DocCompiler.ai

ADMINISTRATOR & DEVELOPER GUIDE

Codex Administrator & Developer Guide

Architecture, Configuration, Security & Developer Reference

A Cyber Panda Solutions LLC product

Version 3.1 2026-05-13 @ f017a23 Generated by DocCompiler.ai

Always-updated version at doccompiler.ai/openai/codex

Source code analysis by Claude Opus 4.6 | 1M token context

About This Document

What Is This?

This is a comprehensive, source-code-verified **Administrator & Developer Guide** for **Codex**, generated by DocCompiler.ai using Claude Opus 4.6 with 1M-token context. Every command, configuration option, and behavior documented here was verified against the actual source code.

The Definitive Reference

This document is designed to be the single source of truth for Codex. Rather than piecing together scattered README fragments, outdated wiki pages, forum posts, and version-specific screenshots, you can rely on this as a verified, structured reference backed by automated code review.

Optimized for AI Assistants

This document is structured for consumption by AI assistants. Feed this PDF into your preferred LLM for accurate, grounded answers about Codex — instead of having the model guess or hallucinate from incomplete context.

Version & Updates

Generated on **2026-05-13** (version **3.1**). This document reflects the source code at commit `f017a23`. The always-updated version is permanently available at **doccompiler.ai/openai/codex**. Do not print or download for offline use — this document is updated when the source code changes.

Our Mission

DocCompiler.ai aims to be the definitive wiki for the world's most important open-source projects. Every document is overkill by design: exhaustive, fact-checked, and built to replace the scattered tribal knowledge that slows teams down.

Table of Contents

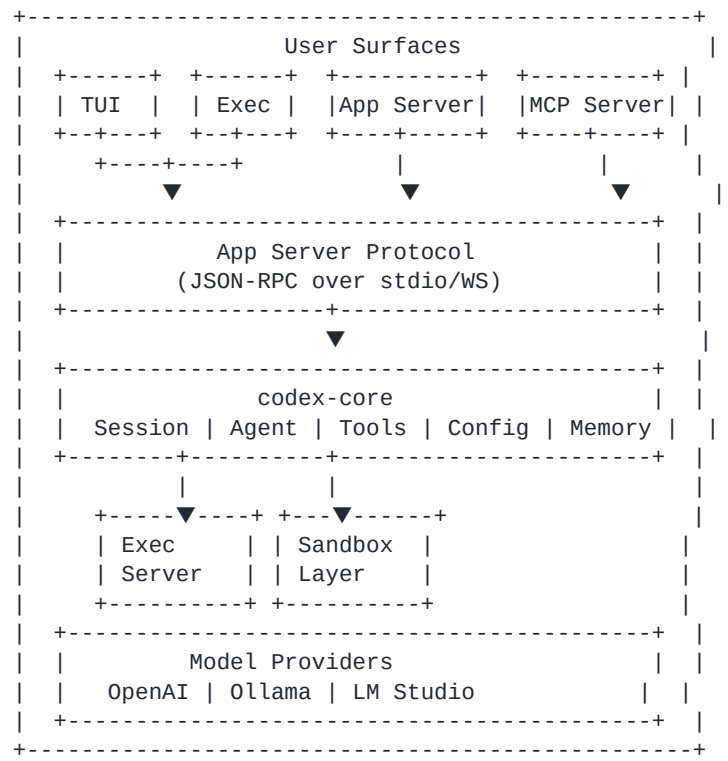
OpenAI Codex -- Administrator & Developer Guide

1. Architecture Overview

OpenAI Codex is a locally-running agentic coding assistant built primarily in Rust, with TypeScript and Python SDK wrappers. The system is structured as a multi-crate Cargo workspace (`codex-rs/`) containing 100+ crates, with a thin npm CLI wrapper (`codex-cli/`) and SDKs (`sdk/typescript`, `sdk/python`).

The architecture follows a layered model: a **core agent engine** processes prompts through the OpenAI Responses API, a **sandboxed execution layer** runs commands and applies file changes, a **protocol layer** defines the typed message contract between components, and multiple **surface layers** (TUI, CLI exec, app server, MCP server) expose the agent to different contexts.

text



Component Inventory

Component	Crate Path	Role
codex-core	`codex-rs/core`	Central agent engine: session management, tool dispatch, memory pipeline, plugin system, config loading

codex-tui	`codex-rs/tui`	Ratatui-based terminal UI: chat widget, approval dialogs, streaming, session history
codex-cli	`codex-rs/cli`	CLI entry point: argument parsing, subcommand dispatch
codex-exec	`codex-rs/exec`	Non-interactive batch mode with JSONL output
codex-app-server	`codex-rs/app-server`	WebSocket/stdio server bridging UI layers to core
codex-app-server-protocol	`codex-rs/app-server-protocol`	Shared protocol types (JSON-RPC, generated TypeScript bindings)
codex-app-server-client	`codex-rs/app-server-client`	Client library for connecting to the app server
codex-exec-server	`codex-rs/exec-server`	Sandboxed command execution service
codex-protocol	`codex-rs/protocol`	Core protocol types: sessions, approvals, events, models, permissions
codex-config	`codex-rs/config`	TOML config schema, config layering, requirements enforcement
codex-sandboxing	`codex-rs/sandboxing`	Cross-platform sandbox abstraction
codex-linux-sandbox	`codex-rs/linux-sandbox`	Linux Landlock + bubblewrap + seccomp sandbox
codex-windows-sandbox-rs	`codex-rs/windows-sandbox-rs`	Windows restricted-token sandbox with ACL management
codex-mcp-server	`codex-rs/mcp-server`	Codex as an MCP server (stdio transport)
codex-codex-mcp	`codex-rs/codex-mcp`	MCP client: connects to external MCP servers
codex-rmcp-client	`codex-rs/rmcp-client`	Low-level MCP client with OAuth support
codex-login	`codex-rs/login`	Authentication: ChatGPT OAuth, API keys, keyring/file storage
codex-hooks	`codex-rs/hooks`	Event hook system: pre/post tool use, session lifecycle
codex-execpolicy	`codex-rs/execpolicy`	Starlark-based execution policy engine
codex-execpolicy-legacy	`codex-rs/execpolicy-legacy`	Legacy execution policy checker
codex-apply-patch	`codex-rs/apply-patch`	Custom structured patch format applicator
codex-state	`codex-rs/state`	SQLite-backed state DB (WAL mode, schema migrations)

codex-thread-store	`codex-rs/thread-store`	Conversation thread persistence (local + remote)
codex-rollout	`codex-rs/rollout`	Session/rollout tracking for memory pipeline
codex-models-manager	`codex-rs/models-manager`	Model catalog, collaboration mode presets
codex-model-provider	`codex-rs/model-provider`	Model provider abstraction layer
codex-model-provider-info	`codex-rs/model-provider-info`	Provider metadata and capability info
codex-ollama	`codex-rs/ollama`	Ollama local model provider integration
codex-lmstudio	`codex-rs/lmstudio`	LM Studio local model provider integration
codex-cloud-tasks	`codex-rs/cloud-tasks`	Cloud-hosted agent task infrastructure
codex-cloud-tasks-client	`codex-rs/cloud-tasks-client`	HTTP client for cloud tasks API
codex-skills	`codex-rs/skills`	Bundled skills and sample agents
codex-core-skills	`codex-rs/core-skills`	Remote skill fetching and installation
codex-core-plugins	`codex-rs/core-plugins`	Plugin marketplace and lifecycle
codex-plugin	`codex-rs/plugin`	Plugin interface definitions
codex-connectors	`codex-rs/connectors`	HTTP connectors with caching
codex-network-proxy	`codex-rs/network-proxy`	Managed network proxy for sandbox traffic
codex-secrets	`codex-rs/secrets`	Secret detection and sanitization
codex-analytics	`codex-rs/analytics`	Telemetry and analytics
codex-otel	`codex-rs/otel`	OpenTelemetry integration
codex-feedback	`codex-rs/feedback`	User feedback submission

codex-git-utils	`codex-rs/git-utils`	Git operations (apply, diff, worktree)
codex-file-search	`codex-rs/file-search`	Fast file search across directories
codex-tools	`codex-rs/tools`	Shared tool schema and Responses API primitives
codex-features	`codex-rs/features`	Feature flag management
codex-arg0	`codex-rs/arg0`	Multi-binary dispatch via `argv[0]`
codex-instructions	`codex-rs/instructions`	User instruction loading (AGENTS.md)
codex-responses-api-proxy	`codex-rs/responses-api-proxy`	Reverse proxy for OpenAI Responses API
codex-chatgpt	`codex-rs/chatgpt`	ChatGPT integration and `apply` command
codex-backend-client	`codex-rs/backend-client`	Backend API client
SDK (TypeScript)	`sdk/typescript`	`@openai/codex-sdk`: async thread/turn API over CLI subprocess
SDK (Python)	`sdk/python`	Runtime bootstrap: installs platform-specific CLI binary

Technology Stack

Layer	Technology
Core Language	Rust (2021 edition)
Build System	Cargo workspace + Bazel (hybrid)
TUI Framework	Ratatui
CLI Parser	clap (derive)
Async Runtime	Tokio
Database	SQLite (via rusqlite, WAL mode)
Serialization	serde + serde_json + toml
HTTP	reqwest + hyper
WebSocket	tokio-tungstenite
TLS	rustls
Sandbox (Linux)	Landlock LSM + bubblewrap + seccomp

Sandbox (macOS)	Seatbelt (sandbox-exec)
Sandbox (Windows)	Restricted tokens + ACLs + ConPTY
MCP	rmcp crate (Model Context Protocol)
Policy Engine	Starlark (execution policy rules)
OpenTelemetry	codex-otel crate
TypeScript SDK	Node.js + tsup
Python SDK	pip + GitHub Releases
Package Distribution	npm (CLI + proxy + SDK), standalone installers
License	Apache-2.0

> **Note**
>
> The `codex-rs/arg0` crate enables a single binary to serve multiple entry points. The binary detects how it was invoked by name (e.g., `codex`, `codex-exec`, `codex-linux-sandbox`, `codex-mcp-server`) and dispatches to the correct sub-entry-point. This powers the platform-specific sandbox helpers without requiring separate binaries.

Data Flow: Interactive Session

1. User launches `codex` -> `codex-cli` parses args -> dispatches to `codex-tui`
2. TUI initializes `codex-app-server` (in-process or via stdio/WebSocket)
3. App server creates a `codex-core` session with the user's config
4. User types a prompt -> app server forwards to core session
5. Core session sends the prompt to the OpenAI Responses API
6. Model response may include tool calls (shell, apply-patch, MCP, web search)
7. For shell commands: `codex-exec-server` executes inside `codex-sandboxing`
8. Approval flow: TUI presents approval dialog -> user accepts/declines
9. Tool output feeds back into the conversation -> model continues
10. Session state persists to `codex-state` SQLite DB

2. System Requirements

Runtime Requirements

Component	Requirement
OS	macOS 12+ (Intel/Apple Silicon), Linux (kernel 5.13+ for Landlock), Windows 10+ (x64/arm64)

RAM	512 MB minimum for CLI; more for large codebases
Disk	~100 MB for binary + config/state
Network	Outbound HTTPS to `api.openai.com` (or configured provider)
Git	Optional but recommended for code review, session tracking
Node.js	16+ (npm distribution only)

Build Requirements

Component	Requirement
Rust	Stable toolchain (edition 2021)
Cargo	Workspace-aware build
Bazel	Optional (used for some CI targets)
Node.js	16+ (for TypeScript SDK/protocol generation)
pnpm	10.x (package management)
Python	3.x (for Python SDK, install scripts)
Platform SDKs	Windows SDK (for `codex-windows-sandbox-rs`), Xcode CLT (macOS)

> **Important**
>
> The Linux sandbox requires kernel 5.13+ for Landlock LSM support. On older kernels, the sandbox falls back to a more permissive mode with a warning. WSL1 is explicitly unsupported -- WSL2 is required.

3. Configuration Reference

Config File Locations

File	Location	Purpose
User config	`~/.codex/config.toml`	Personal settings, model choice, sandbox mode
Project config	`.codex/config.toml`	Project-specific overrides
Managed config	`managed_config.toml` (system path)	Enterprise/MDM-managed settings
Requirements	`requirements.toml` (system path)	Enforced policy constraints
Auth credentials	`~/.codex/auth.json` or keyring	OAuth tokens, API keys
State DB	`~/.codex/` (SQLite)	Session history, memory, rollouts

Config Layering (Precedence: highest wins)

1. **CLI/Session overrides** (`-c key=value`) -- highest precedence
2. **MDM managed preferences** (macOS only)
3. **System managed config** (`managed_config.toml`)
4. **Project config** (`.codex/config.toml`)
5. **User config** (`~/.codex/config.toml`) -- lowest precedence

> **Note**
>
> The merge strategy is "first writer wins" via `merge_unset_fields` -- higher-precedence layers fill in values, lower layers only fill gaps. The `apps` field uses a special union-with-disable-propagation: if any layer sets `enabled = false` for an app, it stays disabled regardless of higher layers.

Complete Configuration Keys

Core Settings

Key	Type	Default	Description
`model`	String	(server)	Model selection (e.g., `"gpt-5"`, `"gpt-4o"`)
`review_model`	String	--	Separate model for `/review`
`model_provider`	String	--	Provider key from `model_providers` map
`model_context_window`	Integer	--	Context window size in tokens
`model_auto_compact_token_limit`	Integer	--	Token threshold for auto-compaction
`model_reasoning_effort`	String	--	`"minimal"`, `"low"`, `"medium"`, `"high"`, `"xhigh"`
`plan_mode_reasoning_effort`	String	--	Reasoning effort for plan mode
`model_reasoning_summary`	String	--	Reasoning summary mode
`model_verbosity`	String	--	Responses API verbosity
`model_supports_reasoning_summaries`	Boolean	--	Force-enable reasoning summaries
`model_catalog_json`	Path	--	Path to JSON model catalog (startup-only)
`model_instructions_file`	Path	--	Override built-in model instructions
`personality`	String	--	Model personality preset
`service_tier`	String	--	`"fast"` or `"flex"`

Approval and Sandbox

Key	Type	Default	Description
`approval_policy`	String	`"on-request"`	`"never"`, `"on-request"`, `"on-failure"`, `"untrusted"`
`approvals_reviewer`	String	`"user"`	`"user"` or `"guardian_subagent"`
`sandbox_mode`	String	`"read-only"`	`"read-only"`, `"workspace-write"`, `"danger-full-access"`
`default_permissions`	String	--	Named permissions profile key

Sandbox Workspace-Write Settings

toml

```toml
[sandbox_workspace_write]
writable_roots = ["/path/to/extra/dir"]
network_access = true
exclude_tmpdir_env_var = false
exclude_slash_tmp = false
```

MCP Server Configuration

toml

```toml
[mcp_servers.my-server]
# Stdio transport
command = "node"
args = ["path/to/server.js"]

# OR HTTP transport
# url = "https://mcp.example.com"
```

Key	Type	Default	Description
`mcp_oauth_credentials_store`	String	`"auto"`	`"keyring"`, `"file"`, `"auto"`
`mcp_oauth_callback_port`	Integer	(ephemeral)	Fixed OAuth callback port
`mcp_oauth_callback_url`	String	--	Override OAuth redirect URI

Model Providers

toml

```toml
[model_providers.my-provider]
# Custom provider configuration
# Built-in IDs (openai, ollama, lmstudio) cannot be overridden
```

Key	Type	Default	Description
`openai_base_url`	String	--	Override OpenAI API base URL

`chatgpt_base_url`	String	--	Override ChatGPT base URL
`oss_provider`	String	--	Preferred local provider: `"lmstudio"` or `"ollama"`

Agent Configuration

```toml
[agents]
max_threads = 4
max_depth = 3
job_max_runtime_seconds = 300
```

Web Search

Key	Type	Default	Description
`web_search`	String	`"disabled"`	`"disabled"`, `"cached"`, `"live"`

Session and History

Key	Type	Default	Description
`instructions`	String	--	System instructions injected into every session
`developer_instructions`	String	--	Developer-role instructions
`compact_prompt`	String	--	Custom prompt for history compaction
`commit_attribution`	String	--	Co-author trailer text (empty = disable)
`project_root_markers`	Array	`[".git"]`	Markers for project root detection
`sqlite_home`	Path	--	SQLite DB directory override
`log_dir`	Path	`$CODEX_HOME/log`	Log directory
`project_doc_max_bytes`	Integer	--	Max bytes from AGENTS.md
`tool_output_token_limit`	Integer	--	Token budget for tool outputs
`background_terminal_max_timeout`	Integer	300000	Background terminal timeout (ms)

Authentication

Key	Type	Default	Description
`auth_credentials_store_mode`	String	`"auto"`	`"auto"`, `"keyring"`, `"file"`, `"ephemeral"`

Analytics and Telemetry

toml

```toml
[analytics]
enabled = true

[feedback]
enabled = true

[otel]
# OpenTelemetry configuration

check_for_update_on_startup = true
```

Named Profiles

toml

```toml
[profiles.fast]
model = "gpt-4o"
service_tier = "fast"

[profiles.secure]
sandbox_mode = "read-only"
approval_policy = "untrusted"
```

Activate with `profile = "fast"` or `--profile fast`.

Windows-Specific

toml

```toml
[windows]
# Windows-specific settings
```

Audio (Realtime)

toml

```toml
[audio]
microphone = "default"
speaker = "default"
```

Environment Variables

Variable	Type	Description
`CODEX_HOME`	Path	Config/data directory (default: `~/.codex`, Windows: `%APPDATA%\codex`)
`OPENAI_API_KEY`	String	API key for authentication
`CODEX_SANDBOX`	String	Override sandbox mode
`CODEX_APPLY_GIT_CFG`	String	Git configuration for apply operations
`CODEX_TUI_RECORD_SESSION`	Boolean	Enable session recording

`CODEX_TUI_SESSION_LOG_PATH`	Path	Session log output path
`CODEX_TUI_ROUNDED`	Boolean	Use rounded corners in TUI
`CODEX_OSS_PORT`	Integer	Port for OSS model provider
`CODEX_OSS_BASE_URL`	String	Base URL for OSS model provider
`CODEX_CLOUD_TASKS_MODE`	String	Cloud tasks mode override
`CODEX_CLOUD_TASKS_BASE_URL`	String	Cloud tasks API URL override
`CODEX_STARTING_DIFF`	String	Pre-load diff for cloud tasks
`EDITOR` / `VISUAL`	String	External editor for TUI
`HOME`	Path	Home directory (LM Studio config resolution)
`SBX_DEBUG`	Boolean	Enable sandbox debug logging (Windows)

> **Important**
>
> `CODEX_HOME` determines the root for all persistent state: config, auth credentials, SQLite databases, session logs, and plugin cache. Changing it redirects all state.

Requirements Enforcement

The `requirements.toml` file (deployed via MDM or system management) constrains which config values are valid:

```toml
allowed_sandbox_modes = ["read-only", "workspace-write"]
allowed_approval_policies = ["on-request", "untrusted"]
allowed_approvals_reviewers = ["user"]
```

> **Caution**
>
> Any `allowed_sandbox_modes` list **must include** `read-only` or the config is rejected at startup. This is a safety invariant enforced in the config loader.

4. Deployment & Scaling

Distribution Methods

Method	Target	Artifact
npm	All platforms	`@openai/codex` (meta-package + platform binaries)

Standalone installer	macOS/Linux	`install.sh` -> versioned releases under `~/.codex/`
Standalone installer	Windows	`install.ps1` -> `%LOCALAPPDATA%\Programs\OpenAI\Codex\`
Python SDK	All platforms	`@openai/codex-sdk` (pip, downloads Rust binary from GitHub Releases)
TypeScript SDK	All platforms	`@openai/codex-sdk` (npm, wraps CLI subprocess)

npm Package Structure

The `@openai/codex` npm package is a meta-package that depends on platform-specific optional packages:

- `@openai/codex-linux-x64`
- `@openai/codex-linux-arm64`
- `@openai/codex-darwin-x64`
- `@openai/codex-darwin-arm64`
- `@openai/codex-win32-x64`
- `@openai/codex-win32-arm64`

The staging script (`scripts/stage_npm_packages.py`) downloads native artifacts, hydrates `vendor/` for each package, and writes tarballs to `dist/npm/`.

Enterprise Deployment

For managed environments, deploy configuration via:

1. **Managed config file:** Place `managed_config.toml` at the system config path
2. **MDM (macOS):** Deploy managed preferences via MDM profile
3. **Requirements:** Deploy `requirements.toml` to enforce policy constraints (allowed sandbox modes, approval policies, etc.)
4. **Environment variables:** Set `CODEX_HOME` and `OPENAI_API_KEY` via system environment

App Server Modes

The app server (`codex-app-server`) supports three transport modes:

Mode	URL	Use Case
stdio	`stdio://` (default)	IDE extensions, local embedding
WebSocket	`ws://IP:PORT`	Remote connections, multi-client
Off	`off`	Disable transport (embedding-only)

bash

```
# Run as stdio server (for IDE extensions)
codex app-server --listen stdio://

# Run as WebSocket server
```

```
codex app-server --listen ws://0.0.0.0:4222

# With authentication
codex app-server --listen ws://0.0.0.0:4222 --auth-token-env MY_SECRET
```

> **Warning**
>
> WebSocket transport without authentication exposes the agent to the network. Always use `--auth-token-env` in production.

5. Operations

Session Management

Sessions are persisted in a SQLite database (WAL mode) under `CODEX_HOME`. Each session is a "thread" with:

- Unique thread ID
- Conversation history
- Working directory context
- Git info (branch, repo)
- Token usage counters

List sessions:

bash

```
codex resume --all
```

Clear memories:

bash

```
codex debug clear-memories
```

Memory Pipeline

Codex implements a two-phase memory pipeline that extracts and consolidates learnings across sessions:

Phase 1 (Rollout Extraction): Runs at session start. Claims recent rollouts from the state DB, sends each to the model for structured memory extraction, stores outputs as stage-1 records.

Phase 2 (Global Consolidation): Serialized -- only one consolidation runs at a time. Loads stage-1 outputs, computes a diff against the previous selection, syncs `raw_memories.md` and `rollout_summaries/` under the memories root, then spawns a consolidation sub-agent.

> **Note**
>
> The memory pipeline only runs for non-ephemeral, non-sub-agent root sessions with memory enabled and a state DB available.

Monitoring and Health

Session recording:

```bash
CODEX_TUI_RECORD_SESSION=1 codex
```

Session logs are written to `CODEX_HOME/log/` or the path specified by `CODEX_TUI_SESSION_LOG_PATH`.

Update checking: Codex checks for updates at startup by default. Disable with:

```toml
check_for_update_on_startup = false
```

Sandbox debug logging (Windows):

```bash
SBX_DEBUG=1 codex
```

> **Tip**
> On macOS, use `codex sandbox macos --log-denials -- <command>` to capture and print Seatbelt sandbox denial events after the command exits.

Token Usage Tracking

Token usage is tracked per session and reported on exit. The TUI shows live rate limit snapshots in the status bar. Configure compaction thresholds to manage token budgets:

```toml
model_auto_compact_token_limit = 50000
tool_output_token_limit = 10000
```

6. Performance

Context Window Management

Codex automatically compacts conversation history when approaching the context window limit. The `model_auto_compact_token_limit` setting controls the threshold.

Manual compaction is available in the TUI as the `/compact` command.

Background Terminal Timeout

Long-running background commands are terminated after `background_terminal_max_timeout` (default: 300000 ms / 5 minutes). Adjust for CI or build tasks:

```toml
background_terminal_max_timeout = 600000  # 10 minutes
```

Agent Concurrency

Multi-agent orchestration is controlled by:

```toml
[agents]
max_threads = 4       # Max concurrent agent threads
max_depth = 3         # Max agent nesting depth
job_max_runtime_seconds = 300  # Max runtime per agent job
```

Model Provider Latency

- `service_tier = "fast"` routes to lower-latency infrastructure
- `service_tier = "flex"` routes to higher-availability but potentially slower infrastructure
- Local providers (`ollama`, `lmstudio`) eliminate network latency but depend on local GPU

7. Security

Sandbox Architecture

Codex implements defense-in-depth sandboxing across all supported platforms:

Linux (Landlock + bubblewrap + seccomp):

- Creates isolated user namespaces via bubblewrap (`--unshare-user --unshare-net`)
- Applies Landlock LSM rules for filesystem access control
- Seccomp filters restrict system calls
- Network isolation via namespace separation; managed proxy for approved traffic
- Falls back to vendored bubblewrap if system copy is not available
- WSL1 is explicitly blocked (cannot create user namespaces)

macOS (Seatbelt):

- Uses `sandbox-exec` with custom Seatbelt profiles
- Filesystem restrictions based on sandbox mode
- Network control per sandbox policy

Windows (Restricted Tokens + ACLs):

- Creates restricted process tokens with dropped privileges
- ACL-based filesystem access control via Windows security APIs
- ConPTY for terminal interaction within sandbox
- Three levels: `disabled` (default), `restricted_token`, `elevated`

> **Caution**
> On Windows without the experimental sandbox enabled, `workspace-write` mode is silently downgraded to `read-only`. Check the TUI status bar to verify the effective sandbox level.

Execution Policy

The `codex-execpolicy` crate implements a Starlark-based policy engine that determines which commands are auto-approved vs. escalated for user approval. Policies are defined as Starlark scripts with pattern matching rules.

bash

```
# Check a command against policy rules
codex execpolicy check -r rules.star -- bash -c "rm -rf /"
```

Secret Detection

The `codex-secrets` crate provides regex-based secret detection and sanitization. Patterns are compiled at initialization and applied to:

- Memory pipeline outputs (Phase 1 extraction)
- Tool outputs before display
- Session logs

Authentication Security

Method	Storage	Notes
ChatGPT OAuth	Keyring (preferred) or `auth.json`	Browser-based flow with PKCE
Device Code	Keyring or `auth.json`	For headless environments
API Key	Keyring or `auth.json`	Piped from stdin, never as CLI arg
MCP OAuth	Per `mcp_oauth_credentials_store`	Supports keyring, file, or auto

> **Warning**
>
> The login flow writes a debug log to `CODEX_HOME/log/codex-login.log` with mode `0600` on Unix. Ensure this directory has appropriate permissions.

Network Security

When sandbox mode restricts network access, Codex uses a managed network proxy (`codex-network-proxy`) that:

- Generates per-session CA certificates for MITM inspection
- Applies domain-level allow/deny rules
- Supports HTTP, HTTPS (CONNECT), SOCKS5 protocols
- Routes traffic through the proxy only -- direct connections are blocked in the sandbox namespace

Guardian Auto-Reviewer

The Guardian is an experimental automated approval system (`approvals_reviewer = "guardian_subagent"`) that:

- Spawns a sub-agent to assess risk for each approval request
- Classifies risk as Low/Medium/High/Critical
- Evaluates user authorization level
- Auto-approves low-risk operations
- Escalates high-risk operations to the user
- Provides rationale for each decision

8. Developer Internals

Repository Structure

text

```
openai/codex/
|-- codex-rs/                    # Rust workspace (100+ crates)
|   |-- cli/src/                 # CLI entry point and subcommands
|   |-- tui/src/                 # Terminal UI (Ratatui)
|   |-- core/                    # Agent engine, session, tools, config
|   |   |-- src/                 # Core logic
|   |   |-- templates/           # Prompt templates (memories, review, etc.)
|   |   +-- tests/               # Integration tests
|   |-- app-server/src/          # WebSocket/stdio app server
|   |-- app-server-protocol/     # Protocol types + generated TS bindings
|   |   +-- schema/              # JSON Schema + TypeScript exports
|   |-- exec/src/                # Non-interactive exec mode
|   |-- exec-server/src/         # Sandboxed command execution service
|   |-- protocol/src/            # Core protocol types
|   |-- config/src/              # TOML config schema + layering
|   |-- sandboxing/src/          # Cross-platform sandbox abstraction
|   |-- linux-sandbox/src/       # Linux sandbox (Landlock/bwrap)
|   |-- windows-sandbox-rs/src/  # Windows sandbox
|   |-- mcp-server/src/          # Codex as MCP server
|   |-- codex-mcp/src/           # MCP client
|   |-- rmcp-client/src/         # Low-level MCP client
|   |-- hooks/src/               # Event hook system
|   |-- execpolicy/src/          # Starlark execution policy
|   |-- apply-patch/src/         # Structured patch applicator
|   |-- state/src/               # SQLite state DB
|   |-- login/src/               # Authentication
|   |-- models-manager/src/      # Model catalog
|   |-- network-proxy/src/       # Managed network proxy
|   |-- secrets/src/             # Secret detection
|   |-- skills/src/              # Skills framework
|   |-- tools/src/               # Shared tool primitives
|   +-- ...                      # 60+ utility crates
|-- codex-cli/                   # npm CLI wrapper (JS shim)
|-- sdk/
|   |-- typescript/src/          # TypeScript SDK
|   +-- python/                  # Python SDK (runtime bootstrap)
|-- scripts/
|   |-- install/                 # Standalone installers
|   +-- stage_npm_packages.py    # npm packaging
|-- patches/                     # Vendored patches
+-- tools/                       # Dev tools (argument-comment-lint)
```

Core Module Reference

codex-core (`codex-rs/core/src/`)

The central engine with 116 files and ~1.5M chars. Key internal modules:

- `session/` -- Session lifecycle, turn management, agent orchestration
- `agent/` -- Agent identity, builtins
- `tools/handlers/` -- Tool dispatch: apply_patch, shell exec, MCP, multi-agent, file search
- `config/` -- Runtime config processing (14 files, 533K chars)
- `config_loader/` -- Multi-layer config loading with merge and fingerprinting
- `memories/` -- Two-phase memory pipeline (extraction + consolidation)
- `guardian/` -- Automated approval assessment
- `context_manager/` -- Context window management
- `sandboxing/` -- Core sandbox coordination
- `plugins/` -- Plugin loading and marketplace integration
- `instructions/` -- AGENTS.md and user instruction loading
- `unified_exec/` -- Unified command execution abstraction
- `state/` -- State management helpers

codex-tui (`codex-rs/tui/src/`)

The TUI with 66 files and ~2.2M chars. Key modules:

- `app/` -- Application state machine, event loop, approval handling
- `chatwidget/` -- Chat input/display component
- `bottom_pane/` -- Approval dialogs, MCP elicitation, selection views
- `streaming/` -- Smooth per-frame streaming animation
- `render/` -- Terminal rendering pipeline
- `markdown_stream/` -- Incremental markdown parser
- `onboarding/` -- First-run experience and auth flows
- `history_cell/` -- Conversation history display
- `status/` -- Status bar with rate limits

codex-protocol (`codex-rs/protocol/src/`)

Protocol types with 28 files and ~553K chars:

- `config_types.rs` -- Sandbox policy, approval policy, network policy types
- `models.rs` -- Response items, content items, tool definitions
- `permissions.rs` -- Permission system types and resolution
- `protocol.rs` -- Session events, turn events, streaming events
- `items.rs` -- Conversation item types
- `prompts/` -- Base instructions, sandbox mode prompts, permission prompts

codex-config (`codex-rs/config/src/`)

Configuration with 27 files and ~266K chars:

- `config_types.rs` -- `ConfigToml` struct with all config fields
- `config_requirements.rs` -- Requirements enforcement and validation
- `config_loader/` -- Multi-layer loading: `state.rs`, `layer_io.rs`, `overrides.rs`, `merge.rs`, `fingerprint.rs`, `macos.rs`

Hook System

Six hook event types provide extensibility at key lifecycle points:

Event	Trigger	Key Output Fields
`SessionStart`	New session begins	`additional_context`
`PreToolUse`	Before tool execution	`decision` (approve/block), `updated_input`, `reason`
`PostToolUse`	After tool execution	`decision` (block), `additional_context`
`PermissionRequest`	Permission escalation	`behavior` (allow/deny)
`UserPromptSubmit`	User submits prompt	`decision` (block), `additional_context`
`Stop`	Session ends	`additional_context`

All hooks share a common output wire format:

- `continue` (bool, default true) -- whether to proceed
- `stop_reason` -- human-readable abort reason
- `suppress_output` -- hide tool output from transcript
- `system_message` -- inject text into model context

Hooks are dispatched sequentially. If any hook returns `should_abort_operation() == true`, the chain short-circuits.

Apply-Patch Format

Codex uses a custom structured patch format (not unified diff). The `codex-apply-patch` crate handles parsing and application with operations:

- **AddFile** -- create a new file
- **UpdateFile** -- modify an existing file with context-aware hunks
- **DeleteFile** -- remove a file
- **MoveFile** -- rename/move a file (with optional content update)

Each operation is verified against the current filesystem state before application.

Code Patterns and Conventions

- **Error handling:** `anyhow::Result` for application errors, `thiserror` for library errors
- **Async:** Tokio-based async throughout; `async_trait` for trait async methods
- **Testing:** `#[cfg(test)]` modules with `#[path = "foo_tests.rs"]` sibling pattern
- **Serialization:** `serde` derive with `#[serde(rename_all = "snake_case")]` convention
- **Config:** TOML with `Option<T>` for all fields (no hardcoded defaults in the struct)
- **CLI:** clap derive with `#[arg(...)]` and `#[command(...)]` attributes
- **Workspace dependencies:** All external deps declared in root `Cargo.toml` `[workspace.dependencies]`
- **Build:** Hybrid Cargo + Bazel; `BUILD.bazel` files alongside Cargo.toml
- **Linting:** Workspace-level lint configuration via `[lints]` section
- **Cross-platform:** `#[cfg(target_os = "...")]` conditionals for platform-specific code

Testing Strategy

- **Unit tests:** Sibling `*_tests.rs` files with `#[cfg(test)]` modules
- **Integration tests:** `tests/` directories in crates like `core`, `app-server`, `exec-server`, `mcp-server`
- **Snapshot tests:** Used in TUI (rendering), protocol (serialization), and guardian (assessment)
- **End-to-end:** `codex-app-server-test-client` crate provides a full test harness
- **Apply-patch fixtures:** `codex-rs/apply-patch/tests/fixtures/scenarios/` contains 22+ scenario directories with input/expected/patch triplets
- **Mock model server:** Test infrastructure includes mock OpenAI Responses API servers

SDK Architecture

TypeScript SDK (`@openai/codex-sdk`):

- Wraps the Codex CLI binary as a subprocess
- Runs `codex exec --experimental-json` for JSONL streaming
- Parses JSONL events into typed `ThreadItem` and `ThreadEvent` objects
- Supports `AbortSignal` for cancellation
- Config overrides flattened to TOML dotted-key CLI flags

Python SDK (`sdk/python`):

- Runtime bootstrap only -- not a user-facing API
- Downloads platform-specific Codex binary from GitHub Releases
- Installs as `codex-cli-bin` pip package
- Pinned to specific version (`0.116.0-alpha.1` in source)
- Fallback chain: direct URL -> GitHub API with token -> `gh` CLI

Potential Improvements

Based on code analysis:

- `codex-tools` **extraction:** The `codex-tools` crate is intentionally minimal -- extracting tool primitives from `codex-core` in reviewable increments. Full migration is planned but not yet complete.

- **Config layer complexity:** The 5-layer config merge with special `apps` union semantics and requirements enforcement is powerful but complex. Consider documenting the effective config resolution path for debugging.
- **Windows sandbox maturity:** Windows sandbox defaults to `disabled`. The `restricted_token` and `elevated` levels are functional but less tested than Linux/macOS equivalents.
- **Memory pipeline observability:** Phase 2 consolidation runs a sub-agent with no approvals and local write access. Consider adding observability hooks for enterprise deployments.
- **Apply-patch error recovery:** Partial application failures leave changes in place (`015_failure_after_partial_success_leaves_changes`). Consider a rollback-on-failure mode.

Contributing

The project uses Apache-2.0 license. Key conventions for contributors:

1. **Tests:** Place unit tests in sibling `*_tests.rs` files, not inline
2. **Dependencies:** Add to `[workspace.dependencies]` in root `Cargo.toml`, reference with `{ workspace = true }`
3. **New crates:** Follow the `codex-rs/` workspace pattern; keep `src/lib.rs` exports-only
4. **CLI args:** Use clap derive macros with descriptive `value_name` and `help` attributes
5. **Config fields:** All `Option<T>` -- no hardcoded defaults in the struct
6. **Platform code:** Use `#[cfg(target_os = "...")]` guards; provide stubs for unsupported platforms
7. **ToC updates:** Run `scripts/readme_toc.py --fix` after modifying headings in README

9. Integration Points

App Server Protocol (JSON-RPC)

The app server protocol defines ~200+ typed messages across two versions:

V1 types: `ClientRequest`, `ClientNotification`, `ServerRequest`, `ServerNotification` -- covering initialization, auth, fuzzy file search, conversation summaries, exec approval, patch approval.

V2 types: Extended protocol with threads, config management, MCP server management, plugins, skills, file system operations, command execution, model management, feature flags, and more.

Transport: JSON-RPC 2.0 over stdio or WebSocket.

MCP Integration

Codex supports MCP in both directions:

As MCP client (`codex-codex-mcp`): Connects to configured MCP servers, calls tools, handles OAuth authentication, manages elicitation forms.

As MCP server (`codex-mcp-server`): Exposes Codex capabilities as MCP tools over stdio transport. Other MCP clients can call Codex tools.

TypeScript SDK Integration

typescript

```typescript
import { Codex } from "@openai/codex-sdk";

const codex = new Codex({
  config: {
    model: "gpt-5",
    sandbox_mode: "workspace-write",
  },
});

const thread = codex.startThread({
  workingDirectory: "/path/to/project",
});

// Structured output
const turn = await thread.run("Analyze codebase", {
  outputSchema: {
    type: "object",
    properties: {
      summary: { type: "string" },
      issues: { type: "array", items: { type: "string" } },
    },
    required: ["summary", "issues"],
  },
});
```

Responses API Proxy

The `codex-responses-api-proxy` binary proxies requests to the OpenAI Responses API:

bash

```bash
codex-responses-api-proxy \
  --upstream https://api.openai.com/v1/responses \
  --api-key-file ~/.openai/key \
  --listen 0.0.0.0:8080
```

Flag	Description
`--listen`	Local listen address
`--api-key-file FILE`	File containing API key
`--api-key`	API key (direct)
`--upstream URL`	Upstream Responses API URL
`--body-log-dir DIR`	Directory for request/response body logging

OpenTelemetry

Codex supports OpenTelemetry export via the `codex-otel` crate:

toml

```
[otel]
# OpenTelemetry configuration
```

Metrics and traces cover session lifecycle, tool execution, model calls, and sandbox operations.

> ***DocCompiler.ai*** *v3.1 | Upgraded source coverage | Generated 2026-05-13 | Source:* `openai/codex` *@* `f017a23`
> *A Cyber Panda Solutions LLC product. This document is intentionally comprehensive -- built to be the complete, source-verified reference an AI or human needs without consulting other sources.* ***Do not print:*** *the always-current version lives at doccompiler.ai/openai/codex.*