Reference Card: Docling
Yogesh Haribhau Kulkarni

Installation & Setup

```
# Install via pip:
pip install docling
# With OCR support
pip install docling[ocr]
# Full installation
pip install docling[all]

# Basic imports:
from docling.document_converter import DocumentConverter
from docling.datamodel.base_models import InputFormat
from docling.datamodel.pipeline_options import PdfPipelineOptions
```

Basic Document Conversion

```
# Simple conversion:
converter = DocumentConverter()
result = converter.convert("document.pdf")
# Export to markdown
markdown = result.document.export_to_markdown()
# Export to dict
doc_dict = result.document.export_to_dict()

# Multiple documents:
docs = ["doc1.pdf", "doc2.docx", "doc3.html"]
results = converter.convert_all(docs)
for result in results:
    print(result.document.export_to_markdown())
```

Supported Input Formats

```
# Docling supports multiple formats:
# PDF, DOCX, PPTX, HTML, Images, Markdown, AsciiDoc
from docling.datamodel.base_models import InputFormat

supported = [
    InputFormat.PDF,
    InputFormat.DOCX,
    InputFormat.PPTX,
    InputFormat.HTML,
    InputFormat.IMAGE,
    InputFormat.MD,
    InputFormat.ASCIIDOC
]

# Check format:
source = DocumentConverter.resolve_source("file.pdf")
print(source.format)  # InputFormat.PDF
```

PDF Pipeline Options

```
# Configure PDF processing:
from docling.datamodel.pipeline_options import (
    PdfPipelineOptions, TableFormerMode
)

pipeline_opts = PdfPipelineOptions()
pipeline_opts.do_ocr = True
pipeline_opts.do_table_structure = True
pipeline_opts.table_structure_options.mode = \
    TableFormerMode.ACCURATE

converter = DocumentConverter(
    format_options={InputFormat.PDF: pipeline_opts}
)
```

OCR Configuration

```
# Enable and configure OCR:

from docling.datamodel.pipeline_options import (
    EasyOcrOptions, TesseractOcrOptions
)

# EasyOCR (default)
ocr_opts = EasyOcrOptions(lang=["en", "de"])
pipeline_opts = PdfPipelineOptions()
pipeline_opts.do_ocr = True
pipeline_opts.ocr_options = ocr_opts

# Tesseract OCR
tesseract_opts = TesseractOcrOptions(lang="eng+deu")
pipeline_opts.ocr_options = tesseract_opts
```

Table Extraction Modes

```
# Configure table extraction:
from docling.datamodel.pipeline_options import (
    TableFormerMode, TableStructureOptions
)

# Fast mode (faster, less accurate)
table_opts = TableStructureOptions(
    mode=TableFormerMode.FAST
)

# Accurate mode (slower, more accurate)
table_opts = TableStructureOptions(
    mode=TableFormerMode.ACCURATE
)

pipeline_opts.table_structure_options = table_opts
```

Document Structure Access

```
# Navigate document hierarchy:
result = converter.convert("doc.pdf")
doc = result.document

# Access sections
for section in doc.sections:
    print(section.heading, section.text)

# Access tables
for table in doc.tables:
    print(table.export_to_dataframe())

# Access figures
for figure in doc.figures:
    print(figure.caption, figure.uri)
```

Metadata Extraction

```
# Access document metadata:
result = converter.convert("doc.pdf")

# File metadata
print(result.input.file)
print(result.status)

# Document properties
doc = result.document
print(doc.name)
print(doc.origin)

# Page count
print(len(doc.pages))
```

Export to Markdown

```
# Export with various options:
doc = result.document

# Standard markdown
md = doc.export_to_markdown()

# With image references
md = doc.export_to_markdown(
    image_mode="referenced"
)

# With tables
md = doc.export_to_markdown(
    strict_text=False
)

# Save to file
with open("output.md", "w") as f:
    f.write(md)
```

Export to JSON/Dict

```
# Export structured data:
import json

doc = result.document

# Export to dictionary
doc_dict = doc.export_to_dict()

# Save to JSON
with open("output.json", "w") as f:
    json.dump(doc_dict, f, indent=2)

# Access nested structure
for item in doc_dict.get("body", []):
    print(item["type"], item.get("text"))
```

Export to Document Tree

```
# Work with document tree:
doc = result.document

# Get document tree
tree = doc.export_to_document_tokens()

# Iterate through tokens
for token in tree:
    print(token.text, token.obj_type)

# Access hierarchical structure
root = doc.body
for node in root.children:
    print(node.label, node.text)
```

Page-Level Processing

```
# Process individual pages:
doc = result.document

# Iterate pages
for page in doc.pages:
    print(f"Page {page.page_no}")
    print(f"Size: {page.size}")
```

```
# Get specific page
page_1 = doc.pages[0]

# Page content
for item in page_1.children:
    print(item.label, item.text)
```

Table Handling

```
# Extract and process tables:
import pandas as pd

doc = result.document

# Export tables to DataFrames
for table in doc.tables:
    df = table.export_to_dataframe()
    print(df)

# Access table cells
for table in doc.tables:
    for row in table.grid:
        for cell in row:
            print(cell.text)
```

Figure & Image Extraction

```
# Extract images and figures:
doc = result.document

# Iterate figures
for figure in doc.figures:
    print(f"Caption: {figure.caption}")
    print(f"URI: {figure.uri}")

# Access image data
for picture in doc.pictures:
    if picture.image:
        img = picture.image.pil_image
        img.save(f"image_{picture.self_ref}.png")
```

Text Extraction & Filtering

```
# Extract specific text elements:
doc = result.document

# Get all text
full_text = doc.export_to_markdown()

# Filter by type
headings = [item for item in doc.body.children
          if item.label == "section_header"]

paragraphs = [item for item in doc.body.children
             if item.label == "paragraph"]

# Get text content
texts = [p.text for p in paragraphs]
```

Batch Processing

```
# Process multiple documents efficiently:
from pathlib import Path

converter = DocumentConverter()

# Process directory
docs = list(Path("./docs").glob("*.pdf"))
results = converter.convert_all(docs)
```

```
# Batch export
for result in results:
    out_name = result.input.file.stem + ".md"
    with open(out_name, "w") as f:
        f.write(result.document.export_to_markdown())
```

Error Handling

```
# Handle conversion errors:
from docling.datamodel.base_models import ConversionStatus

result = converter.convert("doc.pdf")

if result.status == ConversionStatus.SUCCESS:
    print("Conversion successful")
    doc = result.document
elif result.status == ConversionStatus.FAILURE:
    print(f"Error: {result.errors}")
elif result.status == ConversionStatus.PARTIAL_SUCCESS:
    print("Partial conversion")
    doc = result.document
```

Custom Pipeline Options

```
# Advanced pipeline configuration:
from docling.datamodel.pipeline_options import (
    PdfPipelineOptions, RapidOcrOptions
)

pipeline_opts = PdfPipelineOptions(
    do_ocr=True,
    do_table_structure=True,
    images_scale=2.0,
    generate_page_images=True,
    generate_picture_images=True
)

# Custom OCR
pipeline_opts.ocr_options = RapidOcrOptions()

converter = DocumentConverter(
    format_options={InputFormat.PDF: pipeline_opts}
)
```

Document Labels & Types

```
# Common document element labels:
# Label types in Docling:
labels = [
    "paragraph",      # Regular text
    "section_header", # Headings
    "title",         # Document title
    "table",          # Tables
    "figure",         # Figures/images
    "caption",        # Captions
    "list_item",      # List items
    "page_header",    # Headers
    "page_footer",    # Footers
    "footnote"        # Footnotes
]
```

Advanced: Custom Backends

```
# Use custom model backends:
from docling_core.types.doc import (
    TableStructureOptions, TableFormerMode
)

# Configure model paths
table_opts = TableStructureOptions(
    mode=TableFormerMode.ACCURATE,
    do_cell_matching=True
)

pipeline_opts = PdfPipelineOptions()
pipeline_opts.table_structure_options = table_opts

# Apply custom backend
converter = DocumentConverter(
    format_options={InputFormat.PDF: pipeline_opts}
)
```

Performance Tips

```
# Optimize processing:
# 1. Disable unnecessary features
pipeline_opts = PdfPipelineOptions(
    do_ocr=False,  # Skip if text-based PDF
    do_table_structure=False,  # Skip if no tables
    generate_page_images=False
)

# 2. Use FAST mode for tables
pipeline_opts.table_structure_options.mode = \
    TableFormerMode.FAST

# 3. Batch process
results = converter.convert_all(docs)

# 4. Process pages selectively
# (Filter pages before conversion if possible)
```

Common Patterns

```
# Useful code patterns:
# Extract only tables from PDF
result = converter.convert("doc.pdf")
tables = result.document.tables
dfs = [t.export_to_dataframe() for t in tables]

# Extract headings hierarchy
headings = [item for item in doc.body.children
            if "header" in item.label]

# Save all images
for i, fig in enumerate(doc.figures):
    if fig.image and fig.image.pil_image:
        fig.image.pil_image.save(f"fig_{i}.png")

# Full pipeline example
converter = DocumentConverter()
result = converter.convert("doc.pdf")
result.document.export_to_markdown()
```

Temporary page!

LaTeX was unable to guess the total number of pages correctly. As there was some unprocessed data that should have been added to the final page this extra page has been added to receive it. If you rerun the document (without altering it) this surplus page will go away, because LaTeX now knows how many pages to expect for this document.