploy OpenAI Models and Set your Deployment ID

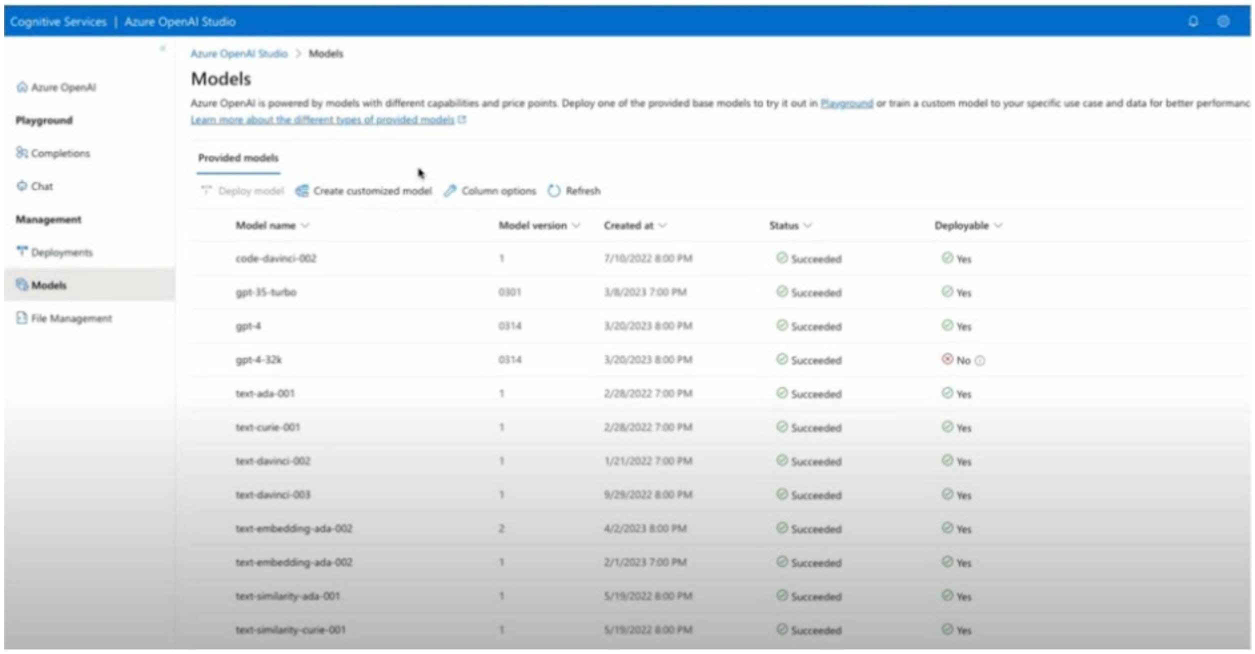
5. Visit the Azure OpenAI Studio to deploy OpenAI models under your created resource.



Showing 1 to 1 of 1 records.

Name	Kind	Location	Custom Domain Name	Pricing tier	Status
OpenAITestAppian2	OpenAI	East US	openaitestappian2	S0	Succeeded

6. Click into your resource to enter the Azure OpenAI Studio. Navigate to “Models” seen as a tab on the left side of the screen. After selecting the best OpenAI Model for your data and use case, deploy your selected chat completions model. The **deployment ID** you create during deployment will be used when configuring your Connected System.



Models

Azure OpenAI is powered by models with different capabilities and price points. Deploy one of the provided base models to try it out in Playground or train a custom model to your specific use case and data for better performanc

Learn more about the different types of provided models

Model name	Model version	Created at	Status	Deployable
code-davinci-002	1	7/10/2022 8:00 PM	Succeeded	Yes
gpt-35-turbo	0301	3/8/2023 7:00 PM	Succeeded	Yes
gpt-4	0314	3/20/2023 8:00 PM	Succeeded	Yes
gpt-4-32k	0314	3/20/2023 8:00 PM	Succeeded	No
text-ada-001	1	2/28/2022 7:00 PM	Succeeded	Yes
text-curie-001	1	2/28/2022 7:00 PM	Succeeded	Yes
text-davinci-002	1	1/21/2022 7:00 PM	Succeeded	Yes
text-davinci-003	1	9/29/2022 8:00 PM	Succeeded	Yes
text-embedding-ada-002	2	4/2/2023 8:00 PM	Succeeded	Yes
text-embedding-ada-002	1	2/1/2023 7:00 PM	Succeeded	Yes
text-similarity-ada-001	1	5/19/2022 8:00 PM	Succeeded	Yes
text-similarity-curie-001	1	5/19/2022 8:00 PM	Succeeded	Yes

Integration

Generate sample records for a Record Type.

ChatGPT Prompt used:
"Create example data for a <relationship> based on the given input. The input consists of an array of JSON objects, each comprising "lastRowData" (representing the last row of a table) and "numberOfRows" (indicating the desired number of generated rows). The resulting JSON array should consist of objects with field names as keys, each containing arrays of sample data. The first object serves as the base table, and subsequent tables are linked through an <relationship>. In cases where tables share field names, ensure they are correlated.To ensure correlation between primary keys in the base and related tables, start data generation for each related table from the next available value after the corresponding primary key in the base table. If tables share field names, ensure that they are correlated. Output the JSON array without introducing newlines within the fields. The generated data should be different from the lastRowData. The fields other than primary key and foreign keys for all the tables should be AI generated. This adjustment aims to guarantee a coherent relationship between primary keys in both the base and related tables. Provide the JSON array without introducing newlines for the following input.The datatype of the fields are:<Fields Info>"

Note: The record type to generate data for needs to have at least one instance of data before using this tool. This is because the Data Generator tool relies on the last row of data in the specified record type's data as a model for the generated data and to ensure the primary key is incremented from the correct value. Additionally, we do not advise generating large amounts of data (more than 45 rows) in a single integration call. To generate more data, we recommend that you make multiple separate calls to the Data Generator tool.

1. One to One Relationship

Inputs:

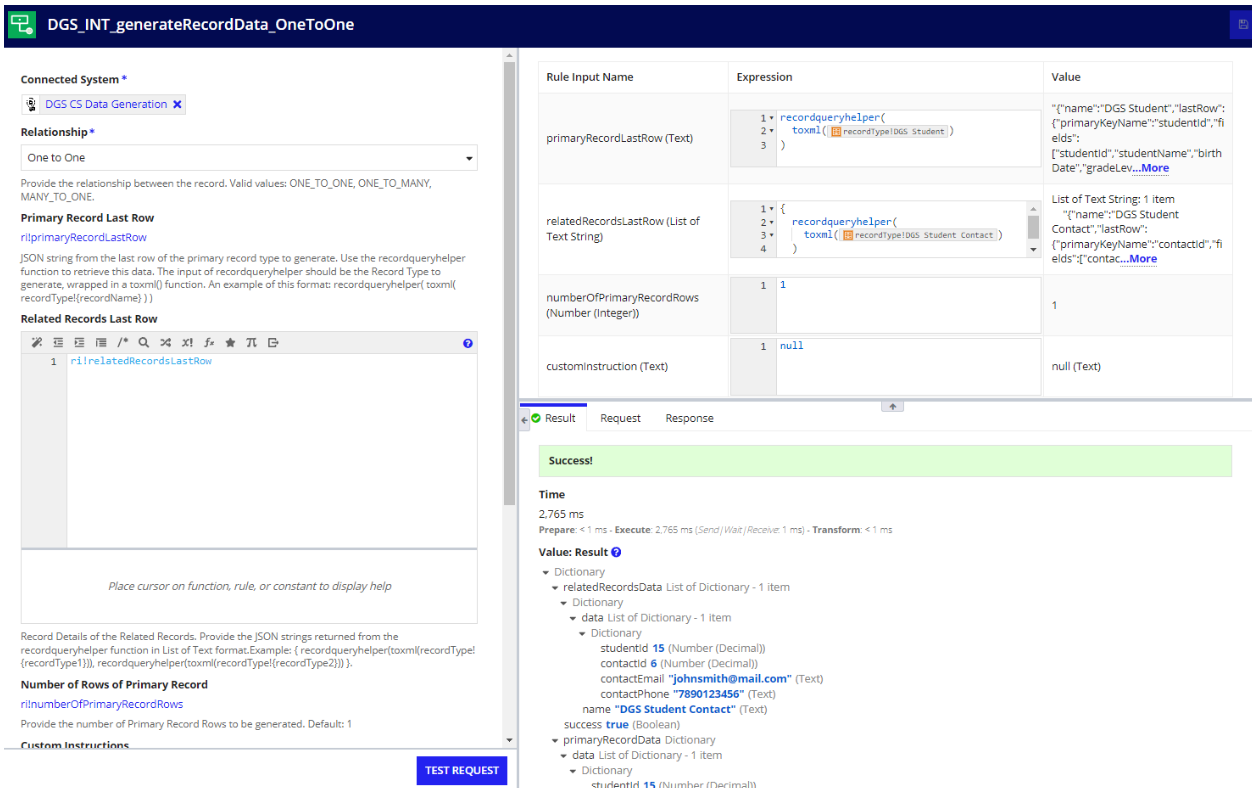
Primary Record Last Row (Text) - Required - JSON string from the last row of the primary record type to generate. Use the recordqueryhelper function to retrieve this data. The input of recordqueryhelper should be the Record Type to generate, wrapped in a toxml() function. An example of this format: recordqueryhelper(toxml(recordType!{recordName}))

Related Records Last Row (List of Text) – Optional - Details of the records related to the primary record. If the primary record generated references another record by a foreign key in a 1 to many relationship, add the related record in a list here. This record input will follow the same format as above where each record must be wrapped in a recordqueryhelper() and a toxml() function. More instructions on this format:

Provide the JSON strings returned from the recordqueryhelper function in List of Text format. Example: { recordqueryhelper(toxml(recordType!{recordType1})), recordqueryhelper(toxml(recordType!{recordType2})) }

<u>Number of Rows of Primary Record</u> (Integer) – Optional - Provide the number of Primary Record Rows to be generated. Default: 1

<u>Custom Instructions</u>(List of Text) – Optional - Provide any additional instructions for the data generation in a list of text. These instructions can be written in natural language. Example: { "all emails should be gmail accounts", "generate any dates in dd-mm-yy format"}



DGS_INT_generateRecordData_OneToOne

Connected System *

DGS CS Data Generation

Relationship *

One to One

Provide the relationship between the record. Valid values: ONE_TO_ONE, ONE_TO_MANY, MANY_TO_ONE.

Primary Record Last Row

ri!primaryRecordLastRow

JSON string from the last row of the primary record type to generate. Use the recordqueryhelper function to retrieve this data. The input of recordqueryhelper should be the Record Type to generate, wrapped in a toxml() function. An example of this format: recordqueryhelper(toxml(recordType!{recordName}))

Related Records Last Row

```
ri!relatedRecordsLastRow
```

Place cursor on function, rule, or constant to display help

Record Details of the Related Records. Provide the JSON strings returned from the recordqueryhelper function in List of Text format.Example: { recordqueryhelper(toxml(recordType!{recordType1})), recordqueryhelper(toxml(recordType!{recordType2})) }.

Number of Rows of Primary Record

ri!numberOfPrimaryRecordRows

Provide the number of Primary Record Rows to be generated. Default: 1

Custom Instructions

TEST REQUEST

Rule Input Name	Expression	Value
primaryRecordLastRow (Text)	`recordqueryhelper(` `toxml( recordType!DGS Student )` `)`	"{"name":"DGS Student","lastRow":{"primaryKeyName":"studentId","fields":["studentId","studentName","birthDate","gradeLev...More
relatedRecordsLastRow (List of Text String)	`{` `recordqueryhelper(` `toxml( recordType!DGS Student Contact )` `)`	List of Text String: 1 item "{"name":"DGS Student Contact","lastRow":{"primaryKeyName":"contactId","fields":["contac...More
numberOfPrimaryRecordRows (Number (Integer))	`1`	1
customInstruction (Text)	`null`	null (Text)

Result Request Response

Success!

Time

2,765 ms

Prepare: < 1 ms - **Execute**: 2,765 ms (*Send / Wait / Receive*: 1 ms) - **Transform**: < 1 ms

Value: Result

```
Dictionary
  relatedRecordsData List of Dictionary - 1 item
    Dictionary
      data List of Dictionary - 1 item
        Dictionary
          studentId 15 (Number (Decimal))
          contactId 6 (Number (Decimal))
          contactEmail "johnsmith@mail.com" (Text)
          contactPhone "7890123456" (Text)
      name "DGS Student Contact" (Text)
  success true (Boolean)
  primaryRecordData Dictionary
    data List of Dictionary - 1 item
      Dictionary
        studentId 15 (Number (Decimal))
```

Output: Dictionary

{relatedRecordsData: {{data: {{studentId: 15, contactId: 6, contactEmail: "emilyjohnson@mail.com", contactPhone: "7890123456"}}, name: "DGS Student Contact"}}, success: true, primaryRecordData: {data: {{studentId: 15, gradeLevel: 9, studentName: "Emily Johnson", birthDate: "2006-05-12"}}, name: "DGS Student"}}

2. One to Many Relationship

Inputs:

Primary Record Last Row (Text) - Required - JSON string from the last row of the primary record type to generate. Use the recordqueryhelper function to retrieve this data. The input of recordqueryhelper should be the Record Type to generate, wrapped in a toxml() function. An example of this format: recordqueryhelper(toxml(recordType!{recordName}))

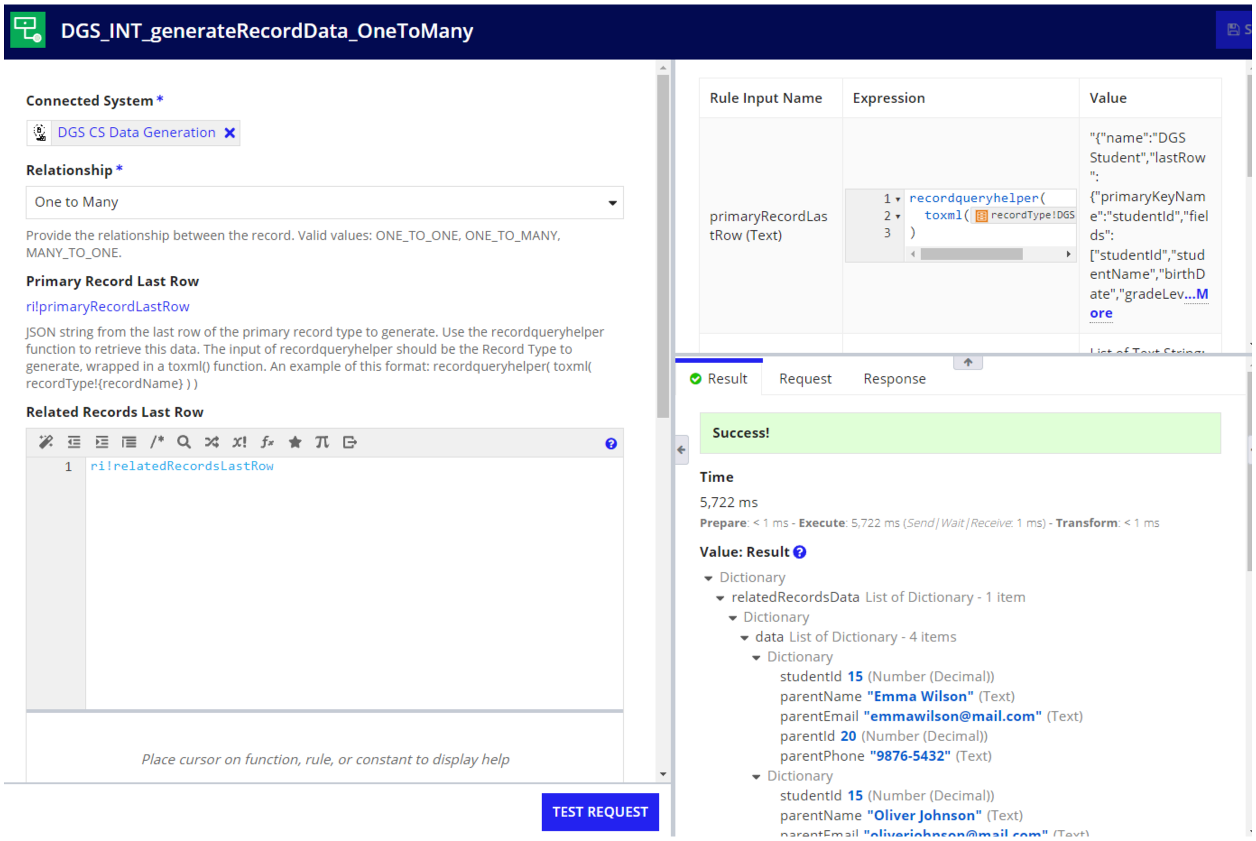
Related Records Last Row (List of Text) – Optional - Details of the records related to the primary record. If the primary record generated references another record by a foreign key in a 1 to many relationship, add the related record in a list here. This record input will follow the same format as above where each record must be wrapped in a recordqueryhelper() and a toxml() function. More instructions on this format:

Provide the JSON strings returned from the recordqueryhelper function in List of Text format. Example: { recordqueryhelper(toxml(recordType!{recordType1})), recordqueryhelper(toxml(recordType!{recordType2})) }

Number of Rows of Primary Record (Integer) – Optional - Provide the number of Primary Record Rows to be generated. Default: 1

Rows per Primary Record (Integer) – Optional - Provide the number of related records to generate for each primary record generated. For example, assume your primary record is a Student record type and your related record type is Parent. If you want each student record to reference two parent records, input 2 here. Default: 1

Custom Instructions(List of Text) – Optional - Provide any additional instructions for the data generation in a list of text. These instructions can be written in natural language. Example: { “all emails should be gmail accounts”, "generate any dates in dd-mm-yy format"}


DGS_INT_generateRecordData_OneToMany

Connected System *

DGS CS Data Generation

Relationship *

One to Many

Provide the relationship between the record. Valid values: ONE_TO_ONE, ONE_TO_MANY, MANY_TO_ONE.

Primary Record Last Row

ri!primaryRecordLastRow

JSON string from the last row of the primary record type to generate. Use the recordqueryhelper function to retrieve this data. The input of recordqueryhelper should be the Record Type to generate, wrapped in a toxml() function. An example of this format: recordqueryhelper(toxml(recordType!{recordName}))

Related Records Last Row

```
ri!relatedRecordsLastRow
```

Place cursor on function, rule, or constant to display help

TEST REQUEST

Rule Input Name	Expression	Value
primaryRecordLastRow (Text)	recordqueryhelper(toxml(recordType!DGS)	"{"name":"DGS Student","lastRow": {"primaryKeyName":"studentId","fields": ["studentId","studentName","birthDate","gradeLev...More

Result | Request | Response

Success!

Time

5,722 ms

Prepare: < 1 ms - Execute: 5,722 ms (Send/Wait/Receive: 1 ms) - Transform: < 1 ms

Value: Result

```
Dictionary
  relatedRecordsData List of Dictionary - 1 item
    Dictionary
      data List of Dictionary - 4 items
        Dictionary
          studentId 15 (Number (Decimal))
          parentName "Emma Wilson" (Text)
          parentEmail "emmawilson@mail.com" (Text)
          parentId 20 (Number (Decimal))
          parentPhone "9876-5432" (Text)
        Dictionary
          studentId 15 (Number (Decimal))
          parentName "Oliver Johnson" (Text)
```

Output: Dictionary

{relatedRecordsData: {{data: {{studentId: 15, parentName: "John Thompson", parentEmail: "johnthompson@mail.com", parentId: 20, parentPhone: "5678-9012"}, {studentId: 15, parentName: "Sarah Johnson", parentEmail: "sarahjohnson@mail.com", parentId: 21, parentPhone: "3456-7890"}, {studentId: 16, parentName: "Michael Smith", parentEmail: "michaelsmith@mail.com", parentId: 22, parentPhone: "6789-0123"}, {studentId: 16, parentName: "Jessica Wilson", parentEmail: "jessicawilson@mail.com", parentId: 23, parentPhone: "9012-3456"}}, name: "DGS Parent"}}, success: true, primaryRecordData: {data: {{studentId: 15, gradeLevel: 10, studentName: "Isabella Thompson", birthDate: "2006-05-20"}, {studentId: 16, gradeLevel: 11, studentName: "Oliver Johnson", birthDate: "2005-09-13"}}, name: "DGS Student"}}

3. Many to One relationship

Inputs:

Primary Record Last Row (Text) - Required - JSON string from the last row of the primary record type to generate. Use the recordqueryhelper function to retrieve this data. The input of recordqueryhelper should be the Record Type to generate, wrapped in a toxml() function. An example of this format: recordqueryhelper(toxml(recordType!{recordName}))

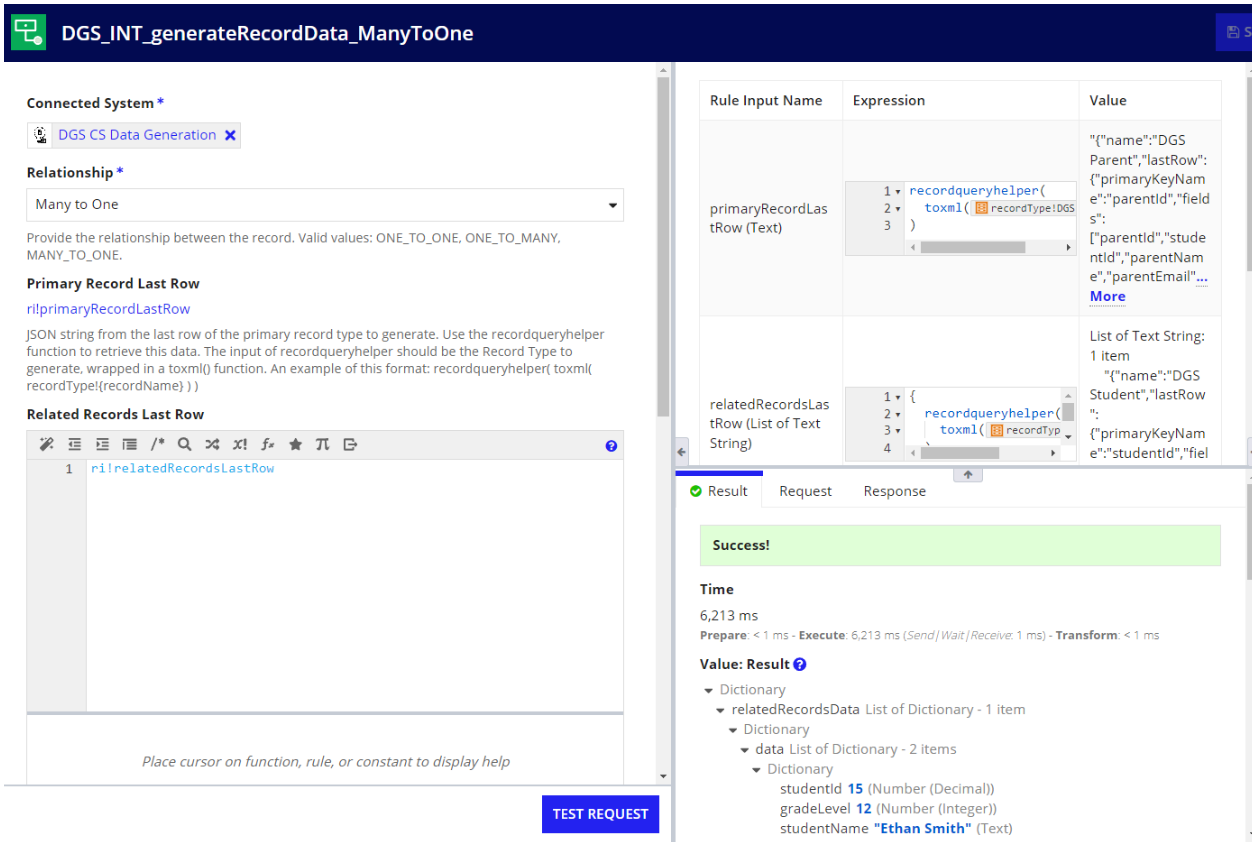
Related Records Last Row (List of Text) – Optional - Details of the records related to the primary record. If the primary record generated references another record by a foreign key in a 1 to many relationship, add the related record in a list here. This record input will follow the same format as above where each record must be wrapped in a recordqueryhelper() and a toxml() function. More instructions on this format:

Provide the JSON strings returned from the recordqueryhelper function in List of Text format. Example: { recordqueryhelper(toxml(recordType!{recordType1})), recordqueryhelper(toxml(recordType!{recordType2})) }

Number of Rows of Primary Record (Integer) – Optional - Provide the number of Primary Record Rows to be generated. Default: 1

Number of Primary Record Rows per Related Record (Integer) – Optional - Provide the number of primary record rows per related record generated. This number will define the "many" in the many to one relationship being generated. For example, if you would like to create student and parent records in a 3:1 relationship, where every every three students reference the same parent, input 3 in this field. Default: 1

Custom Instructions(List of Text) – Optional - Provide any additional instructions for the data generation in a list of text. These instructions can be written in natural language. Example: { "all emails should be gmail accounts", "generate any dates in dd-mm-yy format"}


DGS_INT_generateRecordData_ManyToOne

Connected System *

DGS CS Data Generation

Relationship *

Many to One

Provide the relationship between the record. Valid values: ONE_TO_ONE, ONE_TO_MANY, MANY_TO_ONE.

Primary Record Last Row

ri!primaryRecordLastRow

JSON string from the last row of the primary record type to generate. Use the recordqueryhelper function to retrieve this data. The input of recordqueryhelper should be the Record Type to generate, wrapped in a toxml() function. An example of this format: recordqueryhelper(toxml(recordType!{recordName}))

Related Records Last Row

```
ri!relatedRecordsLastRow
```

Place cursor on function, rule, or constant to display help

TEST REQUEST

Rule Input Name	Expression	Value
primaryRecordLastRow (Text)	recordqueryhelper(toxml(recordType!DGS)	"{"name":"DGS Parent","lastRow":{"primaryKeyName":"parentId","fields":["parentId","studentId","parentName","parentEmail"... More

relatedRecordsLastRow (List of Text String)	{
	  recordqueryhelper(
	    toxml(recordTyp
		List of Text String: 1 item
		"{"name":"DGS Student","lastRow":{"primaryKeyName":"studentId","fiel

Result Request Response

Success!

Time

6,213 ms

Prepare: < 1 ms - **Execute**: 6,213 ms (*Send/Wait/Receive*: 1 ms) - **Transform**: < 1 ms

Value: Result

```
▾ Dictionary
  ▾ relatedRecordsData List of Dictionary - 1 item
    ▾ Dictionary
      ▾ data List of Dictionary - 2 items
        ▾ Dictionary
            studentId 15 (Number (Decimal))
            gradeLevel 12 (Number (Integer))
            studentName "Ethan Smith" (Text)
```

Output: Dictionary

{relatedRecordsData: {{data: {{studentId: 17, gradeLevel: 10, studentName: "Emily Wilson", birthDate: "2004-07-17"}, {studentId: 18, gradeLevel: 11, studentName: "Michael Anderson", birthDate: "2003-09-08"}}, name: "DGS Student"}}, success: true, primaryRecordData: {data: {{studentId: 17, parentName: "Olivia Smith", parentEmail: "oliviasmith@mail.com", parentId: 24, parentPhone: "2734-9502"}, {studentId: 17, parentName: "Emma Johnson", parentEmail: "emmajohnson@mail.com", parentId: 25, parentPhone: "3847-2034"}, {studentId: 17, parentName: "Ava Davis", parentEmail: "avadavis@mail.com", parentId: 26, parentPhone: "1023-8394"}, {studentId: 18, parentName: "Sophia Williams", parentEmail: "sophiawilliams@mail.com", parentId: 27, parentPhone: "3947-5832"}, {studentId: 18, parentName: "Charlotte Brown", parentEmail: "charlottebrown@mail.com", parentId: 28, parentPhone: "2039-4958"}, {studentId: 18, parentName: "Amelia Garcia", parentEmail: "ameliagarcia@mail.com", parentId: 29, parentPhone: "5839-1039"}}, name: "DGS Parent"}}

Data Generator Sample App Setup

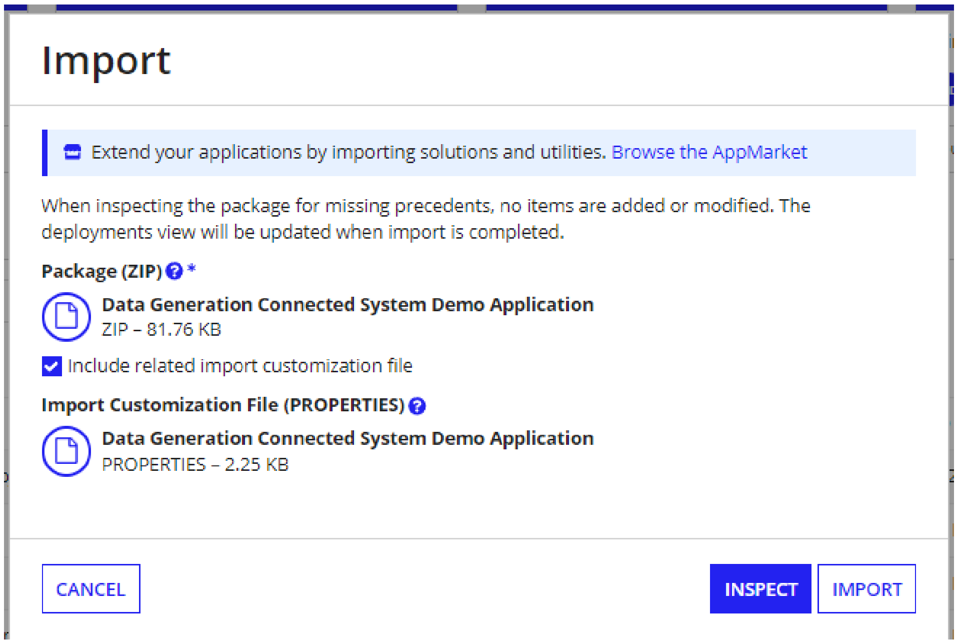
To configure the sample app to begin experimenting with the GenAI Tool: Data Generator, complete the following steps.

1. **Download the necessary files:** Find the Data Generator Connected System from Appian's App Market. Select download and open the zip file to access the Sample Application (.zip), an SQL script (.sql), a Sample App Properties file (.properties), and the plug-in jar file (.jar).
2. **Import the sample data:** In your Appian Cloud Database, select the database where you would like to create a new table for the sample record data. Select Import and "Choose File." Select the SQL file from the downloaded package and hit the "Import" button.
3. **Configure the properties file:** In the properties file downloaded from the AppMarket, add your API key to either the OpenAI or Azure OpenAI API key variable, depending on your preference and your models available.

```
## Connected System: DGS CS Data Generation OpenAI
connectedSystem._a-0000eaac-2ee0-8000-62ea-01ef9001ef90_4507668.openaiApiKey=<--Provide your OpenAI API key here-->

## Connected System: DGS CS Data Generator Azure
connectedSystem._a-0000eafb-e4d0-8000-62fd-01ef9001ef90_4582114.azureAPIKey=<--Provide your Azure API key here-->
```

4. **Import the sample application:** In your Appian designer, select the import button and upload the downloaded sample app zip file and the configured properties file.



Import

Extend your applications by importing solutions and utilities. Browse the AppMarket

When inspecting the package for missing precedents, no items are added or modified. The deployments view will be updated when import is completed.

Package (ZIP) *

Data Generation Connected System Demo Application
ZIP – 81.76 KB

☑ Include related import customization file

Import Customization File (PROPERTIES)

Data Generation Connected System Demo Application
PROPERTIES – 2.25 KB

5. **Finish configuring sample data:** In each of the three sample records (Parent, Student, and Student Contact), configure the data source by selecting "Change Data

Source" in the top right of each Data Model view. Connect to the table in your database that was created with the SQL script in step #2.

6. **Explore the tool:** You can now freely experiment with the Data Generation tool from the sample application. Each relationship available in the integration has been preconfigured in the interface, all you need to do is:
    a. Select the relationship to generate
    b. Enter the number of records to generate
    c. Determine the "Many" in the "One to Many" or "Many to One" relationship, if appropriate
    d. Enter custom instructions if desired, this is optional.
    e. Select "Generate" to view the generated data. Edit the data in the editable grid that appears below to your specifications.
    f. Select "Write to Records" and view the new data in the record view tabs available in the sample app tabs.

Cost Metrics

The below metrics are for a reference on the cost incurred in the Query Documents Integration for the records DGS Parent, DGS Student and DGS Student Contact in a single request. This pricing is consistent between both OpenAI and AzureOpenAI models.

Number of Rows	Tokens used	Cost incurred	
		GPT 3.5 Turbo	GPT 4
5	731	$0.0014	$0.043
10	1044	$0.0020	$0.062
20	1747	$0.0034	$0.105
30	2314	$0.0046	$0.139