Tutorial — xcodex

1. What is xcodex?

xcodex is a terminal-based CLI tool for Apple developers. It consolidates the most important Xcode workflows into a single, keyboard-driven interface — without memorizing flags, without context switching, without opening Xcode.

The tool supports iOS, iPadOS and macOS projects and covers the entire development workflow: build, tests, simulator control, device management, archives, TestFlight/App Store delivery, localization, code coverage, crash analysis and project maintenance.

> **Note:** xcodex is not affiliated with Apple Inc. or Xcode. It is an independent open-source tool and not an official Apple product.

Download

Tutorial as PDF — Download this tutorial as a PDF for offline reading, printing or archiving.	Save Tutorial as PDF
Download xcodex — Download the notarized xcodex.zip directly — no Git clone required. Unzip, set executable permission and launch immediately.	Download xcodex.zip

2. Requirements

Requirement	Notes	Download
`macOS 13 Ventura+`	macOS only	—
`Xcode`	Full install from the Mac App Store (~30 GB). Open once to install components	App Store ↗ · developer.apple.com ↗
`Command Line Tools`	Includes Git, xcodebuild, clang, make, svn	Download ↗ · `xcode-select --install`
`Swift 5.9+`	Included with	—

	Xcode	
`Git`	For cloning the repository	—
Homebrew *(optional)*	For CocoaPods and other tools	brew.sh ↗ · `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`
CocoaPods *(optional)*	Only if your project uses CocoaPods	`brew install cocoapods`
Carthage *(optional)*	Only if your project uses Carthage	`brew install carthage`
xcbeautify *(optional)*	For readable build output	`brew install xcbeautify`
xcodes CLI *(optional)*	For managing Xcode versions	github.com/XcodesOrg/xcodes ↗ · `brew install xcodesorg/made/xcodes`

3. Installation

The following requirements must be met before installation:

Requirement	Notes	Download
`Xcode`	~30 GB, open once to install components	App Store ↗ · developer.apple.com ↗

`Xcode Command Line Tools`	Includes Git, xcodebuild, clang, make, svn	Download ↗️ · `xcode-select --install`

Clone the repository

This command downloads the complete xcodex source code to your Mac. Git creates the directory `~/Developer/xcodex` containing all files. Having a Git clone lets you apply future updates with a single `git pull` — no need to re-download manually.

```
git clone https://github.com/drapatzc/xcodex.git ~/Developer/xcodex
```

Set executable permission

By default, macOS does not allow running downloaded files as programs. `chmod +x` grants the operating system permission to execute the `xcodex` file. This step only needs to be done once after cloning.

```
chmod +x ~/Developer/xcodex/xcodex
```

Set up shell alias (recommended)

A shell alias lets you run `xcodex` from any directory without typing the full path. The alias is saved permanently in your shell's configuration file and is available automatically from the next terminal session onward.

zsh:

```
echo 'alias xcodex="$HOME/Developer/xcodex/xcodex"' >> ~/.zshrc
source ~/.zshrc
```

bash:

```
echo 'alias xcodex="$HOME/Developer/xcodex/xcodex"' >> ~/.bash_profile
source ~/.bash_profile
```

fish:

```
alias xcodex="$HOME/Developer/xcodex/xcodex"
funcsave xcodex
```

> **Which shell am I using?** Run `echo $SHELL` in your terminal. The output shows the path to your active shell — e.g. `/bin/zsh`, `/bin/bash`, or `/usr/local/bin/fish`.

Run

Navigate to your Xcode project's root directory and launch xcodex. On first launch you need to manually select your project or workspace so xcodex has all the information it needs — scheme, device and build configuration.

```
cd YourXcodeProject
xcodex
```

Update

`git pull` downloads the latest changes from the repository and applies them to your local copy. This keeps xcodex up to date without needing to clone it again.

```
cd ~/Developer/xcodex
git pull
```

Download xcodex directly

You can also download the notarized xcodex.zip directly — no Git clone required. Download the ZIP file, unzip it and place the binary in a fixed path (e.g. `~/Developer/xcodex/`). Then only follow steps 2 to 4: set executable permission, configure the shell alias and run xcodex. Updates need to be applied manually by re-downloading.

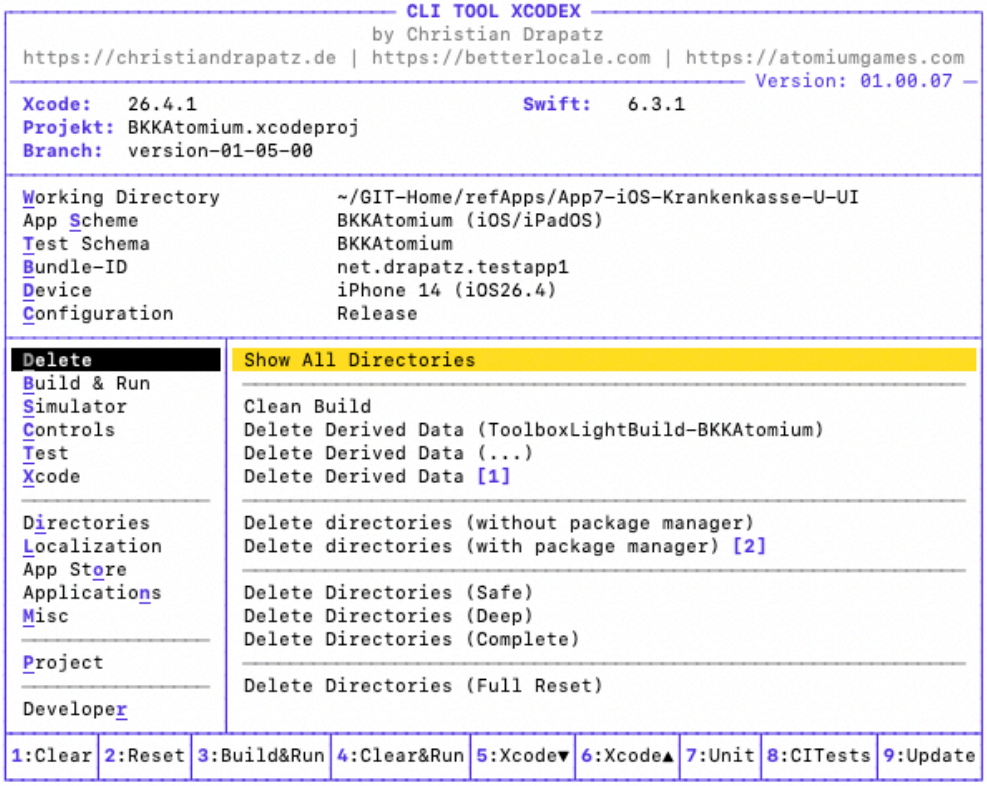
Download xcodex.zip

---

4. The Menu Interface

The interface is split into two panes: command groups on the left, actions on the right. Navigation is entirely keyboard-driven.

At the bottom there is a hotkey bar with 9 quick-access commands, accessible directly via the corresponding number key.



xcodex menu interface

Split-pane interface: command groups on the left, actions on the right

Navigation Keys

Key	Action
↑ / ↓	Navigate within the active pane
← / →	Switch between groups and actions
↵	Execute the selected action
H	Show context-sensitive help for the current action
X	Return to menu
Shift+Q	Quit the application
1–9	Direct access to hotkeys in the bottom bar
Shift+<Letter>	Jump directly to a command group
L	Toggle interface language
Space	Switch between XCode view and Browser view
Shift+A	Open Browser view in the current working directory

`Shift+W`	Jump directly to Browser view

All settings persist automatically between sessions.

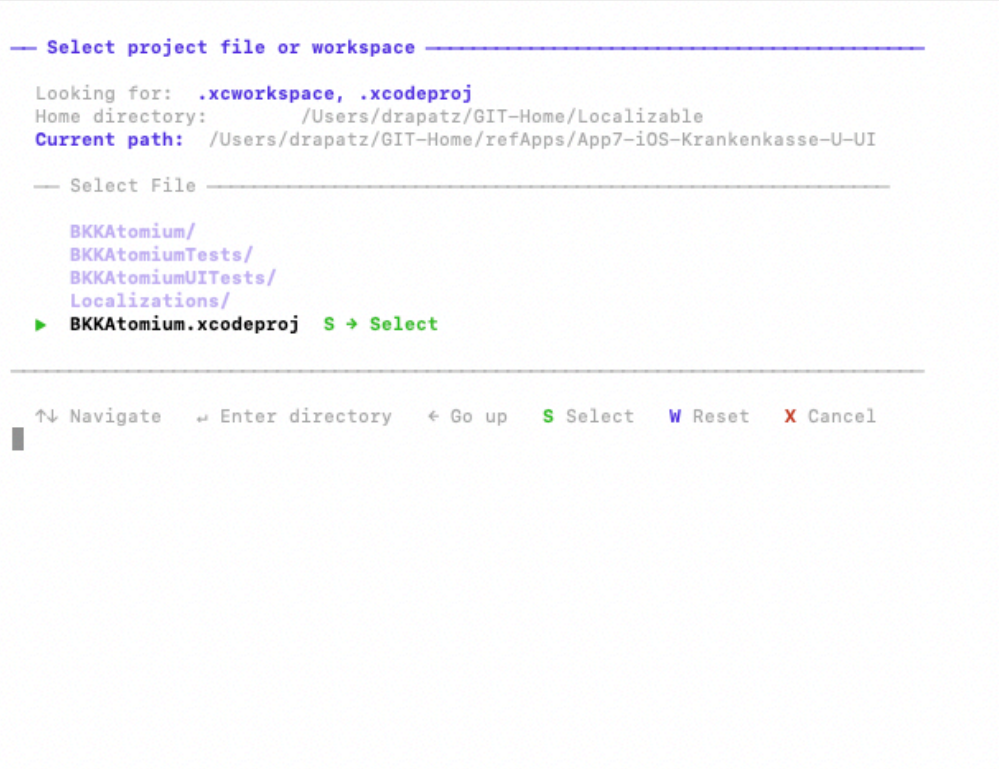
> **Hotkeys vs. commands:** The table below lists all global hotkeys. Unlike commands in the action list — which may prompt for confirmation — hotkeys execute immediately without further input, making them ideal for fast, repetitive workflows.

Global Hotkeys

Hotkey	Command	Description
`1`	Delete DerivedData	Removes the project's DerivedData folder and forces a clean rebuild
`2`	Clear all caches	Clears DerivedData, module cache, and package manager caches (SPM, CocoaPods, Carthage)
`3`	Build & Run	Builds the app and launches it directly on the active device (simulator, physical device, or Mac)
`4`	Quick Reset & Build	Deletes project DerivedData, rebuilds and launches the app — the fastest fix for persistent build errors
`5`	Close Xcode	Quits Xcode without losing terminal focus
`6`	Open project in Xcode	Opens the active project directly in Xcode
`7`	Run unit tests	Starts the unit test run with the configured test scheme
`8`	Automated tests	Opens the menu for automated and combined test runs
`9`	Update project	Triggers a project update (e.g. re-resolve dependencies)

5. Getting Started — Project Setup

On first launch from a project directory, xcodex needs to be configured once.



Project setup in xcodex

Step by step: select project, scheme, device and build configuration

Step 1 — Select working directory

xcodex automatically searches for Xcode files in the current directory. If none is found, a file browser opens for manual selection.

Step 2 — Select app scheme

The scheme controls what gets built and how it is launched. Optionally select a separate test scheme or test plan — useful when tests are organized in their own target.

Step 3 — Select target device

Available simulators and connected physical devices are automatically detected from the current Xcode installation.

Step 4 — Choose build configuration

- **Debug** — daily development, with symbols and logs

- **Release** — App Store builds, with all optimizations enabled

> **Tip:** Scheme, device and configuration can be changed at any time via `S`, `D` and `C` in the menu — no need to restart the tool. All settings are saved automatically.

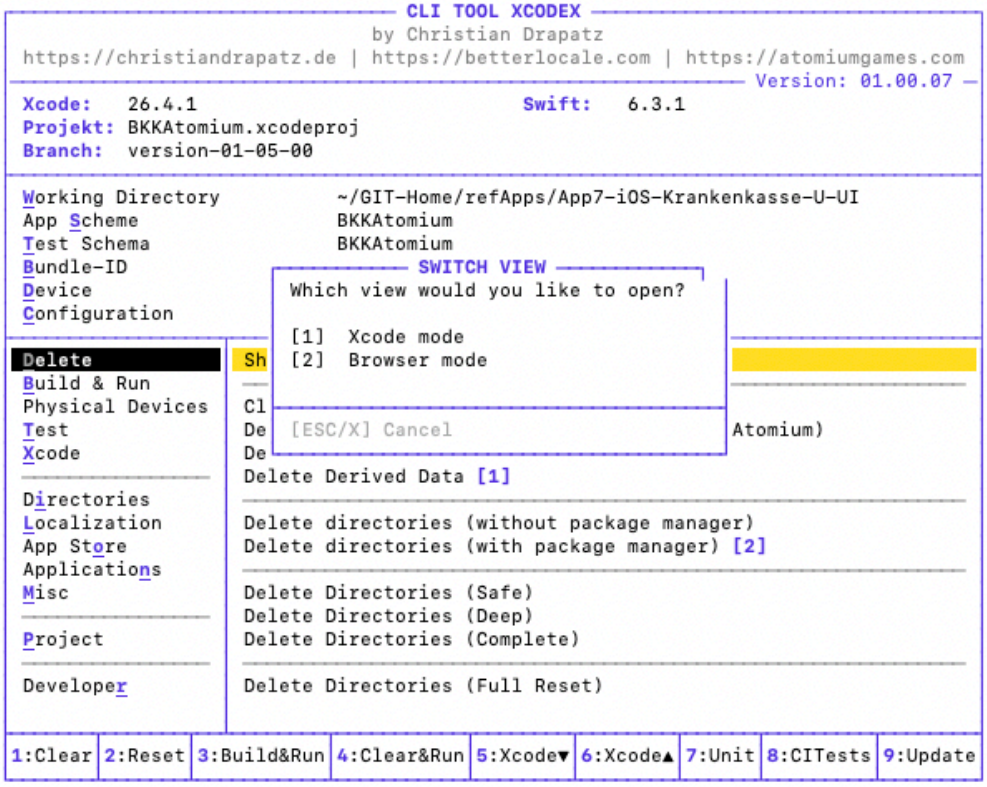
Quick shortcuts in the header

Key	Function
`W`	Select working directory
`S`	Select app scheme
`T`	Select test target
`D`	Select target device
`C`	Switch build configuration (Debug / Release)
`B`	Show and copy bundle ID
`L`	Toggle interface language (German / English)

---

6. XCode View vs. Browser View

xcodex offers two main views you can switch between at any time: the **XCode view** with all developer tools and the **Browser view** — a keyboard-driven file browser.



View selection in xcodex

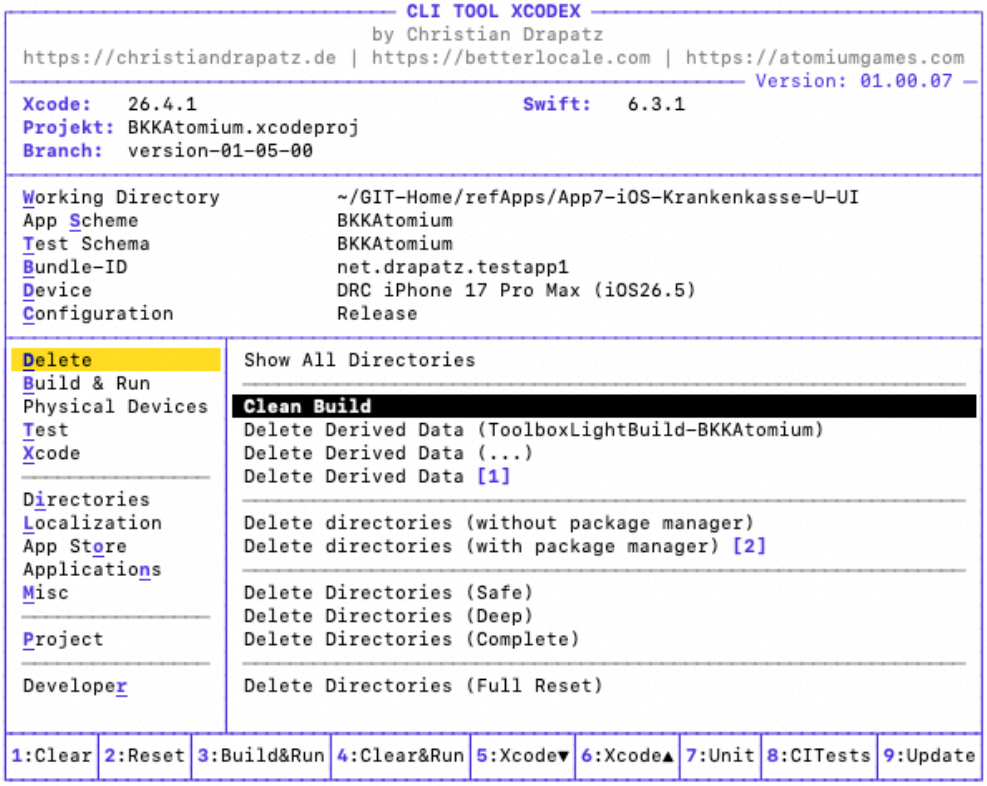
Switch between XCode view and Browser view using the Space key

Key	Action
`Space`	Switch between XCode view and Browser view
`Shift+X`	Jump directly to XCode view
`Shift+W`	Jump directly to Browser view
`Shift+A`	In XCode view: open Browser view with working directory

> **Tip:** Use `Shift+A` in the XCode view to switch directly to the Browser and land automatically in the current project working directory.

7. Clean & Cache

Xcode stores all build artifacts in a shared DerivedData folder. When builds fail for no apparent reason, the app behaves strangely, or errors appear after a refactoring that don't exist in the code — a targeted cleanup helps.



Clean & Cache in xcodex

Clean & Cache: various cleanup options at a glance

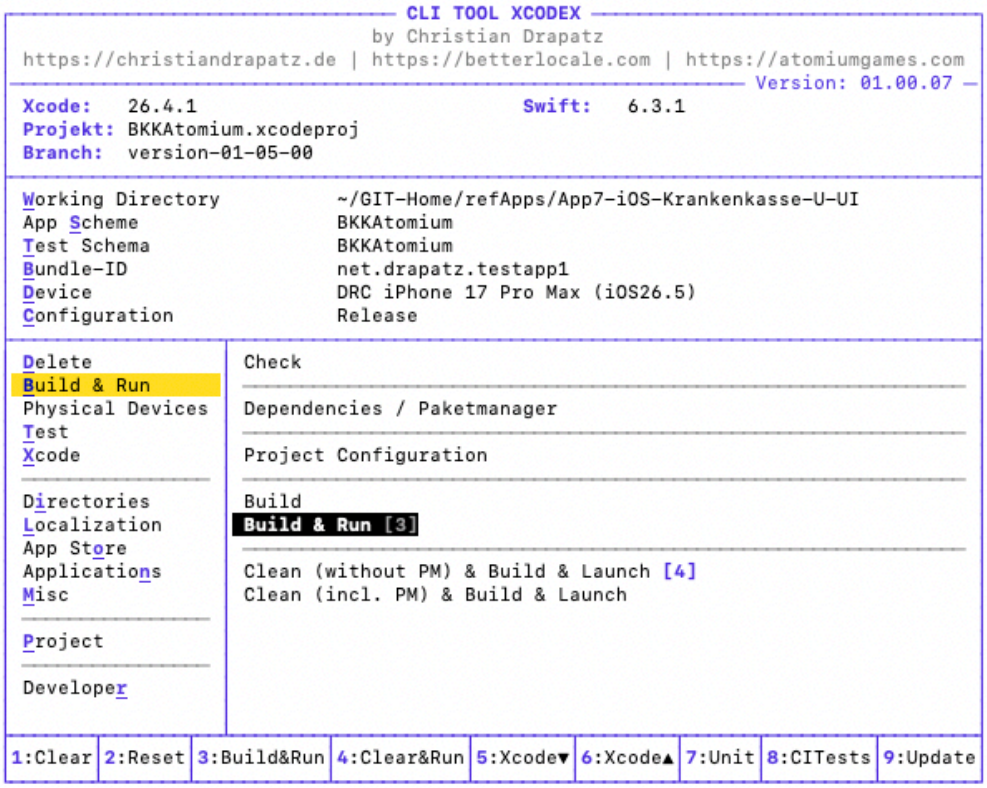
Action	What is cleaned	When to use
`Show all directories`	Cache browser with size display and selective deletion	Overview before cleaning
`Clean build`	Compiled artifacts of the current scheme	Small reset before a clean build
`Delete DerivedData (project)`	Project-specific DerivedData folder	Unexplained build failures
`Delete DerivedData`	Entire DerivedData folder	Full build cache reset
Delete directories (without package managers)	DerivedData, Xcode caches, Simulator cache	Quick reset without touching package managers
Delete directories (with package managers)	+ CocoaPods, Carthage and SPM caches	When package manager caches should also be cleaned
`Delete directories (Safe)`	DerivedData, ModuleCache, SwiftPM, Xcode caches, Simulator cache, Logs, Diagnostic Reports	Comprehensive reset — everything Xcode recreates automatically
`Delete directories (Deep)`	Everything from "Safe" + Archives, iOS DeviceSupport,	Persistent issues; recover a lot of storage

	all simulator devices	
`Delete directories (Complete)`	Everything from "Deep" + CocoaPods, Carthage and SPM cache; optional Simulator Runtimes	Maximum storage recovery
`Delete directories (Restart)`	Full reset including SourcePackages for a clean project restart	When everything needs to be rebuilt from scratch

> **Tip:** For unexplained behavior, try deleting DerivedData first — it resolves the majority of cache-related issues in under 10 seconds.

8. Build & Run

xcodex offers several preconfigured build modes optimized for common development scenarios.



Build & Run overview in xcodex

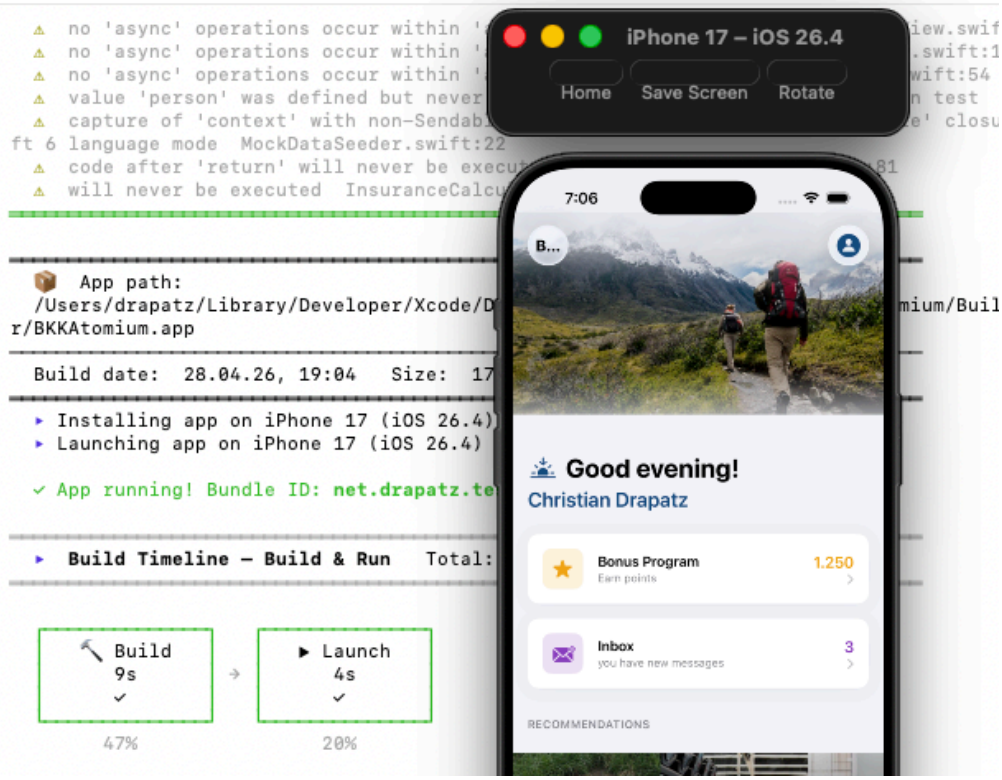
Build & Run: different modes for different situations

Action	What happens	When to use
`Validate`	Checks the project for configuration errors	Before the first build in a new setup
`Dependencies / Package managers`	Resolves SPM, CocoaPods and Carthage dependencies	After cloning or on package errors
`Project configuration`	Shows all build settings for the selected scheme	To diagnose configuration issues
`Build`	Compiles the project without launching	Quick compilation check
`Build & Run`	Builds and launches the app in the simulator or on macOS	Standard development workflow
Clean (without package managers) & Build & Run	Deletes DerivedData and Xcode caches, then rebuilds and launches	Build failures caused by cache issues
Clean (incl. package managers) & Build & Run	Deletes all caches including package manager caches, then rebuilds and launches	Persistent issues or after package manager changes

Understanding build output

During the build, xcodex shows each step with its duration. Errors are highlighted in red with the affected file and line number.

Use the `+` key to add additional target devices, so the app runs on multiple operating systems in parallel.



Build output in xcodex

Build output: structured, compact, with timing per step

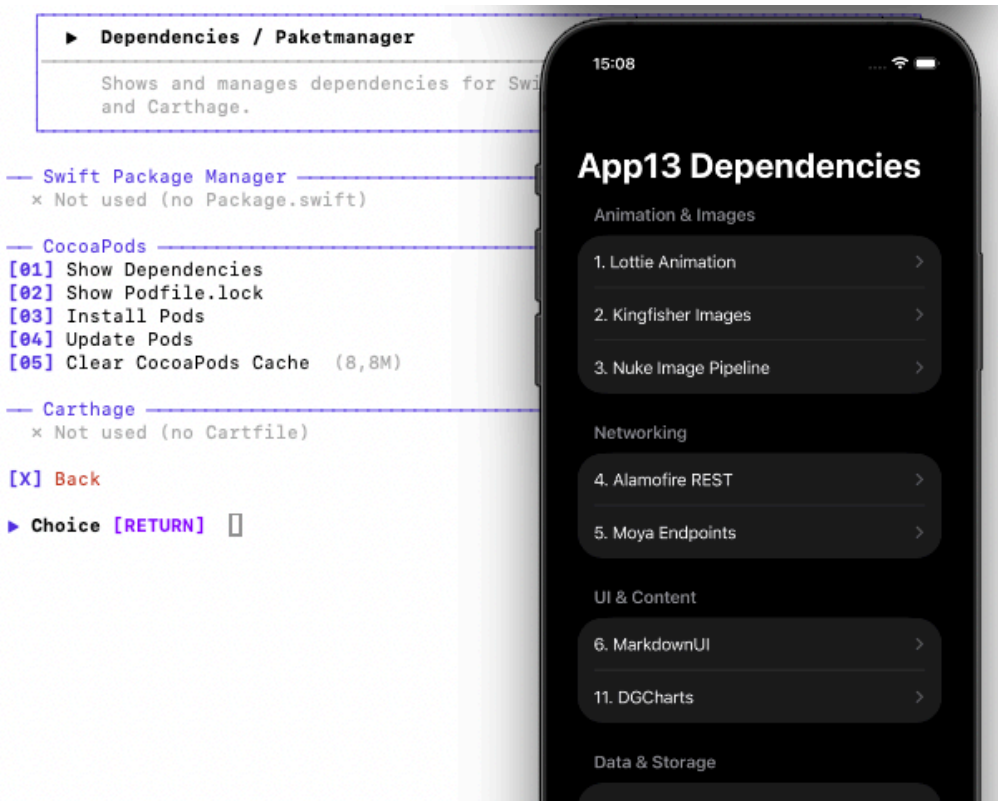
Isolated DerivedData

xcodex builds into its own DerivedData directory, completely separate from Xcode. Both can run simultaneously without interfering with each other.

> **Tip:** Use `v` to validate the project before building — checks that all dependencies are resolved and the scheme is correctly configured.

9. Dependencies

CocoaPods, SPM or Carthage dependencies can be managed directly from xcodex — no terminal experience required.



Dependencies in xcodex

Dependencies: SPM, CocoaPods and Carthage in one menu

Package Manager	Command
Swift Package Manager	`swift package resolve`
CocoaPods	`pod install`
Carthage	`carthage update`

> **Tip:** After a `git pull`, resolve all dependencies before building — this prevents the most common "missing module" errors after branch switches.

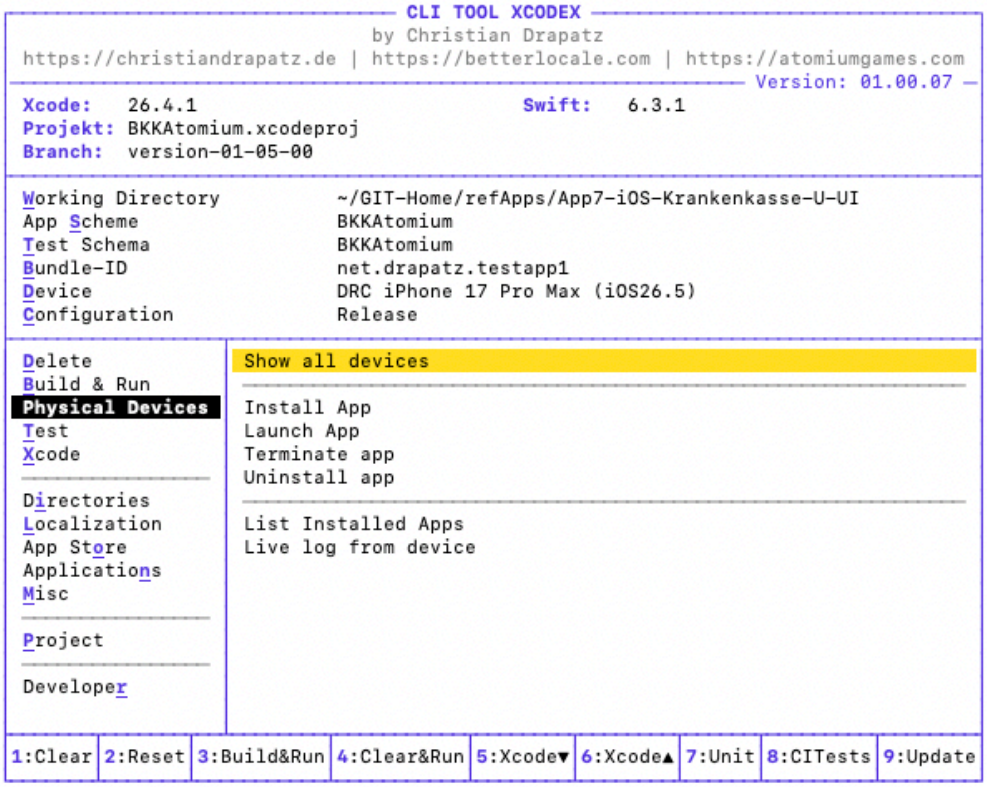
---

10. Simulator & Physical Devices

Depending on the selected target device, xcodex shows either the **Simulator group** or the **Physical Devices group** — never both at the same time. The available actions automatically adapt to the active device.

Simulator

Simulators can fall into an inconsistent state — especially after iOS updates or extended uptime. This group restores control.



Simulator control in xcodex

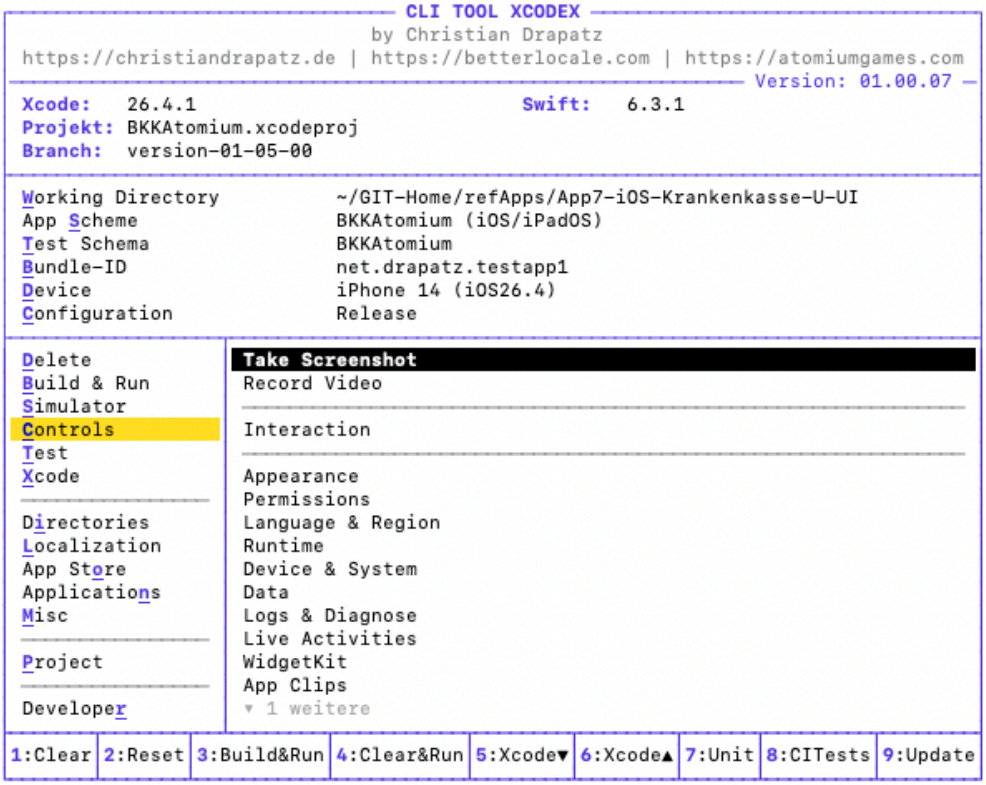
Simulator: reinstall app, reset, close all instances

Action	What happens	When to use
`Create new simulator`	Choose runtime and device type, create simulator via xcrun simctl create	When a specific device/iOS combination is missing
`Launch app`	Launches the built app on the selected simulator	After a build without rebuilding
`Launch app on selected simulator`	Like "Launch app" but with prior simulator selection	Testing on a different device
`Uninstall app & fresh test`	Uninstalls the app and reinstalls it	Clean start without old app state
`Restart simulator`	Shuts down the simulator and restarts it	For a hanging or unstable simulator
`Stop simulator`	Shuts down the running simulator	Free up resources
`Reset simulator`	Resets the current simulator to	Completely remove app data

	factory state	and settings
`Show simulator status`	All simulators grouped by runtime, color-coded (Booted / Shutdown)	Overview of installed simulators
`Erase simulator contents`	Deletes all content of all simulators (xcrun simctl erase all)	Full reset of all simulator data
`Delete unavailable simulators`	Removes simulators without an installed runtime	Cleanup after Xcode updates
`Stop all simulators`	Stops all running simulators at once	Free up resources, before a build
`Delete all simulator devices`	Removes all simulator device entries (xcrun simctl delete all)	Full restart of simulator infrastructure
`Delete simulator caches`	Removes temporary CoreSimulator caches	Display glitches or behavior after iOS updates

Remote control the simulator

After the app launches, simulator settings can be changed directly from xcodex — no need to dig through the simulator's own menus.



Simulator controls in xcodex

Controls: Dark Mode, permissions, status bar, push notifications

1. Interaction

#	Command	Testing value
01	Test push notification	Simulates an APNs payload without a real server — ideal for notification UX tests.
02	Open deep link	Opens a URL scheme directly, without manually navigating the app.

2. Appearance

#	Command	Testing value
01	Toggle Dark / Light Mode	Checks that all assets and colors respond correctly to both modes.
02	Mock status bar	Sets defined values (time, signal strength) for screenshots and UI tests.
03	Reset status bar	Restores the status bar mock back to the real system state.
04	Set Dynamic Type	Tests layout stability at extreme font sizes.
05	Toggle Bold Text	Checks readability with the bold-text accessibility setting enabled.
06	Toggle Reduce Motion	Ensures animations are correctly disabled.
07	Toggle Increase Contrast	Validates contrast ratios for accessibility.
08	Set Display Zoom *(iPhone only)*	Tests layout in enlarged display mode.
09	Toggle Reduce Transparency	Checks UI elements with blur effects disabled.
10	Toggle On/Off Labels	Ensures toggles are recognisable without colour

		alone.
11	Differentiate Without Color	Tests that the app does not rely exclusively on colour to convey information.

3. Permissions

#	Command	Testing value
01	Set permissions	Granular control of individual privacy grants.
02	Reset all app permissions	Resets the app's permission state to "not asked" — for permission-flow tests.
03–19	Quick Grants (Camera, Microphone, Location, Photos, Contacts, Calendar, Reminders, Motion, Health, Network, Bluetooth, HomeKit, Media Library, Siri, Speech, Local Network, User Tracking)	Grant a single permission in one step — saves dialog clicks on every test run.
20	Grant all permissions	Sets the entire permission list to "allowed" — for quick tests beyond the permission flow.

4. Language & Region

#	Command	Testing value
01	Set app language	Tests app localisation without changing the system language.
02	Set system language	Tests global language effects (keyboard, system dialogs).
03	Toggle 24h format	Validates time display in both formats.
04	Reset system language	Resets all language settings to factory defaults.

5. Runtime

#	Command	Testing value
01	Slow Animations	Slows down animations — transition errors become visible.
02	Highlight Clipped Views	Shows clipped UI elements immediately highlighted in red.
03	Set Launch Arguments	Injects -argument flags for feature flags and test modes without a build.
04	Set Env Variables	Passes environment variables to the app on launch.
05	Simulate Memory Pressure	Tests app behaviour under memory pressure (warning levels).
06	Test URL Scheme	Opens a deep link and checks the handling.
07	Core Data Debug	Enables SQL logging for Core Data — shows all queries.
08	Localization Debug	Enables double length / pseudolanguage — layout problems become immediately visible.
09	SwiftUI Debug	Shows SwiftUI render boundaries and invalidations in the console.
10	Background Fetch	Triggers a background fetch manually — without a 20-minute wait.
11	Background Launch	Launches the app in background mode.
12	Metal Validation	Enables Metal API validation for GPU code.

6. Device & System

#	Command	Testing value
01	Set location	Simulates GPS coordinates without a real location.
02	Simulate battery status	Tests low-battery warnings and

		charge-level logic.
03	Set cellular mode	Simulates EDGE, LTE, 5G or no signal.
04	Reset IDFA	Resets the advertising ID — for tracking tests.
05	Trigger iCloud sync	Forces an iCloud sync without waiting.
06	Set timezone	Tests timezone-sensitive logic (calendar, countdowns).
07	Mock status bar time	Sets a fixed time for consistent screenshots.
08	Stop location simulation	Ends the simulated GPS location.
09	Set Wi-Fi bars	Tests UI at various Wi-Fi signal strengths.
10	Set cellular bars	Tests UI at various mobile signal strengths.
11	Set carrier name	Sets a custom carrier name in the status bar.
12	Stage Manager *(iPad only)*	Tests the layout in Stage Manager mode.
13	Set audio route	Switches between speaker, headphones, Bluetooth.

7. Data

#	Command	Testing value
01	Add media	Imports photos/videos into the simulator's photo library.
02	Clear simulator keychain	Removes all stored keychain entries for the app.
03	Open app data folder	Opens the container directly in Finder — for manual file debugging.
04	Reset app UserDefaults	Deletes all stored UserDefaults values for the app.
05	Delete all app data	Removes the entire app

		container.
06	Open App Group container	Opens the shared container in Finder.
07	Show app defaults	Lists all current UserDefaults keys and values.
08	Open SQLite database	Opens the app's database directly in an external editor.
09	Show keychain entries	Lists all stored keychain entries for the app.
10	Install root certificate	Installs a custom CA certificate for SSL pinning tests.
11	Measure app container	Shows the size of the app container on disk.
12	Export app container	Exports the container as an archive for offline analysis.
13	Edit UserDefaults key	Changes individual keys directly — without an app restart.

8. Logs & Diagnostics

#	Command	Testing value
01	Stream app logs	Live log output from the app in the console.
02	Filter console	Filters the system console to relevant subsystems.
03	Open crash log	Opens the app's most recent crash report.
04	Create diagnostic report	Creates a full simulator diagnostic report.
05	Show app version	Shows the bundle version and build number of the installed app.
06	Stream system logs	Full system console of the simulator.
07	Validate Privacy Manifest	Checks the PrivacyInfo.xcprivacy for

		completeness.
08	SwiftData Debug	Enables SQL logging for SwiftData.
09	Create sysdiagnose	Creates a full system diagnostic archive.

9. Live Activities

#	Command	Testing value
01	Start Live Activity	Starts a Live Activity with a custom payload.
02	Update Live Activity	Sends an update to a running Live Activity.
03	End Live Activity	Ends a Live Activity.

10. WidgetKit

#	Command	Testing value
01	Reload widget timelines	Forces a timeline refresh without waiting.
02	Clear widget cache	Clears the internal WidgetKit cache.

11. App Clips

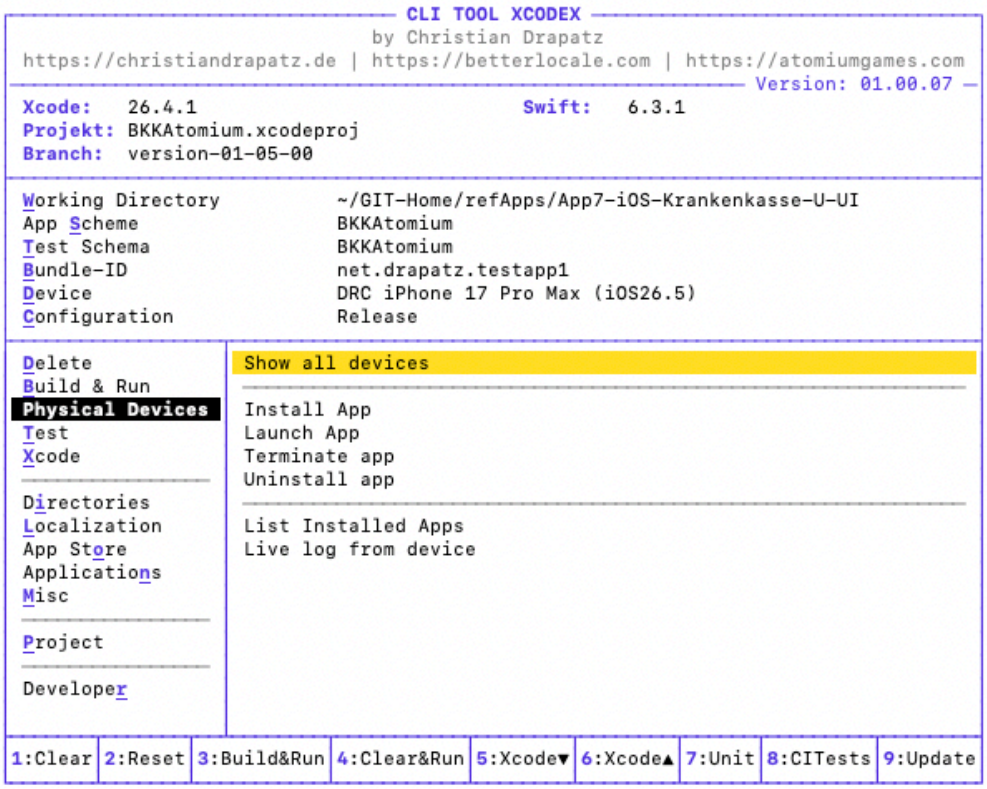
#	Command	Testing value
01	Test App Clip URL	Opens an App Clip experience via URL.
02	Reset App Clip experience	Resets the stored App Clip state.

> **Tip:** Set the status bar to "9:41, full battery, full signal" before creating App Store screenshots — exactly as Apple does in official presentations.

> **Note:** A full simulator reset deletes all data — including Keychain entries, settings and test accounts. Make sure no important test data will be lost.

Physical Devices

This group controls directly connected iOS devices via USB or Wi-Fi. It is only visible when a physical device is selected as the target.



Physical Devices in xcodex

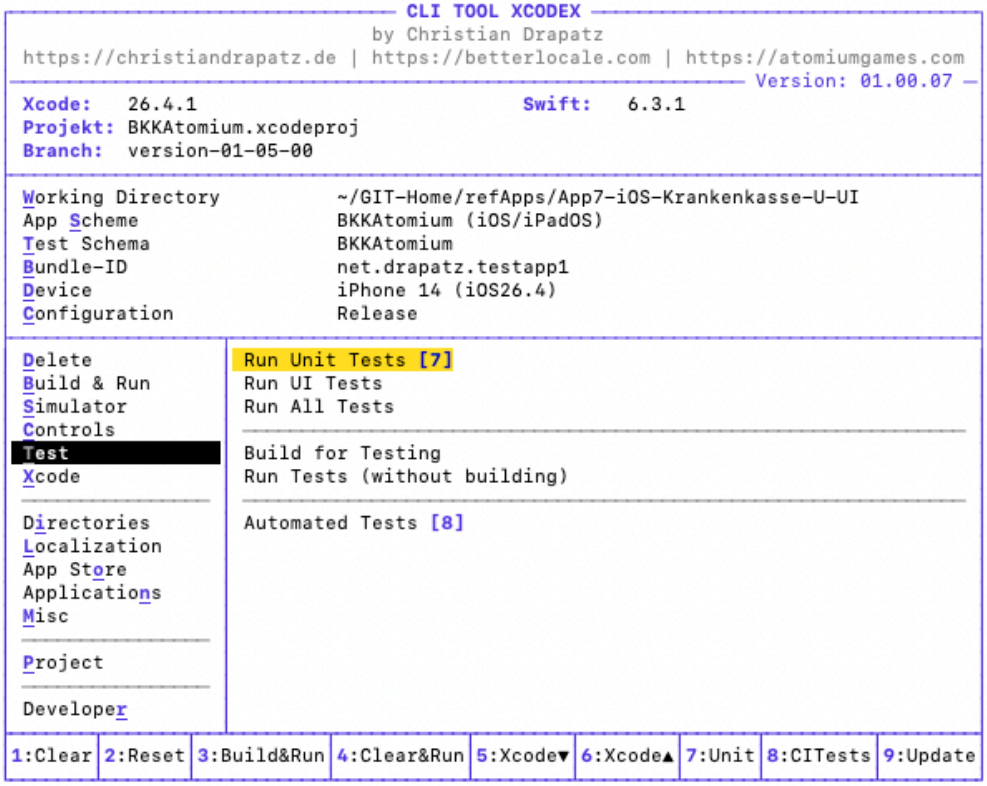
Full device management for physical iOS devices directly in the terminal

Action	Description
`Show all devices`	Lists all connected physical devices with name, UDID and iOS version
`Install App`	Installs the last built app bundle on the connected device (device must be unlocked and trusted)
`Launch App`	Launches the installed app on the connected device
`Terminate app`	Terminates the app process on the device without manually operating it
`Uninstall app`	Completely removes the app from the device including all local data
`List Installed Apps`	Displays bundle identifiers and names of all installed apps on the device

`Live log from device`	Streams console output from the device in real time — stop with Ctrl+C

11. Tests

xcodex separates unit tests and UI tests so the right run can be started for each situation — fast unit tests in the development loop, UI tests before commits or releases.



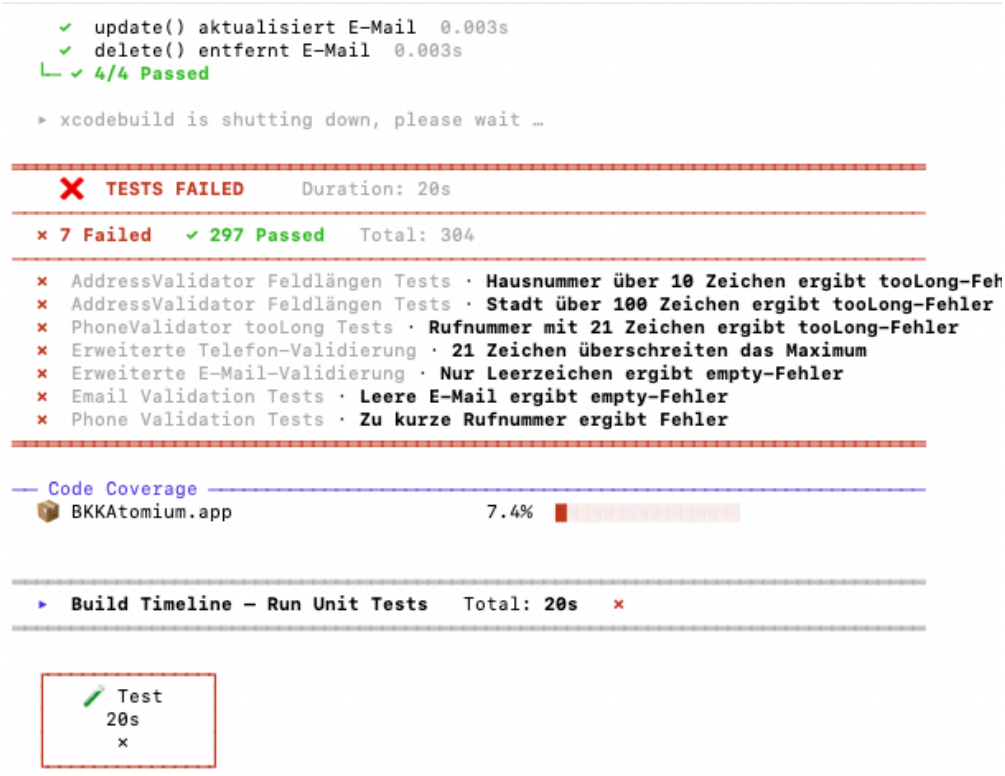
Tests in xcodex

Tests: unit tests, UI tests and all tests with one action

Test type	Description
`Unit Tests`	Fast, no simulator launch required
`UI Tests`	Launches the app in the simulator and runs interaction tests
`All Tests`	Combined run

Result states: ✅ **Passed** · ⚠️ **Skipped** · ❌ **Failed**

Re-run failed tests individually



Unit tests and re-run in xcodex

Failed tests are highlighted in red and can be re-run individually

After a run, xcodex lists all results. Failed tests can be re-run individually — without repeating the entire test suite. Typical workflow:

24. Run all tests
25. Identify failed tests
26. Implement fix
27. Re-run only the affected tests
28. On success: run full suite to confirm

Code Coverage

The code coverage view shows you directly how well your project is covered by tests. xcodex can run test suites with coverage enabled, evaluate the results and show you which areas are already well tested and where gaps remain. This helps you spot weaknesses early — before they surface in a review, a CI/CD pipeline or a release.

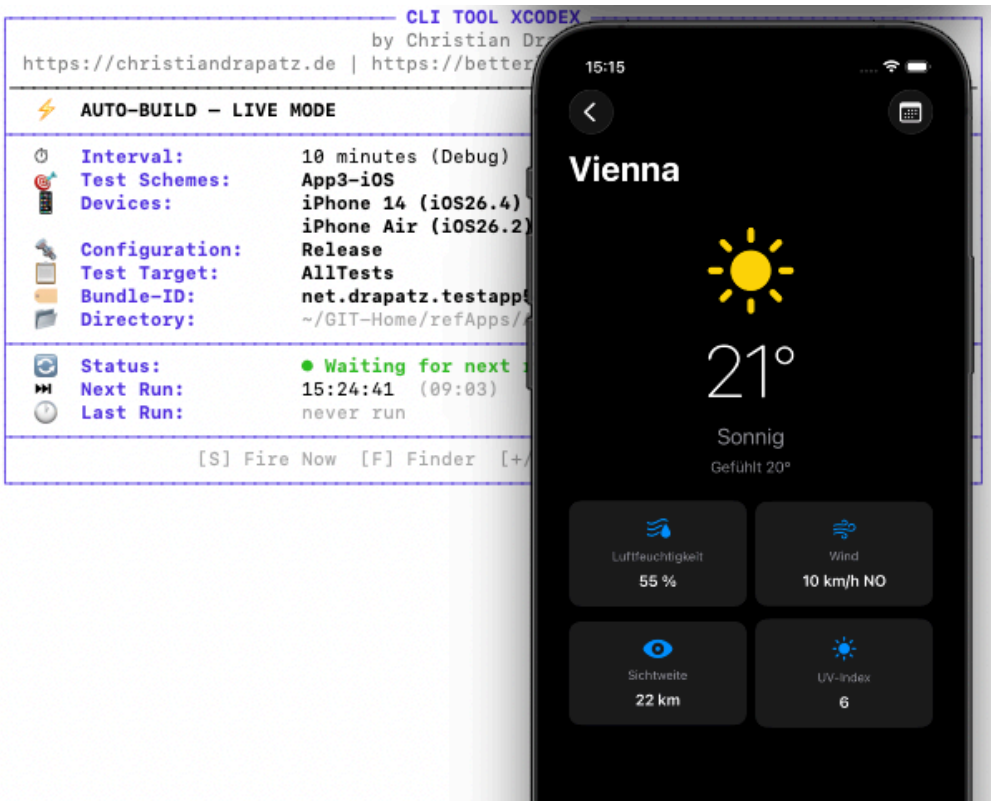
Code coverage in xcodex

Code coverage per file: quickly identify where tests are missing

Coverage	Assessment
80–100 %	Well covered — critical paths are tested
50–80 %	Room for improvement, especially for complex logic
< 50 %	High risk on changes — tests are missing

Automated Tests

The automated tests view bundles recurring test runs into a clear terminal workflow. You can run unit tests selectively and test different combinations of scheme, device and configuration. This lets you repeat typical pre-pull-request or pre-release checks faster, without having to reassemble long xcodebuild commands each time.



Automated tests in xcodex

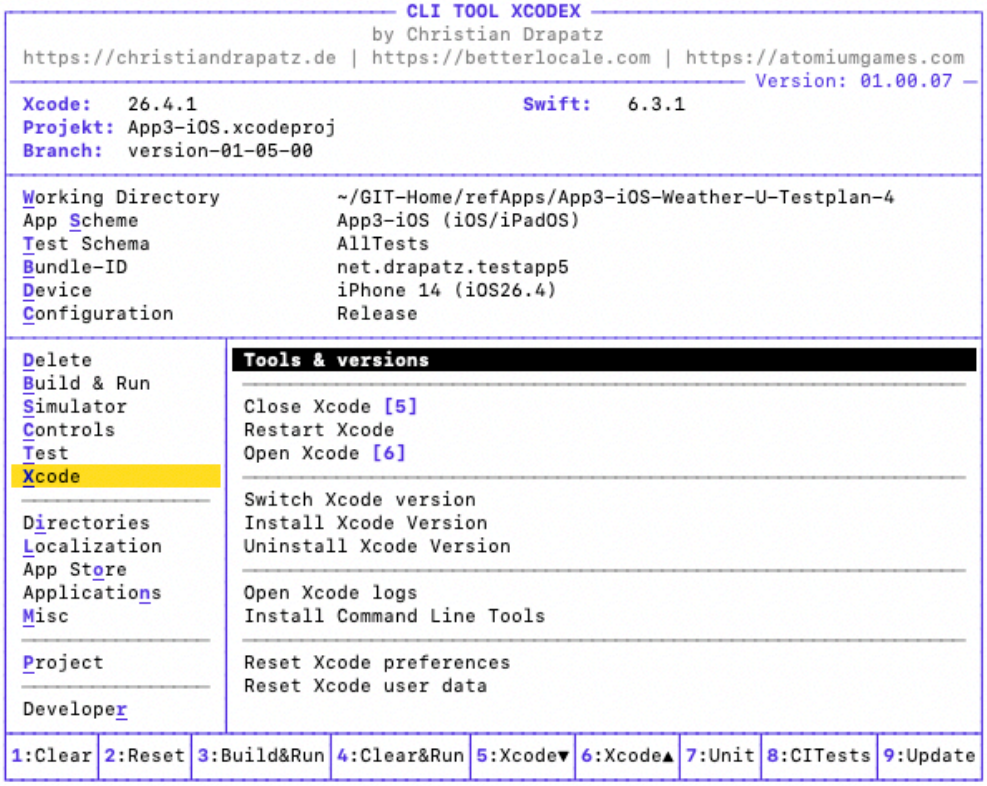
Automated tests run completely independently of Xcode

Automated tests run fully independently of Xcode — Xcode can be open simultaneously without affecting the test run. Ideal for:

- **In parallel with development** — tests for feature A while working on feature B
- **QA without Xcode knowledge** — testers can run test suites independently
- **Second clone** — test the last commit without touching your own development environment

12. Xcode Integration

This group contains quick-access commands for Xcode itself — for switching between the terminal and the IDE.



Xcode integration in xcodex

Open and close Xcode directly from xcodex

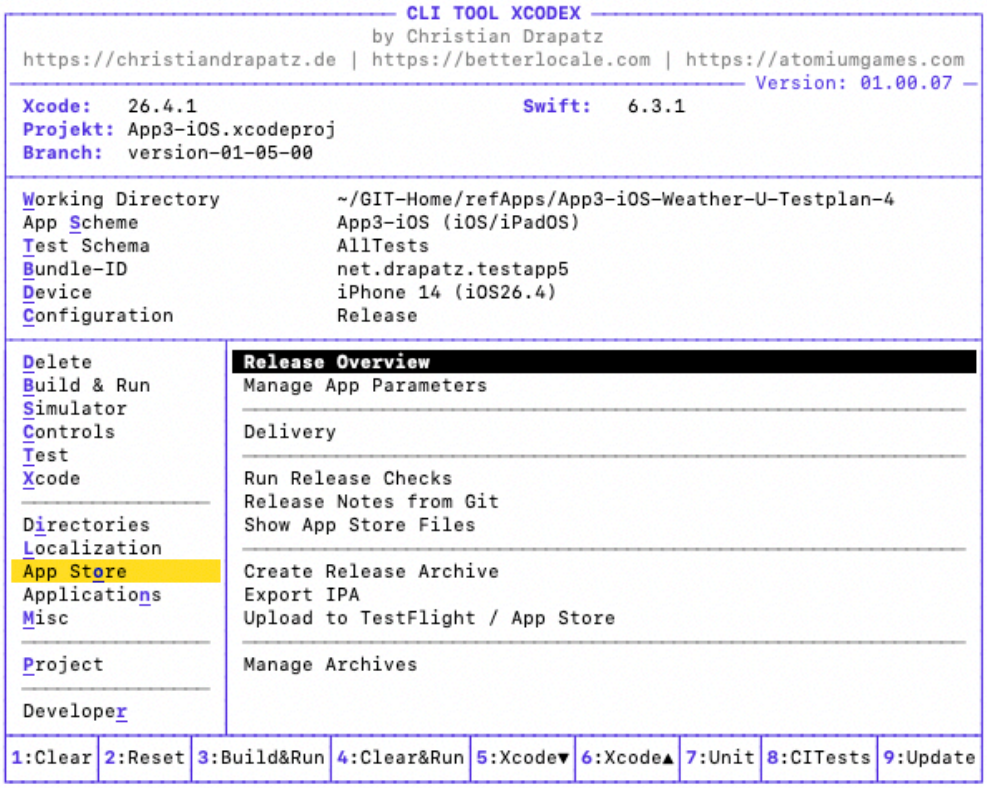
Action	What happens	When to use
`Tools & versions`	Shows installed versions of Xcode, Swift, CocoaPods, SwiftLint and more	Quick overview of the development environment
`Close Xcode`	Quits Xcode	Before cache operations or switching Xcode versions
`Restart Xcode`	Closes Xcode and reopens it	For a hanging Xcode or after settings changes
`Open Xcode`	Opens the current project in Xcode	Quick access without Finder
`Switch Xcode version`	Switches the active Xcode version via xcode-select	When multiple Xcodes are installed and a specific one is needed
`Install Xcode version`	Installs a new Xcode version via Xcodes	Install a new Xcode version alongside the existing one
`Uninstall Xcode version`	Removes an installed Xcode version via Xcodes	Delete old versions and free up storage
`Open Xcode logs`	Opens ~/Library/Logs/Xcode in Finder	For unexplained errors or crashes in Xcode

`Install Command Line Tools`	Starts the macOS system dialog for CLT installation	After a macOS update or on a new Mac
`Reset Xcode preferences`	Deletes com.apple.dt.Xcode.plist after confirmation	For UI freezes or broken keyboard shortcuts in Xcode
`Reset Xcode user data`	Removes ~/Library/Developer/Xcode/UserData/ after confirmation (Keybindings, Themes, Snippets)	When Xcode customizations cause problems

> **Tip:** Close Xcode before a Full Reset & Build — this prevents file locks on the DerivedData folder.

13. App Store & Distribution

xcodex covers the entire distribution process — from archiving to upload. This saves the manual work in Xcode Organizer and enables uploads directly from the terminal.



App Store distribution in xcodex

App Store: archive, validate and upload in a structured workflow

Prerequisite: App-specific password

For uploads to TestFlight or App Store Connect, xcodex requires an app-specific password:

29. Open appleid.apple.com
30. Go to *Security → App-Specific Passwords*
31. Create a new password (e.g. "xcodex Upload")
32. Enter the password once in xcodex settings

Typical release workflow

App Store delivery in xcodex

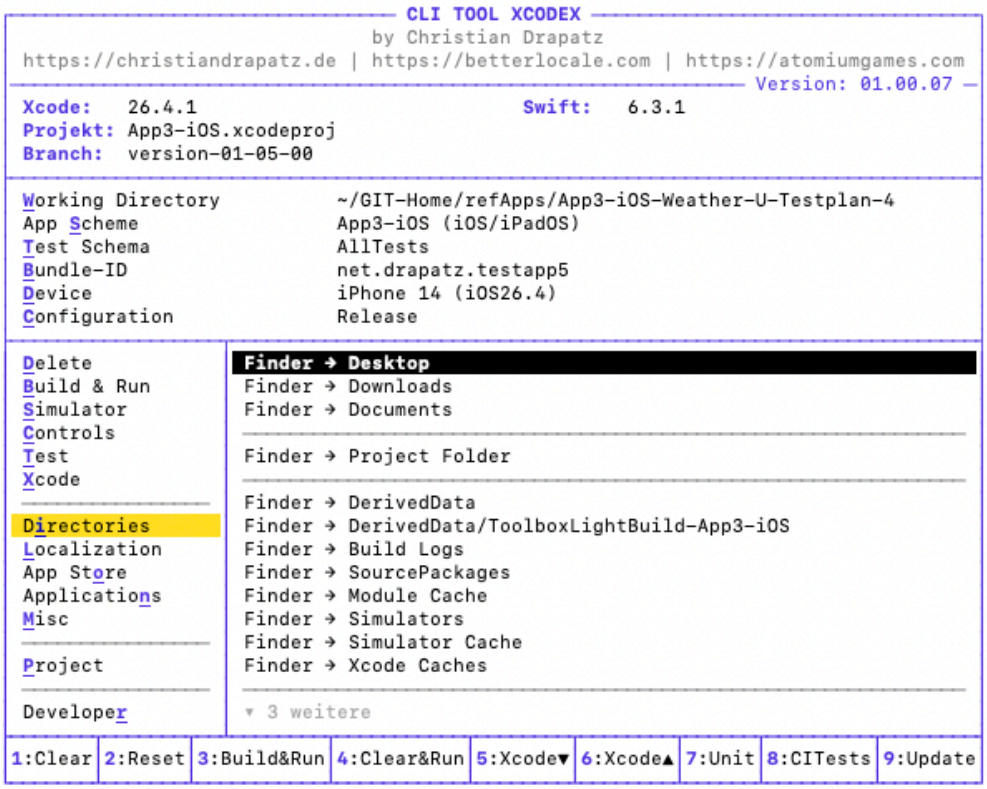
Release workflow: validating before upload prevents common submission errors

33. Full Reset & Build with Release configuration
34. Archive
35. Validate — fix errors if any
36. Upload to TestFlight — for beta testing
37. After approval: submit to App Store

> **Note:** For macOS apps, the archive is automatically notarized and stapled (Notarize & Staple) before upload — a step that is error-prone when done manually.

14. Directories

Xcode distributes its data across several nested Library paths. xcodex opens the most important folders with a single keystroke directly in Finder.



Directories in xcodex

Directories: DerivedData, archives, simulators and provisioning profiles in one click

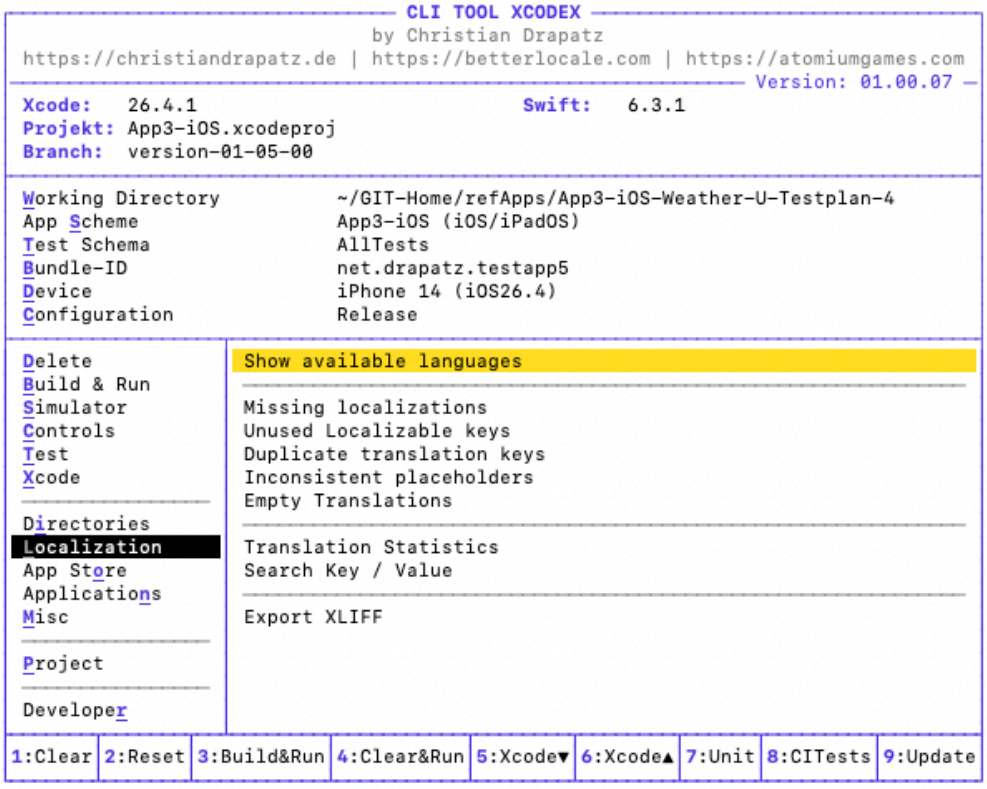
Action	Path	Content
`Finder → Desktop`	`~/Desktop`	Desktop files
`Finder → Downloads`	`~/Downloads`	Downloads folder
`Finder → Documents`	`~/Documents`	Documents
`Finder → Project folder`	current working directory	.xcworkspace / .xcodeproj and project files

Finder → DerivedData	~/Library/Developer/Xcode/DerivedData	Entire DerivedData of all projects
Finder → DerivedData/\<ProjectName\>	~/Library/Developer/Xcode/DerivedData/<ProjectName>-…	DerivedData of the current project
Finder → Build Logs	…/DerivedData/<ProjectName>/Logs/Build	Build logs of the current project
Finder → SourcePackages	…/DerivedData/<ProjectName>/SourcePackages	Cloned SPM packages of the current project
Finder → Module Cache	~/Library/Developer/Xcode/DerivedData/ModuleCache.noindex	Pre-compiled module cache
Finder → Xcode Caches	~/Library/Caches/com.apple.dt.Xcode	Xcode's own cache data
Finder → Simulators	~/Library/Developer/CoreSimulator/Devices	All simulator devices and app data
Finder → Simulator Cache	~/Library/Developer/CoreSimulator/Caches	Temporary simulator caches
Finder → Archives	~/Library/Developer/Xcode/Archives	Xcode archives (.xcarchive)
Finder → DiagnosticReports	~/Library/Logs/DiagnosticReports	Crash and diagnostic reports
Finder → iOS Device Logs	~/Library/Logs/CrashReporter	Crash logs

		from connected devices
Finder → Provisioning Profiles	`~/Library/MobileDevice/Provisioning Profiles`	Installed provisioning profiles

15. Localization

Localization errors often only surface at runtime — when the app has already been shipped. xcodex checks the `.xcstrings` file fully statically, without a build process and without Xcode. Missing keys, duplicate entries and inconsistencies between languages are made immediately visible.



Localization in xcodex

Localization: missing keys, duplicate keys and consistency check

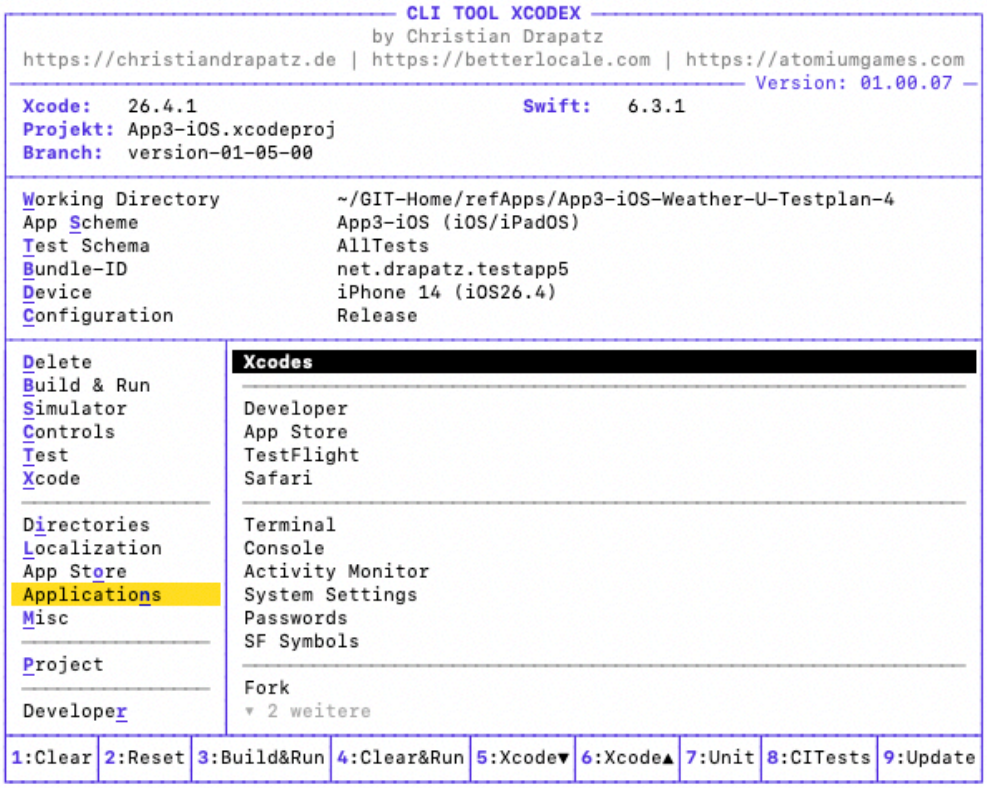
Action	What happens	When to use

`Show available languages`	Lists all detected language codes from .lproj folders and .xcstrings files	Overview of which languages are present in the project
`Missing localizations`	Shows keys that are completely missing in at least one language	Check before a release whether all languages are complete
`Unused localizable keys`	Shows keys present in localization files but not referenced in code	Clean up outdated or forgotten keys
`Duplicate translation keys`	Finds keys that appear multiple times in the same file	For unexplained translation errors
`Inconsistent placeholders`	Detects keys where placeholders (%@, %d etc.) differ between languages	Prevents crashes from incorrect format arguments
`Empty translations`	Finds keys with an existing but empty entry ("" or whitespace only)	Complements the missing key check — empty values are otherwise overlooked
`Translation statistics`	Shows the translation progress per language as a progress bar from .xcstrings files	Quick overview of how complete each language is
`Search key / value`	Interactive search in .xcstrings by key name or translation value	Quickly find a specific text or key
`Export XLIFF`	Runs xcodebuild -exportLocalizations and places a .xcloc package on the desktop	Create translation packages for external translators

> **Note:** xcodex checks the `.xcstrings` file directly — no dependency on Xcode or a build process.

16. Apps

The "Apps" command group lets you open frequently used macOS apps directly from the toolbox — no Dock, Spotlight or Finder required. Only apps that are actually installed appear in the list. Apps are launched via their full path, which is more reliable on macOS 26+ than the `open -a` command.



Apps in xcodex

Open frequently used macOS apps directly from the toolbox

App	Group	Purpose
`Xcodes`	Apple Developer Tools	Manage, install and switch Xcode versions
`Developer`	Apple Developer Tools	Apple Developer Portal and documentation
`App Store`	Apple Developer Tools	Open the App Store
`TestFlight`	Apple Developer Tools	Manage and test beta apps
`Safari`	Apple Developer Tools	Web browser for documentation and research
`Terminal`	System Tools	Command line
`Console`	System Tools	Read system logs and diagnostic messages
`Activity Monitor`	System Tools	Monitor CPU, memory and process usage
`System Settings`	System Tools	macOS settings
`Passwords`	System Tools	Manage macOS passwords

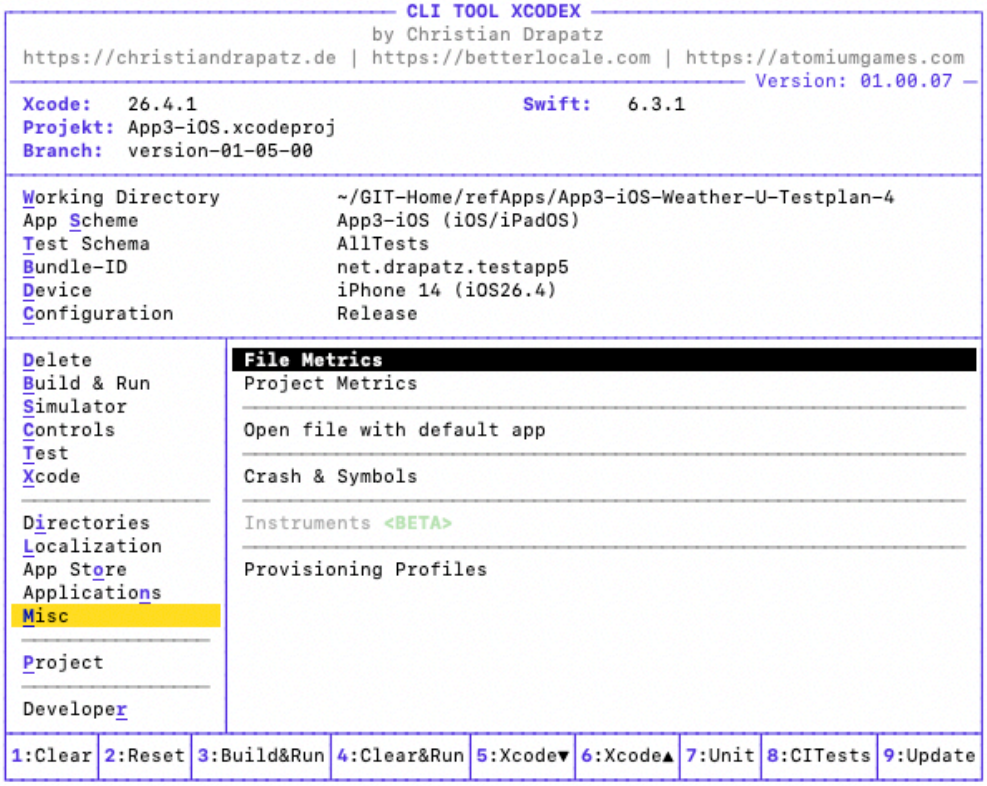
		and Keychain
`SF Symbols`	System Tools	Browse and copy Apple SF Symbols
`Fork`	Third-party	Git client
`Insomnia`	Third-party	REST and API testing
`Cyberduck`	Third-party	FTP/SFTP/S3 file transfer
`DevCleaner`	Third-party	Clean Xcode caches and developer data

> Apps that are not installed are hidden. The list adapts automatically to the software available on your system.

---

17. Miscellaneous

Miscellaneous bundles functions that appear less frequently in the daily workflow but can be critical at the right moment. This includes analyzing crash reports, launching Instruments sessions, and managing provisioning profiles. Especially with hard-to-reproduce crashes or build issues right before a release, these tools are invaluable — and accessible directly from xcodex.



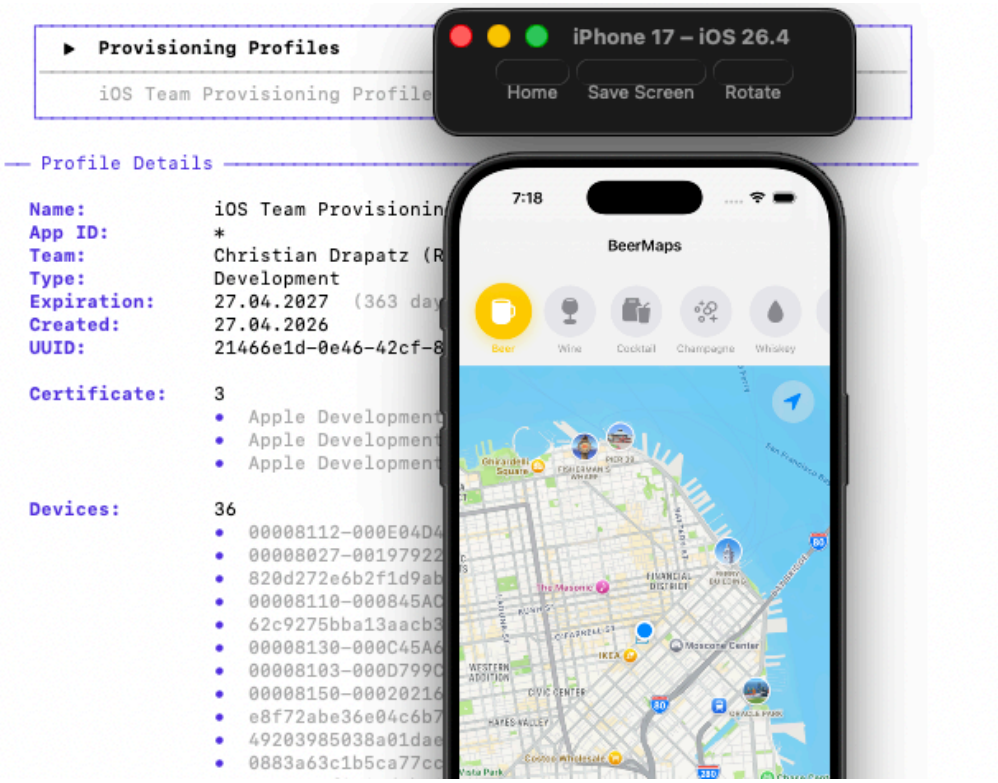
Miscellaneous in xcodex

Crash reports, Instruments and provisioning profile management

- **Analyze Crash Reports** — symbolizes `.crash` files and shows readable stack traces. Requires the matching `.dSYM` archive
- **Instruments (Beta)** — starts profiling sessions directly from xcodex
- **Manage Provisioning Profiles** — lists all local profiles and allows removal of expired ones

Provisioning profiles at a glance

This view shows you all locally available provisioning profiles at a glance. You can see immediately which ones are valid, which app they belong to and where problems exist. This quickly reveals the cause when builds fail locally — without having to click through Xcode.



Provisioning profiles in xcodex

All local provisioning profiles at a glance — remove expired ones directly

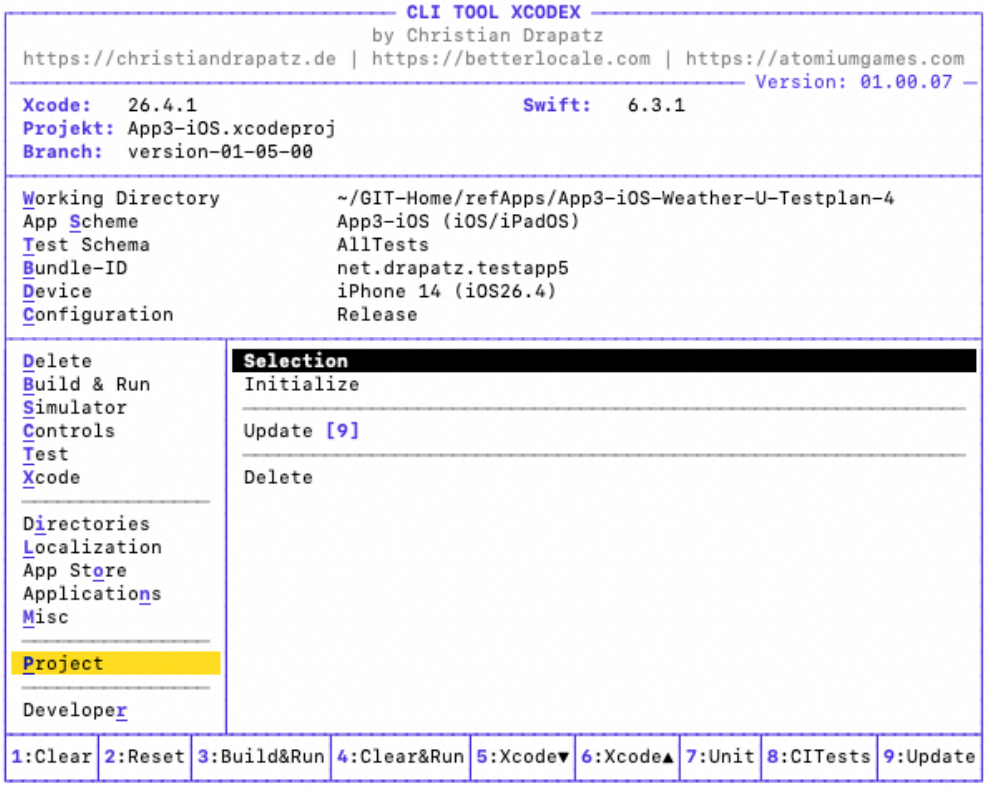
Column	Information
Name	Human-readable profile name
App ID	Bundle Identifier the profile applies to
Type	Development, AdHoc, App Store, Enterprise
Expiry date	When the profile becomes invalid
Devices	Number of registered devices (Development/AdHoc)

> **Tip:** Check the profile list before a release build — an expired distribution profile will block the upload to App Store Connect.

18. Isolated Development Environments

This feature clones a repository into a separate directory on disk. Your own development environment remains completely untouched. You can, for example, check out a feature

branch in parallel and test it with xcodex — while Xcode continues working on your own branch.



Isolated development environments in xcodex

Isolated environments: QA and testers can build and test independently

Workflow

Step	Description
1. Select	On first launch, a JSON configuration file must be selected (see below). If a file was already loaded, xcodex displays it — you simply confirm whether to keep using it or choose a new one.
2. Clean directory	If the target directory already exists, xcodex asks whether to delete it first. This is useful to start with a clean state.
3. Clone repository	The repository is cloned and placed in the configured target directory. This step can be triggered again at any time via the **Initialize** command in the main menu.
4. Set working directory	After cloning, xcodex asks whether to use the cloned directory as the working directory. This corresponds to pressing `A` in the main menu:

	xcodex locates the Xcode project in the cloned directory and connects it as the active project.
5. Update	With **Update**, any branch can be selected. All local changes in the temporary directory are discarded — the branch is reset to the latest commit. This allows you to compare and test older versions or other branches.
6. Delete	With **Delete**, the entire cloned directory is removed. Your own development environment remains untouched.

> **Example scenario:** Xcode is working on your own feature branch. In parallel, xcodex clones the develop branch into a separate directory. Via **Set working directory**, xcodex connects this state as the active project — tests, builds and analyses now run against the develop state without affecting your own work.

Typical use cases

- **QA and testers** — independently build and test the current develop branch without Xcode knowledge
- **Android developers** — try out the iOS app in a defined state
- **UX designers** — quickly launch a feature branch on the simulator
- **Code reviews** — second clone for testing the review branch without touching your own environment

Configuration file

Since xcodex cannot know which repository to clone or where, you need to create a JSON configuration file in advance and place it anywhere on disk. This file is loaded once via **Select** and saved — on the next launch it is automatically pre-selected.

Field	Meaning
`description`	Freely chosen name for this configuration — displayed as a label in xcodex
`base`	Base directory into which all repositories are cloned (must end with `/`)
`repositories[].name`	Display name of the repository
`repositories[].url`	Git URL of the repository (SSH or HTTPS)
`repositories[].destination`	Subdirectory within `base` into which the repository is cloned

The full path of the cloned repository is: `base` + `destination`. In the example below: `/Users/username/Downloads/tempApps/refApps/`

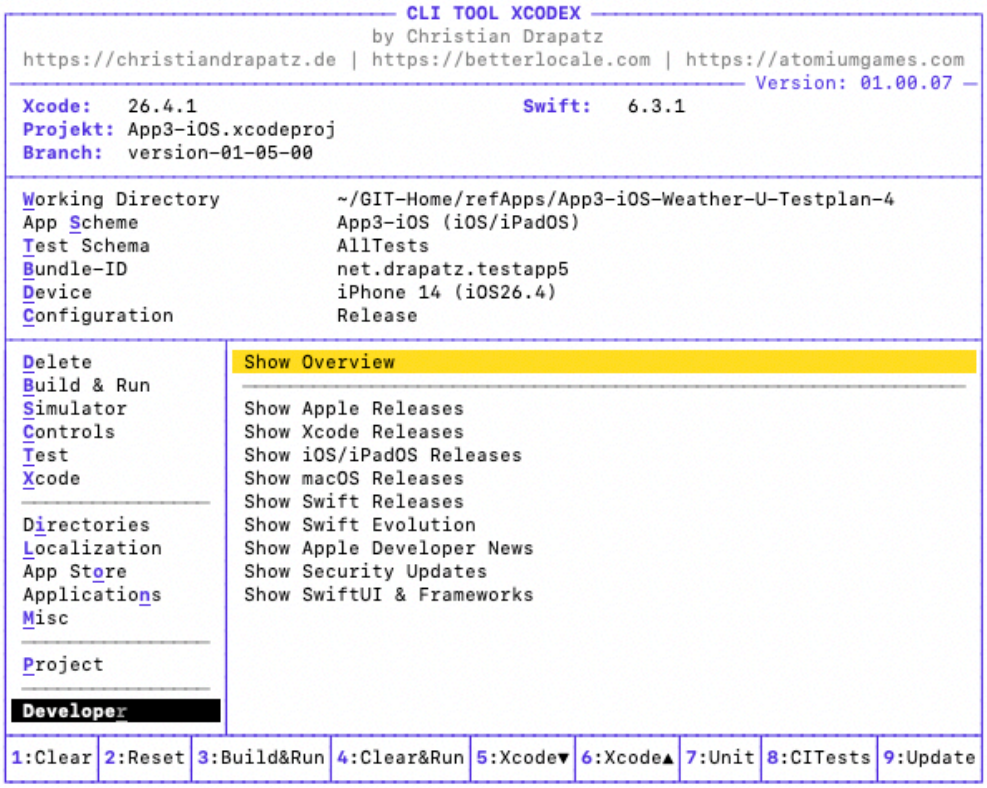
```
{
  "description": "ReferenceApps",
  "base": "/Users/username/Downloads/tempApps/",
  "repositories": [
    {
      "name": "ReferenceApps-Repository",
      "url": "git@github.com:example/repo.git",
      "destination": "refApps"
    }
  ]
}
```

> **Note:** The file can have any name (e.g. `xcodex-env.json`) and be stored anywhere on disk. A fixed location such as the home directory or a project folder is recommended so it can be easily found again.

---

19. Developer Resources

The last command group bundles the most frequently needed documentation and management pages for Apple developers — from Apple Developer Documentation to App Store Connect and the Swift Package Index.



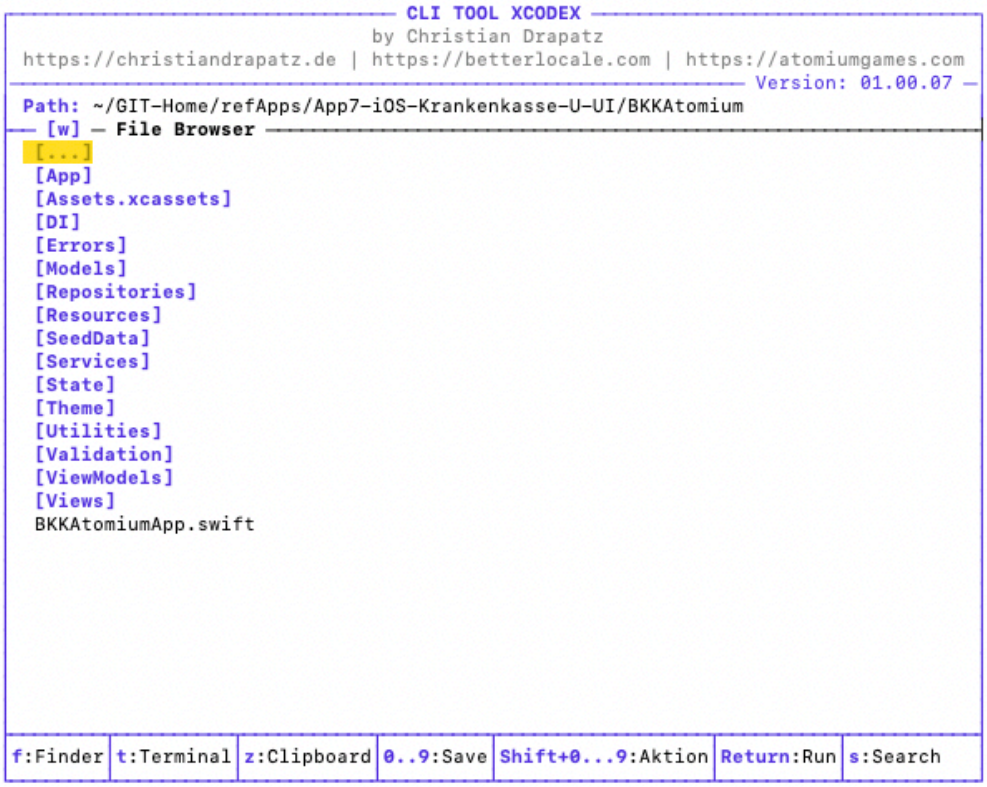
Developer resources in xcodex

Direct access to Apple documentation, App Store Connect and Swift Package Index

Resource	Content
`Apple Developer Documentation`	Official documentation for all Apple frameworks and APIs
Swift Documentation & Evolution Proposals	Language specification and active language proposals
`Human Interface Guidelines`	Design guidelines for iOS, iPadOS and macOS
`App Store Connect`	App management, TestFlight and sales reports
`Apple Developer Portal`	Certificates, provisioning profiles and devices
`Swift Package Index`	Directory of all Swift packages

20. Browser View

The Browser view is a fully keyboard-driven file browser directly in xcodex. You navigate through directories, open files, set favorites and perform searches or filters — all without a mouse and without switching context.



Browser View

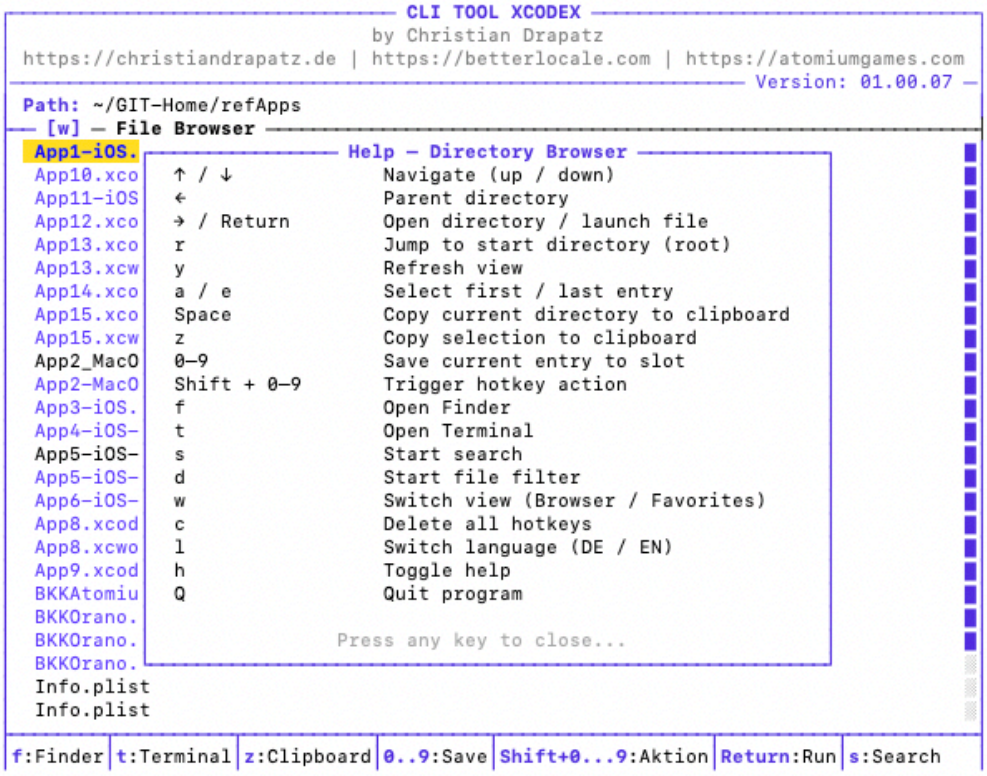
Keyboard-driven file browser directly in xcodex

Hotkeys

Key	Action
`Z`	Jump to the project working directory
`0-9`	Open favorite
`Shift+0-9`	Save current directory as favorite
`F`	Start search
`T`	Activate filter
`S`	Confirm selection / set directory
`D`	Use selected directory as working directory
`W`	Open in Finder

Help

The Browser view includes context-sensitive help. Press `H` to open the help overlay. It shows all available keyboard shortcuts and actions for the current view — clearly laid out and accessible without leaving the Browser view.

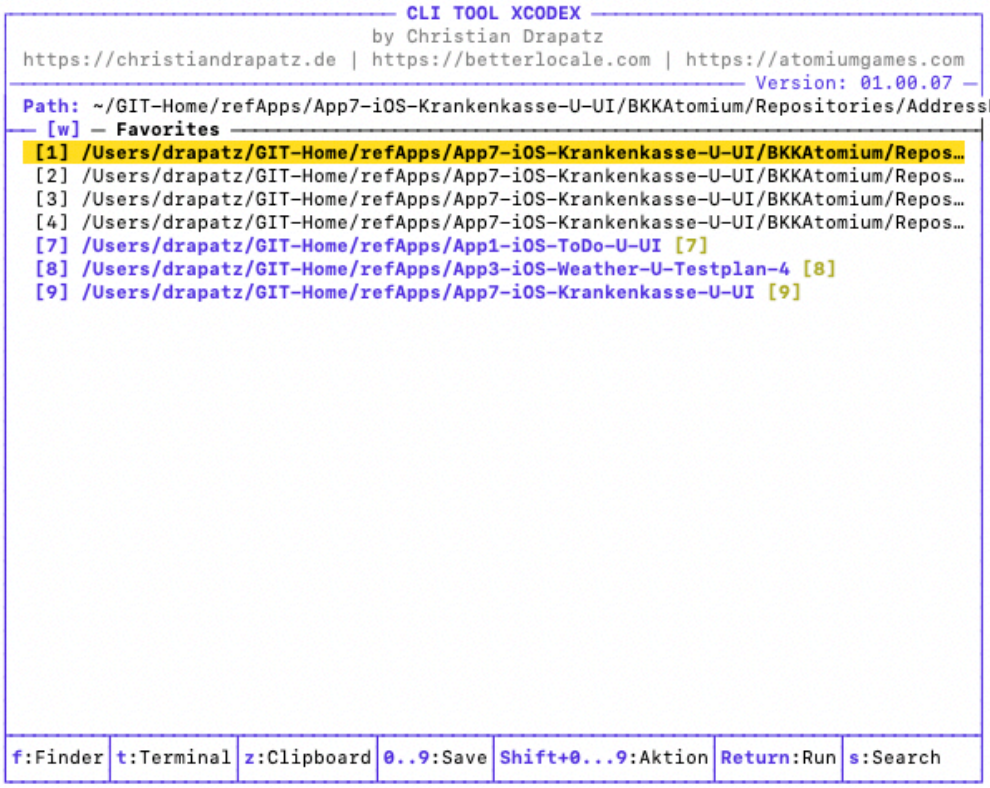


Help in Browser View

Context-sensitive help — all keyboard shortcuts at a glance

Favorites

You can set up to 10 quick-access shortcuts for frequently used directories. Use keys 0–9 to save your favorites and Shift + 0–9 to jump directly to them. Alternatively, press W to switch quickly between the browser and favorites view.

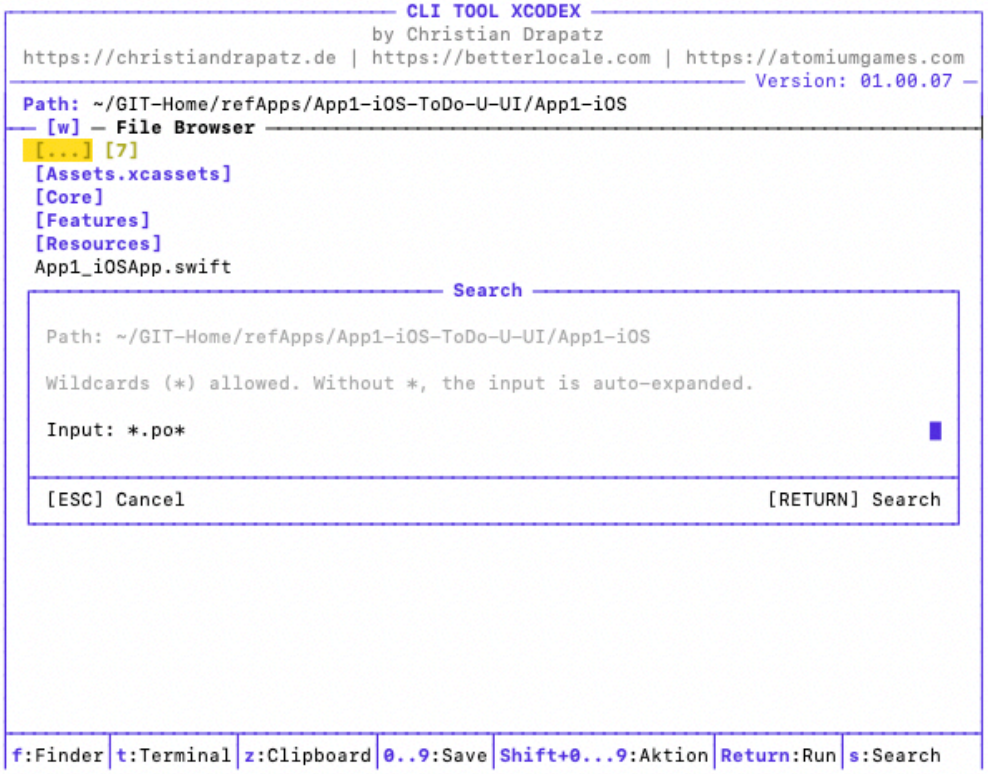


Favorites

Save directories as favorites and access them directly with a number key

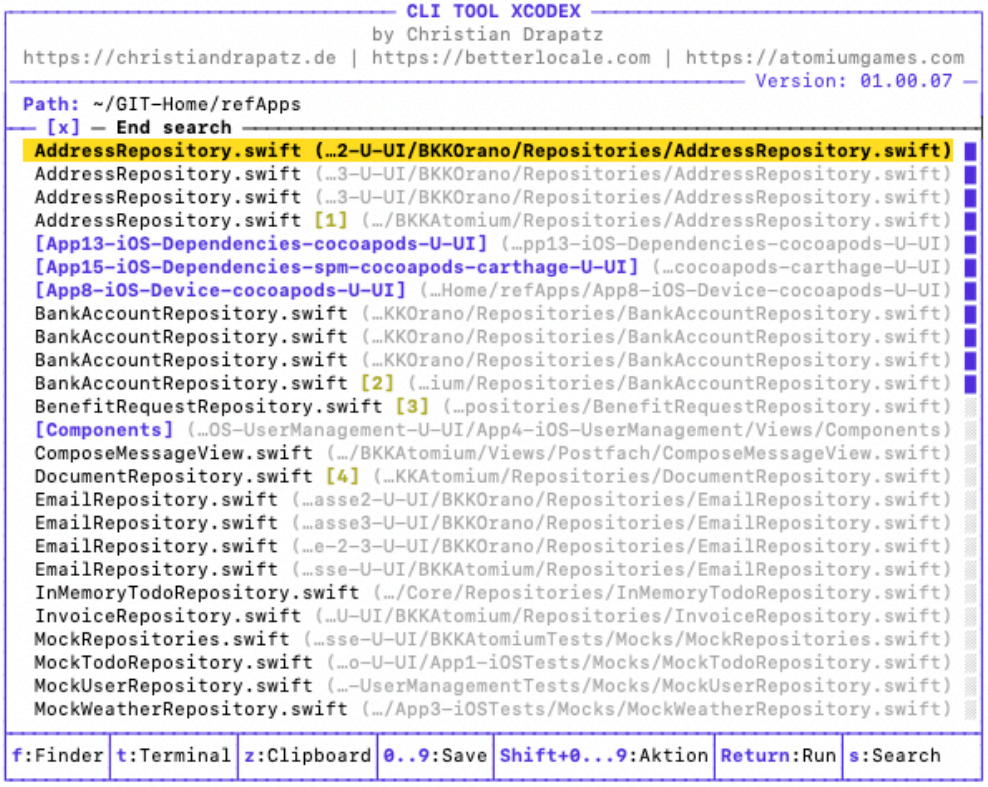
Search

The built-in search scans the current directory including all subdirectories. Wildcards such as `*` are supported, letting you filter flexibly by filename. Results are shown with the full path and can be opened directly.



Search input

Directory search: enter search term

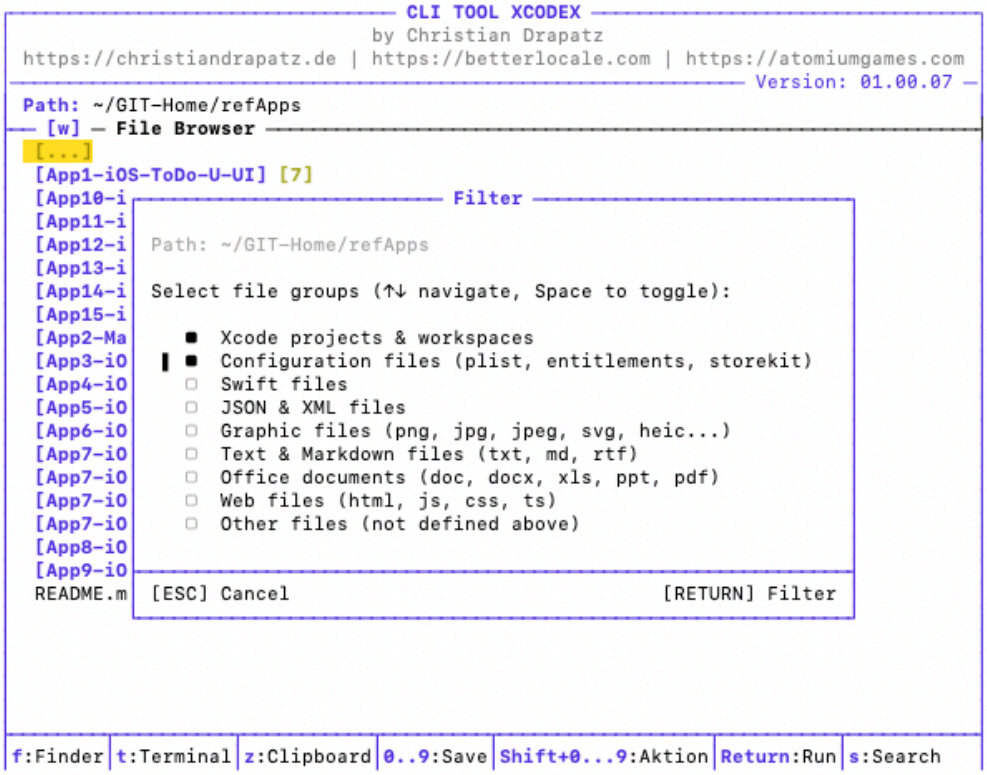


Search results

Search results: direct navigation to found directories

Filter

With filters you can display specific file types, for example Xcode project files or JSON files. You get a clear list with path information and can open the desired files immediately.



Filter input

Filter: narrow the file list by term

Filtered view

Filtered directory view — only matching entries are shown

Quick Access

- **Copy path (Z)** — Copies the absolute path to the clipboard. Useful for external tools such as Claude Code.
- **Open in Finder or Terminal (W)** — Opens the current directory directly. Ideal for continuing work with Claude Code.
- **Open file (Return)** — Opens Xcode projects, Office files or other formats immediately with the appropriate application.

These small features save time and support your daily workflow without interrupting your flow.

---

21. Common Issues

No scheme found

The working directory is not set correctly or the `.xcworkspace`/`.xcodeproj` file is not in the current directory. Launch xcodex from the project root.

Simulator not available

Open Xcode and install the desired simulator platforms. Then restart xcodex — the device list is updated automatically.

Build fails

Take the error message and affected file from the red output. For unexplained errors, delete DerivedData first (*Clean & Cache*) and then build again.

Xcode Command Line Tools missing

```
xcode-select --install
```

Restart the terminal after installation.

Push notification not working

The app must have Push Notifications enabled in the entitlements file and be correctly registered in the Apple Developer Portal. In the simulator, notifications are supported without an APNs certificate.

Permission error on launch

xcodex is not executable. Solution:

```
chmod +x /path/to/xcodex
```

"missing module" error after branch switch

Resolve dependencies after the branch switch: *Build & Run → Resolve All Dependencies*.

---

22. Everyday Tips

Tip	Explanation
Use an alias	Add `xcodex` to `~/.zshrc` so the command works from any directory
Launch from the project directory	xcodex detects the project file automatically when started in the root directory of the project
Before pull requests	Run local builds and tests before creating a PR — this avoids failing CI pipelines

Use isolated DerivedData	xcodex builds into an isolated directory, so parallel Xcode builds are not affected
After `git pull`	Resolve dependencies first (see Dependencies section), then build
Before a release build	Check provisioning profiles and clear simulator cache
Update regularly	`git pull` in the xcodex directory keeps the tool up to date

23. Disclaimer

xcodex is an independent open-source tool and is not affiliated with Apple Inc. "Xcode" and "TestFlight" are registered trademarks of Apple Inc. Use of this tool is at your own risk. No liability is accepted for damages arising from its use.