Code2PDF

Alhaan

2025-05-06 19:34:00

Contents

Project Overview

No project description provided.

Codebase Statistics

- **Total files:** 12
- **Total lines of code:** 568
- **Languages used:** plaintext (2), python (8), html (1), javascript (1)
- **Generated by:** code2pdf v0.2

Directory Structure

```
doc_complie/
    └── code2pdf_installer.iss
    └── my_reference.txt
    └── setup.py
    ├── code2pdf/
        └── code2pdf.py
        └── constants.py
        └── file_utils.py
        └── parser_utils.py
        └── pdf_utils.py
    ├── test/
        └── app.py
        └── index.html
        └── run_test.py
        └── script.js
```

D:/Programming/doc_complie/code2pdf_installer.iss

```
[Setup]
AppName=Code2PDF
AppVersion=1.0
DefaultDirName={pf}\Code2PDF
DefaultGroupName=Code2PDF
OutputBaseFilename=Code2PDFInstaller
Compression=lzma
SolidCompression=yes
```

```
[Files]
Source: "dist\code2pdf.exe"; DestDir: "{app}"; Flags: ignoreversion
Source: "installers\pandoc-3.6.4-windows-x86_64.msi"; DestDir: "{tmp}"; Flags:
 ↪  deleteafterinstall
Source: "installers\basic-miktex-24.1-x64.exe"; DestDir: "{tmp}"; Flags:
 ↪  deleteafterinstall

[Icons]
Name: "{group}\Code2PDF"; Filename: "{app}\code2pdf.exe"
Name: "{group}\Uninstall Code2PDF"; Filename: "{uninstallexe}"

[Run]
; Install Pandoc silently
Filename: "msiexec.exe"; Parameters: "/i ""{tmp}\pandoc-3.6.4-windows-x86_64.msi""
 ↪  /quiet"; StatusMsg: "Installing Pandoc..."

; Install MiKTeX silently in unattended mode
Filename: "{tmp}\basic-miktex-24.1-x64.exe"; Parameters: "--unattended";
 ↪  StatusMsg: "Installing MiKTeX..."

; Small delay to ensure MiKTeX is fully initialized (optional but safer)
Filename: "ping.exe"; Parameters: "127.0.0.1 -n 6 > nul"; Flags: runhidden

[Registry]
Root: HKCU; Subkey: "Environment"; \
    ValueType: string; ValueName: "Path"; \
    ValueData: "{olddata};{app}"; Flags: preservestringtype uninsdeletevalue
```

D:/Programming/doc_complie/my_reference.txt

```
Run this install the package globally
C:\Users\ADMIN\AppData\Local\Programs\Python\Python39\Scripts\pip.exe install
 ↪  --user .

Place where markdown is saved
C:\Users\Admin\AppData\Local\Temp\combined_code.md
```

D:/Programming/doc_complie/setup.py

```
from setuptools import setup, find_packages
```

```
setup(
    name="code2pdf",
    version="1.0",
    packages=find_packages(),  # Automatically find the 'code2pdf' package
    entry_points={
        'console_scripts': [
            'code2pdf = code2pdf.code2pdf:main',
        ],
    },
    install_requires=['pathspec'],
)
```

D:/Programming/doc_complie/code2pdf/code2pdf.py

```
import argparse
import os

from .file_utils import (
    load_ignore_spec,
    format_path,
    get_language,
    generate_markdown,
    get_included_and_ignored_files,  # <-- add import
)
from .pdf_utils import (
    check_dependencies,
    convert_to_pdf_pandoc,
)
from .constants import (
    EXCLUDE_DIRS,
    EXCLUDE_FILES,
    DEFAULT_OUTPUT,
    TEMP_MD,
)
from .parser_utils import create_parser

def main():
    # Main entry point: parses arguments, generates markdown, and converts to PDF.
    # Handles user input and orchestrates the codebase-to-PDF workflow.

    parser = create_parser()  # use modularized parser
    args = parser.parse_args()
```

```
    folder_path = os.path.abspath(args.folder)

    if not os.path.isdir(folder_path):
        print(f"❌ Error: '{folder_path}' is not a valid directory.")
        return

    print(f"📂 Scanning: {folder_path}")
    ignore_spec = load_ignore_spec(folder_path)
    include_exts = set(args.include.split(',')) if args.include else None
    exclude_exts = set(args.exclude.split(',')) if args.exclude else None

    # Show included and ignored files before proceeding
    included, ignored = get_included_and_ignored_files(
        folder_path, ignore_spec=ignore_spec,
        include_exts=include_exts, exclude_exts=exclude_exts
    )
    print("\nFiles to be INCLUDED in the PDF:")
    for f in included:
        print("  +", os.path.relpath(f, folder_path))
    print("\nFiles to be IGNORED:")
    for f in ignored:
        print("  -", os.path.relpath(f, folder_path))
    print("\nProceed with PDF generation? (y/n): ", end="")
    choice = input().strip().lower()
    if choice != "y":
        # Suggest ignore file command
        ignore_file = ".code2pdfignore" if os.path.exists(os.path.join(folder_path,
↪   ".code2pdfignore")) else ".gitignore"
        print(f"\nTo customize which files are ignored, edit or create
        ↪  '{ignore_file}' in the project folder.")
        print(f"Example: echo '*.log' >> {ignore_file}")
        print("Aborted.")
        return

    # Determine output PDF path: if user gave just a filename, put it in the
    ↪  scanned folder
    output_arg = args.output
    if not os.path.isabs(output_arg):
        output_pdf = os.path.join(folder_path, output_arg)
    else:
        output_pdf = output_arg

    generate_markdown(
```

```
        folder_path, args.title, args.author,
        skip_empty=args.skip_empty,
        ignore_spec=ignore_spec,
        include_exts=include_exts,
        exclude_exts=exclude_exts,
        project_desc=args.project_desc,
        tool_version="0.2"
    )
    print(f"Markdown compiled. Converting to PDF...")
    convert_to_pdf_pandoc(TEMP_MD, output_pdf)
    # md_to_pdf(TEMP_MD, output_pdf)
    # markdown_to_pdf(TEMP_MD, output_pdf)
    print(f"PDF saved as: {output_pdf}")

if __name__ == '__main__':
    main()
```

D:/Programming/doc_complie/code2pdf/constants.py

```
import os
import tempfile

# Constants for directories and files to exclude
EXCLUDE_DIRS = {
    '.git', 'node_modules', '__pycache__', 'venv', '.venv',
    '.mypy_cache', '.pytest_cache', '.idea', '.vscode', 'env',
    'dist', 'build', '*.egg-info', 'installers',
    'out', 'tmp', 'temp', 'cache', 'logs', '.tox', '.coverage', 'dist',
    'coverage.xml', 'htmlcov', 'pytest_cache', 'code2pdf.egg-info'
}
EXCLUDE_FILES = {
    '.DS_Store', 'Thumbs.db'
}

DEFAULT_OUTPUT = 'codebase.pdf'
TEMP_MD = os.path.join(tempfile.gettempdir(), 'combined_code.md')
```

D:/Programming/doc_complie/code2pdf/file_utils.py

```
import os
import pathspec
from datetime import datetime
```

```
from .constants import EXCLUDE_DIRS, EXCLUDE_FILES, TEMP_MD

# Loads ignore patterns from .gitignore or .code2pdfignore if present.
# Returns a pathspec object or None if no ignore file is found.
def load_ignore_spec(input_dir):
    ignore_files = ['.gitignore', '.code2pdfignore']
    for ignore_file in ignore_files:
        ignore_path = os.path.join(input_dir, ignore_file)
        if os.path.exists(ignore_path):
            with open(ignore_path, 'r') as f:
                return pathspec.PathSpec.from_lines('gitwildmatch', f.readlines())
    return None

# Formats a file path for markdown display, optionally with backticks.
# Converts backslashes to slashes and escapes underscores if needed.
def format_path(path, backtick=True, escape_underscore=True):
    # Only escape underscores if requested (default True)
    path = path.replace('\\', '/')
    if escape_underscore:
        path = path.replace('_', r'\_')
    return f"`{path}`" if backtick else path

# Returns the programming language for a given filename based on its extension.
# Used for syntax highlighting in markdown code blocks.
def get_language(filename):
    ext = os.path.splitext(filename)[1].lower()
    language_map = {
        '.py': 'python',
        '.js': 'javascript',
        '.java': 'java',
        '.cpp': 'cpp',
        '.c': 'c',
        '.html': 'html',
        '.css': 'css',
        '.md': 'markdown',
        '.json': 'json',
        '.xml': 'xml',
        '.sh': 'bash',
        '.bat': 'batch',
        '.txt': 'plaintext',
    }
    return language_map.get(ext, 'plaintext')
```

```python
# Compute codebase statistics (file count, line count, languages)
def compute_codebase_stats(input_dir, ignore_spec=None):
    stats = {
        "total_files": 0,
        "total_lines": 0,
        "languages": {},
    }
    for root, dirs, files in os.walk(input_dir):
        dirs[:] = [d for d in dirs if d not in EXCLUDE_DIRS]
        files[:] = [f for f in files if f not in EXCLUDE_FILES]
        if ignore_spec:
            files = [f for f in files if not
↪   ignore_spec.match_file(os.path.join(root, f))]
        for file in files:
            ext = os.path.splitext(file)[1].lower()
            lang = get_language(file)
            file_path = os.path.join(root, file)
            try:
                with open(file_path, 'r', encoding='utf-8') as f:
                    lines = f.readlines()
            except Exception:
                continue
            stats["total_files"] += 1
            stats["total_lines"] += len(lines)
            stats["languages"][lang] = stats["languages"].get(lang, 0) + 1
    return stats

# Generate directory tree as markdown
def get_directory_tree(input_dir, ignore_spec=None, prefix=""):
    tree_lines = []
    for root, dirs, files in os.walk(input_dir):
        level = root.replace(input_dir, '').count(os.sep)
        indent = '    ' * level
        subdir = os.path.basename(root)
        if level == 0:
            tree_lines.append(f"{subdir}/")
        else:
            tree_lines.append(f"{indent}├── {subdir}/")
        dirs[:] = [d for d in dirs if d not in EXCLUDE_DIRS]
        files[:] = [f for f in files if f not in EXCLUDE_FILES]
        if ignore_spec:
            files = [f for f in files if not
↪   ignore_spec.match_file(os.path.join(root, f))]
```

```
        for f in files:
            tree_lines.append(f"{indent}    └── {f}")
    return "\n".join(tree_lines)

# Returns two lists: (included_files, ignored_files)
def get_included_and_ignored_files(input_dir, ignore_spec=None, include_exts=None,
↪   exclude_exts=None, include_names=None, exclude_names=None):
    included = []
    ignored = []
    for root, dirs, files in os.walk(input_dir):
        dirs[:] = [d for d in dirs if d not in EXCLUDE_DIRS]
        files[:] = [f for f in files if f not in EXCLUDE_FILES]
        orig_files = list(files)
        if ignore_spec:
            files = [f for f in files if not
↪   ignore_spec.match_file(os.path.join(root, f))]
        for file in orig_files:
            ext = os.path.splitext(file)[1].lower()
            file_path = os.path.join(root, file)
            if ignore_spec and ignore_spec.match_file(file_path):
                ignored.append(file_path)
                continue
            if include_exts and ext not in include_exts:
                ignored.append(file_path)
                continue
            if exclude_exts and ext in exclude_exts:
                ignored.append(file_path)
                continue
            if include_names and file not in include_names:
                ignored.append(file_path)
                continue
            if exclude_names and file in exclude_names:
                ignored.append(file_path)
                continue
            included.append(file_path)
    return included, ignored

# Walks the input directory and generates a markdown file with code listings.
# Applies ignore rules, file filters, and formats each file as a markdown section.
def generate_markdown(input_dir, title, author, out_file=TEMP_MD,
                      skip_empty=False, ignore_spec=None,
                      include_exts=None, exclude_exts=None,
                      include_names=None, exclude_names=None,
```

```
                    project_desc=None, tool_version="0.2"):
    stats = compute_codebase_stats(input_dir, ignore_spec)
    dir_tree = get_directory_tree(input_dir, ignore_spec)
    with open(out_file, 'w', encoding='utf-8') as md_file:
        md_file.write(f"""---
title: "{title}"
author: "{author}"
date: "{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"
titlepage: true
titlepage-color: "003366"
titlepage-text-color: "FFFFFF"
titlepage-rule-color: "FFFFFF"
titlepage-rule-height: 2
logo: ""
toc: true
toc-own-page: true
toc-depth: 3
colorlinks: true
linkcolor: blue
fontsize: 12pt
geometry: margin=1in
mainfont: TeX Gyre Pagella
monofont: Fira Mono
...

""")  # <-- Ensure a blank line after the YAML block

        md_file.write(f"""# Project Overview

{project_desc or "No project description provided."}

## Codebase Statistics

- **Total files:** {stats['total_files']}
- **Total lines of code:** {stats['total_lines']}
- **Languages used:** {', '.join(f"{lang} ({count})" for lang, count in
 ↪  stats['languages'].items())}
- **Generated by:** code2pdf v{tool_version}

## Directory Structure

{dir_tree}
```

```
""")

        for root, dirs, files in os.walk(input_dir):
            dirs[:] = [d for d in dirs if d not in EXCLUDE_DIRS]
            files[:] = [f for f in files if f not in EXCLUDE_FILES]

            if ignore_spec:
           files = [f for f in files if not ignore_spec.match_file(os.path.join(root, f))]

            for file in files:
                ext = os.path.splitext(file)[1].lower()
                if include_exts and ext not in include_exts:
                    continue
                if exclude_exts and ext in exclude_exts:
                    continue
                if include_names and file not in include_names:
                    continue
                if exclude_names and file in exclude_names:
                    continue

                file_path = os.path.join(root, file)
                try:
                    with open(file_path, 'r', encoding='utf-8') as f:
                        content = f.read()
                except UnicodeDecodeError:
                    # Skip files that can't be decoded as UTF-8
                    continue

                if skip_empty and not content.strip():
                    continue

                # Print feedback for the user
           print(f"Adding to markdown: {os.path.relpath(file_path, input_dir)}")

                # Do NOT escape underscores for code block headers
           md_file.write(f"## {format_path(file_path, backtick=True, escape_underscore=Fa
                md_file.write(f"```{get_language(file)}\n")
                md_file.write(content)
                md_file.write("\n```\n\n")
```

D:/Programming/doc_complie/code2pdf/parser_utils.py

```python
import argparse
from .constants import DEFAULT_OUTPUT

# Creates and returns the argument parser for the CLI.
# Encapsulates all argument definitions for modularity and reuse.
def create_parser():
    parser = argparse.ArgumentParser(
        description="📄 Convert the current directory's codebase into a syntax-highlighted PDF."
    )
    parser.add_argument(
        "folder", nargs="?", default=".",
        help="Directory to scan (default: current dir)"
    )
    parser.add_argument(
        "-o", "--output", default=DEFAULT_OUTPUT,
        help="Output PDF filename (default: codebase.pdf)"
    )
    parser.add_argument(
        "--title", default="📄 Project Codebase",
        help="Title for the PDF document"
    )
    parser.add_argument(
        "--author", default="Anonymous",
        help="Author name"
    )
    parser.add_argument(
        "--skip-empty", action="store_true",
        help="Skip empty files instead of showing them in the PDF"
    )
    parser.add_argument(
        "--no-toc", action="store_false",
        help="Disable table of contents in the PDF"
    )
    parser.add_argument(
        "--no-highlight", action="store_false",
        help="Disable syntax highlighting in the PDF"
    )
    parser.add_argument(
        "--include", help="Comma-separated list of file extensions to include (e.g. py,js,html)"
    )
```

```
    parser.add_argument(
        "--exclude", help="Comma-separated list of file extensions to exclude (e.g.
         ↪  log,pyc)"
    )
    parser.add_argument(
        "--project-desc", default=None,
        help="Project description to include in the PDF"
    )
    return parser
```

D:/Programming/doc_complie/code2pdf/pdf_utils.py

```
import subprocess
import os
import threading
import sys
import time

# Checks for required external dependencies (pandoc, xelatex).
# Exits the program if any are missing.
def check_dependencies():
    try:
        subprocess.run(["pandoc", "--version"], check=True, stdout=subprocess.PIPE,
 ↪  stderr=subprocess.PIPE)
        subprocess.run(["xelatex", "--version"], check=True,
 ↪  stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    except FileNotFoundError as e:
        print(f"❌ Error: Missing dependency: {e.filename}. Please install it and
         ↪  try again.")
        exit(1)

def _spinner(msg, stop_event):
    spinner = "|/-\\"
    idx = 0
    sys.stdout.write(msg)
    sys.stdout.flush()
    while not stop_event.is_set():
        sys.stdout.write(spinner[idx % len(spinner)])
        sys.stdout.flush()
        time.sleep(0.1)
        sys.stdout.write('\b')
        idx += 1
    sys.stdout.write(' \n')
```

```
    sys.stdout.flush()

# Converts a markdown file to PDF using pandoc and xelatex with the eisvogel
 ↪  template for a professional look.
def convert_to_pdf_pandoc(md_file, output_pdf):
    try:
        # Try to find eisvogel.latex in various locations
        eisvogel_path = os.path.abspath("eisvogel.latex")
        use_eisvogel = True
        if not os.path.isfile(eisvogel_path):
            pandoc_user_template_unix =
 ↪  os.path.expanduser("~/.pandoc/templates/eisvogel.latex")
            pandoc_user_template_win = os.path.join(
                os.environ.get("APPDATA", ""), "pandoc", "templates",
 ↪  "eisvogel.latex"
            )
            if os.path.isfile(pandoc_user_template_unix):
                eisvogel_path = pandoc_user_template_unix
            elif os.path.isfile(pandoc_user_template_win):
                eisvogel_path = pandoc_user_template_win
            else:
                print("⚠ Warning: 'eisvogel.latex' template not found. Falling back
                 ↪  to Pandoc's default template.")
                use_eisvogel = False

        pandoc_cmd = [
            "pandoc", md_file, "-o", output_pdf,
            "--pdf-engine=xelatex",
            "--highlight-style=pygments",
            "--variable", "mainfont=Times New Roman",
            "--variable", "monofont=Consolas",
            "--variable", "geometry:margin=1in",
            "--variable", "colorlinks=true",
            "--variable", "linkcolor=blue",
            "--toc",
            "--toc-depth=3",
        ]
        if use_eisvogel:
            print("Using eisvogel template for PDF generation.")
            pandoc_cmd.insert(pandoc_cmd.index("--highlight-style=pygments"),
 ↪  "--template")
            pandoc_cmd.insert(pandoc_cmd.index("--template") + 1, eisvogel_path)
```

```
        stop_event = threading.Event()
        spinner_thread = threading.Thread(target=_spinner, args=("Generating PDF...
↪   ", stop_event))
        spinner_thread.start()
        try:
            subprocess.run(pandoc_cmd, check=True)
        finally:
            stop_event.set()
            spinner_thread.join()
        print("PDF generation complete.")
    except subprocess.CalledProcessError as e:
        print(f"❌ Error: Failed to convert markdown to PDF using pandoc. {e}")
        exit(1)

# Converts markdown to PDF using reportlab (simple, no highlighting).
# Renders each line of the HTML-converted markdown.
'''
def markdown_to_pdf(md_file, pdf_file):
    try:
        with open(md_file, 'r', encoding='utf-8') as f:
            md_content = f.read()
        html_content = markdown.markdown(md_content)
        c = canvas.Canvas(pdf_file, pagesize=letter)
        text_object = c.beginText(72, 720)
        text_object.setFont("Helvetica", 12)
        for line in html_content.splitlines():
            text_object.textLine(line)
        c.drawText(text_object)
        c.save()
    except Exception as e:
        print(f"❌ Error: Failed to convert markdown to PDF using reportlab. {e}")
        exit(1)
'''
# Converts markdown to PDF using pdfkit (via HTML).
# Reads markdown, converts to HTML, and generates a PDF.
''' def md_to_pdf(md_path, pdf_path):
    try:
        with open(md_path, 'r', encoding='utf-8') as f:
            md_content = f.read()
        html_content = markdown.markdown(md_content)
        pdfkit.from_string(html_content, pdf_path)
    except Exception as e:
        print(f"❌ Error: Failed to convert markdown to PDF using pdfkit. {e}")
```

```
        exit(1)
        '''
```

D:/Programming/doc_complie/test/app.py

```
def sample_function():
    return "Hello, World!"

def test_sample_function():
    assert sample_function() == "Hello, World!"

def test_sample_function_fail():
    assert sample_function() == "Hello, Python!"
```

D:/Programming/doc_complie/test/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    hello world
    This is a test file to check if its working or not.
</body>
</html>
```

D:/Programming/doc_complie/test/run_test.py

```
import sys
import os

# Ensure the parent directory is at the front of sys.path for local imports
sys.path.insert(0, os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from code2pdf.code2pdf import main

if __name__ == "__main__":
```

```
    # Simulate CLI arguments for local code2pdf
    sys.argv = [
        "code2pdf",
        "test",  # folder to scan
        "-o", "test_code2pdf.pdf",
        "--title", "Test Folder Codebase",
        "--author", "Test Runner"
    ]
    main()
```

D:/Programming/doc_complie/test/script.js

```
function test() {
    console.log("test");
}

function test2() {
    console.log("test2");
}
```