Model Context Protocol

Model Context Protocol (MCP) connects models to tools and context. Use it to give Codex access to third-party documentation, or to let it interact with developer tools like your browser or Figma.

Codex supports MCP servers in both the CLI and the IDE extension.

Supported MCP features

- **STDIO servers**: Servers that run as a local process (started by a command).
  - Environment variables
- **Streamable HTTP servers**: Servers that you access at an address.
  - Bearer token authentication
  - OAuth authentication (run `codex mcp login <server-name>` for servers that support OAuth)

Connect Codex to an MCP server

Codex stores MCP configuration in `~/.codex/config.toml` alongside other Codex configuration settings.

The CLI and the IDE extension share this configuration. Once you configure your MCP servers, you can switch between the two Codex clients without redoing setup.

To configure MCP servers, choose one option:

1. **Use the CLI**: Run `codex mcp` to add and manage servers.
2. **Edit `config.toml`**: Update `~/.codex/config.toml` directly.

Configure with the CLI

Add an MCP server

```
codex mcp add <server-name> --env VAR1=VALUE1 --env VAR2=VALUE2 --
        <stdio server-command>
```

For example, to add Context7 (a free MCP server for developer documentation), you can run the following command:

```
codex mcp add context7 -- npx -y @upstash/context7-mcp
```

Other CLI commands

To see all available MCP commands, you can run `codex mcp --help`.

Terminal UI (TUI)

In the `codex` TUI, use `/mcp` to see your active MCP servers.

Configure with config.toml

For more fine-grained control over MCP server options, edit `~/.codex/config.toml`. In the IDE extension, select **MCP settings** > **Open config.toml** from the gear menu.

Configure each MCP server with a `[mcp_servers.<server-name>]` table in the configuration file.

STDIO servers

- `command` (required): The command that starts the server.
- `args` (optional): Arguments to pass to the server.
- `env` (optional): Environment variables to set for the server.
- `env_vars` (optional): Environment variables to allow and forward.
- `cwd` (optional): Working directory to start the server from.

Streamable HTTP servers

- `url` (required): The server address.
- `bearer_token_env_var` (optional): Environment variable name for a bearer token to send in `Authorization`.
- `http_headers` (optional): Map of header names to static values.
- `env_http_headers` (optional): Map of header names to environment variable names (values pulled from the environment).

Other configuration options

- `startup_timeout_sec` (optional): Timeout (seconds) for the server to start. Default: `10`.
- `tool_timeout_sec` (optional): Timeout (seconds) for the server to run a tool. De-

fault: `60`.
- `enabled` (optional): Set `false` to disable a server without deleting it.
- `enabled_tools` (optional): Tool allow list.
- `disabled_tools` (optional): Tool deny list (applied after `enabled_tools`).

config.toml examples

```toml
[mcp_servers.context7]
command = "npx"
args = ["-y", "@upstash/context7-mcp"]

[mcp_servers.context7.env]
MY_ENV_VAR = "MY_ENV_VALUE"

[mcp_servers.figma]
url = "https://mcp.figma.com/mcp"
bearer_token_env_var = "FIGMA_OAUTH_TOKEN"
http_headers = { "X-Figma-Region" = "us-east-1" }

[mcp_servers.chrome_devtools]
url = "http://localhost:3000/mcp"
enabled_tools = ["open", "screenshot"]
disabled_tools = ["screenshot"] # applied after enabled_tools
startup_timeout_sec = 20
tool_timeout_sec = 45
enabled = true
```

Examples of useful MCP servers

The list of MCP servers keeps growing. Here are a few common ones:

- Context7: Connect to up-to-date developer documentation.
- Figma Local and Remote: Access your Figma designs.
- Playwright: Control and inspect a browser using Playwright.
- Chrome Developer Tools: Control and inspect Chrome.
- Sentry: Access Sentry logs.
- GitHub: Manage GitHub beyond what `git` supports (for example, pull requests and issues).