Getting Started

This guide will help you get started with the Mindee Python OCR SDK to easily extract data from your documents.

You can view the source code on GitHub ↗, and the package on PyPI ↗.

Prerequisite

- Download and install Python ↗. This library is officially supported on Python `3.7` to `3.12`.
- Download and install pip package manager ↗.

Installation

To quickly get started with the Python OCR SDK anywhere, the preferred installation method is via `pip`.

```
pip install mindee
```

Development Installation

If you'll be modifying the source code, you'll need to install the development requirements to get started.

1. First clone the repo.

```
git clone git@github.com:mindee/mindee-api-python.git
```

2. Then navigate to the cloned directory and install all development requirements.

```
cd mindee-api-python
pip install -e ".[dev,test]"
```

Updating the Version

It is important to always check the version of the Mindee OCR SDK you are using, as new and updated features won't work on old versions.

To check the installed version:

```
pip show mindee
```

To get the latest version:

```
pip install mindee --upgrade
```

To install a specific version:

```
pip install mindee==<your_version>
```

Usage

To get started with Mindee's APIs, you need to create a `Client` and you're ready to go.

Let's take a deep dive into how this works.

Initializing the Client

The `Client` centralizes document configurations in a single object.

The `Client` requires your API key.

You can either pass these directly to the constructor or through environment variables.

Pass the API key directly

```
from mindee import Client
#  Init with your API key
mindee_client = Client(api_key="my-api-key")
```

Set the API key in the environment

API keys should be set as environment variables, especially for any production deployment.

The following environment variable will set the global API key:

```
MINDEE_API_KEY="my-api-key"
```

Then in your code:

```
from mindee import Client
#  Init without an API key
mindee_client = Client()
```

Setting the Request Timeout

The request timeout can be set using an environment variable:

```
MINDEE_REQUEST_TIMEOUT=200
```

Loading a Document File

Before being able to send a document to the API, it must first be loaded.

You don't need to worry about different MIME types, the library will take care of handling
all supported types automatically.

Once a document is loaded, interacting with it is done in exactly the same way, regardless
of how it was loaded.

There are a few different ways of loading a document file, depending on your use case:

- Path
- File Object
- Base64

- Bytes
- URL

Path

Load from a file directly from disk. Requires an absolute path, as a string.

```
input_source = PathInput("/path/to/the/invoice.pdf")
```

File Object

A normal Python file object with a path. **Must be in binary mode**.

```
with open("/path/to/the/receipt.jpg", 'rb') as fo:
     input_doc = mindee_client.source_from_file(fo)
```

Base64

Requires a base64 encoded string.

Note: The original filename is required when calling the method.

```
b64_string =
"/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAgGBgcGBQgHBwcJCQgKDBQNDAsLD....
"
input_doc = mindee_client.source_from_b64string(b64_string,
"receipt.jpg")
```

Bytes

Requires raw bytes.

Note: The original filename is required when calling the method.

```
raw_bytes = b"%PDF-1.3\n%\xbf\xf7\xa2\xfe\n1 0 ob..."
input_doc = mindee_client.source_from_bytes(raw_bytes,
"invoice.pdf")
```

Loading from bytes is useful when using FastAPI `UploadFile` objects.

```
@app.post("/process-file")
async def upload(upload: UploadFile):
    input_doc = mindee_client.source_from_bytes(
        upload.file.read(),
        filename=upload.filename
    )
```

URL

Allows sending an URL directly.

Note: No local operations can be performed on the input (such as removing pages from a PDF).

```
input_doc =
mindee_client.source_from_url(url="https://www.example.com/invoice
.pdf")
```

Sending a File

To send a file to the API, we need to specify how to process the document. This will determine which API endpoint is used and how the API return will be handled internally by the library.

More specifically, we need to set a `mindee.product` class as the first parameter of the `parse` method.

This is because the `parse` method's' return type depends on its first argument.

Product classes inherit from the base `mindee.parsing.common.inference` class.

More information is available in each document-specific guide.

Off-the-Shelf Documents

Simply setting the correct class and passing the input document is enough:

```
result = mindee_client.parse(product.InvoiceV4, input_doc)
```

Custom Documents (docTI & Custom APIs)

The endpoint to use must be created beforehand and subsequently passed to the `endpoint` argument of the `enqueue_and_parse` method:

```
custom_endpoint = mindee_client.create_endpoint(
    "my-endpoin-url",
    "my-account-name",
    # "my-version" # optional
)
result =
mindee_client.enqueue_and_get_inference(product.GeneratedV1,
input_doc, endpoint=custom_endpoint)
```

This is because the `GeneratedV1` class is enough to handle the return processing, but the actual endpoint needs to be specified.

Processing the Response

Results of a prediction can be retrieved in two different places:

- Document level predictions
- Page level predictions

Document Level Prediction

The `document` attribute is an object specific to the type of document being processed.
It is an instance of the `Document` class, to which a generic type is given.

It contains the data extracted from the entire document, all pages combined. It's possible to have the same field in various pages, but at the document level only the highest confidence field data will be shown (this is all done automatically at the API level).

Usage:

```
print(resp.document)
```

A `document`'s fields (attributes) can be accessed through it's `prediction` attribute, which have types that can vary from one product to another.
These attributes are detailed in each product's respective guide.

Page Level Prediction

The `pages` attribute is a list of `Page` objects. `Page` is a wrapper around elements that extend the `Document` class.
The `prediction` of a `Page` inherits from the product's own `Document`, and adds all page-specific fields to it.

The order of the elements in the list matches the order of the pages in the document.

All response objects have a `pages` property, regardless of the number of pages.
Single-page documents will have a single entry.

Iteration over `pages` is done like with any list, for example:

```
for page in resp.pages:
    print(page)
```

Previous
Python OCR SDK

Next
Command Line Interface