GenAI Tool: Data Generator

Connected System Plugin for Appian
Appian Corporation

Version 1.0.0

Table of Contents

Overview 3
Features 3
  Chat Completion Model: OpenAI 4
  Chat Completion Model: Azure OpenAI 7
Integration 11
  1. One to One Relationship 11
  2. One to Many Relationship 12
  3. Many to One relationship 14
Data Generator Sample App Setup 17
Cost Metrics 18
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
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.

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.

To find the appropriate Chat Completions model:

1. Visit [https://platform.openai.com/docs/models/model-endpoint-compatibility](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.
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.

![Azure OpenAI Service Documentation](image1.png)

Additional resources

2. Apply for access to Azure OpenAI services by completing the form [here](image2.png). You will need your subscription ID from the previous step.

![Request Access to Azure OpenAI Service](image3.png)

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.
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.
**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})) }

**Number of Rows of Primary Record** (Integer) – Optional - Provide the number of Primary Record Rows to be generated. 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" }

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" }`
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" }"
Output: Dictionary

```json
{relatedRecordsData: [{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: [{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-000000000000-0000-0000-0000-000000000000.openaiApiKey=-- Provide your OpenAI API key here -->
   # Connected System: DGS CS Data Generator Azure
   connectedSystem = a-000000000000-0000-0000-0000-000000000000.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.

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.

<table>
<thead>
<tr>
<th>Number of Rows</th>
<th>Tokens used</th>
<th>Cost incurred</th>
<th>GPT 3.5 Turbo</th>
<th>GPT 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>GPT 3.5 Turbo</td>
<td>GPT 4</td>
</tr>
<tr>
<td>5</td>
<td>731</td>
<td>$0.0014</td>
<td>$0.043</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1044</td>
<td>$0.0020</td>
<td>$0.062</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1747</td>
<td>$0.0034</td>
<td>$0.105</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>2314</td>
<td>$0.0046</td>
<td>$0.139</td>
<td></td>
</tr>
</tbody>
</table>