ChatGPT Projects CLI Documentation (v5)

Introduction

ChatGPT Projects CLI is a command-line interface (CLI) tool designed to facilitate isolated project management and AI-assisted coding workflows using OpenAI's GPT models. It emulates a "website-parity" experience similar to ChatGPT's web interface but with enhanced project isolation, artifact extraction, and cost/token awareness. The tool ensures strict scoping: GPT interactions are limited to the current project's files, preventing accidental leakage of unrelated data.

Key principles:

- **Isolation**: Each project is a self-contained sandbox with files, artifacts, conversations, and metadata.
- **Transparency**: Token estimation, cost projection, and automatic artifact saving promote efficient usage.
- **Robustness**: Handles API variations (Responses vs. Chat Completions), streaming fallbacks, and error retries.
- **Extensibility**: Supports model switching, with aliases for common GPT variants and live API model listing.

This documentation is technical, focusing on engineering aspects, including token management, pricing calculations, and model integration. It includes diagrams for architectural visualization.

Dependencies:

- openai (core API client)
- python-dotenv (for API key loading)
- tiktoken (optional, for precise token estimation; falls back to heuristic if unavailable)
- Standard libraries: os, json, cmd, shutil, etc.

Install via:

text

```
pip install --upgrade openai python-dotenv tiktoken
```

Set OPENAI_API_KEY in .env or environment.

Run: python ChatGPTProjects_v5.py

Features

- **Project Management**: Create/open projects, add/remove/list files (scoped to /files subdir).
- **Chat Integration**: Send queries to GPT with optional file context inclusion (-n for none, -s for selective).
- **Artifact Extraction**: Automatically detects and saves code blocks from responses as versioned files in /artifacts.
- **Conversation Logging**: Per-chat JSON files + consolidated project history for auditability.
- **Model Flexibility**: Aliases (e.g., gpt5 → gpt-5) and direct API model IDs; temperature control (0.2 default, adjustable).
- **Token & Cost Awareness**: Estimates via tiktoken (or char/4 heuristic); pricing table for USD projections.
- **Streaming Support**: Prefers Responses API for real-time output; auto-falls back to non-streaming or Chat Completions if restricted.
- **Multi-Line Input**: -m flag for extended prompts.
- **Export & Sync**: Export files/artifacts; auto-sync metadata with filesystem.

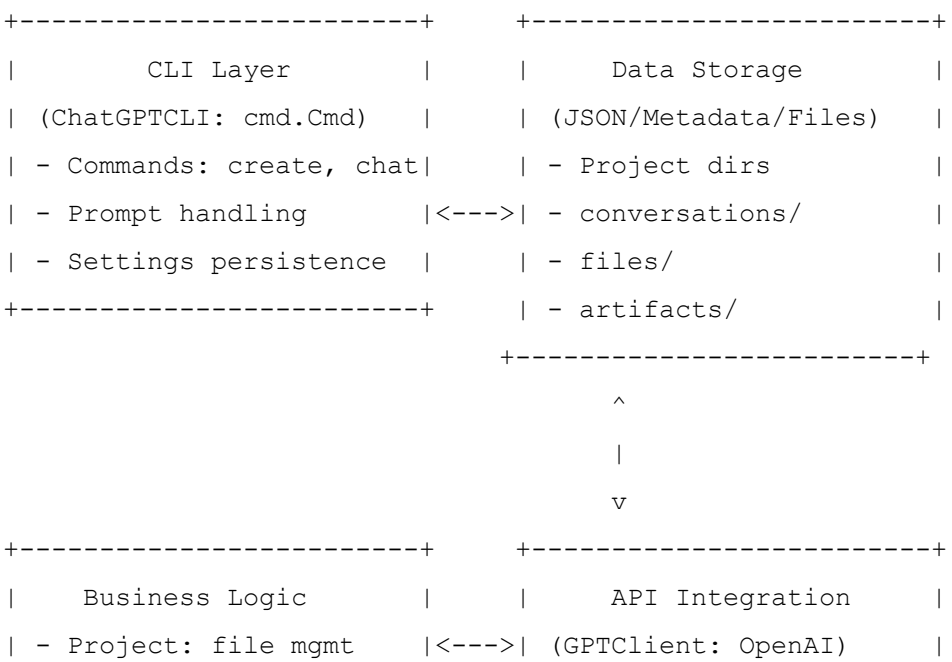
Architecture

The system is built around a modular CLI (cmd.Cmd subclass) interacting with core classes: Project, ArtifactManager, and GPTClient. It uses a REPL-style loop for commands.

High-Level Architecture Diagram

Below is an ASCII art representation of the system's layered architecture:

text



```
| - ArtifactManager: code |     | - Responses/Chat Comps  |
|   extraction/saving     |     | - Streaming fallbacks   |
+-------------------------+     | - Model listing         |
                                +-------------------------+
```

- **CLI Layer**: User-facing REPL with command parsing (e.g., do_chat handles flags like -s).
- **Business Logic**: Manages state (e.g., Project ensures isolation via files_path).
- **API Integration**: Abstracts OpenAI calls, with robust error handling (e.g., no temperature for Responses API).
- **Data Storage**: Filesystem-based persistence (e.g., metadata.json tracks file additions).

Chat Workflow Diagram

A sequence diagram (in Mermaid-like markdown) illustrating a typical chat flow:

GPTClientArtifactManagerProjectChatGPTCLIUserGPTClientArtifactManagerProjectChatGPTCLIUserchat -s file.py -- "Analyze code"get_project_context(selected_files=["file.py"])Context string (file contents)stream(model="gpt-5", system=INSTRUCTIONS, user=Context + Message)Streamed deltas (response text)Print deltas in real-timeextract_code_blocks(response)Save artifacts to /artifactssave_conversation(messages, response, artifacts)Conversation saved

- **Flow Notes**: Context building checks token limits; artifacts are extracted via regex (triple-backticks with language).
- **Error Paths**: If streaming fails (e.g., 400 error), retry non-streaming; if Responses API fails, fallback to Chat Completions.

Project Directory Structure

An example tree diagram (ASCII art):

text

```
ChatGPT_Projects/
├── my_project/                 # Project root
│   ├── files/                  # User-added files (isolated scope)
│   │   ├── script.py           # Added via 'add path/to/script.py'
│   │   └── data.json
│   ├── artifacts/              # Auto-extracted code from responses
│   │   ├── MyClass_20250808.py  # Versioned artifact
│   │   └── utils_20250808.js
│   ├── conversations/          # Per-chat logs
```

```
|   |   ├── conversation_20250808.json
|   |   └── project_history.json # Consolidated history
|   └── metadata.json            # File metadata (added timestamps, sizes)
└── .settings.json               # CLI settings (last project, model)
```

- **Isolation Engineering**: All paths are relative to project_path; no global filesystem access.
- **Sync Mechanism**: sync_files() reconciles metadata with actual files (adds missing, removes deleted).

Usage Guide

Basic Workflow

1. create my_project → Creates isolated dir.
2. add /path/to/file.py → Copies file to /files.
3. files → Lists scoped files.
4. chat "Improve this code" → Sends with all files as context.
5. Response streams; code blocks saved as artifacts.
6. artifacts → View extracted files.
7. history 5 → View last 5 chats in consolidated history.

Advanced Options

- Selective Context: chat -s file1.py,file2.js -- "Refactor these"
- No Context: chat -n "General question"
- Multi-Line: chat -m (type lines, end with EOF).
- Model Switch: model 4o (aliases: gpt5, 4.1, 4o, mini).
- Export: export artifacts ~/backup (copies to path).

Full command help: help in CLI.

Technical Details

Token Limits and Estimation

- **Global Limit**: Set to TOKEN_LIMIT = 200_000 (adjustable; based on GPT-5 context window).
- **Estimation Engine**: Prefers tiktoken for precise encoding (o200k_base for modern models). Fallback: len(text) / 4 heuristic (conservative; assumes ~4 chars/token).
- **Budget Checks**: Before chat, sums context + message + overhead (2000 tokens). If > limit - 5000, aborts. Selective inclusion (-s) allows manual trimming.
- **Overhead Breakdown**:
    - System prompt: ~200 tokens.

    - File wrappers: ~50 tokens per file (e.g., "--- File: name ---").
    - Response buffer: Reserved for output.
- **Visualization**: tokens command outputs a table with per-file tokens, colored icons (🟢 <20k, 🟡 <50k, 🔴 >50k), and total projection.

Engineering Note: Token checks prevent API errors; future enhancements could include auto-truncation.

Pricing Calculations

- **Table**: PRICING_USD_PER_1K dict (input/output rates per model). Example:
    - GPT-5: $0.00 input / $0.00 output (placeholder; update post-release).
    - GPT-4o: $5.00 input / $15.00 output.
- **One-Shot Estimation**: (in_tokens / 1000) * input_rate + (out_tokens / 1000) * output_rate.
    - Defaults: 50% input ratio, 200k total tokens.
- **Hourly Projection**: Assumes throughput (e.g., 30 tokens/sec), balanced input/output: tps * 3600 * (avg_rate / 1000).
- **Command**: pricing --model 4o --estimate-tokens 100000 --in-ratio 0.6 --throughput-tps 50.
- **Accuracy**: Based on estimates; actual costs vary by API. No real-time billing query (API limitation).

Engineering Note: Pricing is static; integrate OpenAI's usage API for post-chat actuals in future versions.

Model Choice and Integration

- **Aliases**: Mapped in MODELS dict (e.g., mini → gpt-4o-mini). Custom IDs allowed via model exact-id.
- **Listing**: models pulls live from client.models.list() (abbreviated to 30 for brevity).
- **API Preference**: Responses API first (for website-like behavior; no temperature sent to avoid 400s). Fallback to Chat Completions if unavailable or restricted.
- **Temperature**: 0.2 default (creativity balance); sent only to Chat Completions.
- **Streaming Logic**:
    - Try Responses streaming.
    - On failure (e.g., "stream not allowed"), retry non-stream.
    - On content-type errors, switch to instructions param.
    - Ultimate fallback: Chat Completions (streaming or non).
- **System Prompt**: Fixed (SYSTEM_INSTRUCTIONS + website-style guidelines) for consistent engineering-focused responses.

Engineering Note: Robustness via try-except chains; handles org-level restrictions (e.g., no streaming for some plans).

Limitations & Tips

- **Limits**: No image/multimodal support (text-only). Token overflows require manual intervention.
- **Performance**: Streaming may lag on slow networks; non-stream fallback ensures completion.
- **Security**: Files are copied (not linked); avoid sensitive data.
- **Tips**:
    - Monitor tokens before large chats.
    - Use -s for efficiency in big projects.
    - Update pricing table for new models.
    - For artifacts: Regex-based extraction skips small snippets (<100 chars).
- **Extending**: Add custom commands by subclassing ChatGPTCLI; integrate other LLMs via adapter pattern.

This tool bridges CLI simplicity with AI power, engineered for reliability in coding workflows. Contributions welcome!