Tutorial — xcodex

1. Was ist xcodex?

xcodex ist ein terminalbasiertes CLI-Tool für Apple-Entwickler. Es fasst die wichtigsten Xcode-Workflows in einer einzigen, tastaturgesteuerten Oberfläche zusammen — ohne Flags auswendig zu lernen, ohne Kontextwechsel, ohne Xcode öffnen zu müssen.

Das Tool unterstützt iOS-, iPadOS- und macOS-Projekte und deckt den gesamten Entwicklungsalltag ab: Build, Tests, Simulatorsteuerung, Geräteverwaltung, Archive,

TestFlight/App-Store-Auslieferung, Lokalisierung, Code Coverage, Crash-Analyse und Projektpflege.

> **Hinweis:** xcodex steht in keiner Verbindung zu Apple Inc. oder Xcode. Es ist ein eigenständiges Open-Source-Werkzeug und kein offizielles Apple-Produkt.

Download

Tutorial als PDF — Dieses Tutorial als PDF herunterladen — zum Offline-Lesen, Drucken oder Archivieren.	Tutorial als PDF speichern
xcodex herunterladen — Das notarisierte xcodex.pkg Installationspaket kannst du direkt herunterladen. Öffne das Paket und führe die Installation durch — xcodex liegt danach unter `/usr/local/bin/xcodex`.	xcodex.pkg herunterladen

2. Voraussetzungen

Voraussetzung	Hinweis	Download
`Xcode`	~30 GB, einmal öffnen zum Einrichten der Komponenten	App Store ↗ · developer.apple.com ↗
`Xcode Command Line Tools`	Enthält Git, xcodebuild, clang, make, svn	Download ↗ · `xcode-select --install`
Homebrew *(optional)*	Für CocoaPods und weitere Tools	brew.sh ↗ · `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`
CocoaPods	Nur wenn	`brew install cocoapods`

(optional)	das Projekt CocoaPods nutzt	
Carthage *(optional)*	Nur wenn das Projekt Carthage nutzt	`brew install carthage`
xcbeautify *(optional)*	Für lesbare Build-Ausgaben	`brew install xcbeautify`
xcodes CLI *(optional)*	Für die Verwaltung von Xcode Versionen	github.com/XcodesOrg/xcodes ↗️ · `brew install xcodesorg/made/xcodes`

3. Installation

Es gibt zwei Möglichkeiten, xcodex zu installieren:

- **Git-Repository** — vollständiger Quellcode, Updates einfach per `git pull`
- **Installationspaket** — notarisierte xcodex.pkg direkt herunterladen und per Installer einrichten

1) Git-Repository

1.1) Repository klonen

Dieser Befehl lädt den gesamten Quellcode von xcodex auf deinen Mac herunter. Git erstellt dabei das Verzeichnis `~/Developer/xcodex` mit allen Dateien. Du benötigst Git, um Updates später einfach per `git pull` einzuspielen, ohne das Tool erneut manuell herunterzuladen.

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

1.2) Ausführungsrechte setzen

macOS erlaubt standardmäßig nicht das direkte Ausführen von heruntergeladenen Dateien. Mit `chmod +x` gibst du dem Betriebssystem die Erlaubnis, die Datei `xcodex` als Programm zu starten. Dieser Schritt muss nur einmalig nach dem Klonen durchgeführt werden.

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

1.3) Shell-Alias einrichten (empfohlen)

Ein Shell-Alias sorgt dafür, dass du `xcodex` aus jedem beliebigen Verzeichnis starten kannst — ohne den vollständigen Pfad eintippen zu müssen.

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
```

> **Welche Shell nutze ich?** Im Terminal den Befehl `echo $SHELL` ausführen. Die Ausgabe zeigt den Pfad zur aktiven Shell — z. B. `/bin/zsh`, `/bin/bash` oder `/usr/local/bin/fish`.

1.4) Ausführen

Wechsle in das Wurzelverzeichnis deines Xcode-Projekts und starte xcodex. Beim ersten Start musst du dein Projekt oder Workspace manuell auswählen, damit das Tool alle nötigen Informationen erhält — Schema, Gerät und Konfiguration.

```
cd YourXcodeProject
xcodex
```

1.5) Aktualisieren

Mit `git pull` lädt Git die neuesten Änderungen aus dem Repository herunter und wendet sie auf deine lokale Kopie an.

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

2) Installationspaket

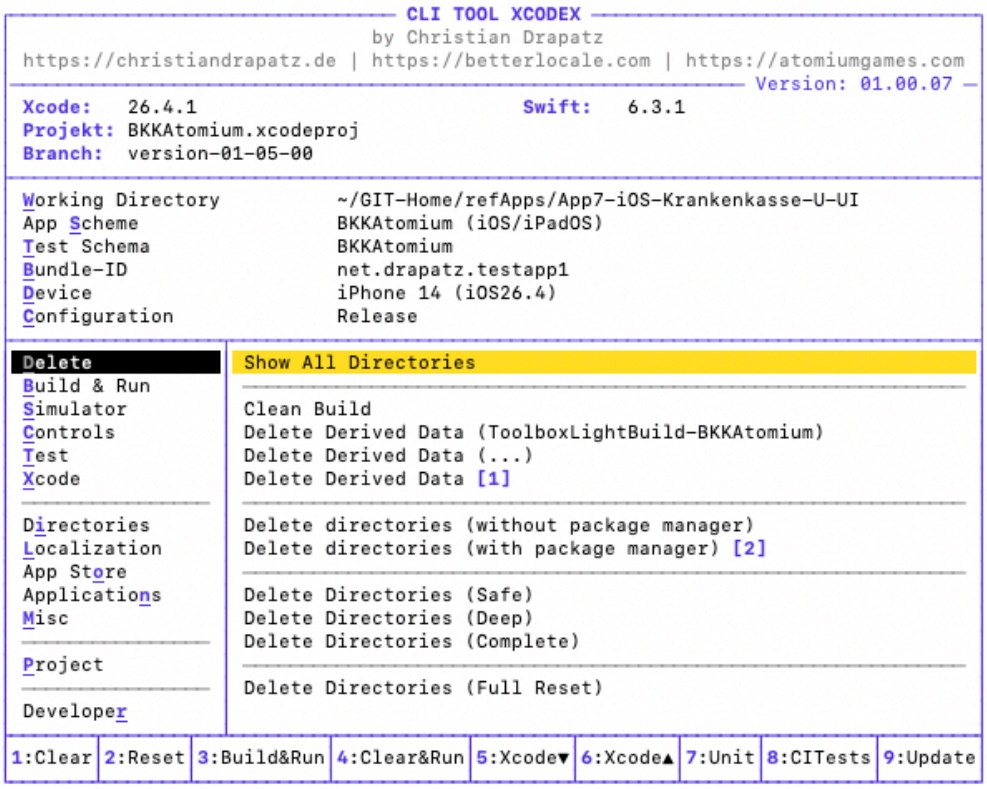
Das notarisierte xcodex.pkg Installationspaket kannst du direkt herunterladen. Apple hat die Datei geprüft und freigegeben, sodass sie den Sicherheitsanforderungen von macOS entspricht und ohne Warnmeldungen öffnet. Öffne das Paket und führe die Installation durch — xcodex wird automatisch nach `/usr/local/bin/xcodex` kopiert.

xcodex.pkg herunterladen

4. Die Menü-Oberfläche

Die Oberfläche ist zweigeteilt: links die Befehlsgruppen, rechts die zugehörigen Aktionen. Navigation erfolgt ausschließlich per Tastatur.

Am unteren Rand befindet sich eine Hotkey-Leiste mit 9 Schnellzugriffsbefehlen, die sich direkt über die entsprechende Zifferntaste aufrufen lassen.



Menü-Oberfläche von xcodex

Split-Pane-Interface: Befehlsgruppen links, Aktionen rechts

Navigationstasten

Taste	Aktion
↑ / ↓	Innerhalb einer Spalte navigieren
← / →	Zwischen Gruppen und Aktionen wechseln
↵	Aktion ausführen
H	Kontexthilfe zur aktuellen Aktion anzeigen

`X`	Zurück zum Menü
`Shift+Q`	Programm beenden
`1–9`	Direktzugriff auf Hotkeys in der Leiste unten
`Shift+<Buchstabe>`	Befehlsgruppe direkt anspringen
`L`	Sprache der Oberfläche umschalten
`Space`	Zwischen XCode-Ansicht und Browser-Ansicht wechseln
`Shift+A`	Browser-Ansicht direkt im Arbeitsverzeichnis öffnen
`Shift+W`	Direkt zur Browser-Ansicht springen

Alle Einstellungen bleiben automatisch zwischen den Sitzungen erhalten.

> **Hotkeys vs. Befehle:** Die folgende Tabelle listet alle globalen Hotkeys. Im Gegensatz zu den Befehlen in der Aktionsliste — die teils Rückfragen stellen — führen Hotkeys Aktionen sofort ohne weitere Eingaben aus und eignen sich daher besonders für schnelle, wiederkehrende Abläufe.

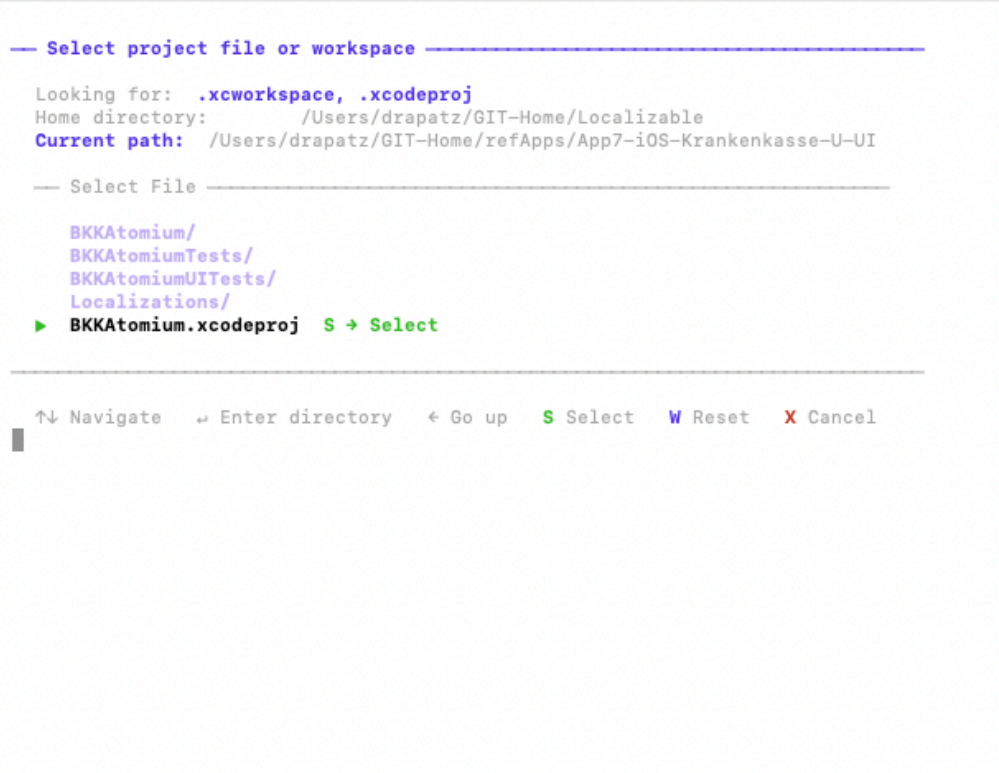
Globale Hotkeys

Hotkey	Befehl	Beschreibung
`1`	DerivedData löschen	Entfernt den DerivedData-Ordner des Projekts und erzwingt einen sauberen Rebuild
`2`	Alle Caches löschen	Löscht DerivedData, Module-Cache und Package-Manager-Caches (SPM, CocoaPods, Carthage)
`3`	Build & Run	Baut die App und startet sie direkt auf dem aktiven Gerät (Simulator, physisches Gerät oder Mac)
`4`	Quick Reset & Build	Löscht DerivedData des Projekts, baut neu und startet die App — schneller Weg bei hartnäckigen Build-Fehlern
`5`	Xcode schließen	Beendet Xcode, ohne den

		Terminal-Fokus zu verlieren
6	Projekt in Xcode öffnen	Öffnet das aktive Projekt direkt in Xcode
7	Unit-Tests ausführen	Startet den Unit-Test-Durchlauf mit dem konfigurierten Test-Schema
8	Automatisierte Tests	Öffnet das Menü für automatisierte und kombinierte Testläufe
9	Projekt aktualisieren	Löst ein Projekt-Update aus (z. B. Abhängigkeiten neu auflösen)

5. Erste Schritte — Projekt einrichten

Bevor du deine App auf einem Simulator oder echten Gerät bauen oder testen kannst, musst du xcodex einmal konfigurieren.



Projekt einrichten in xcodex

Schritt-für-Schritt: Projekt auswählen, Schema, Gerät und Konfiguration festlegen

Schritt 1 — Arbeitsverzeichnis auswählen

Im Hauptmenü drückst du `A`. Es öffnet sich der Datei-Browser, in dem du dein Xcode-Projekt (`.xcodeproj`) oder Workspace (`.xcworkspace`) auswählst.

Mit `S` bestätigst du die Auswahl. Anschließend wirst du gefragt, ob der Xcode-Cache (DerivedData) gelöscht werden soll.

Schritt 2 — App-Schema festlegen

Jetzt wählst du das App-Schema aus. Das Schema bestimmt, was gebaut wird und wie die App gestartet wird.

Schritt 3 — Test-Schema oder Testplan festlegen

Falls vorhanden, kannst du ein separates Test-Schema oder einen Testplan auswählen. Das ist sinnvoll, wenn deine Tests in einem eigenen Target oder über Testpläne organisiert sind.

Schritt 4 — Zielgerät auswählen

Als Nächstes wählst du das Gerät aus, auf dem gebaut oder getestet werden soll. Verfügbare Simulatoren sowie angeschlossene physische Geräte werden automatisch aus deiner Xcode-Installation erkannt.

Schritt 5 — Build-Konfiguration wählen

- **Debug** — für die tägliche Entwicklung mit Logs und Debug-Informationen
- **Release** — für optimierte Builds, z. B. für den App Store

Danach ist alles eingerichtet und du kannst direkt mit Build, Tests und weiteren Funktionen in xcodex starten.

> **Tipp:** Schema, Gerät und Konfiguration lassen sich jederzeit über `S`, `G` und `K` im Menü ändern, ohne das Tool neu zu starten. Alle Einstellungen werden automatisch gespeichert.

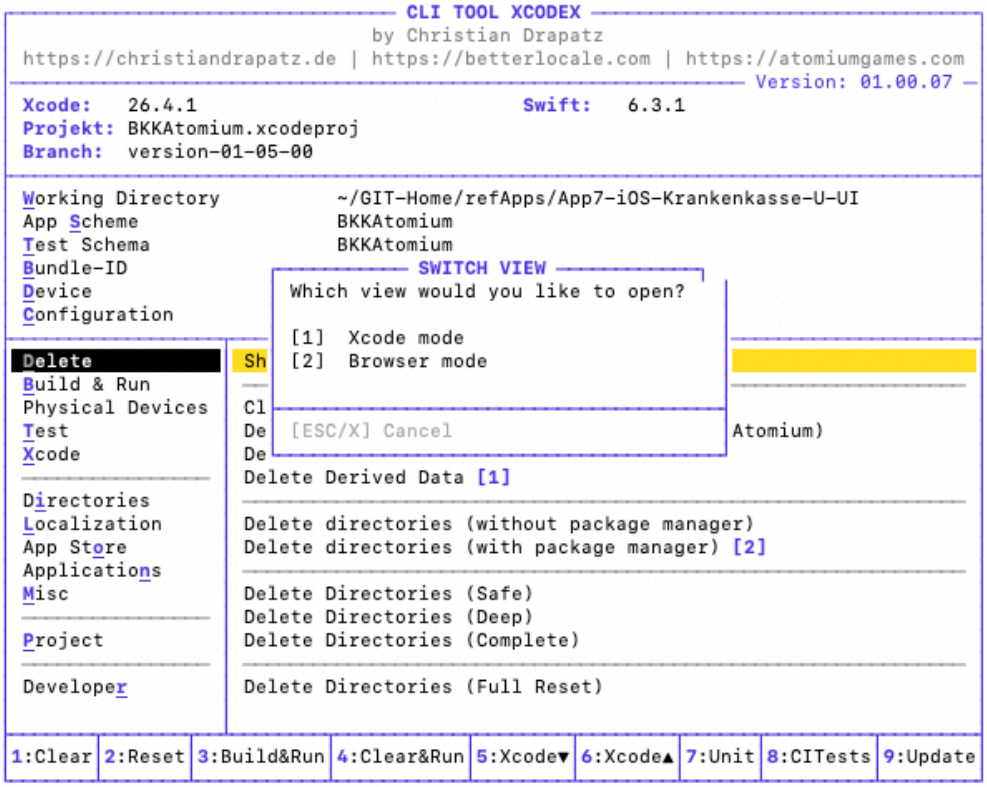
Schnell-Shortcuts im Header

Taste	Funktion
`A`	Arbeitsverzeichnis wählen
`S`	App-Schema auswählen
`T`	Test-Ziel auswählen

`G`	Zielgerät auswählen
`K`	Build-Konfiguration wechseln (Debug / Release)
`B`	Bundle ID anzeigen und kopieren
`L`	Sprache der Oberfläche umschalten (Deutsch / Englisch)

6. XCode-Ansicht vs. Browser-Ansicht

xcodex hat zwei Hauptansichten: die **XCode-Ansicht** mit allen Entwickler-Werkzeugen und die **Browser-Ansicht** — ein tastaturgesteuerter Datei-Browser. Kein Kontextwechsel, keine Maus, kein Fensterwechsel.



Ansichtsauswahl in xcodex

Mit der Leertaste zwischen XCode-Ansicht und Browser-Ansicht wechseln

Taste	Aktion
`Leertaste`	Zwischen XCode-Ansicht und Browser-Ansicht

	wechseln
`Shift+X`	Direkt zur XCode-Ansicht springen
`Shift+W`	Direkt zur Browser-Ansicht springen
`Shift+A`	Browser-Ansicht direkt im Arbeitsverzeichnis öffnen

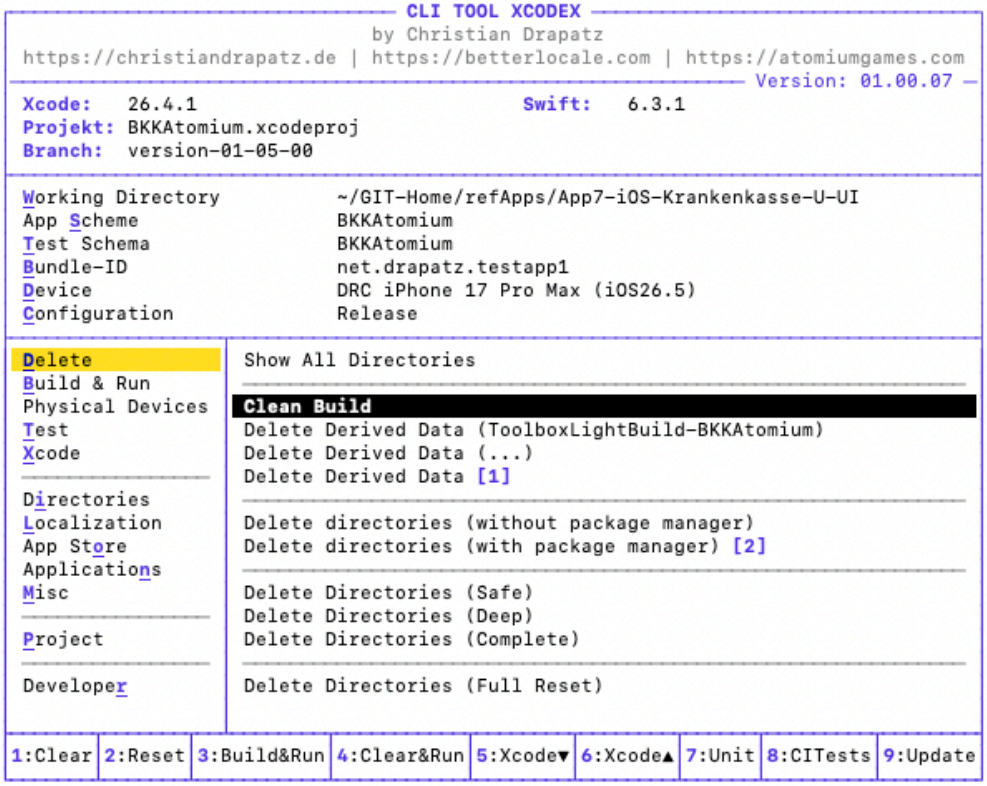
Die **XCode-Ansicht** ist der Ausgangspunkt: Hier baust du, testest, archivierst und verwaltest Simulatoren. Die **Browser-Ansicht** ergänzt sie als schneller Datei-Navigator — ideal zum Prüfen von Verzeichnissen, Öffnen von Dateien oder Setzen eines neuen Arbeitsverzeichnisses, ohne das Terminal zu verlassen.

Wechsel zwischen den Ansichten passiert nahtlos: Die Terminalgröße bleibt identisch, kein Springen, kein Scrollen. Mit `Shift+A` öffnet die Browser-Ansicht sofort im aktuellen Arbeitsverzeichnis des Projekts.

---

7. Clean & Cache

Xcode speichert alle Build-Artefakte in einem gemeinsamen DerivedData-Ordner. Wenn Builds ohne erkennbaren Grund fehlschlagen, die App sich seltsam verhält oder nach einem Refactoring Fehler erscheinen, die im Code nicht existieren — dann hilft ein gezieltes Bereinigen.



Clean & Cache in xcodex

Clean & Cache: verschiedene Bereinigungs-Optionen auf einen Blick

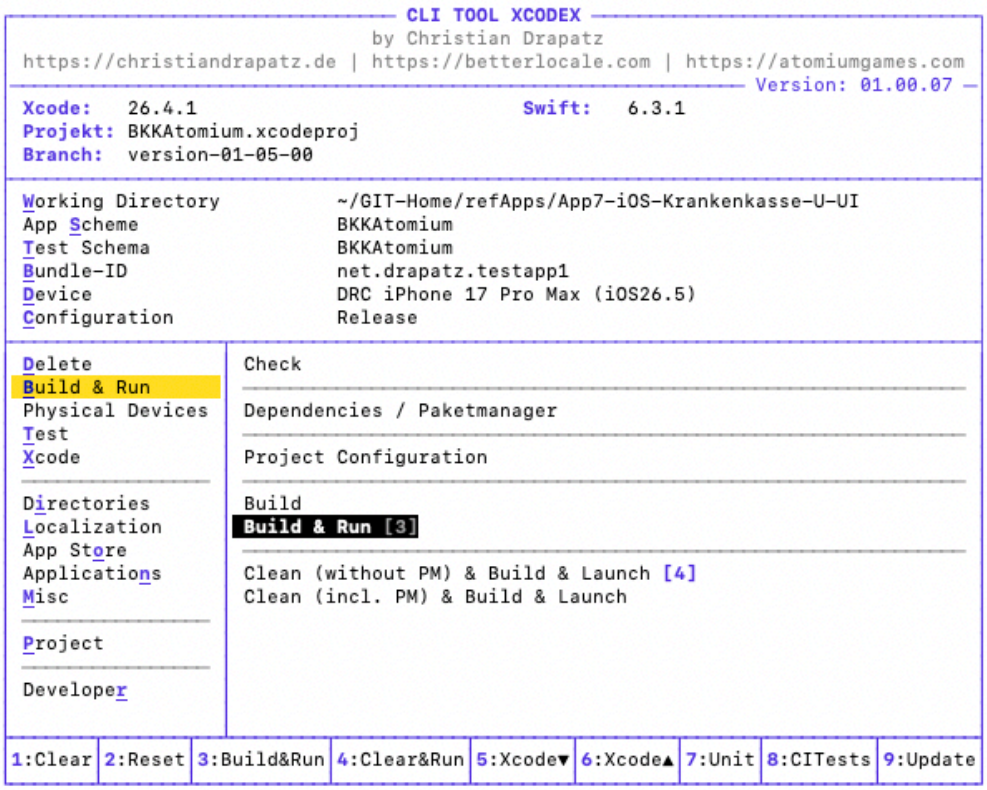
Aktion	Was wird bereinigt	Wann sinnvoll
`Alle Verzeichnisse anzeigen`	Cache-Browser mit Größenanzeige und gezieltem Löschen	Überblick vor dem Löschen
`Build bereinigen`	Kompilierte Artefakte des aktuellen Schemas	Kleiner Reset vor einem sauberen Build
Derived Data (Projektname) löschen	Projektspezifischer DerivedData-Ordner	Bei unerklärlichen Build-Fehlern
Derived Data löschen	Gesamter DerivedData-Ordner	Vollständiger Build-Cache-Reset
Verzeichnisse löschen (ohne Paketmanager)	DerivedData, Xcode-Caches, Simulator-Cache	Schneller Reset ohne Paketmanager anzufassen
Verzeichnisse löschen (mit Paketmanager)	+ CocoaPods-, Carthage- und SPM-Cache	Wenn auch Paketmanager-Caches bereinigt werden sollen
Verzeichnisse löschen (Sicher)	DerivedData, ModuleCache, SwiftPM, Xcode-Caches, Simulator-Cache, Logs, Diagnostic Reports	Umfassender Reset — alles was Xcode automatisch neu anlegt

Verzeichnisse löschen (Tief)	Alles aus „Sicher" + Archives, iOS DeviceSupport, alle Simulator-Geräte	Hartnäckige Probleme; viel Speicher zurückgewinnen
Verzeichnisse löschen (Komplett)	Alles aus „Tief" + CocoaPods-, Carthage- und SPM-Cache; optional Simulator-Runtimes	Maximale Speicherfreigabe
Verzeichnisse löschen (Neustart)	Kompletter Reset inkl. SourcePackages für einen sauberen Projekt-Neustart	Wenn alles neu aufgebaut werden soll

> **Tipp:** Bei unerklärlichem Verhalten zuerst DerivedData löschen — das löst die Mehrheit aller Cache-bedingten Probleme in unter 10 Sekunden.

8. Build & Run

xcodex bietet mehrere vorkonfigurierte Build-Modi, die für häufige Entwicklungsszenarien optimiert sind.



Build & Run Übersicht in xcodex

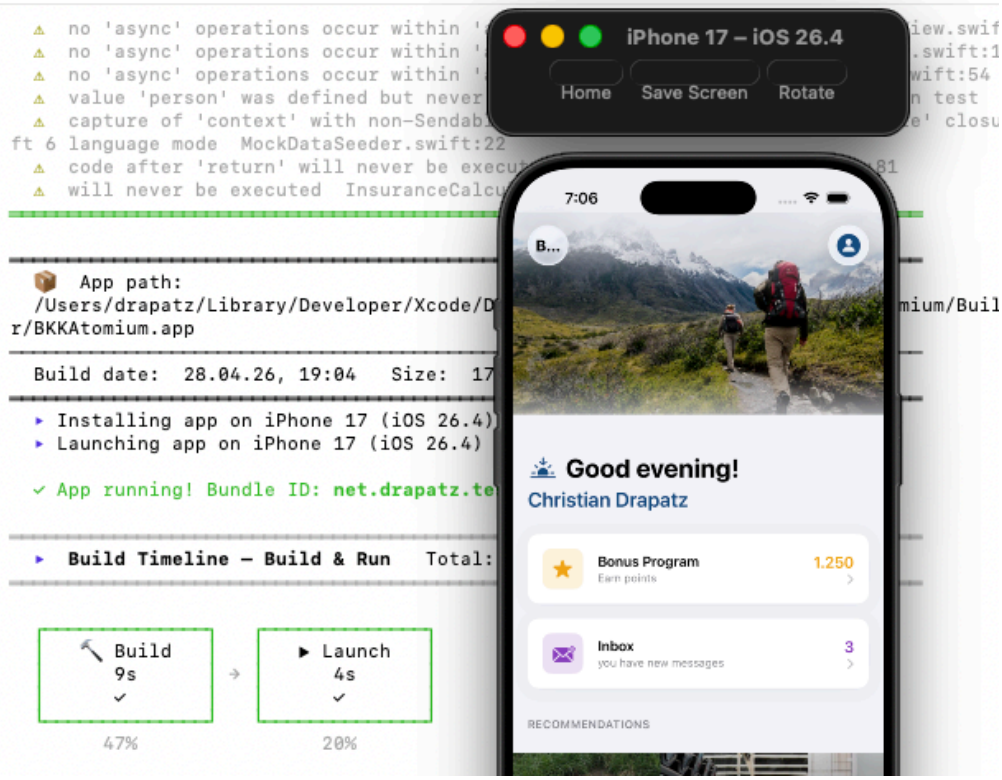
Build & Run: verschiedene Modi für unterschiedliche Situationen

Aktion	Was passiert	Wann sinnvoll
Überprüfen	Prüft das Projekt auf Konfigurationsfehler	Vor dem ersten Build in einem neuen Setup
`Dependencies / Paketmanager`	Löst SPM-, CocoaPods- und Carthage-Abhängigkeiten auf	Nach dem Klonen oder bei Paketfehlern
`Projektkonfiguration`	Zeigt alle Build-Einstellungen des gewählten Schemas	Zur Diagnose von Konfigurationsproblemen
`Bauen`	Kompiliert das Projekt ohne zu starten	Schnelle Kompilierungsprüfung
`Bauen & starten`	Baut und startet die App im Simulator oder auf macOS	Normaler Entwicklungsworkflow
Bereinigen (ohne Paketmanager) & Bauen & Starten	Löscht DerivedData und Xcode-Caches, baut dann neu und startet	Bei Build-Fehlern durch Cache-Probleme
Bereinigen (inkl. Paketmanager) & Bauen & Starten	Löscht alle Caches inkl. Paketmanager-Caches, baut dann neu und startet	Bei hartnäckigen Problemen oder nach Paketmanager-Änderungen

Build-Ausgabe verstehen

Während des Builds zeigt xcodex jeden Schritt mit seiner Laufzeit an. Bei einem Fehler wird die Fehlermeldung rot hervorgehoben und mit der betroffenen Datei und Zeile angezeigt.

Mit der Taste `+` lassen sich weitere Zielgeräte hinzufügen, sodass die App parallel auf mehreren Betriebssystemen läuft.



Build-Ausgabe in xcodex

Build-Ausgabe: strukturiert, komprimiert, mit Zeitangaben pro Schritt

Isoliertes DerivedData

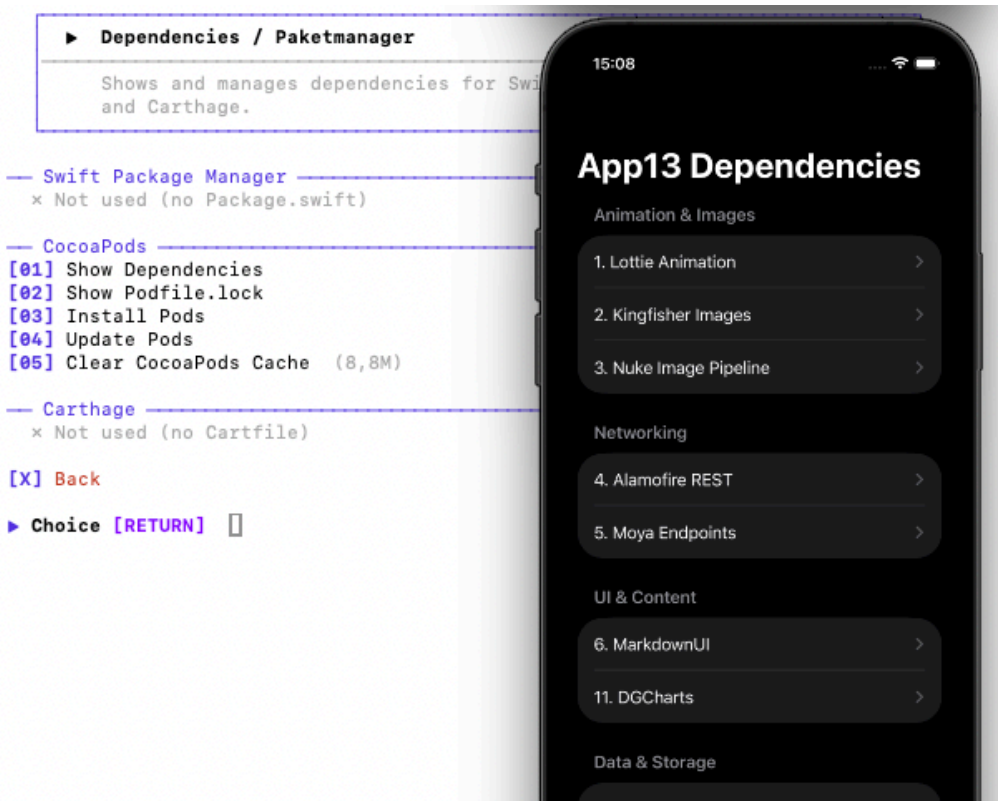
xcodex baut in ein eigenes DerivedData-Verzeichnis, das vollständig von Xcode getrennt ist. Beide können gleichzeitig laufen, ohne sich gegenseitig zu beeinflussen.

> **Tipp:** Mit `v` Projekt validieren lässt sich vor dem Bauen prüfen, ob alle Abhängigkeiten aufgelöst sind und das Schema korrekt konfiguriert ist.

---

9. Abhängigkeiten

Wer CocoaPods, SPM oder Carthage nutzt, kann Abhängigkeiten direkt aus xcodex heraus verwalten — ohne Terminal-Kenntnisse.



Abhängigkeiten in xcodex

Dependencies: SPM, CocoaPods und Carthage in einem Menü

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

> **Tipp:** Nach einem `git pull` zuerst „Alle Dependencies auflösen" ausführen, bevor gebaut wird — das verhindert die häufigsten „missing module"-Fehler nach Branch-Wechseln.

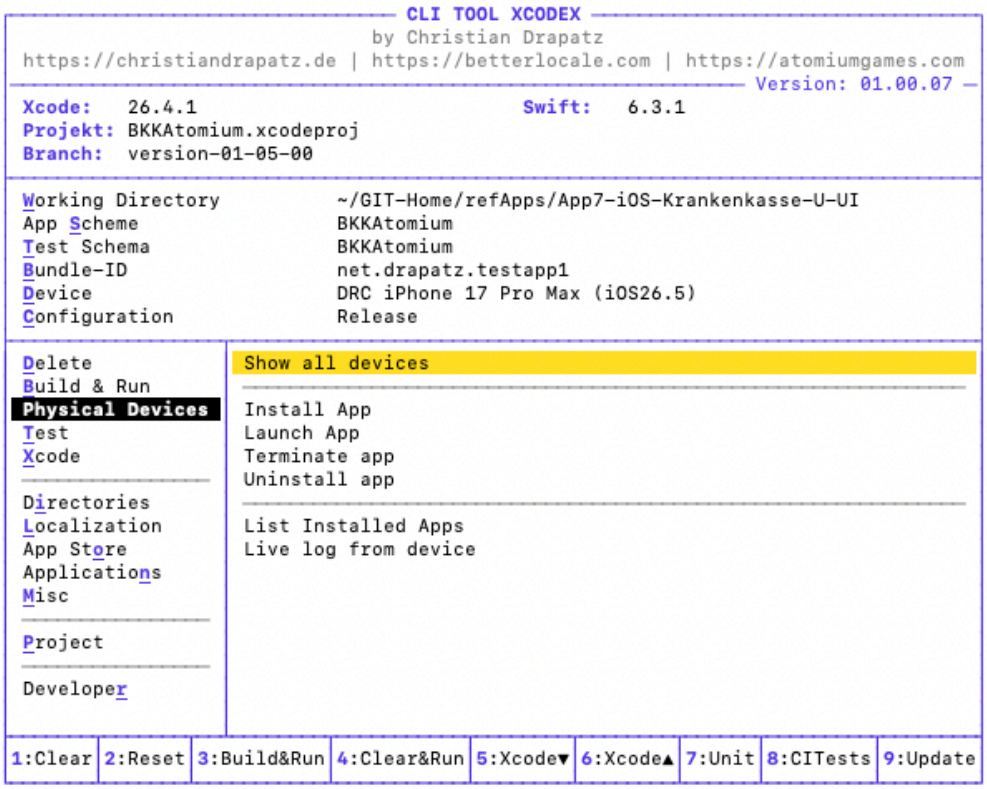
---

10. Simulator & Physische Geräte

Je nach ausgewähltem Zielgerät zeigt xcodex entweder die **Simulator-Gruppe** oder die **Physische-Geräte-Gruppe** an — nie beide gleichzeitig. Die verfügbaren Aktionen passen sich automatisch an das aktive Gerät an.

Simulator

Simulatoren können in einen inkonsistenten Zustand geraten — besonders nach iOS-Updates oder langen Laufzeiten. Diese Gruppe gibt die Kontrolle zurück.



Simulator-Steuerung in xcodex

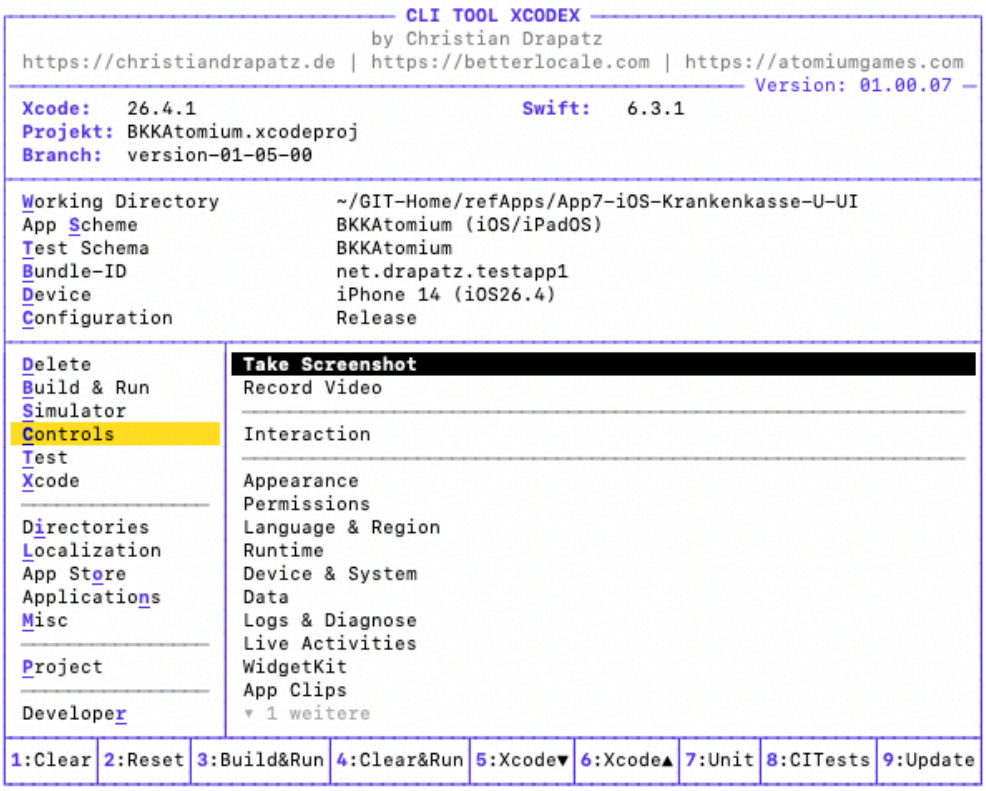
Simulator: App neu installieren, zurücksetzen, alle Instanzen schließen

Aktion	Was passiert	Wann sinnvoll
`Neuen Simulator erstellen`	Runtime und Gerätetyp wählen, Simulator via xcrun simctl create anlegen	Wenn ein bestimmtes Gerät/iOS-Kombination fehlt
`App starten`	Startet die gebaute App auf dem gewählten Simulator	Nach einem Build ohne erneutes Bauen
App auf ausgewähltem Simulator starten	Wie „App starten", aber mit vorheriger Simulator-Auswahl	Zum Testen auf einem anderen Gerät
`App deinstallieren & frisch testen`	Deinstalliert die App und installiert sie neu	Sauberer Start ohne alten App-Zustand
`Simulator neu starten`	Fährt den Simulator herunter und startet ihn neu	Bei hängendem oder instabilem Simulator
`Simulator stoppen`	Fährt den laufenden Simulator herunter	Ressourcen freigeben

Simulator zurücksetzen	Setzt den aktuellen Simulator auf Werkszustand zurück	App-Daten und Einstellungen vollständig entfernen
`Simulator Status anzeigen`	Alle Simulatoren nach Runtime gruppiert, farbkodiert (Booted / Shutdown)	Überblick über installierte Simulatoren
Simulator Inhalte zurücksetzen	Löscht alle Inhalte aller Simulatoren (xcrun simctl erase all)	Vollständiger Reset aller Simulator-Daten
Nicht verfügbare Simulatoren löschen	Entfernt Simulatoren ohne installierte Runtime	Aufräumen nach Xcode-Updates
`Alle Simulatoren stoppen`	Stoppt alle laufenden Simulatoren auf einmal	Ressourcen freigeben, vor einem Build
Alle Simulator-Geräte löschen	Entfernt alle Simulator-Geräteeinträge (xcrun simctl delete all)	Vollständiger Neustart der Simulator-Infrastruktur
Simulator-Caches löschen	Entfernt temporäre CoreSimulator-Caches	Darstellungsfehler oder Verhalten nach iOS-Updates

Simulator fernsteuern

Nach dem App-Start lassen sich Simulator-Einstellungen direkt aus xcodex heraus ändern — ohne den Simulator-Menübaum zu durchsuchen.



Simulator-Controls in xcodex

Controls: Dark Mode, Berechtigungen, Statusleiste, Push Notifications

1. Interaktion

#	Befehl	Nutzen beim Testen
01	Push-Notification testen	Simuliert einen APNs-Payload, ohne echten Server — ideal für Notification-UX-Tests.
02	Deep Link öffnen	Öffnet einen URL-Scheme direkt, ohne die App manuell zu navigieren.

2. Erscheinungsbild

#	Befehl	Nutzen beim Testen
01	Dark / Light Mode umschalten	Prüft, ob alle Assets und Colors korrekt auf beide Modi reagieren.
02	Status Bar mocken	Setzt definierte Werte (Uhrzeit, Signalstärke) für Screenshots und UI-Tests.

03	Status Bar zurücksetzen	Setzt den Status Bar-Mock zurück auf den echten Systemzustand.
04	Dynamic Type setzen	Testet Layout-Stabilität bei extremen Schriftgrößen.
05	Bold Text umschalten	Prüft Lesbarkeit bei aktivierter Fett-Schrift-Barrierefreiheit.
06	Reduce Motion umschalten	Stellt sicher, dass Animationen korrekt deaktiviert werden.
07	Increase Contrast umschalten	Validiert Kontrastverhältnisse für Barrierefreiheit.
08	Display Zoom setzen *(iPhone only)*	Testet Layout bei vergrößertem Display-Modus.
09	Reduce Transparency umschalten	Prüft UI-Elemente mit deaktiviertem Blur-Effekt.
10	On/Off-Labels umschalten	Sichert, dass Toggles auch ohne Farbe erkennbar sind.
11	Differentiate Without Color	Testet, ob die App nicht ausschließlich auf Farbe als Informationsträger setzt.

3. Berechtigungen

#	Befehl	Nutzen beim Testen
01	Berechtigungen setzen	Granulare Steuerung einzelner Privacy-Grants.
02	Alle App-Berechtigungen zurücksetzen	Setzt den App-Berechtigungsstatus auf „nicht gefragt" — für Permission-Flow-Tests.
03–19	Quick Grants (Kamera, Mikrofon, Standort, Fotos, Kontakte, Kalender, Erinnerungen, Bewegung, Health, Netzwerk, Bluetooth, HomeKit, Mediathek, Siri, Sprache, Lokales Netz, User-Tracking)	Einzelne Berechtigung in einem Schritt erteilen — spart Dialog-Klicks bei jedem Testlauf.
20	Alle Berechtigungen erteilen	Gesamte Berechtigungsliste auf

		„erlaubt" — für Schnelltests jenseits des Permission-Flows.

4. Sprache & Region

#	Befehl	Nutzen beim Testen
01	App-Sprache setzen	Prüft Lokalisierung der App ohne Systemsprache zu ändern.
02	Systemsprache setzen	Testet globale Spracheffekte (Tastatur, Systemdialoge).
03	24h-Format umschalten	Validiert Zeitdarstellung in beiden Formaten.
04	Systemsprache zurücksetzen	Setzt alle Spracheinstellungen auf Werksstandard zurück.

5. Laufzeit

#	Befehl	Nutzen beim Testen
01	Slow Animations	Verlangsamt Animationen — Fehler im Übergang werden sichtbar.
02	Clipped Views hervorheben	Zeigt abgeschnittene UI-Elemente sofort rot markiert.
03	Launch Arguments setzen	Injiziert -Argument-Flags für Feature-Flags und Test-Modi ohne Build.
04	Env-Variablen setzen	Übergibt Umgebungsvariablen an die App beim Start.
05	Memory Pressure simulieren	Testet das Verhalten der App bei Speicherdruck (Warnstufen).
06	URL-Scheme testen	Öffnet einen Deep Link und prüft die Verarbeitung.
07	Core Data Debug	Aktiviert SQL-Logging für Core Data — zeigt alle Queries.
08	Localization Debug	Aktiviert doppelte Länge / Pseudosprache — Layout-Probleme werden sofort

		sichtbar.
09	SwiftUI Debug	Zeigt SwiftUI-Render-Grenzen und Invalidierungen in der Konsole.
10	Background Fetch	Löst einen Hintergrund-Fetch manuell aus — ohne 20-Minuten-Wartezeit.
11	Background Launch	Startet die App im Hintergrundmodus.
12	Metal Validation	Aktiviert Metal-API-Validierung für GPU-Code.

6. Gerät & System

#	Befehl	Nutzen beim Testen
01	Standort setzen	Simuliert GPS-Koordinaten ohne echten Standort.
02	Batteriestatus simulieren	Testet Low-Battery-Warnungen und Ladestand-Logik.
03	Cellular-Modus setzen	Simuliert EDGE, LTE, 5G oder kein Signal.
04	IDFA zurücksetzen	Setzt die Advertising-ID zurück — für Tracking-Tests.
05	iCloud-Sync auslösen	Erzwingt einen iCloud-Sync ohne Wartezeit.
06	Zeitzone setzen	Testet zeitzonensensitive Logik (Kalender, Countdowns).
07	Status Bar Uhrzeit mocken	Setzt eine feste Uhrzeit für konsistente Screenshots.
08	Standortsimulation stoppen	Beendet den simulierten GPS-Standort.
09	WLAN-Balken setzen	Testet UI bei verschiedenen WLAN-Signalstärken.
10	Cellular-Balken setzen	Testet UI bei verschiedenen Mobilfunk-Signalstärken.
11	Netzanbieter-Name setzen	Setzt einen eigenen Carrier-Namen in der Status Bar.

12	Stage Manager *(iPad only)*	Testet das Layout im Stage-Manager-Modus.
13	Audio-Route setzen	Wechselt zwischen Lautsprecher, Kopfhörer, Bluetooth.

7. Daten

#	Befehl	Nutzen beim Testen
01	Medien hinzufügen	Importiert Bilder/Videos in die Foto-Bibliothek des Simulators.
02	Simulator-Keychain leeren	Entfernt alle gespeicherten Keychain-Einträge der App.
03	App-Datenordner öffnen	Öffnet den Container direkt im Finder — für manuelles Datei-Debugging.
04	App UserDefaults zurücksetzen	Löscht alle gespeicherten UserDefaults-Werte der App.
05	Alle App-Daten löschen	Entfernt den kompletten App-Container.
06	App-Group-Container öffnen	Öffnet den geteilten Container im Finder.
07	App-Defaults anzeigen	Listet alle aktuellen UserDefaults-Keys und -Werte.
08	SQLite-Datenbank öffnen	Öffnet die App-Datenbank direkt in einem externen Editor.
09	Keychain-Einträge anzeigen	Listet alle gespeicherten Keychain-Einträge der App.
10	Root-Zertifikat installieren	Installiert ein eigenes CA-Zertifikat für SSL-Pinning-Tests.
11	App-Container messen	Zeigt die Größe des App-Containers auf Disk.
12	App-Container exportieren	Exportiert den Container als Archiv für Offline-Analyse.
13	UserDefaults-Key bearbeiten	Ändert einzelne Keys direkt — ohne App-Neustart.

8. Logs & Diagnose

#	Befehl	Nutzen beim Testen
01	App-Logs streamen	Live-Log-Ausgabe der App in der Konsole.
02	Konsole filtern	Filtert die Systemkonsole auf relevante Subsysteme.
03	Crash-Log öffnen	Öffnet den letzten Crash-Report der App.
04	Diagnose-Report erstellen	Erstellt einen vollständigen Simulator-Diagnosebericht.
05	App-Version anzeigen	Zeigt Bundle-Version und Build-Nummer der installierten App.
06	System-Logs streamen	Vollständige Systemkonsole des Simulators.
07	Privacy Manifest validieren	Prüft das PrivacyInfo.xcprivacy auf Vollständigkeit.
08	SwiftData Debug	Aktiviert SQL-Logging für SwiftData.
09	Sysdiagnose erstellen	Erstellt ein vollständiges System-Diagnose-Archiv.

9. Live Activities

#	Befehl	Nutzen beim Testen
01	Live Activity starten	Startet eine Live Activity mit eigenem Payload.
02	Live Activity aktualisieren	Sendet ein Update an eine laufende Live Activity.
03	Live Activity beenden	Beendet eine Live Activity.

10. WidgetKit

#	Befehl	Nutzen beim Testen
01	Widget-Timelines neu laden	Erzwingt ein Timeline-Refresh ohne Wartezeit.

02	Widget-Cache leeren	Leert den internen WidgetKit-Cache.

11. App Clips

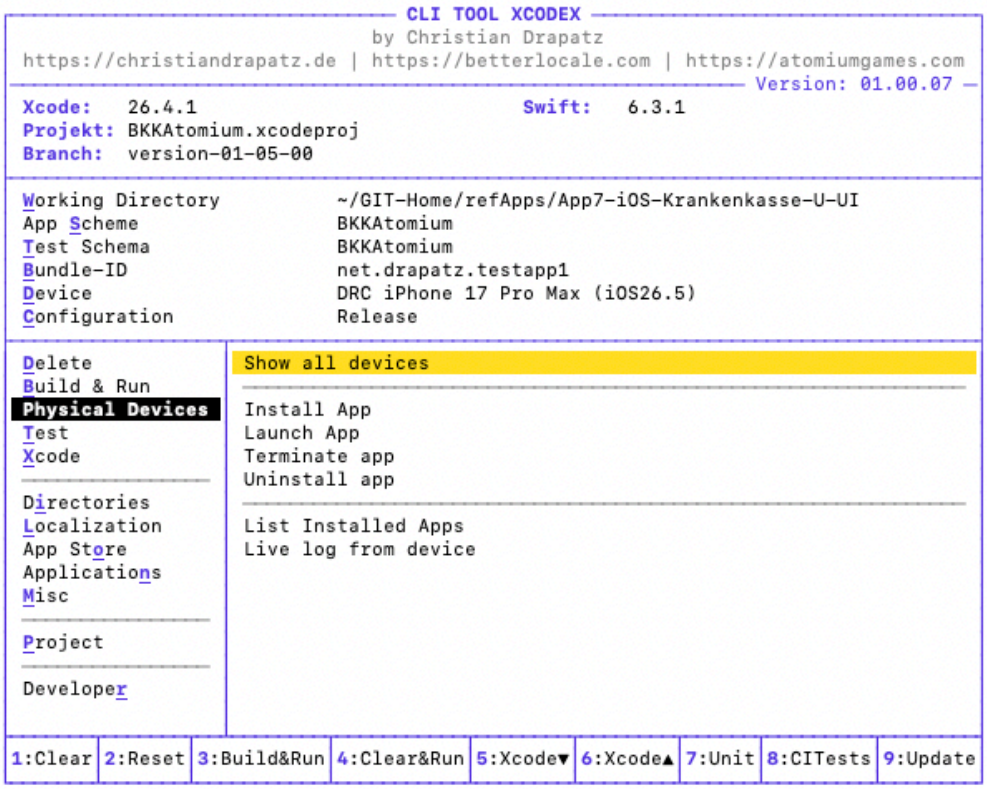
#	Befehl	Nutzen beim Testen
01	App Clip URL testen	Öffnet eine App-Clip-Experience via URL.
02	App-Clip-Experience zurücksetzen	Setzt den gespeicherten App-Clip-Zustand zurück.

> **Tipp:** Statusleiste auf „9:41 Uhr, voll geladen, voller Empfang" setzen, bevor App-Store-Screenshots erstellt werden — genau wie Apple es in offiziellen Präsentationen macht.

> **Hinweis:** Der vollständige Simulator-Reset löscht alle Daten auf dem Simulator — inklusive Keychain-Einträge, Einstellungen und Test-Accounts. Vorher sicherstellen, dass keine wichtigen Test-Daten verloren gehen.

Physische Geräte

Die Gruppe steuert direkt angeschlossene iOS-Geräte über USB oder WLAN. Sie ist nur sichtbar, wenn ein physisches Gerät als Ziel ausgewählt ist.



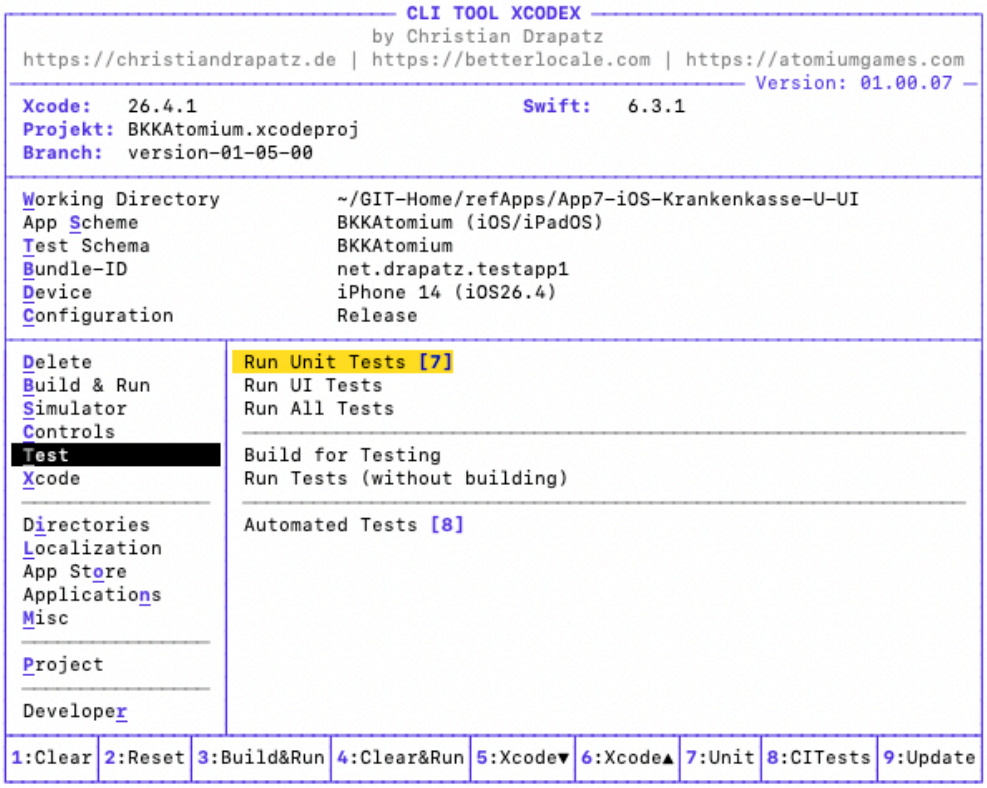
Physische Geräte in xcodex

Vollständige Geräteverwaltung für physische iOS-Geräte direkt im Terminal

Aktion	Beschreibung
Alle Geräte anzeigen	Listet alle verbundenen physischen Geräte mit Name, UDID und iOS-Version auf
`App installieren`	Installiert den zuletzt gebauten App-Bundle auf dem verbundenen Gerät (Gerät muss entsperrt und vertrauenswürdig sein)
`App starten`	Startet die installierte App auf dem verbundenen Gerät
`App beenden`	Beendet den App-Prozess auf dem Gerät, ohne das Gerät manuell zu bedienen
`App deinstallieren`	Entfernt die App vollständig vom Gerät einschließlich aller lokalen Daten
`Installierte Apps auflisten`	Zeigt Bundle-Identifier und Namen aller installierten Apps auf dem Gerät
Live-Log vom Gerät	Streamt Konsolenausgaben des Geräts in Echtzeit — beenden mit Ctrl+C

11. Tests

xcodex trennt Unit-Tests und UI-Tests, damit der jeweils passende Durchlauf gestartet werden kann — schnelle Unit-Tests im Entwicklungsloop, UI-Tests vor Commits oder Releases.



Tests in xcodex

Tests: Unit-Tests, UI-Tests und alle Tests auf einen Klick

Testart	Beschreibung
`Unit-Tests`	Schnell, kein Simulator-Start nötig
`UI-Tests`	Startet die App im Simulator, führt Interaktionstests durch
`Alle Tests`	Kombinierter Durchlauf

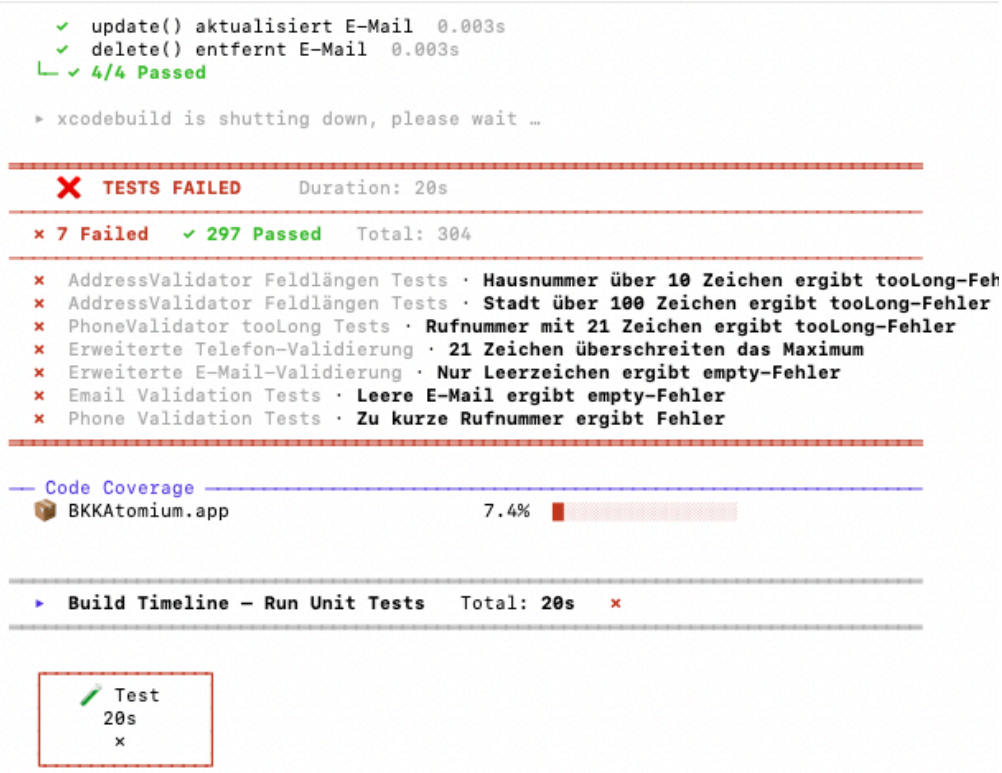
Testergebnisse & Auswertung

- ✅ **Bestanden** – Test erfolgreich ausgeführt
- ⚠️ **Übersprungen** – Test wurde nicht ausgeführt
- ❌ **Fehlgeschlagen** – Test ist fehlgeschlagen

Tests erweitern

Mit der Taste `+` kannst du Tests auf weitere Geräte ausweiten. So lässt sich das Verhalten deiner App parallel auf unterschiedlichen Simulatoren oder Geräten prüfen.

Fehlgeschlagene Tests einzeln wiederholen



Unit-Tests und Wiederholung in xcodex

Fehlgeschlagene Tests werden rot markiert und können einzeln neu gestartet werden

Nach einem Durchlauf listet xcodex alle Ergebnisse auf. Fehlgeschlagene Tests können einzeln neu gestartet werden — ohne die gesamte Test-Suite zu wiederholen. Typischer Workflow:

24. Alle Tests starten
25. Fehlgeschlagene Tests identifizieren
26. Fix implementieren
27. Nur die betroffenen Tests erneut ausführen
28. Bei Erfolg: vollständigen Durchlauf zur Bestätigung starten

Code Coverage

Mit der Code-Coverage-Ansicht siehst du direkt, wie gut dein Projekt durch Tests abgedeckt ist. xcodex kann Testläufe mit aktivierter Coverage starten, Ergebnisse auswerten und dir zeigen, welche Bereiche bereits gut getestet sind und wo noch Lücken bestehen.

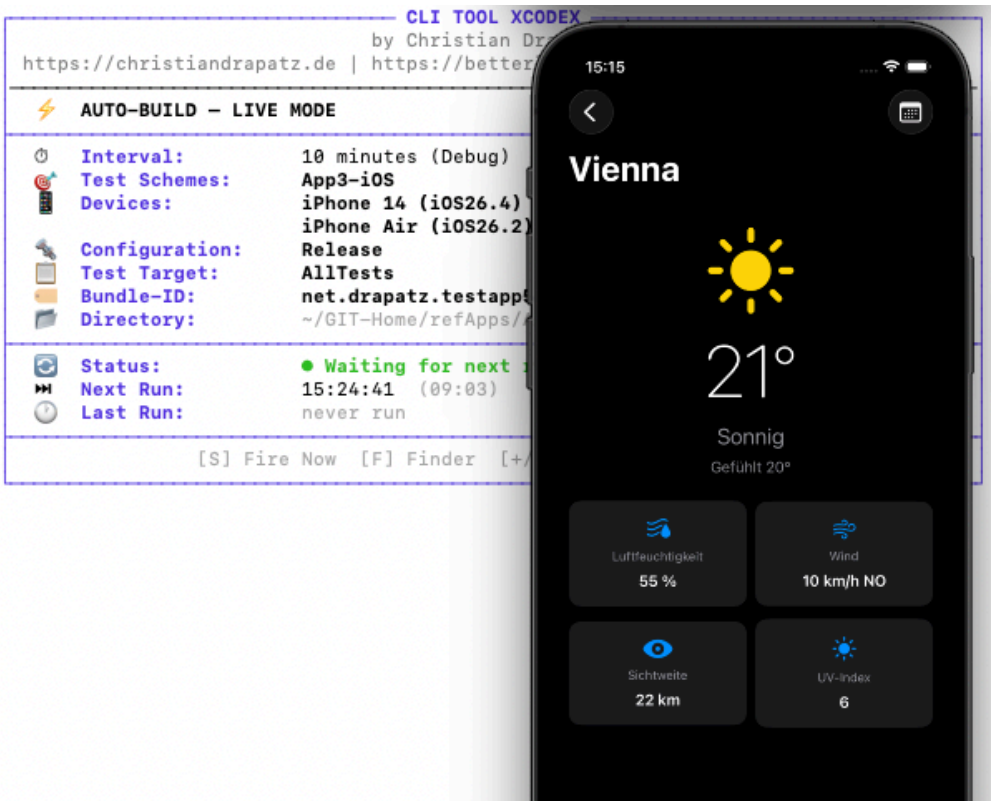
Code Coverage in xcodex

Code Coverage auf Dateiebene: schnell identifizieren, wo Tests fehlen

Abdeckung	Bewertung
`80–100 %`	Gut abgedeckt — kritische Pfade sind getestet
`50–80 %`	Verbesserungspotenzial, besonders bei komplexer Logik
`< 50 %`	Hohes Risiko bei Änderungen — Tests fehlen

Automatisierte Tests

Die Ansicht für automatisierte Tests bündelt wiederkehrende Testläufe in einem klaren Terminal-Workflow. Du kannst Unit-Tests gezielt ausführen und verschiedene Kombinationen aus Schema, Device und Konfiguration testen.



Automatisierte Tests in xcodex

Automatisierte Tests laufen vollständig unabhängig von Xcode

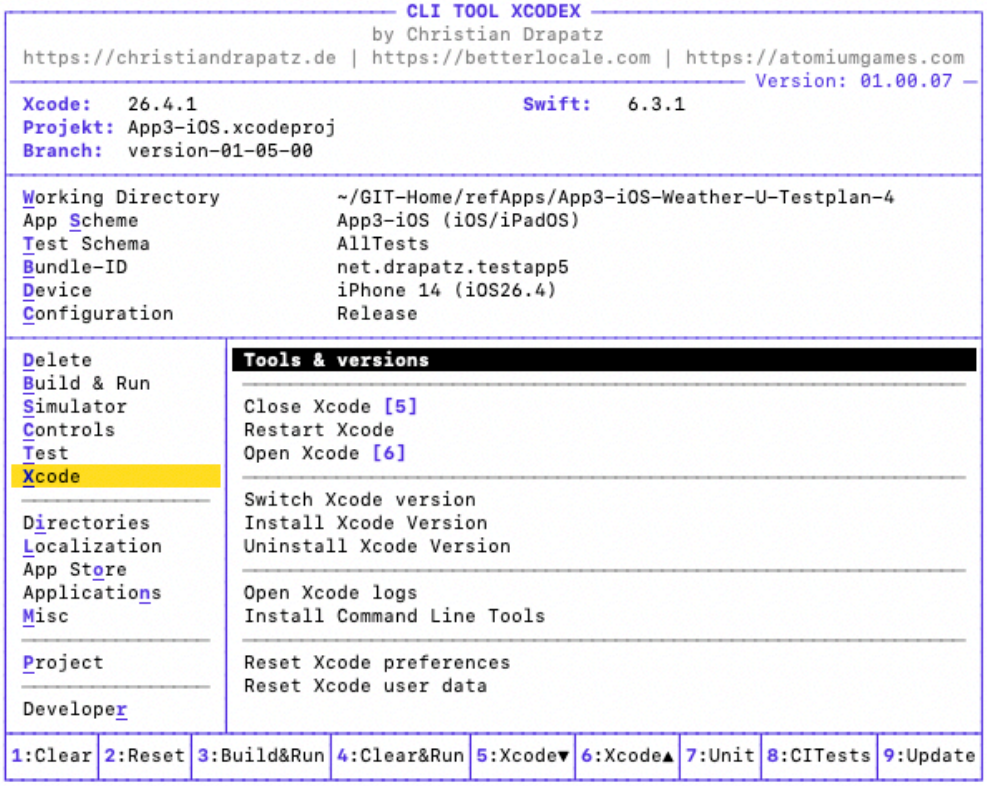
Die automatisierten Tests laufen vollständig unabhängig von Xcode — Xcode kann gleichzeitig geöffnet sein, ohne Auswirkungen auf den Test-Durchlauf. Ideal für:

- **Parallel zur Entwicklung** — Tests für Feature A, während man an Feature B arbeitet
- **QA ohne Xcode-Kenntnisse** — Tester können eigenständig Test-Durchläufe starten
- **Zweiter Clone** — letzten Commit testen, ohne die eigene Entwicklungsumgebung zu berühren

---

12. Xcode-Integration

Diese Gruppe enthält Schnellzugriff-Befehle für Xcode selbst — für den Wechsel zwischen Terminal und IDE.



Xcode-Integration in xcodex

Xcode öffnen und schließen direkt aus xcodex

Aktion	Was passiert	Wann sinnvoll
`Tools & Versionen`	Zeigt installierte Versionen von Xcode, Swift, CocoaPods, SwiftLint u.a.	Schneller Überblick über die Entwicklungsumgebung
`Xcode schließen`	Beendet Xcode	Vor Cache-Aktionen oder einem Xcode-Wechsel
`Xcode neu starten`	Schließt Xcode und öffnet es danach neu	Bei hängendem Xcode oder nach Einstellungsänderungen
Xcode öffnen	Öffnet das aktuelle Projekt in Xcode	Schneller Einstieg ohne Finder
`Xcode-Version wechseln`	Wechselt die aktive Xcode-Version via xcode-select	Wenn mehrere Xcodes installiert sind und gezielt eine genutzt werden soll
`Xcode-Version installieren`	Installiert eine neue Xcode-Version via Xcodes	Neue Xcode-Version parallel zur bestehenden installieren
`Xcode-Version deinstallieren`	Entfernt eine installierte Xcode-Version via Xcodes	Alte Versionen löschen und Speicher freigeben
Xcode-Protokolle öffnen	Öffnet ~/Library/Logs/Xcode im	Bei unerklärlichen Fehlern

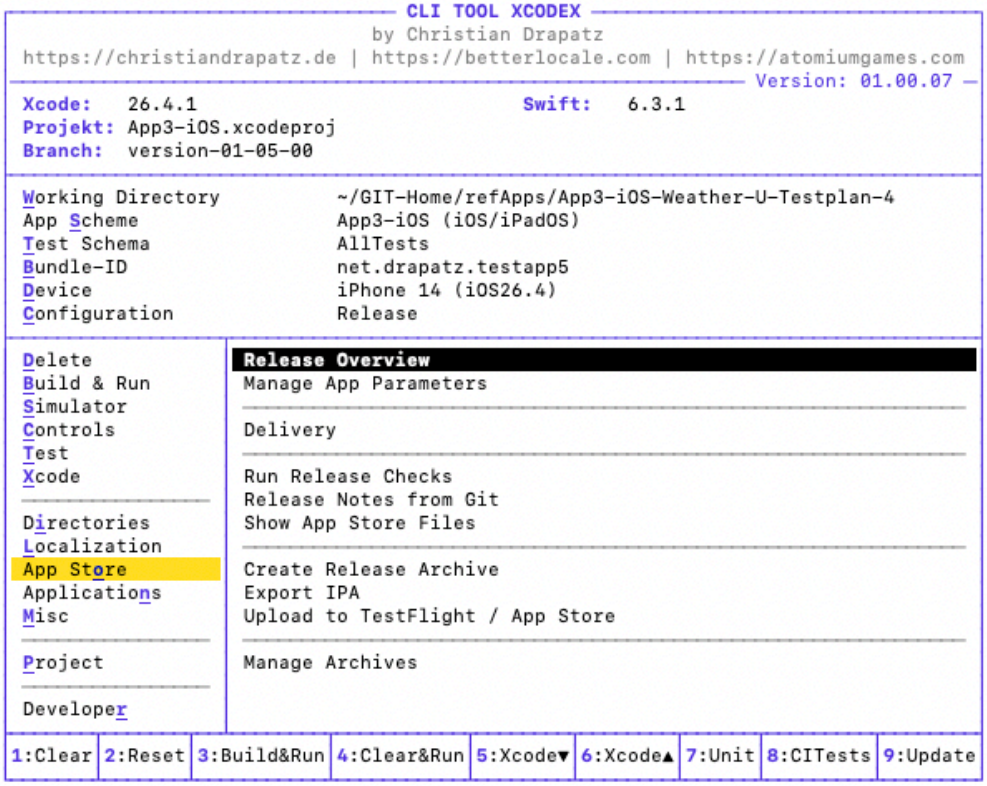
	Finder	oder Abstürzen in Xcode
`Command Line Tools installieren`	Startet den macOS-Systemdialog zur CLT-Installation	Nach einem macOS-Update oder auf einem neuen Mac
Xcode-Einstellungen zurücksetzen	Löscht com.apple.dt.Xcode.plist nach Bestätigung	Bei UI-Hängern oder fehlerhaften Tastaturkürzeln in Xcode
Xcode-Benutzerdaten zurücksetzen	Entfernt ~/Library/Developer/Xcode/UserData/ nach Bestätigung (Keybindings, Themes, Snippets)	Wenn Xcode-Anpassungen Probleme verursachen

> **Tipp:** Vor einem Full Reset & Build zuerst Xcode schließen — das verhindert Dateisperren auf den DerivedData-Ordner.

---

13. App Store & Distribution

xcodex deckt den gesamten Distributionsprozess ab — vom Archivieren bis zum Upload. Das spart die manuelle Arbeit im Xcode Organizer und ermöglicht Uploads direkt aus dem Terminal.



App Store Distribution in xcodex

App Store: Archivieren, Validieren und Hochladen in einem strukturierten Workflow

Voraussetzung: App-spezifisches Passwort

Für den Upload zu TestFlight oder App Store Connect benötigt xcodex ein App-spezifisches Passwort:

29. Öffne appleid.apple.com
30. Gehe zu *Sicherheit → App-spezifische Passwörter*
31. Erstelle ein neues Passwort (z. B. „xcodex Upload")
32. Das Passwort wird einmalig in den xcodex-Einstellungen hinterlegt

Typischer Release-Workflow

App Store Auslieferung in xcodex

Release-Workflow: Validieren vor dem Upload schützt vor häufigen Einreichungsfehlern

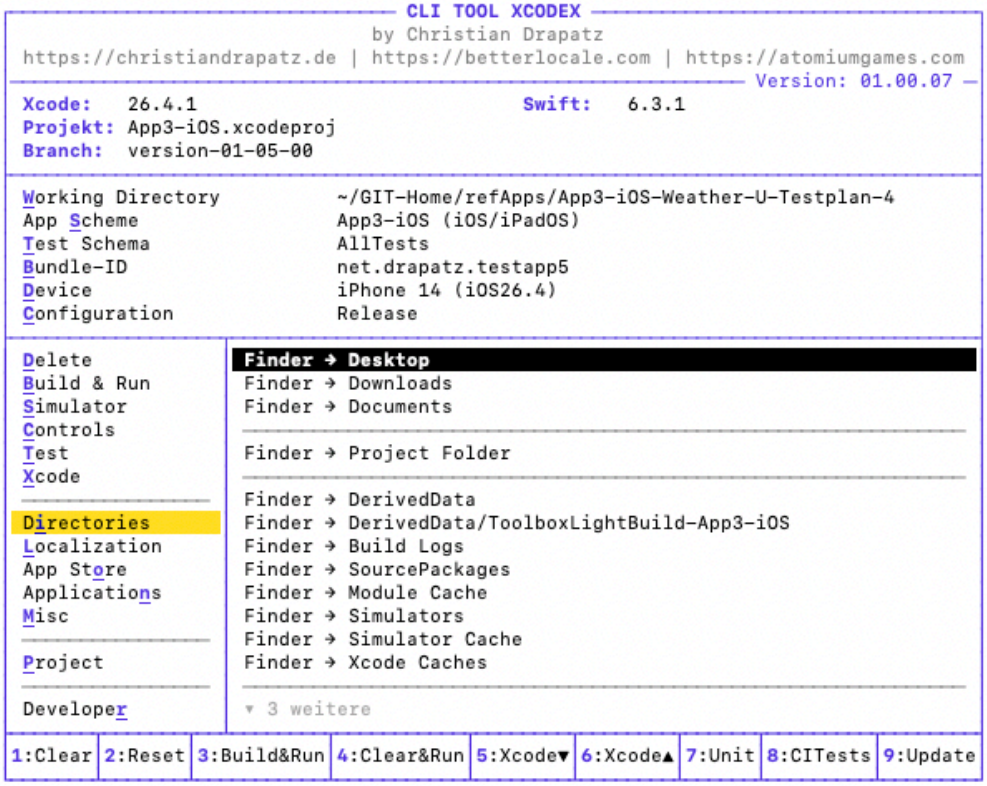
33. Full Reset & Build mit Release-Konfiguration
34. Archivieren
35. Validieren — Fehler korrigieren, falls vorhanden
36. Zu TestFlight hochladen — für Beta-Tests
37. Nach Freigabe: In den App Store einreichen

> **Hinweis:** Für macOS-Apps wird das Archiv automatisch notarisiert und mit dem Notarisierungs-Ticket gestapelt (Notarize & Staple), bevor es hochgeladen wird — ein Schritt, der manuell fehleranfällig ist.

---

14. Verzeichnisse

Xcode verteilt seine Daten über mehrere verschachtelte Library-Pfade. xcodex öffnet die wichtigsten Ordner mit einem Tastendruck direkt im Finder.



Verzeichnisse in xcodex

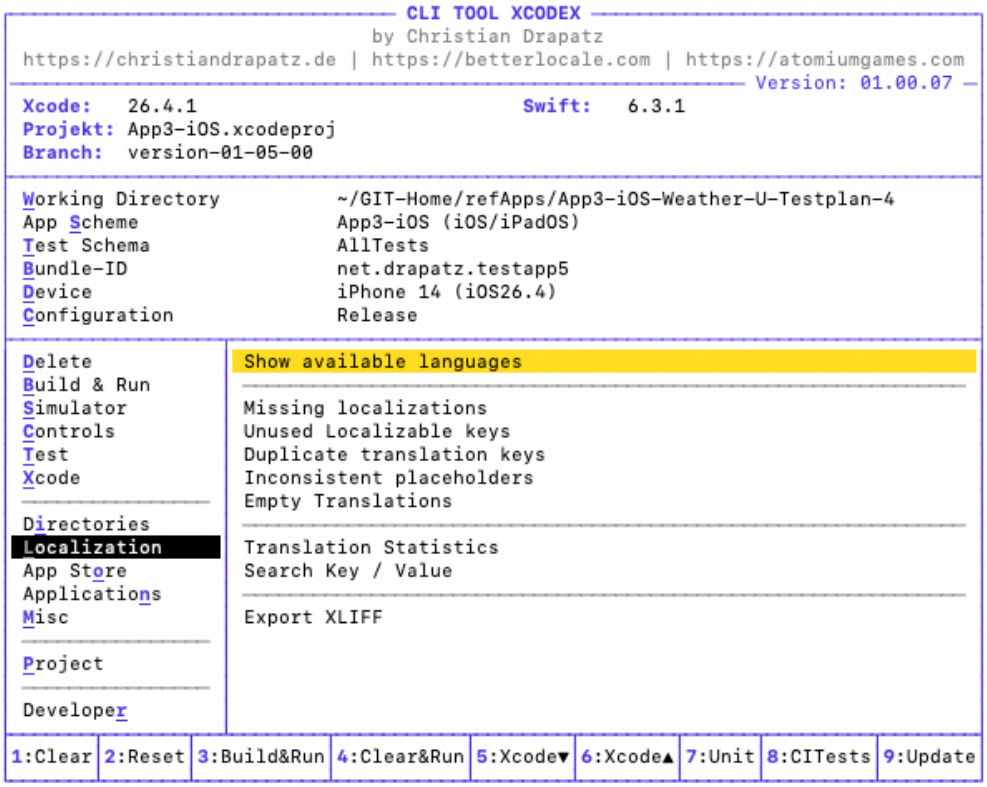
Verzeichnisse: DerivedData, Archive, Simulatoren und Provisioning Profiles auf einen Klick

Aktion	Pfad	Inhalt
Finder → Schreibtisch	`~/Desktop`	Desktop-Dateien
Finder → Downloads	`~/Downloads`	Download-Ordner
Finder → Dokumente	`~/Documents`	Dokumente
Finder → Projektordner	aktuelles Arbeitsverzeichnis	.xcworkspace / .xcodeproj und Projektdateien
Finder → DerivedData	`~/Library/Developer/Xcode/DerivedData`	Gesamtes DerivedData aller Projekte
Finder → DerivedData/\<Projektname\>	`~/Library/Developer/Xcode/DerivedData/<Projektname>-…`	DerivedData des aktuellen Projekts

Finder → Build Logs	…/DerivedData/<Projektname>/Logs/Build	Build-Protokolle des aktuellen Projekts
Finder → SourcePackages	…/DerivedData/<Projektname>/SourcePackages	Geklonte SPM-Pakete des aktuellen Projekts
Finder → Module Cache	~/Library/Developer/Xcode/DerivedData/ModuleCache.noindex	Vorkompilierter Modul-Cache
Finder → Xcode Caches	~/Library/Caches/com.apple.dt.Xcode	Xcode-eigene Cache-Daten
Finder → Simulatoren	~/Library/Developer/CoreSimulator/Devices	Alle Simulator-Geräte und App-Daten
Finder → Simulator Cache	~/Library/Developer/CoreSimulator/Caches	Temporäre Simulator-Caches
Finder → Archives	~/Library/Developer/Xcode/Archives	Xcode-Archive (.xcarchive)
Finder → DiagnosticReports	~/Library/Logs/DiagnosticReports	Crash- und Diagnose-Berichte des Systems
Finder → iOS Device Logs	~/Library/Logs/CrashReporter	Crash-Logs von angeschlossenen Geräten
Finder → Provisioning Profiles	~/Library/MobileDevice/Provisioning Profiles	Installierte Provisioning Profiles

15. Lokalisierung

Fehler in der Lokalisierung fallen oft erst zur Laufzeit auf – dann, wenn die App bereits ausgeliefert ist. xcodex prüft die `.xcstrings`-Datei vollständig statisch, ohne Build-Prozess und ohne Xcode. Fehlende Keys, doppelte Einträge und Inkonsistenzen zwischen Sprachen werden sofort sichtbar gemacht.



Lokalisierung in xcodex

Lokalisierung: fehlende Keys, doppelte Keys und Konsistenz-Prüfung

Aktion	Was passiert	Wann sinnvoll
Verfügbare Sprachen anzeigen	Listet alle erkannten Sprachcodes aus .lproj-Ordnern und .xcstrings-Dateien	Überblick welche Sprachen im Projekt vorhanden sind
`Fehlende Lokalisierungen`	Zeigt Keys, die in mindestens einer Sprache komplett fehlen	Vor einem Release prüfen ob alle Sprachen vollständig sind
`Ungenutzte Localizable-Keys`	Zeigt Keys, die in den Lokalisierungsdateien vorhanden, aber im Code nicht referenziert werden	Aufräumen von veralteten oder vergessenen Keys
Doppelte Übersetzungsschlüssel	Findet Keys, die mehrfach in derselben Datei vorkommen	Bei unerklärlichen Übersetzungsfehlern
`Inkonsistente Platzhalter`	Erkennt Keys bei denen sich die Platzhalter (%@, %d u.a.)	Verhindert Abstürze durch falsche Format-Argumente

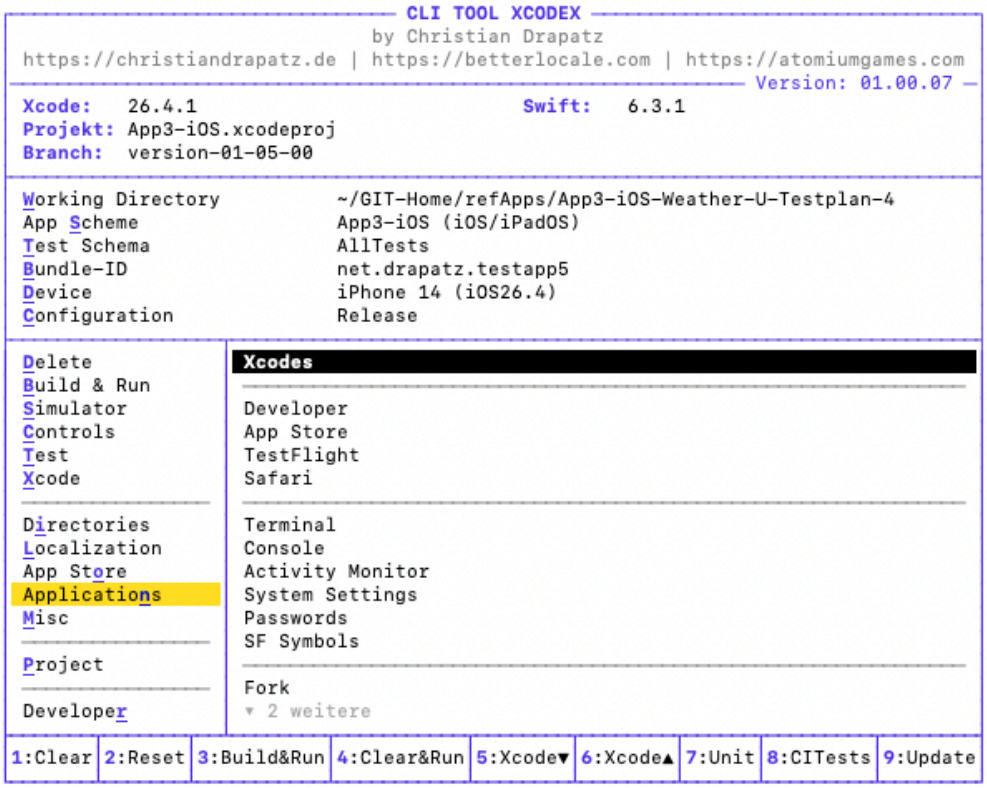
	zwischen Sprachen unterscheiden	
Leere Übersetzungen	Findet Keys mit vorhandenem, aber leerem Eintrag ("" oder nur Leerzeichen)	Ergänzt die Prüfung auf fehlende Keys — leere Werte werden sonst übersehen
Übersetzungsstatistik	Zeigt den Übersetzungsgrad pro Sprache als Fortschrittsbalken aus .xcstrings-Dateien	Schneller Überblick wie vollständig jede Sprache übersetzt ist
Schlüssel / Wert suchen	Interaktive Suche in .xcstrings nach Key-Name oder Übersetzungswert	Einen bestimmten Text oder Key schnell finden
`XLIFF exportieren`	Führt xcodebuild -exportLocalizations aus und legt ein .xcloc-Paket auf dem Desktop ab	Übersetzungspakete für externe Übersetzer erstellen

> **Hinweis:** xcodex prüft die `.xcstrings`-Datei direkt — keine Abhängigkeit von Xcode oder einem Build-Prozess.

---

16. Programme

Die Befehlsgruppe „Programme" ermöglicht es, häufig verwendete macOS-Apps direkt aus der Toolbox heraus zu öffnen — ohne Dock, Spotlight oder Finder. Nur tatsächlich installierte Apps erscheinen in der Liste.



Programme in xcodex

Häufig verwendete macOS-Apps direkt aus der Toolbox öffnen

App	Gruppe	Zweck
`Xcodes`	Apple Entwicklertools	Xcode-Versionen verwalten, installieren und wechseln
`Developer`	Apple Entwicklertools	Apple Developer Portal und Dokumentation
`App Store`	Apple Entwicklertools	App Store öffnen
`TestFlight`	Apple Entwicklertools	Beta-Apps verwalten und testen
`Safari`	Apple Entwicklertools	Webbrowser für Dokumentation und Recherche
`Terminal`	Systemwerkzeuge	Kommandozeile
`Konsole`	Systemwerkzeuge	Systemlogs und Diagnosemeldungen lesen
Aktivitätsanzeige	Systemwerkzeuge	CPU-, Speicher- und Prozessauslastung überwachen
`Systemeinstellungen`	Systemwerkzeuge	macOS-Einstellungen

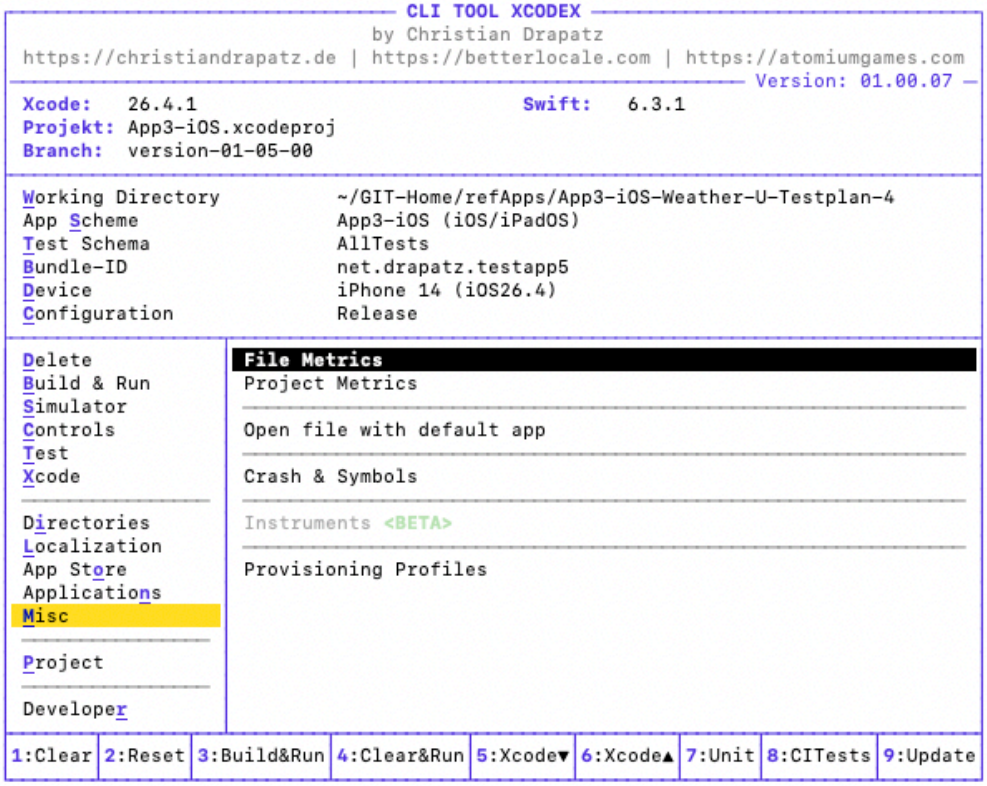
Passwörter	Systemwerkzeuge	macOS-Passwörter und Keychain verwalten
`SF-Symbole`	Systemwerkzeuge	Apple SF Symbols durchsuchen und kopieren
`Fork`	Drittanbieter	Git-Client
`Insomnia`	Drittanbieter	REST- und API-Tests
`Cyberduck`	Drittanbieter	FTP/SFTP/S3-Dateiübertragung
`DevCleaner`	Drittanbieter	Xcode-Caches und Entwicklerdaten bereinigen

> Nicht installierte Apps werden ausgeblendet. Die Liste passt sich automatisch an die vorhandene Software an.

---

17. Verschiedenes

Unter Verschiedenes sind Funktionen gebündelt, die seltener im täglichen Workflow vorkommen, aber im richtigen Moment entscheidend sein können. Dazu gehören die Analyse von Crash Reports, das Starten von Instruments-Sessions und die Verwaltung von Provisioning Profiles.



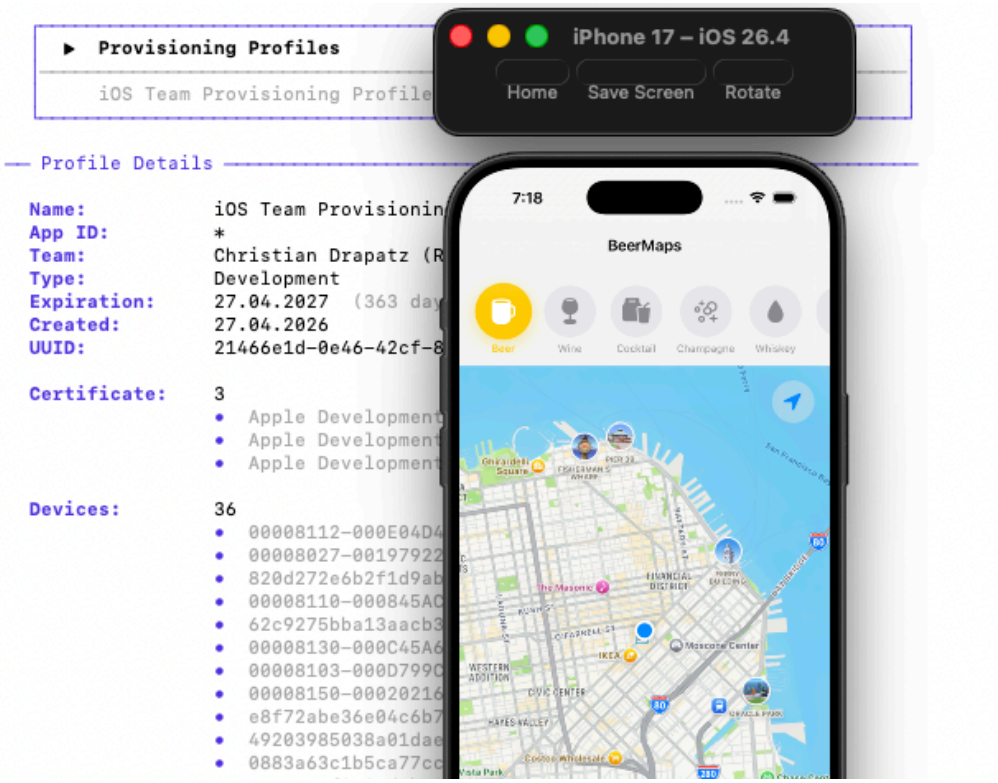
Verschiedenes in xcodex

Crash Reports, Instruments und Provisioning Profiles

- **Crash Reports analysieren** — symbolisiert `.crash`-Dateien und zeigt lesbare Stack Traces. Erfordert das passende `.dSYM`-Archiv
- **Instruments (Beta)** — startet Profiling-Sessions direkt aus xcodex heraus
- **Provisioning Profiles verwalten** — listet alle lokalen Profile auf und erlaubt die Bereinigung abgelaufener Profile

Provisioning Profiles im Überblick

Diese Ansicht zeigt dir alle lokal verfügbaren Provisioning Profiles auf einen Blick. Du erkennst sofort, welche gültig sind, zu welcher App sie gehören und wo es Probleme gibt.



Provisioning Profiles in xcodex

Alle lokalen Provisioning Profiles auf einen Blick — abgelaufene direkt entfernen

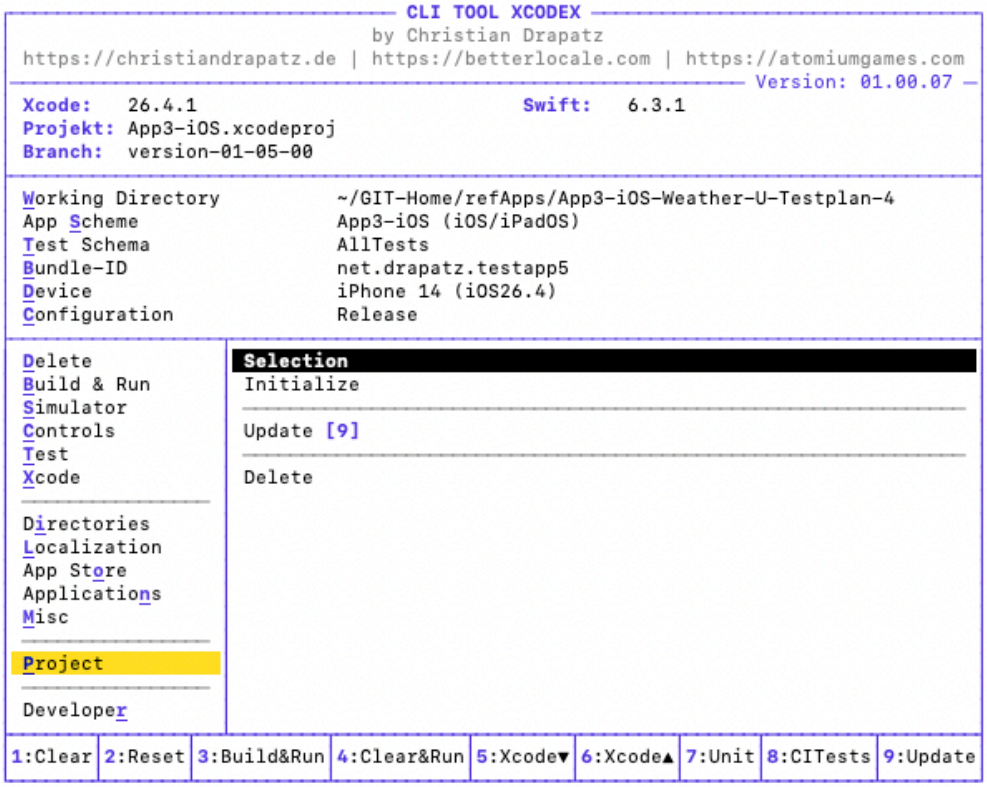
Spalte	Information
`Name`	Lesbarer Profilname
`App ID`	Bundle Identifier, für den das Profil gilt
`Typ`	Development, AdHoc, App Store, Enterprise
`Ablaufdatum`	Wann das Profil ungültig wird
Geräte	Anzahl der zugeordneten Geräte (bei Development/AdHoc)

> **Tipp:** Vor einem Release-Build die Profilliste prüfen — ein abgelaufenes Distribution-Profil blockiert den Upload zu App Store Connect.

18. Entwicklungsumgebungen isolieren

Diese Funktion klont ein Repository in ein separates Verzeichnis auf der Festplatte. Die eigene Entwicklungsumgebung bleibt dabei vollständig unberührt. Man kann so z. B. einen

Feature-Branch parallel auschecken und mit xcodex testen — während Xcode weiterhin am eigenen Branch arbeitet.



Entwicklungsumgebungen in xcodex

Isolierte Entwicklungsumgebungen: QA und Tester können eigenständig bauen und testen

Workflow

Schritt	Beschreibung
1. Auswahl	Beim ersten Start muss eine JSON-Konfigurationsdatei ausgewählt werden (siehe unten). Wurde bereits eine Datei geladen, zeigt xcodex diese an — man bestätigt einfach, ob man sie weiter verwenden möchte, oder wählt eine neue.
2. Verzeichnis bereinigen	Falls das Zielverzeichnis bereits existiert, fragt xcodex, ob es vorher gelöscht werden soll. Das ist sinnvoll, um mit einem sauberen Stand zu beginnen.
3. Repository klonen	Das Repository wird geklont und im konfigurierten Zielverzeichnis abgelegt. Dieser Schritt kann jederzeit erneut über den Befehl **Initialisieren** im Hauptmenü angestoßen werden.

4. Arbeitsverzeichnis festlegen	Nach dem Klonen fragt xcodex, ob das geklonte Verzeichnis als Arbeitsverzeichnis verwendet werden soll. Das entspricht dem Workflow beim Drücken von `A` im Hauptmenü.
5. Aktualisieren	Mit **Aktualisieren** kann ein beliebiger Branch ausgewählt werden. Alle lokalen Änderungen im temporären Verzeichnis werden dabei verworfen — der Branch wird auf den neuesten Commit zurückgesetzt.
6. Löschen	Mit **Löschen** wird das gesamte geklonte Verzeichnis entfernt. Die eigene Entwicklungsumgebung bleibt unberührt.

> **Beispiel-Szenario:** Xcode arbeitet am eigenen Feature-Branch. Parallel dazu klont xcodex den develop-Branch in ein separates Verzeichnis. Über **Arbeitsverzeichnis festlegen** verbindet xcodex diesen Stand als aktives Projekt — Tests, Builds und Analysen laufen nun gegen den develop-Stand, ohne die eigene Arbeit zu beeinflussen.

Typische Anwendungsfälle

- **QA und Tester** — eigenständig den aktuellen develop-Branch bauen und testen, ohne Xcode-Kenntnisse
- **Android-Entwickler** — iOS-App in einem definierten Stand ausprobieren
- **UX-Designer** — Feature-Branch schnell auf dem Simulator starten
- **Code-Reviews** — zweiter Clone zum Testen des Review-Branches, ohne die eigene Umgebung zu berühren

Konfigurationsdatei

Da xcodex nicht wissen kann, welches Repository geklont werden soll und wohin, muss man vorab eine JSON-Konfigurationsdatei erstellen und an einem beliebigen Ort auf der Festplatte ablegen. Diese Datei wird einmalig über **Auswahl** geladen und anschließend gespeichert.

Feld	Bedeutung
`description`	Frei wählbarer Name für diese Konfiguration — wird in xcodex als Bezeichnung angezeigt
`base`	Basisverzeichnis, in das alle Repositories geklont werden (muss mit `/` enden)
`repositories[].name`	Anzeigename des Repositories

`repositories[].url`	Git-URL des Repositories (SSH oder HTTPS)
`repositories[].destination`	Unterordner innerhalb von `base`, in den geklont wird

Das vollständige Verzeichnis des geklonten Repositories ergibt sich aus: `base` + `destination`. Im Beispiel unten also: `/Users/drapatz/Downloads/tempApps/refApps/`

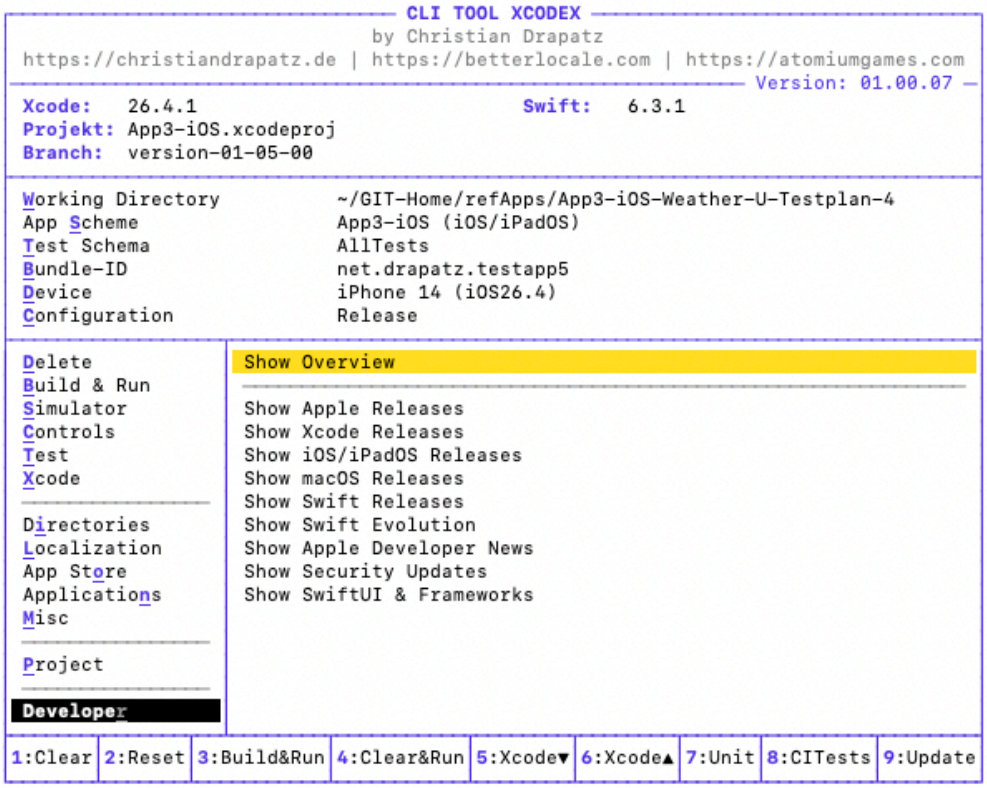
```
{
  "description": "RefenzApps",
  "base": "/Users/drapatz/Downloads/tempApps/",
  "repositories": [
    {
      "name": "RefenzApps-Repository",
      "url": "git@github.com:beispiel/repo.git",
      "destination": "refApps"
    }
  ]
}
```

> **Hinweis:** Die Datei kann einen beliebigen Namen tragen (z. B. `xcodex-env.json`) und an jedem Ort auf der Festplatte liegen. Empfehlenswert ist ein fester Ablageort wie das Home-Verzeichnis oder ein Projektordner.

---

19. Developer-Ressourcen

In der letzten Befehlsgruppe sind die am häufigsten benötigten Dokumentations- und Verwaltungsseiten für Apple-Entwickler gebündelt — von der Apple Developer Documentation über App Store Connect bis zum Swift Package Index.



Developer-Ressourcen in xcodex

Direktzugriff auf Apple-Dokumentation, App Store Connect und Swift Package Index

Ressource	Inhalt
`Apple Developer Documentation`	Offizielle Dokumentation aller Apple-Frameworks und APIs
Swift Documentation & Evolution Proposals	Sprachspezifikation und aktive Sprachvorschläge
`Human Interface Guidelines`	Design-Richtlinien für iOS, iPadOS und macOS
`App Store Connect`	App-Verwaltung, TestFlight und Verkaufsberichte
`Apple Developer Portal`	Zertifikate, Provisioning Profiles und Geräte
`Swift Package Index`	Verzeichnis aller Swift-Pakete

20. Browser-Ansicht

Die Browser-Ansicht ist ein vollständig per Tastatur bedienbarer Datei-Browser direkt in xcodex. Du navigierst durch Verzeichnisse, öffnest Dateien, setzt Favoriten und führst Suchen oder Filter aus — alles ohne Maus und ohne Kontextwechsel.

Browser-Ansicht

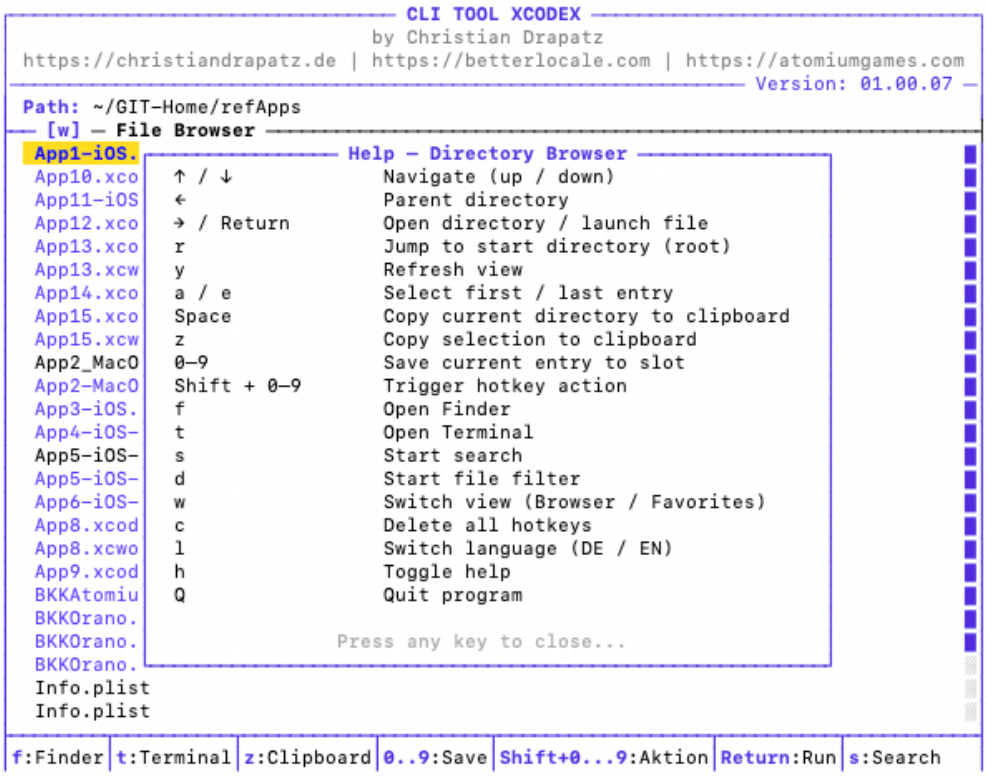
Tastaturgesteuerter Datei-Browser direkt in xcodex

Hotkeys

Taste	Aktion
`Z`	Zum Arbeitsverzeichnis des Projekts wechseln
`0-9`	Favorit aufrufen
`Shift+0-9`	Aktuelles Verzeichnis als Favorit speichern
`F`	Suche starten
`T`	Filter aktivieren
`S`	Auswahl bestätigen / Verzeichnis setzen
`D`	Ausgewähltes Verzeichnis als Arbeitsverzeichnis übernehmen
`W`	In Finder öffnen

Hilfe

In der Browser-Ansicht steht dir eine kontextsensitive Hilfe zur Verfügung. Drücke `H`, um das Hilfe-Overlay einzublenden.

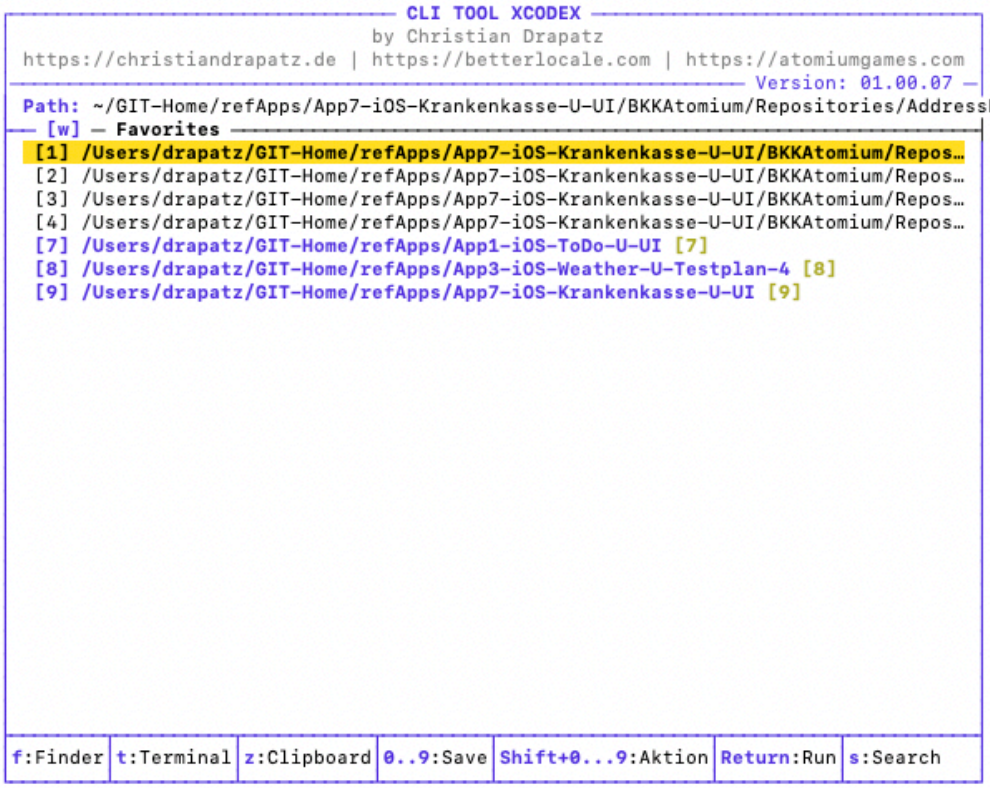


Hilfe in der Browser-Ansicht

Kontextsensitive Hilfe — alle Tastenkürzel auf einen Blick

Favoriten

Für häufig genutzte Verzeichnisse kannst du dir bis zu 10 Schnellzugriffe anlegen. Mit den Tasten 0–9 speicherst du deine Favoriten, mit Shift + 0–9 springst du direkt dorthin.

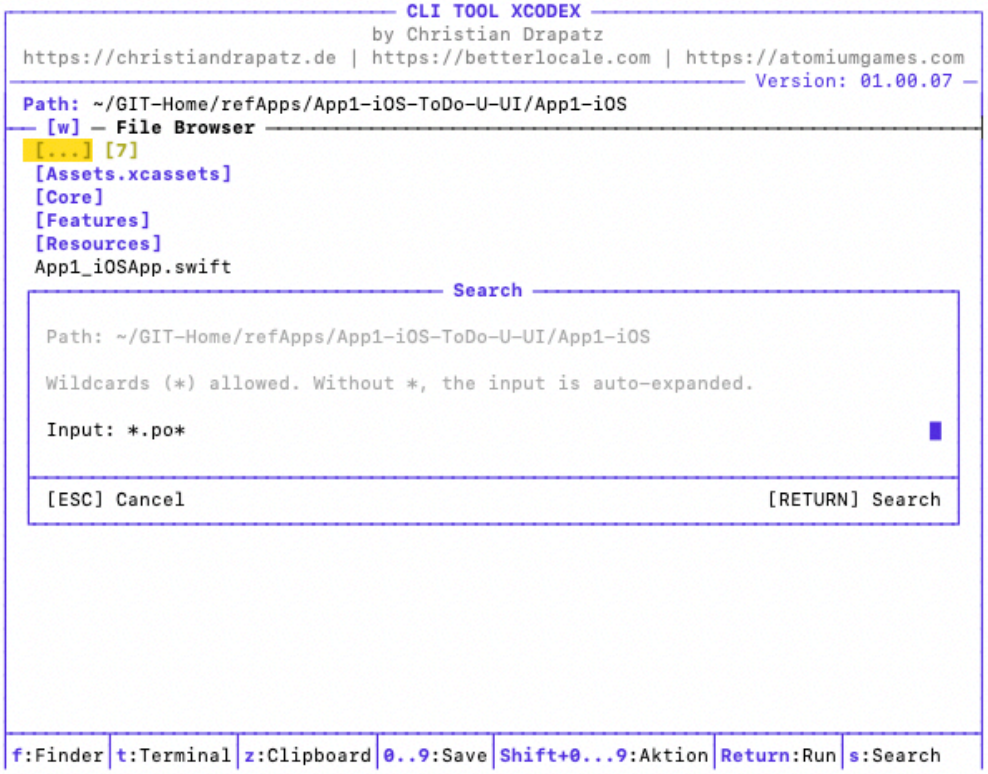


Favoriten

Verzeichnisse als Favoriten speichern und direkt per Ziffer aufrufen

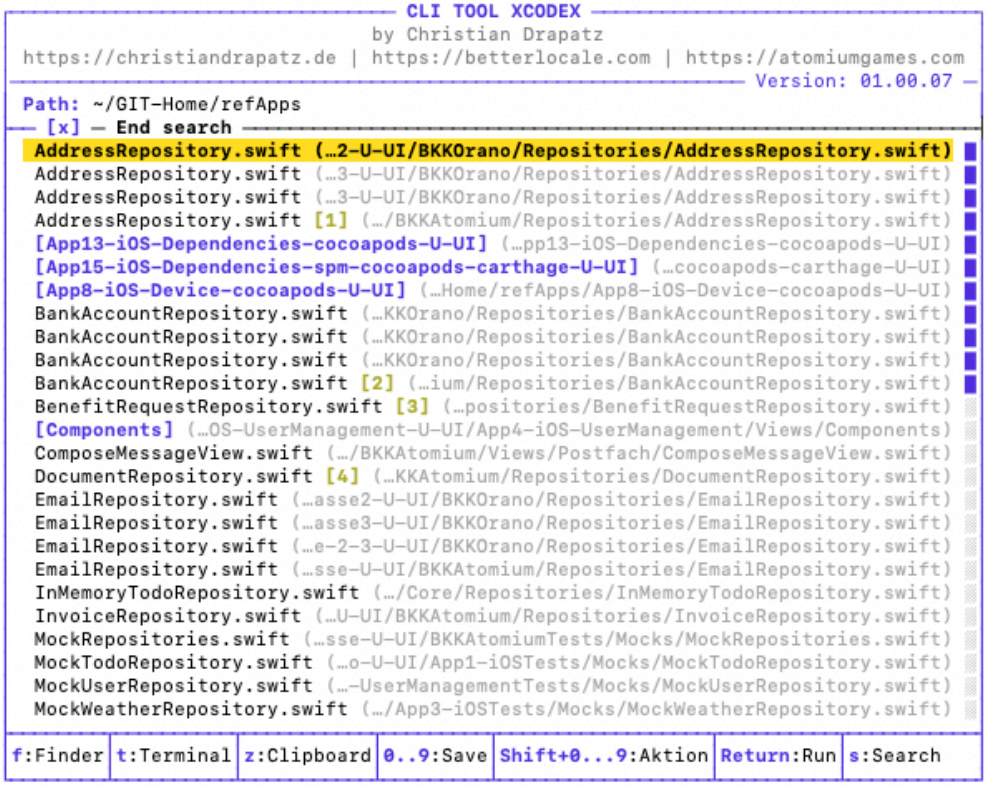
Suche

Die integrierte Suche durchsucht das aktuelle Verzeichnis inklusive aller Unterordner. Wildcards wie `*` werden unterstützt, sodass du flexibel nach Dateinamen filtern kannst.



Sucheingabe

Verzeichnissuche: Suchbegriff eingeben

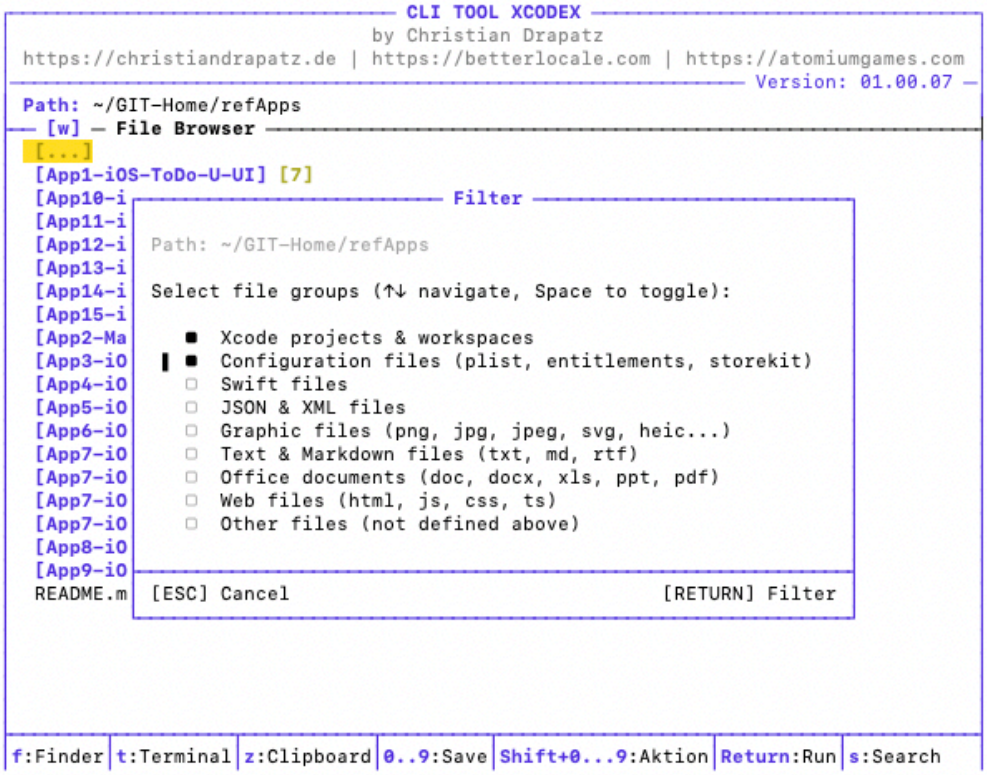


Suchergebnisse

Suchergebnisse: direkte Navigation zu gefundenen Verzeichnissen

Filter

Mit Filtern kannst du gezielt bestimmte Dateitypen anzeigen, zum Beispiel Xcode-Projektdateien oder JSON-Dateien.



Filtereingabe

Filter: Dateiliste nach Begriff einschränken

Gefilterte Ansicht

Gefilterte Verzeichnisansicht — nur passende Einträge werden angezeigt

Schnellzugriff

- **Pfad kopieren (Z)** — Kopiert den absoluten Pfad in die Zwischenablage. Nützlich für externe Tools wie Claude Code.
- **In Finder oder Terminal öffnen (W)** — Öffnet das aktuelle Verzeichnis direkt. Ideal für die Weiterarbeit mit Claude Code.
- **Datei öffnen (Return)** — Öffnet Xcode-Projekte, Office-Dateien oder andere Formate sofort mit dem passenden Programm.

Diese kleinen Funktionen sparen Zeit und unterstützen dich im Alltag, ohne den Flow zu unterbrechen.

---

21. Häufige Probleme

Kein Schema gefunden

Das Arbeitsverzeichnis ist nicht korrekt gesetzt oder die `.xcworkspace`/`.xcodeproj`-Datei liegt nicht im aktuellen Verzeichnis. xcodex aus dem Projekt-Root starten.

Simulator nicht verfügbar

Xcode neu öffnen und die gewünschten Simulator-Plattformen installieren. Anschließend xcodex neu starten — die Geräteliste wird automatisch aktualisiert.

Build schlägt fehl

Fehlermeldung und betroffene Datei aus der roten Ausgabe entnehmen. Bei unerklärlichen Fehlern zuerst DerivedData löschen (*Clean & Cache*) und dann erneut bauen.

Xcode Command Line Tools fehlen

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

Nach der Installation Terminal neu starten.

Push Notification funktioniert nicht

Die App muss Push Notifications im entitlements-File aktiviert haben und korrekt im Apple Developer Portal registriert sein. Im Simulator werden Notifications auch ohne APNs-Zertifikat unterstützt.

Rechteproblem beim Starten

xcodex ist nicht ausführbar. Lösung:

```
chmod +x /pfad/zu/xcodex
```

„missing module"-Fehler nach Branch-Wechsel

Abhängigkeiten nach dem Branch-Wechsel neu auflösen: *Build & Run → Alle Dependencies auflösen*.

---

22. Tipps für den Alltag

Tipp	Erklärung
Alias verwenden	`xcodex` in `~/.zshrc` einrichten, damit der Befehl aus jedem Verzeichnis funktioniert
Aus dem Projektverzeichnis starten	xcodex erkennt die Projektdatei automatisch, wenn es im Root-Verzeichnis des Projekts gestartet wird

Vor Pull Requests	Lokale Builds und Tests ausführen, bevor ein PR erstellt wird — vermeidet fehlschlagende CI-Pipelines
Isoliertes DerivedData	xcodex baut in ein isoliertes Verzeichnis, sodass parallele Xcode-Builds nicht beeinflusst werden
Nach `git pull`	Zuerst Dependencies auflösen (Sektion Abhängigkeiten), dann bauen
Vor einem Release-Build	Provisioning Profiles prüfen und Simulator-Cache leeren
Regelmäßig aktualisieren	`git pull` im xcodex-Verzeichnis hält das Tool auf dem neuesten Stand

23. Haftungsausschluss

xcodex ist ein unabhängiges Open-Source-Werkzeug und steht in keiner Verbindung zu Apple Inc. „Xcode" und „TestFlight" sind eingetragene Marken von Apple Inc. Die Nutzung des Tools erfolgt auf eigene Verantwortung. Für Schäden, die aus der Nutzung entstehen, wird keine Haftung übernommen.