

[MCP Registry](#) / [Markdown](#)

Markdown

By [microsoft](#) · ☆ 147 387

Convert various file formats (PDF, Word, Excel, images, audio) to Markdown.

[Install MCP server](#)

MarkItDown

pypi v0.1.6

downloads 224k/day

Built by AutoGen Team

⚠ Important

MarkItDown performs I/O with the privileges of the current process. Like `open()` or `requests.get()`, it will access resources that the process itself can access. Sanitize your inputs in untrusted environments, and call the narrowest `convert_*` function needed for your use case (e.g., `convert_stream()`, or `convert_local()`). See the [Security Considerations](#) section of the documentation for more information.

MarkItDown is a lightweight Python utility for converting various files to Markdown for use with LLMs and related text analysis pipelines. To this end, it is most comparable to [textract](#), but with a focus on preserving important document structure and content as Markdown (including: headings, lists, tables, links, etc.) While the output is often reasonably presentable and human-friendly, it is meant to be consumed by text analysis tools -- and may not be the best option for high-fidelity document conversions for human consumption.

MarkItDown currently supports the conversion from:

- PDF
- PowerPoint
- Word
- Excel
- Images (EXIF metadata and OCR)
- Audio (EXIF metadata and speech transcription)

- HTML
- Text-based formats (CSV, JSON, XML)
- ZIP files (iterates over contents)
- Youtube URLs
- EPubs
- ... and more!

Why Markdown?

Markdown is extremely close to plain text, with minimal markup or formatting, but still provides a way to represent important document structure. Mainstream LLMs, such as OpenAI's GPT-4o, natively "*speak*" Markdown, and often incorporate Markdown into their responses unprompted. This suggests that they have been trained on vast amounts of Markdown-formatted text, and understand it well. As a side benefit, Markdown conventions are also highly token-efficient.

Prerequisites

Markdown requires Python 3.10 or higher. It is recommended to use a virtual environment to avoid dependency conflicts.

With the standard Python installation, you can create and activate a virtual environment using the following commands:

```
python -m venv .venv
source .venv/bin/activate
```



If using `uv`, you can create a virtual environment with:

```
uv venv --python=3.12 .venv
source .venv/bin/activate
# NOTE: Be sure to use 'uv pip install' rather than just 'pip install' to instal
```



If you are using Anaconda, you can create a virtual environment with:

```
conda create -n markdown python=3.12
conda activate markdown
```



Installation

To install MarkItDown, use pip: `pip install 'markdown[all]'` . Alternatively, you can install it from the source:

```
git clone git@github.com:microsoft/markitdown.git
cd markitdown
pip install -e 'packages/markitdown[all]'
```



Usage

Command-Line

```
markitdown path-to-file.pdf > document.md
```



Or use `-o` to specify the output file:

```
markitdown path-to-file.pdf -o document.md
```



You can also pipe content:

```
cat path-to-file.pdf | markitdown
```



Optional Dependencies

MarkItDown has optional dependencies for activating various file formats. Earlier in this document, we installed all optional dependencies with the `[all]` option. However, you can also install them individually for more control. For example:

```
pip install 'markdown[pdf, docx, pptx]'
```



will install only the dependencies for PDF, DOCX, and PPTX files.

At the moment, the following optional dependencies are available:

- `[all]` Installs all optional dependencies
- `[pptx]` Installs dependencies for PowerPoint files
- `[docx]` Installs dependencies for Word files
- `[xlsx]` Installs dependencies for Excel files
- `[xls]` Installs dependencies for older Excel files

- `[pdf]` Installs dependencies for PDF files
- `[outlook]` Installs dependencies for Outlook messages
- `[az-doc-intel]` Installs dependencies for Azure Document Intelligence
- `[az-content-understanding]` Installs dependencies for Azure Content Understanding
- `[audio-transcription]` Installs dependencies for audio transcription of wav and mp3 files
- `[youtube-transcription]` Installs dependencies for fetching YouTube video transcription

Plugins

MarkItDown also supports 3rd-party plugins. Plugins are disabled by default. To list installed plugins:

```
markitdown --list-plugins
```



To enable plugins use:

```
markitdown --use-plugins path-to-file.pdf
```



To find available plugins, search GitHub for the hashtag `#markitdown-plugin`. To develop a plugin, see `packages/markitdown-sample-plugin`.

markitdown-ocr Plugin

The `markitdown-ocr` plugin adds OCR support to PDF, DOCX, PPTX, and XLSX converters, extracting text from embedded images using LLM Vision — the same `llm_client` / `llm_model` pattern that MarkItDown already uses for image descriptions. No new ML libraries or binary dependencies required.

Installation:

```
pip install markitdown-ocr
pip install openai # or any OpenAI-compatible client
```



Usage:

Pass the same `llm_client` and `llm_model` you would use for image descriptions:

```
from markitdown import MarkItDown
from openai import OpenAI

md = MarkItDown(
```



```
enable_plugins=True,
llm_client=OpenAI(),
llm_model="gpt-4o",
)
result = md.convert("document_with_images.pdf")
print(result.text_content)
```

If no `llm_client` is provided the plugin still loads, but OCR is silently skipped and the standard built-in converter is used instead.

See [packages/markitdown-ocr/README.md](https://github.com/mcp/mcp/blob/main/packages/markitdown-ocr/README.md) for detailed documentation.

Azure Content Understanding

[Azure Content Understanding](#) provides higher-quality conversion with structured field extraction (YAML front matter), multi-modal support (documents, images, audio, video), and configurable analyzers.

Install: `pip install 'markitdown[az-content-understanding]'`

When to use Content Understanding

Content Understanding is ideal when you need capabilities beyond what built-in or Document Intelligence converters provide:

- **Audio and video files** — CU is the only option for video, and the higher-quality cloud option for audio. Built-in converters have no video support and only basic audio transcription.
- **Structured field extraction** — [Prebuilt](#) or [custom-built](#) analyzers extract domain-specific fields (invoice amounts, receipt dates, contract clauses) serialized as YAML front matter. Neither built-in nor Doc Intel integration exposes fields.
- **Higher-quality document extraction** — Cloud-based layout analysis and OCR for scanned PDFs, complex tables, and multi-page documents.
- **Single API for all modalities** — One `cu_endpoint` handles documents, images, audio, and video with automatic analyzer routing.

Capability	Built-in converters	Azure Document Intelligence	Azure Content Understanding
Document conversion	Offline, format-specific extraction	Cloud layout extraction	Cloud multimodal extraction
Structured fields	Not available	Not exposed by this integration	YAML front matter from analyzer fields

Capability	Built-in converters	Azure Document Intelligence	Azure Content Understanding
Custom analyzers	Not available	Not configurable in this integration	Supported with <code>cu_analyzer_id</code>
Audio and video	Basic audio, no video	Not supported	Audio and video analyzers
Cost	Local compute only	Billable Azure API calls	Billable Azure API calls

CLI:

```
markdown path-to-file.pdf --use-cu --cu-endpoint "<content_understanding_endpc
```



Python API:

```
from markdown import MarkItDown

# Zero-config – auto-selects analyzer per file type
md = MarkItDown(cu_endpoint="<content_understanding_endpoint>")
result = md.convert("report.pdf") # documents → prebuilt-documentSearch
result = md.convert("meeting.mp4") # video → prebuilt-videoSearch
result = md.convert("call.wav") # audio → prebuilt-audioSearch
print(result.markdown)
```

With a custom analyzer (for domain-specific field extraction):

```
md = MarkItDown(
    cu_endpoint="<content_understanding_endpoint>",
    cu_analyzer_id="my-invoice-analyzer",
)
result = md.convert("invoice.pdf")
print(result.markdown)
# Output includes YAML front matter with extracted fields:
# ---
# contentType: document
# fields:
#   VendorName: CONTOSO LTD.
#   InvoiceDate: '2019-11-15'
# ---
# <!-- page 1 -->
# ...
```

When `cu_analyzer_id` is set, the converter automatically scopes it to compatible file types based on the analyzer's modality. Incompatible types (e.g., audio files with a document analyzer) auto-route to default prebuilt analyzers.

Cost note: Each `convert()` call for a CU-routed format is a billable Azure API call. Use `cu_file_types` to restrict which formats route to CU:

```
from markdown.converters import ContentUnderstandingFileType

md = MarkItDown(
    cu_endpoint="<content_understanding_endpoint>",
    cu_file_types=[ContentUnderstandingFileType.PDF], # only PDFs use CU
)
```



More information about Azure Content Understanding can be found [here](#).

Azure Document Intelligence

To use Microsoft Document Intelligence for conversion:

```
markdown path-to-file.pdf -o document.md -d -e "<document_intelligence_endpoir
```



More information about how to set up an Azure Document Intelligence Resource can be found [here](#)

Python API

Basic usage in Python:

```
from markdown import MarkItDown

md = MarkItDown(enable_plugins=False) # Set to True to enable plugins
result = md.convert("test.xlsx")
print(result.text_content)
```



Document Intelligence conversion in Python:

```
from markdown import MarkItDown

md = MarkItDown(docintel_endpoint="<document_intelligence_endpoint>")
result = md.convert("test.pdf")
print(result.text_content)
```



To use Large Language Models for image descriptions (currently only for pptx and image files), provide `llm_client` and `llm_model` :

```
from markdown import Markdown
from openai import OpenAI

client = OpenAI()
md = Markdown(llm_client=client, llm_model="gpt-4o", llm_prompt="optional cust
result = md.convert("example.jpg")
print(result.text_content)
```



Docker

```
docker build -t markdown:latest .
docker run --rm -i markdown:latest < ~/your-file.pdf > output.md
```



Contributing

This project welcomes contributions and suggestions. Most contributions require you to agree to a

Contributor License Agreement (CLA) declaring that you have the right to, and actually do, grant us

the rights to use your contribution. For details, visit <https://cla.opensource.microsoft.com>.

When you submit a pull request, a CLA bot will automatically determine whether you need to provide

a CLA and decorate the PR appropriately (e.g., status check, comment). Simply follow the instructions

provided by the bot. You will only need to do this once across all repos using our CLA.

This project has adopted the [Microsoft Open Source Code of Conduct](#).

For more information see the [Code of Conduct FAQ](#) or

contact opencode@microsoft.com with any additional questions or comments.

How to Contribute

You can help by looking at issues or helping review PRs. Any issue or PR is welcome, but we have also marked some as 'open for contribution' and 'open for reviewing' to help facilitate community contributions. These are of course just suggestions and you are welcome to contribute in any way you like.

	All	Especially Needs Help from Community
Issues	All Issues	Issues open for contribution
PRs	All PRs	PRs open for reviewing

Running Tests and Checks

- Navigate to the MarkItDown package:

```
cd packages/markitdown
```



- Install hatch in your environment and run tests:

```
pip install hatch # Other ways of installing hatch: https://hatch.pypa.io/
hatch shell
hatch test
```



(Alternative) Use the Devcontainer which has all the dependencies installed:

```
# Reopen the project in Devcontainer and run:
hatch test
```



- Run pre-commit checks before submitting a PR: `pre-commit run --all-files`

Security Considerations

MarkItDown performs I/O with the privileges of the current process. Like `open()` or `requests.get()` , it will access resources that the process itself can access.

Sanitize your inputs: Do not pass untrusted input directly to MarkItDown. If any part of the input may be controlled by an untrusted user or system, such as in hosted or server-side applications, it must be validated and restricted before calling MarkItDown. Depending on your environment, this may include restricting file paths, limiting URI schemes and network destinations, and blocking access to private, loopback, link-local, or metadata-service addresses.

Call only the conversion method you need: Prefer the narrowest conversion API that fits your use case. Markdown's `convert()` method is intentionally permissive and can handle local files, remote URLs, and byte streams. If your application only needs to read local files, call `convert_local()` instead. If you need more control over URI fetching, call `requests.get()` yourself and pass the response object to `convert_response()`. For maximum control, open a stream to the input you want converted and call `convert_stream()`.

Contributing 3rd-party Plugins

You can also contribute by creating and sharing 3rd party plugins. See `packages/markitdown-sample-plugin` for more details.

Trademarks

This project may contain trademarks or logos for projects, products, or services. Authorized use of Microsoft

trademarks or logos is subject to and must follow

[Microsoft's Trademark & Brand Guidelines](#).

Use of Microsoft trademarks or logos in modified versions of this project must not cause confusion or imply Microsoft sponsorship.

Any use of third-party trademarks or logos are subject to those third-party's policies.

Resources

 [microsoft / markitdown](#)

 [Contact support](#)