PyPDFLoader

```
class langchain_community.document_loaders.pdf.PyPDFLoader(
  file_path: str | PurePath,
  password: str | bytes | None = None,
  headers: dict | None = None,
  extract_images: bool = False,
  *,
  mode: Literal['single', 'page'] = 'page',
  images_parser: BaseImageBlobParser | None = None,
  images_inner_format: Literal['text', 'markdown-img', 'html-img'] =
  'text',
  pages_delimiter: str = '\n\x0c',
  extraction_mode: Literal['plain', 'layout'] = 'plain',
  extraction_kwargs: dict | None = None,
) #
```

[source]

Load and parse a PDF file using 'pypdf' library.

> This class provides methods to load and parse PDF documents, supporting various configurations such as handling password-protected files, extracting images, and defining extraction mode. It integrates the pypdf library for PDF processing and offers both synchronous and asynchronous document loading.
>
> **Examples:**
>
> Setup:
>
> ```
> pip install -U langchain-community pypdf
> ```
>
> Instantiate the loader:

> ```
> from langchain_community.document_loaders import PyPDFLoader
>
> loader = PyPDFLoader(
>     file_path = "./example_data/layout-parser-paper.pdf",
>     # headers = None
>     # password = None,
>     mode = "single",
>     pages_delimiter = "
> ```

“,

> > # extract_images = True, # images_parser = RapidOCRBlobParser(),
>
>)

Lazy load documents:

```
docs = []
docs_lazy = loader.lazy_load()

for doc in docs_lazy:
    docs.append(doc)
print(docs[0].page_content[:100])
print(docs[0].metadata)
```

Load documents asynchronously:

```
docs = await loader.aload()
print(docs[0].page_content[:100])
print(docs[0].metadata)
```

Initialize with a file path.

Parameters:

- **file_path** (str | PurePath) – The path to the PDF file to be loaded.
- **headers** (dict | None) – Optional headers to use for GET request to download a file from a web path.
- **password** (str | bytes | None) – Optional password for opening encrypted PDFs.

- **mode** (Literal['single', 'page']) – The extraction mode, either “single” for the entire document or “page” for page-wise extraction.
- **pages_delimiter** (str) – A string delimiter to separate pages in single-mode extraction.
- **extract_images** (bool) – Whether to extract images from the PDF.
- **images_parser** (BaseImageBlobParser | None) – Optional image blob parser.
- **images_inner_format** (Literal['text', 'markdown-img', 'html-img']) – The format for the parsed output. - “text” = return the content as is - “markdown-img” = wrap the content into an image markdown link, w/ link pointing to (![body)(#)] - “html-img” = wrap the content as the alt text of an tag and link to (<img alt="{body}" src="#"/>)
- **extraction_mode** (Literal['plain', 'layout']) – “plain” for legacy functionality, “layout” extract text in a fixed width format that closely adheres to the rendered layout in the source pdf
- **extraction_kwargs** (dict | None) – Optional additional parameters for the extraction process.

Returns:

This method does not directly return data. Use the load, lazy_load or aload methods to retrieve parsed documents with content and metadata.

Attributes

`source`	

Methods

`__init__`(file_path[, password, headers, ...])	Initialize with a file path.
`alazy_load`()	A lazy loader for Documents.
`aload`()	Load data into Document objects.
`lazy_load`()	Lazy load given path as pages.
`load`()	Load data into Document objects.
`load_and_split`([text_splitter])	Load Documents and split into chunks.

```
__init__(
  file_path: str | PurePath,
  password: str | bytes | None = None,
  headers: dict | None = None,
  extract_images: bool = False,
  *,
  mode: Literal['single', 'page'] = 'page',
  images_parser: BaseImageBlobParser | None = None,
  images_inner_format: Literal['text', 'markdown-img', 'html-img'] =
  'text',
  pages_delimiter: str = '\n\x0c',
  extraction_mode: Literal['plain', 'layout'] = 'plain',
  extraction_kwargs: dict | None = None,
  ) → None #
```
[source]

Initialize with a file path.

Parameters:

- **file_path** (str | PurePath) – The path to the PDF file to be loaded.
- **headers** (dict | None) – Optional headers to use for GET request to download a file from a web path.
- **password** (str | bytes | None) – Optional password for opening encrypted PDFs.
- **mode** (Literal['single', 'page']) – The extraction mode, either “single” for the entire document or “page” for page-wise extraction.
- **pages_delimiter** (str) – A string delimiter to separate pages in single-mode extraction.
- **extract_images** (bool) – Whether to extract images from the PDF.
- **images_parser** (BaseImageBlobParser | None) – Optional image blob parser.
- **images_inner_format** (Literal['text', 'markdown-img', 'html-img']) – The format for the parsed output. - “text” = return the content as is - “markdown-img” = wrap the content into an image markdown link, w/ link pointing to (![body)(#)] - “html-img” = wrap the content as the alt text of an tag and link to (<img alt=”{body}” src=”#”/>)
- **extraction_mode** (Literal['plain', 'layout']) – “plain” for legacy functionality, “layout” extract text in a fixed width format that closely adheres to the rendered layout in the source pdf
- **extraction_kwargs** (dict | None) – Optional additional parameters for the extraction process.

Returns:

This method does not directly return data. Use the load, lazy_load or aload methods to retrieve parsed documents with content and metadata.

Return type:

None

`async alazy_load() → AsyncIterator[Document]` #

A lazy loader for Documents.

Return type:

AsyncIterator[Document]

`async aload() → list[Document]` #

Load data into Document objects.

Return type:

list[Document]

`lazy_load() → Iterator[Document]` # [source]

Lazy load given path as pages. Insert image, if possible, between two paragraphs. In this way, a paragraph can be continued on the next page.

Return type:

Iterator[Document]

`load() → list[Document]` #

Load data into Document objects.

Return type:

list[Document]

```
load_and_split(
  text_splitter: TextSplitter | None = None,
) → list[Document] #
```

Load Documents and split into chunks. Chunks are returned as Documents.

Do not override this method. It should be considered to be deprecated!

Parameters:

text_splitter (Optional[TextSplitter]) – TextSplitter instance to use for splitting documents. Defaults to RecursiveCharacterTextSplitter.

Returns:

List of Documents.

Return type:

list[Document]

Examples using PyPDFLoader

- Apache Cassandra
- Azure Cosmos DB No SQL
- Build a PDF ingestion and Question/Answering system
- Google Cloud Storage File
- Google Vertex AI Vector Search

© Copyright 2023, LangChain Inc.