

---

# Process Documents with Python Using the Document AI API

---

## ✓ Objectives


- Enable and use the **Cloud Document AI API**
- Create and test a **General Form Parser** processor
- Configure **Vertex AI Workbench** for Document AI API calls
- Perform **synchronous** and **asynchronous** document processing
- Extract **key-value pairs** from form documents
- Use **OCR** processors to extract raw text
- Save and display the structured output

## Task 1. Create and test a general form processor

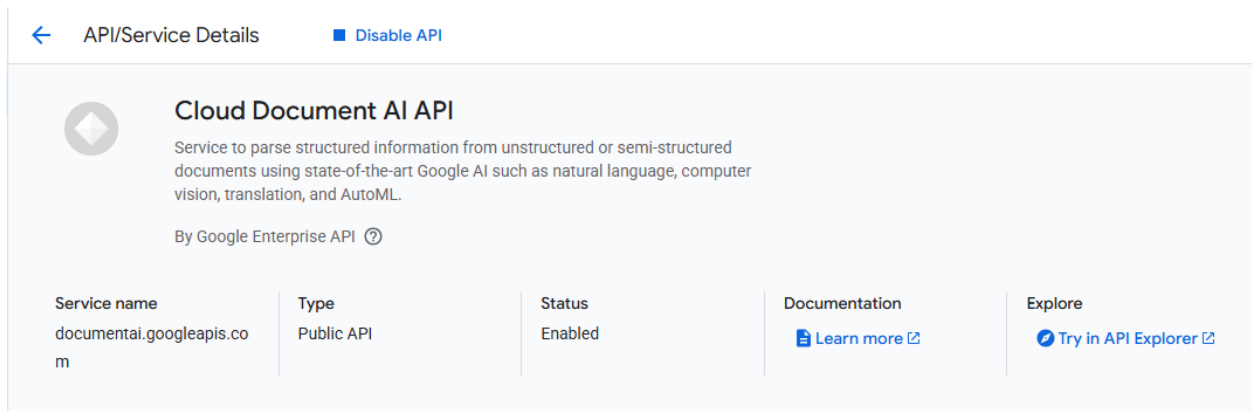
---

Enable the Cloud Document AI API


Before you can begin using Document AI, you must enable the API.

1. In Cloud Console, from the **Navigation menu** () , click **APIs & services > Library**.
2. Search for **Cloud Document AI API**, then click the **Enable** button to use the API in your Google Cloud project.

If the Cloud Document AI API is already enabled you will see the **Manage** button and you can continue with the rest of the lab.




API/Service Details ■ Disable API



### Cloud Document AI API


Service to parse structured information from unstructured or semi-structured documents using state-of-the-art Google AI such as natural language, computer vision, translation, and AutoML.

By Google Enterprise API 

Service name documentai.googleapis.com	Type Public API	Status Enabled	Documentation <a href="#">Learn more</a>	Explore <a href="#">Try in API Explorer</a>
---	--------------------	-------------------	---	--

Create a general form processor

Create a Document AI processor using the Document AI form parser.


1. In the console, on the **Navigation menu** () , click **Document AI > Overview**.
2. Click **Explore processors** and click **Create Processor** for **Form Parser**, which is a type of general processor.
3. Specify the processor name as **form-parser** and select the region **US (United States)** from the list.
4. Click **Create** to create the general form-parser processor.
5. Make a note of the Processor ID as you will need to update variables in JupyterLab notebooks with the Processor ID in later tasks.

---

## Create processor

---

### Form Parser

Extract text and spatial structure from documents, including tabular content through OCR [Learn more](#) 

Processor name \*

form-parser

Must start with a letter. Can use letters, numbers, spaces, dashes, and underscores.

Region

US (United States)



✓ **ADVANCED OPTIONS**

**CREATE**

CANCEL

←
form-parser
○ DISABLE PROCESSOR
☰ ACTIVITY
⋮
🔍

PROCESSOR DETAILS
MANAGE VERSIONS

### Basic information

Name	form-parser
ID	c57417a233dfb9d8
Status	✔ Enabled
Processor Type	Form Parser
Created	Jun 24, 2025, 11:10:43 AM
Encryption Type	Google-managed
Region	us

### Prediction

Prediction endpoint ⓘ <https://us-documentai.googleapis.com/v1/projects/131629384457/locations/us/processors/c57417a233dfb9d8:process> 📄

Test your processor

## Task 2. Configure your Vertex AI Workbench instance to perform Document AI API calls

Next, connect to JupyterLab running on the Vertex AI Workbench instance that was created for you when the lab was started, then configure that environment for the remaining lab tasks.

1. In the Google Cloud console, on the **Navigation menu** (☰), click **Vertex AI > Workbench**.
2. Find the Workbench instance name instance and click on the **Open JupyterLab** button.
3. Click **Terminal** to open a terminal shell inside the Vertex AI Workbench instance.
4. Enter the following command in the terminal shell to import the lab files into your Vertex AI Workbench instance:

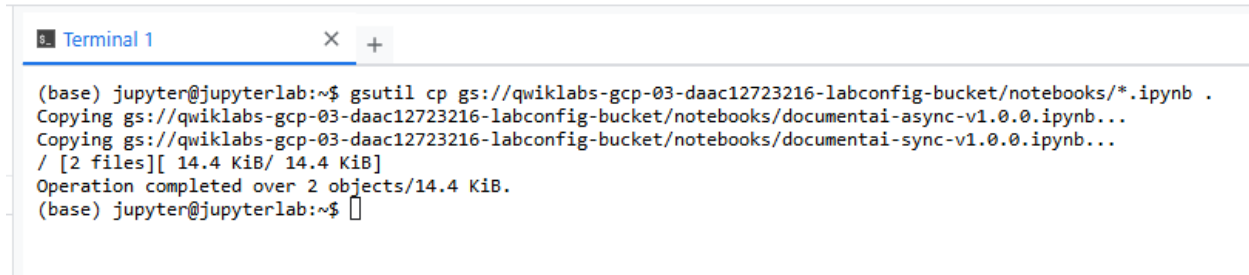
```
gsutil cp notebook_files_path .
```

Terminal 1
×
+

```
(base) jupyter@jupyterlab:~$ gsutil cp gs://qwiklabs-gcp-03-daac12723216-labconfig-bucket/notebooks/*.ipynb .
Copying gs://qwiklabs-gcp-03-daac12723216-labconfig-bucket/notebooks/documentai-async-v1.0.0.ipynb...
Copying gs://qwiklabs-gcp-03-daac12723216-labconfig-bucket/notebooks/documentai-sync-v1.0.0.ipynb...
/ [2 files][ 14.4 KiB/ 14.4 KiB]
Operation completed over 2 objects/14.4 KiB.
(base) jupyter@jupyterlab:~$
```

5. Enter the following command in the terminal shell to install the Python client libraries required for Document AI and other required libraries:

```
python -m pip install --upgrade google-cloud-core google-cloud-documentai google-cloud-storage prettytable
```

A terminal window titled 'Terminal 1' with a close button and a plus sign. The terminal shows the following output:

```
(base) jupyter@jupyterlab:~$ gsutil cp gs://qwiklabs-gcp-03-daac12723216-labconfig-bucket/notebooks/*.ipynb .  
Copying gs://qwiklabs-gcp-03-daac12723216-labconfig-bucket/notebooks/documentai-async-v1.0.0.ipynb...  
Copying gs://qwiklabs-gcp-03-daac12723216-labconfig-bucket/notebooks/documentai-sync-v1.0.0.ipynb...  
/ [2 files][ 14.4 KiB/ 14.4 KiB]  
Operation completed over 2 objects/14.4 KiB.  
(base) jupyter@jupyterlab:~$
```

6. Enter the following command in the terminal shell to import the sample health intake form:

```
gsutil cp form_path form.pdf
```

7. In the notebook interface open the JupyterLab notebook called notebook name.
8. In the **Select Kernel** dialog, choose **Python 3** from the list of available kernels.

### Task 3. Make a synchronous process document request

---

Make a process document call using a synchronous Document AI API call. For processing large amounts of documents at a time you can also use the asynchronous API which you will use in a later task.

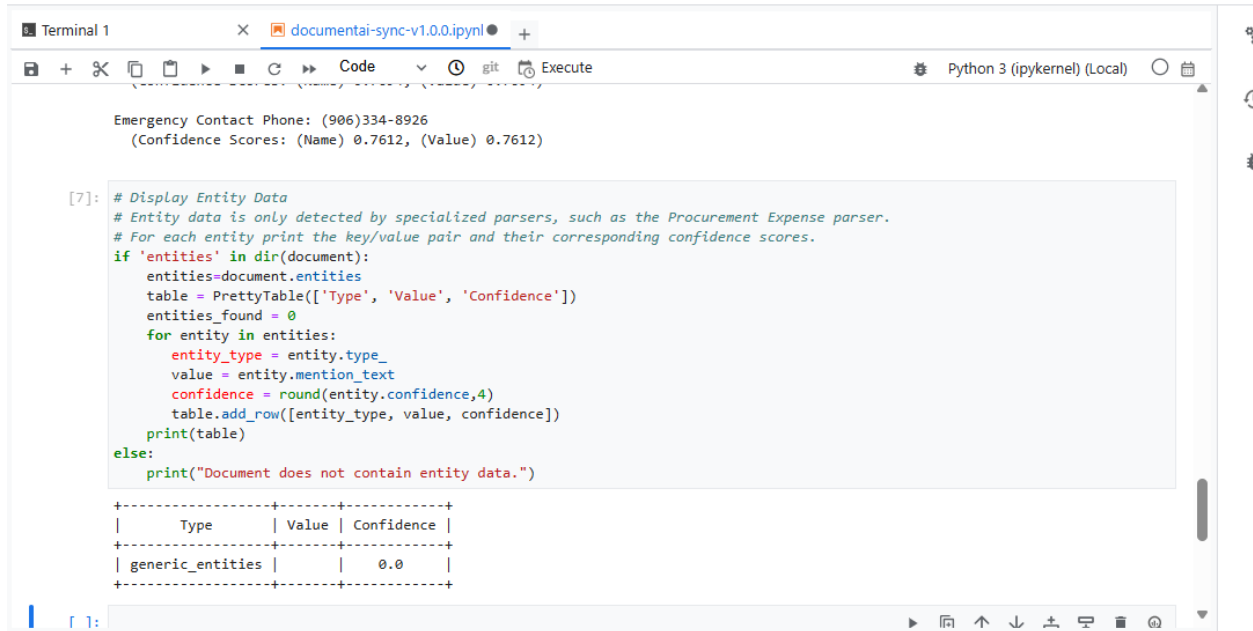
Review the Python code for synchronous Document AI API calls

### Task 4. Run the synchronous Document AI Python code

---

Execute the code to make synchronous calls to the Document AI API in the JupyterLab notebook.

1. In the second **Set your Processor ID** code cell replace the `PROCESSOR_ID` placeholder text with the Processor ID for the **form-parser** processor you created in an earlier step.
2. Select the first cell, click the **Run** menu and then click **Run Selected Cell and All Below** to run all the code in the notebook.



```
Emergency Contact Phone: (906)334-8926
(Confidence Scores: (Name) 0.7612, (Value) 0.7612)

[7]: # Display Entity Data
# Entity data is only detected by specialized parsers, such as the Procurement Expense parser.
# For each entity print the key/value pair and their corresponding confidence scores.
if 'entities' in dir(document):
    entities=document.entities
    table = PrettyTable(['Type', 'Value', 'Confidence'])
    entities_found = 0
    for entity in entities:
        entity_type = entity.type_
        value = entity.mention_text
        confidence = round(entity.confidence,4)
        table.add_row([entity_type, value, confidence])
    print(table)
else:
    print("Document does not contain entity data.")

+-----+-----+-----+
|      Type      | Value | Confidence |
+-----+-----+-----+
| generic_entities |      | 0.0        |
+-----+-----+-----+
```

3. In the JupyterLab menu click **File** and then click **Save Notebook** to save your progress.

**Task 5. Create a Document AI Document OCR processor**

---

In this task you will create a Document AI processor using the general Document OCR parser.

- 1. From the **Navigation menu**, click **Document AI > Overview**.
- 2. Click **Explore Processors** and then click **Create Processor** for **Document OCR**. This is a type of general processor.
- 3. Specify the processor name as **ocr-processor** and select the region **US (United States)** from the list.
- 4. Click **Create** to create your processor.
- 5. Make a note of the **processor ID**. You will need to specify this in a later task.

## Create processor

### Document OCR

Extract text from documents with world-class accuracy, supporting over 200 languages and 50 languages for handwriting recognition [Learn more](#)

Processor name \*

ocr-processor

Must start with a letter. Can use letters, numbers, spaces, dashes, and underscores.

Region

US (United States)

✓ ADVANCED OPTIONS

CREATE

CANCEL

### Task 6. Prepare your environment for asynchronous Document AI API calls

In this task you upload the sample JupyterLab notebook to test asynchronous Document AI API calls and copy some sample forms for the lab to Cloud Storage for asynchronous processing.

1. Click the **Terminal** tab to re-open the terminal shell inside the Vertex AI Workbench instance.
2. Create a Cloud Storage bucket for the input documents and copy the sample W2 forms into the bucket:

```
export PROJECT_ID="$(gcloud config get-value core/project)" export  
BUCKET="${PROJECT_ID}"_doc_ai_async gsutil mb gs://${BUCKET} gsutil -m cp  
async_files_path gs://${BUCKET}/input
```

3. In the notebook interface open the JupyterLab notebook called notebook name.
4. In the **Select Kernel** dialog, choose **Python 3** from the list of available kernels.

```
(base) jupyter@jupyterlab:~$ export PROJECT_ID="$(gcloud config get-value core/project)"
export BUCKET="${PROJECT_ID}"_doc_ai_async
gsutil mb gs://${BUCKET}
gsutil -m cp gs://qwiklabs-gcp-03-daac12723216-labconfig-bucket/async/*. * gs://${BUCKET}/input
Creating gs://qwiklabs-gcp-03-daac12723216-doc_ai_async/...
Copying gs://qwiklabs-gcp-03-daac12723216-labconfig-bucket/async/google_invoice.pdf [Content-Type=application/pdf]...
Copying gs://qwiklabs-gcp-03-daac12723216-labconfig-bucket/async/office-depot-receipt.pdf [Content-Type=application/pdf]...
/ [2/2 files][ 1.5 MiB/ 1.5 MiB] 100% Done
Operation completed over 2 objects/1.5 MiB.
(base) jupyter@jupyterlab:~$
```

## Task 7. Make an asynchronous process document request

Review the Python code for asynchronous Document AI API calls

Take a minute to review the Python code in the notebook name notebook.

Run the asynchronous Document AI Python code

Use the sample code provided for you in the Jupyterlab notebook to process documents asynchronously using a Document AI batch processing request.

1. In the second code cell replace the `PROCESSOR_ID` placeholder text with the Processor ID for the **form-parser** processor you created in an earlier step.
2. Select the first cell, click the **Run** menu and then click **Run Selected Cell and All Below** to run all the code in the notebook.
3. As the code cells execute, you can step through the notebook reviewing the code and the comments that explain how the asynchronous request object is created and used.

The screenshot shows the JupyterLab interface with a notebook titled "Document AI Asynchronous API". The notebook contains three code cells:

```
[1]: # Import Libraries
from google.cloud import documentai_v1beta3 as documentai
from google.cloud import storage
from prettytable import PrettyTable

import re
import os
import pandas as pd

[7]: # Set your Processor ID
processor_id = "c57417a233dfb9d8" # TODO: Replace with a valid Processor ID

[8]: # Set your variables
project_id = %system: gcloud config get-value core/project
```

```
        table.add_row([entity_type, value, confidence])
    print(table)
    else:
        print('No entity data returned by the Document AI processor for file'+blob.name)
    else:
        print(f"Skipping non-supported file type {blob.name}")

Fetched file 1:output/17533071060967412362/0/google_invoice-0.json
+-----+
|      Type      | Value | Confidence |
+-----+
| generic_entities |      | 0.0        |
+-----+
Fetched file 2:output/17533071060967412362/1/office-depot-receipt-0.json
+-----+
|      Type      | Value | Confidence |
+-----+
| generic_entities |      | 0.0        |
+-----+
```

4. In the JupyterLab menu click **File** and then click **Save Notebook** to save your progress.

### Skills Demonstrated

---

- Google Cloud Console navigation
- Vertex AI Workbench and JupyterLab usage
- Cloud Storage setup and file management
- Synchronous vs asynchronous API integration
- Text and form data extraction with confidence scores
- Working with Python and Google Cloud SDK

### Conclusion

---

This project provided practical experience with Google Cloud Document AI, including both **general form** and **OCR processors**. Successfully processed documents both synchronously and asynchronously, extracted structured data, and evaluated the capabilities of each processor type.