Process Documents with Python Using the Document AI API

**Cloud Document AI API**

1 hourFree

**GSP925**



**Overview**

The Document AI API is a document understanding solution that takes unstructured data, such as documents and emails, and makes the data easier to understand, analyze, and consume.

In this lab, you will use the Document AI API with Python to create various processors, including a general form processor and a Document OCR processor, then make synchronous and asynchronous calls to the API using Python. This lab creates a Vertex AI Notebooks instance for you that you will use with JupyterLab notebooks to work with the Document AI Python Client modules.

**Objectives**

In this lab you will learn how to:

* Enable the Document AI API and create processors.
* Install the client library for Python in a Vertex AI Notebooks instance.
* Parse data from a scanned form using Python to make a synchronous API call.
* Parse data from scanned forms using Python to make an asynchronous API call.

**Setup and Requirements**

**Before you click the Start Lab button**

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

**What you need**

To complete this lab, you need:

* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

**Note:** If you already have your own personal Google Cloud account or project, do not use it for this lab.

**Note:** If you are using a Chrome OS device, open an Incognito window to run this lab.

Activate Cloud Shell

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.



Click **Continue**.



It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT\_ID*. For example:



gcloud is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

gcloud auth list

Copied!

content\_copy

(Output)

ACTIVE: \*

ACCOUNT: student-01-xxxxxxxxxxxx@qwiklabs.net

To set the active account, run:

$ gcloud config set account `ACCOUNT`

You can list the project ID with this command:

gcloud config list project

Copied!

content\_copy

(Output)

[core]

project = <project\_ID>

(Example output)

[core]

project = qwiklabs-gcp-44776a13dea667a6

For full documentation of gcloud see the [gcloud command-line tool overview](https://cloud.google.com/sdk/gcloud" \t "_blank).

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

In this task you will enable the Document AI API and create and test a general form processor. The general form processor will process any type of document and extract all the text content it can identify in the document. It is not limited to printed text, it can handle handwritten text and text in any orientation, supports a number of languages, and understands how form data elements are related to each other so that you can extract key:value pairs for form fields that have text labels.

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** (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.

Check that the Cloud Document AI API has been enabled.

Check my progress

Create a general form processor

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

1. In the console, on the **Navigation menu** (Navigation menu), click **Document AI > Overview**.
2. Click **Create processor** and select **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.

This will create the processor and return to the processor details page that will display the processor ID, status, and the prediction endpoint.

Make a note of the Processor ID as you will need to update variables in JupyterLab notebooks with the Processor ID in later tasks.

**Task 2. Configure your Vertex AI Notebooks instance to perform Document AI API calls**

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

1. In the Cloud Console, on the **Navigation menu**, click **Vertex AI > Notebooks**.
2. Click **Open Jupyterlab** to open the JupyterLab console on your Vertex AI Notebooks instance.
3. Click **Terminal** to open a terminal shell inside the Vertex AI Notebooks instance.
4. Enter the following command in the terminal shell to import the lab files into your Vertex AI Notebooks instance:

gsutil cp gs://cloud-training/gsp925/\*.ipynb .

Copied!

content\_copy

1. 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

Copied!

content\_copy

You should see output indicating that the libraries have been installed successfully.

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

gsutil cp gs://cloud-training/gsp925/health-intake-form.pdf form.pdf

Copied!

content\_copy

1. In the notebook interface open the JupyterLab notebook called documentai-sync.ipynb.

Check that the Vertex AI instance has been prepared for synchronous Document AI API calls.

Check my progress

**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

Take a minute to review the Python code in the documentai-sync.ipynb notebook.

1. The first code block imports the required libraries and initializes some variables.

from google.cloud import documentai\_v1beta3 as documentai

from google.cloud import storage

from prettytable import PrettyTable

project\_id = %system gcloud config get-value core/project

project\_id = project\_id[0]

location = 'us'

file\_path = 'form.pdf'

1. The **Set your Processor ID** code cell sets the Processor ID that you have to manually set before you can process documents with the notebook.

processor\_id = 'PROCESSOR\_ID' # TODO: Replace with a valid Processor ID

You will need the Document AI processor ID of the processor you created in Task 1 for this step. If you did not save it, then in the Cloud Console tab open the **Navigation menu** (Navigation menu), click **Document AI > Processors**, then click the name of your processor to open the details page. From here you can copy the processor ID.

1. The **Process Document Function** code cell defines the process\_document function that is used to make a synchronous call to a Document AI processor. The function creates a Document AI API client object.

The processor name required by the API call is created using the project\_id,locations, and processor\_id parameters and the sample PDF document is read in and stored in a mime\_type structure.

The function creates a request object that contains the full processor name of the document and uses that object as the parameter for a synchronous call to the Document AI API client. If the request is successful the document object that is returned will include properties that contain the entities detected in the form.

def process\_document(

project\_id=project\_id, location=location,

processor\_id=processor\_id, file\_path=file\_path

):

# Instantiates a client

client = documentai.DocumentProcessorServiceClient()

# The full resource name of the processor, e.g.:

# projects/project-id/locations/location/processor/processor-id

# You must create new processors in the Cloud Console first

name = f"projects/{project\_id}/locations/{location}/processors/{processor\_id}"

with open(file\_path, "rb") as image:

image\_content = image.read()

# Read the file into memory

document = {"content": image\_content, "mime\_type": "application/pdf"}

# Configure the process request

request = {"name": name, "document": document}

# Use the Document AI client to process the sample form

result = client.process\_document(request=request)

return result.document

1. The **Process Document** code cell calls the process\_document function, saves the response in the document variable, and prints the raw text that has been detected. All of the processors will report some data for the document.text property.

document=process\_document()

# print all detected text.

# All document processors will display the text content

print("Document processing complete.")

print("Text: {}".format(document.text))

1. The **Get Text Function** code cell defines the get\_text() function that retrieves the text for a named element using the text\_anchor start\_index and end\_index properties of the named element's text\_segments. This function is used to retrieve the form name and form value for form data if that data is returned by the processor.

def get\_text(doc\_element: dict, document: dict):

"""

Document AI identifies form fields by their offsets

in document text. This function converts offsets

to text snippets.

"""

response = ""

# If a text segment spans several lines, it will

# be stored in different text segments.

for segment in doc\_element.text\_anchor.text\_segments:

start\_index = (

int(segment.start\_index)

if segment in doc\_element.text\_anchor.text\_segments

else 0

)

end\_index = int(segment.end\_index)

response += document.text[start\_index:end\_index]

return response

1. The **Display Form Data** cell iterates over all pages that have been detected and for each form\_field detected it uses the get\_text() function to retrieve the field name and field value. Those values are then printed out, along with their corresponding confidence scores. Form data will be returned by processors that use the general form parser or the specialized parsers but will not be returned by processors that were created with the Document OCR parser.

document\_pages = document.pages

print("Form data detected:\n")

# For each page fetch each form field and display fieldname, value and confidence scores

for page in document\_pages:

print("Page Number:{}".format(page.page\_number))

for form\_field in page.form\_fields:

fieldName=get\_text(form\_field.field\_name,document)

nameConfidence = round(form\_field.field\_name.confidence,4)

fieldValue = get\_text(form\_field.field\_value,document)

valueConfidence = round(form\_field.field\_value.confidence,4)

print(fieldName+fieldValue +" (Confidence Scores: (Name) "+str(nameConfidence)+", (Value) "+str(valueConfidence)+")\n")

1. The **Display Entity Data** cell extracts entity data from the document object and displays the entity type, value, and confidence properties for each entity detected. Entity data is *only* returned by processors that use specialized Document AI parsers such as the Procurement Expense parser. The general form parser and the Document OCR parser will not return entity data.

if 'entities' in dir(document):

entities = document.entities

# Grab each key/value pair and their confidence scores.

table = PrettyTable(['Type', 'Value', 'Confidence'])

for entity in entities:

entity\_type = entity.type\_

value = entity.mention\_text

confience = round(entity.confidence,4)

table.add\_row([entity\_type, value, confience])

print(table)

else:

print("Document does not contain entity data.")

**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.

If you have used the sample health intake form, you will data similar to the following for the output cell for the form data:

Form data detected:

Page Number:1

Phone #: (906) 917-3486

(Confidence Scores: (Name) 1.0, (Value) 1.0)

...

Date:

9/14/19

(Confidence Scores: (Name) 0.9999, (Value) 0.9999)

...

Name:

Sally

Walker

(Confidence Scores: (Name) 0.9973, (Value) 0.9973)

...

If you are able to create a specialised processor the final cell will display entity data, otherwise it will show an empty table.

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

Check that a document has been processed using the synchronous Cloud Document API.

Check my progress

**Task 4. Create a Document AI Document OCR processor**

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

1. In the Cloud Console, on the **Navigation menu**, click **Document AI > Overview**.
2. Click **Create Processor** and then select **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 use need to specify this in a later task.

**Task 5. 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 Notebooks 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 gs://cloud-training/gsp925/async/\*.\* gs://${BUCKET}/input

Copied!

content\_copy

1. In the notebook interface open the JupyterLab notebook called documentai-async.py.

Check that the Vertex AI instance has been prepared for asynchronous Document AI API calls.

Check my progress

**Task 6. 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 documentai-async.py notebook.

1. The first code cell imports the required libraries.

from google.cloud import documentai\_v1beta3 as documentai

from google.cloud import storage

import re

import os

import pandas as pd

import simplejson as json

1. The **Set your Processor ID** code cell sets the Processor ID that you have to manually set before you can process documents with the notebook.

processor\_id = "PROCESSOR\_ID" # TODO: Replace with a valid Processor ID

1. The **Set your variables** code cell defines the parameters that will be used to make the asynchronous call, including the location of the input and output Cloud Storage buckets that will be used for the source data and output files. You will update the placeholder values in this cell for the PROJECT\_ID and the PROCESSOR\_ID in the next section of the lab before you run the code. The other variables contain defaults for the processor location, input Cloud Storage Bucket, and output Cloud Storage bucket that you do not need to change.

project\_id = %system gcloud config get-value core/project

project\_id = project\_id[0]

location = 'us' # Replace with 'eu' if processor does not use 'us' location

gcs\_input\_bucket = project\_id+"\_doc\_ai\_async" # Bucket name only, no gs:// prefix

gcs\_input\_prefix = "input/" # Input bucket folder e.g. input/

gcs\_output\_bucket = project\_id+"\_doc\_ai\_async" # Bucket name only, no gs:// prefix

gcs\_output\_prefix = "output/" # Input bucket folder e.g. output/

timeout = 300

1. The **Define Google Cloud client objects** code cell initializes the Document AI and Cloud Storage clients.

client\_options = {"api\_endpoint": "{}-documentai.googleapis.com".format(location)}

client = documentai.DocumentProcessorServiceClient(client\_options=client\_options)

storage\_client = storage.Client()

1. The **Create input configuration** code cell creates the input configuration array parameter for the source data that will be passed to the asynchronous Document AI request as an input configuration. This array stores the Cloud Storage source location, and the mime type, for each of the files that are found in the input Cloud Storage location.

blobs = storage\_client.list\_blobs(gcs\_input\_bucket, prefix=gcs\_input\_prefix)

input\_configs = []

print("Input Files:")

for blob in blobs:

if ".pdf" in blob.name:

source = "gs://{bucket}/{name}".format(bucket = gcs\_input\_bucket, name = blob.name)

print(source)

input\_config = documentai.types.document\_processor\_service.BatchProcessRequest.BatchInputConfig(

gcs\_source=source, mime\_type="application/pdf"

)

input\_configs.append(input\_config)

1. The **Create output configuration** code cell creates the output parameter for the asynchronous request containing the output Cloud Storage bucket location and stores that as a Document AI batch output configuration.

destination\_uri = f"gs://{gcs\_output\_bucket}/{gcs\_output\_prefix}"

output\_config = documentai.types.document\_processor\_service.BatchProcessRequest.BatchOutputConfig(

gcs\_destination=destination\_uri

)

1. The **Create the Document AI API request** code cell builds the asynchronous Document AI batch process request object using the input and output configuration objects.

name = f"projects/{project\_id}/locations/{location}/processors/{processor\_id}"

request = documentai.types.document\_processor\_service.BatchProcessRequest(

name=name,

input\_configs=input\_configs,

output\_config=output\_config,

)

1. The **Start the batch (asynchronous) API operation** code cell makes an asynchronous document process request by passing the request object to the batch\_process\_documents() method. This is an asynchronous call so you use the result() method to force the notebook to wait until the background asynchronous job has completed.

operation = client.batch\_process\_documents(request)

# Wait for the operation to finish

operation.result(timeout=timeout)

print ("Batch process completed.")

1. The **Fetch list of output files** cell enumerates the objects in the output bucket location as defined in the destination\_uri variable.
2. The **Display detected text from asynchronous output JSON files** cell loads each output JSON file that is found as a Document AI document object and the text data detected by the Document OCR processor is printed out.
3. The **Display entity data** cell will display any entity data that is found, however, entity data is only available for processors that were created using a specialized parser. Entity data will not be displayed with the general Document AI OCR parser used in this task.

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 notebook will take a minute or two to wait for the asynchronous batch process operation to complete at the **Start the batch (asynchronous) API operation** code cell. While the batch process API call itself is asynchronous the notebook uses the result method to force the notebook to wait until the asynchronous call has completed before enumerating and displaying the output data.

If the asynchronous job takes longer than expected and times out you may have to run the remaining cells again to display the output. These are the cells after the **Start the batch (asynchronous) API operation** cell.

Your output will contain text listing the Document AI data detected in each file. The Document OCR parser does not detect form or entity data so there will be no form or entity data produced. If you can create a specialised processor then you will also see entity data printed out by the final cell.

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

Document processing complete.

Text: FakeDoc M.D.

HEALTH INTAKE FORM

Please fill out the questionnaire carefully. The information you provide will be used to complete

your health profile and will be kept confidential.

Date:

Sally

Walker

Name:

9/14/19

...

Check that a document has been processed using the asynchronous Cloud Document API.

Check my progress

**Congratulations**

You've successfully made synchronous and asynchronous calls to the the Document AI API. In this lab, you enabled the Document AI API and created processors. You installed the client library for Python in a Vertex AI Notebooks instance, parsed data from a scanned form using Python to make a synchronous API call, and parsed data from scanned forms using Python to make an asynchronous API call.

Finish Your Quest

Finish your quest

This self-paced lab is part of the Detect Manufacturing Defects using Visual Inspection AI skill badge quest. A quest is a series of related labs that form a learning path. Completing this quest earns you a badge to recognize your achievement. Share your badge on your resume and social platforms, and announce your accomplishment using #GoogleCloudBadge. You can make your badge public and link to it in your online resume or social media account. Enroll in a quest and get immediate completion credit if you've taken this lab. [See other available quests](http://google.qwiklabs.com/catalog).

Next steps/learn more

To learn more about using the Document AI APIs, read [the guide](https://cloud.google.com/document-ai/docs/w9-parser). To learn more about using the **Document AI APIs**, read [the guide](https://cloud.google.com/document-ai/docs/apis).

Google Cloud Training & Certification

...helps you make the most of Google Cloud technologies. [Our classes](https://cloud.google.com/training/courses) include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and virtual options to suit your busy schedule. [Certifications](https://cloud.google.com/certification/) help you validate and prove your skill and expertise in Google Cloud technologies.

**Manual Last Updated March 25, 2022**

**Lab Last Tested September 28, 2021**

Copyright 2022 Google LLC All rights reserved. Google and the Google logo are trademarks of Google LLC. All other company and product names may be trademarks of the respective companies with which they are associated.