

Rebound Retro Studio – Rebound Retro Studio – User's Manual

Studio adds generated tool tables to chapter 3 and keyboard shortcut tables to chapter 7 inside the app. Those tables are not in the repository Markdown and are not included in this web/PDF edition.

1. What Studio is

Rebound Retro Studio is a workbench for writing programs for old computers on a modern Mac: today the **Commodore 64**, the **MEGA65** and the **Amiga**. You write the code here, press one key, and the program runs — in an emulator on this Mac, or on the real machine sitting next to you.

You do not need to know anything about "cross-compiling" or "toolchains" to start. Those words describe what happens behind the button; this manual explains them exactly when you need them and not before.

Quick start

1. Open the downloaded DMG. Drag **Rebound Retro Studio.app** to **Applications**, then open the app from there.
2. On a fresh setup, first launch opens the setup wizard: **Welcome to Rebound Retro Studio**. Check the projects folder, tick a machine, then press **Set up**.
Already configured? Use **File ► Set Up Machines...**
3. In **Languages**, use the download buttons for one language and its dependencies. In **Emulators**, download an emulator and complete any ROM configuration and mounted-folder instructions. **Real hardware** is optional.
4. In **Test your first program**, use **Open ...** in the editor to open an editable sample. Or choose a matching sample under **Sample projects** on the start page.
5. With the sample's code tab active, **⌘B** builds and **⌘R** runs it in the configured

Next, chapter 2 explains setup; chapter 4 covers projects and targets; chapter 5 covers build/run/deploy; chapter 6 covers editing. For problems, see chapter 9.

The four things on screen

- **The editor** in the middle — where you type. It comes with two keyboard styles (chapter 6) and knows the assembly languages of all three machines, plus Commodore BASIC and C.
- **The sidebar** on the left, with three panels: **Explorer** (your project's files and attached folders), **Docs** (reference pages, including this one) and **Outline** (the headings or supported labels of the file you are reading).
- **The terminal** below the editor — a real shell. With a project open, new sessions start there; existing sessions keep their directory across project switches.
- **The dock** at the very bottom: **Build**, where errors appear, **Source Control**, a small git panel, and **Search**, for a term across the whole project and the headers it includes.

The one idea to hold on to

A project has one or more **targets**. A target says which machine the program is for, which tool turns the source into a program, and where it runs — an emulator or the real hardware. Everything else in Studio hangs off that: the Docs panel shows the active target's machine, **Build** uses its tool, **Run** uses its destination.

What Studio is not

It is not an emulator or an assembler. It drives those — VICE, KickAssembler, Xemu, and the others named in chapter 3 — and tells you when one is missing. With a project open, **Window ► File Manager** opens a **File Manager** tab, which also transfers files. If a sentence in a banner starts with a lowercase letter and names a key, it is Studio telling you the next thing to press.

2. First run

Studio opens its **setup wizard** when no shared tools, no shared machines and no recent projects exist. Its first page, **Welcome to Rebound Retro Studio**, shows **your projects folder** — where new projects and sample copies go — with **Change...** to pick another. The same setting lives in **Settings ► General**. On that same page, tick **Commodore 64**, **MEGA65**, **Amiga**, or several. Nothing is selected for you, and **Set up** stays inactive until you tick a machine. The choices name the languages you can write and where the program can run — an emulator on this Mac, and a C64 Ultimate, MEGA65 or Apollo Vampire V4.

Set up walks your selected machines in order. Later (**Esc**) skips the welcome.

File ► Set Up Machines... brings this list back whenever you need it, with configured machines marked. Skipping without configuring anything or opening a project means the welcome can appear again on the next launch.

The start page

With no project open you see the start page: **Recent projects**, including **+ New project**, **Sample projects**, and **Getting started**.

The wizard

Each machine has a short journey:

1. **Languages** — start with one language. Download its tools or choose tools already installed. Java appears underneath **KickAssembler** because it runs the assembler; you do not need to learn Java. On the C64 **petcat** for BASIC is part of VICE, so that one download also provides the emulator assembly programs run in.
2. **Emulators** — choose an emulator. The Amiga offers **Amiberry** and **FS-UAE** as alternatives, not two requirements. **FS-UAE Launcher** is not a third choice; it is FS-UAE's own front end — the window that writes the machine configuration and imports Kickstart ROMs — and it is offered underneath FS-UAE, as its own download, installed beside it. For Amiga runs, choose **Amiberry config (.uae)** or **FS-UAE config (.fs-uae)** and the **Deploy** directory (DEV · laptop volume). The wizard explains additional ROM requirements.

3. **Real hardware** — optional, on every machine. Local builds and emulator use do not depend on completing it. The MEGA65 finds its machine by itself (below), once it is on the same network and you have pressed **SHIFT + E** on it to switch on its network access; a MEGA65 that was just switched on is not listening yet. The C64 Ultimate and the Apollo each take a host name — see A C64 Ultimate or an Apollo below.
4. **Test your first program** — which language (when more than one is set up), then two numbered steps: an optional check of the tools, and opening your own editable copy.

The step names are navigation: you can move backwards or skip ahead. **Close** (**Esc**) leaves the current machine; if others are queued, the button names the next one instead. Escape still works while the manual address field has focus.

What the marks mean

These are the tool-row marks. Navigation marks also account for saved hardware settings, required Amiga emulator configuration and a successful sample proof.

Mark	Meaning
Empty circle	Not set up yet — normal on a new Mac
Checked circle	Installed/available; the detail line explains what was checked
Muted attention mark	An installed tool is waiting for a dependency, or macOS blocked it (see below)
Red attention mark	A real failure: the selected file is wrong or could not run

A missing tool is **not an error**. Installing a second emulator or an optional hardware utility is not necessary to make the first one useful. “Emulator installed” does not mean its ROMs or configuration have been tested; “hardware transport installed” does not mean a real machine answered.

Studio searches standard install locations and your `PATH` and checks tools where `emulators` lists them. VICE is identified through the `petcat` belonging to the `emulators` without opening an emulator just to ask its version. Different

versions are reported, not automatically rejected, except that Studio requires Java 21 or newer and does not suggest an unreadable Java version.

Keeping what Studio finds

Moving between steps or closing the wizard saves discovered tool paths into **empty settings only**. A path you chose by hand is never overwritten. **Choose installed tool...** is shown on every tool, but is disabled while a tool installation runs. **Search again** picks up software installed outside Studio, including newly installed Apple developer tools.

On the **Test your first program** step, **Check the tools** (step 1, optional) builds a disposable copy of the sample with your saved tools and discards it. It reports the actual result; an installed checkmark alone is not a build test. **Open ... in the editor** (step 2) copies the sample into your **projects folder** — the step names the exact path — and opens it. In the editor, **⌘B** builds your copy and **⌘R** runs it. The **?** beside **2. Open it and start writing** names the emulator, or says that one still needs installing. A chosen, installed emulator is selected where the sample provides a matching target.

Copy ... to your projects folder, beside it, only copies: the page shows “Copied to ” and the setup stays open, so the next queued machine is still one press away. Pressing it again says “Already in your projects folder: ” and copies nothing, and **Open** afterwards opens that same copy instead of making a second one.

Opening a sample closes the setup journey. Any machines still waiting are named before the action and in the status bar afterwards; use **File ► Set Up Machines...** to continue with them.

Downloading a tool

Studio downloads a pinned archive and verifies its SHA-256 digest. VICE, Amiberry, FS-UAE, its Launcher and M65Connect go under **~/Applications/<bundle>-<version>/** other tools, including Xemu, go under **~/.local/share/rebound-retro-studio/tools/**. It does not replace a system installation. Progress distinguishes downloading, verification, unpacking and, for source packages, compilation. Once a tool is installed this way its download button **DEV · laptop** on while the tool still waits for a dependency (KickAssembler for

Java); buttons that cannot be pressed are drawn dimmed. The examples below name embedded download versions. A verified hosted manifest can update them; the button shows the version currently offered.

- **C64 BASIC and VICE:** one VICE download provides both `petcat` and the emulator.
- **C64 assembly:** KickAssembler and a Temurin 21 JDK have separate downloads. An installed JDK of **Java 21 or newer** can be used instead; an older Java is shown as older than the version Studio is tested with, and the Temurin 21 download is offered. macOS's Java stub is not a JDK. If an old Java path was set by hand, clear it or choose the newer Java; installing Temurin alone does not override that path.
- **MEGA65 assembly:** Download & build ACME 0.97 compiles pinned source for this Mac. It requires Apple's developer tools first; Studio explains that prerequisite and does not launch the developer-tools installer itself.
- **Amiga emulators:** Amiberry and FS-UAE downloads retain their complete app bundles, and they are installed into `~/Applications` so you can open them from Finder like any other program. They include AROS replacement ROMs, not original Kickstart/AmigaOS files. FS-UAE uses a `.fs-uae` text configuration.
- **FS-UAE Launcher** is offered under **Optional**: A window to set the Amiga up; FS-UAE runs a configuration, while the Launcher is the window that writes one, imports your Kickstart ROMs and starts the emulator. Studio installs it **beside FS-UAE in the same folder**, which is exactly how the Launcher finds the emulator — keep them together if you move them.
- **Amiga assembly and C:** Download & build `vasm 2.0f` compiles the assembler for this Mac; Download & build the **Amiga C toolchain** (`vbcc`, `vasm`, `vlink`) 0.9h fetches five vendor archives — the `vbcc` compiler, `vasm`, `vlink`, the AmigaOS C library with its headers, and the configuration a Unix host needs — and assembles them into one directory. Both need Apple's developer tools; install refuses before downloading if those tools are absent.
- **Amiga NDK:** the row under the C toolchain fetches Hyperion's NDK 3.2 R4 from Aminet: `NDK3.2.1ha` (digest-pinned; Studio reads `.1ha` itself) and uses `Include_H` for `#include <proto/exec.h>` and friends. The headers are Hyperion's licensed package; Studio fetches them for you and never redistributes them. A copy you unpacked yourself can be chosen with `FilePicker`.

- **MEGA65 transport tools:** one M65Connect download (digest-pinned, from the MEGA65 file host) provides both `etherload` and `mega65_ftp`. It installs the whole `M65Connect.app` into `~/Applications/m65connect-2.5/`. A copy you installed yourself is still found by discovery or can be chosen with the file picker. A copy you downloaded with a web browser may be blocked by macOS; its row then says macOS blocked this tool because it was downloaded in a web browser and offers **Allow M65Connect's tools**, which removes the browser's download flag from that app when you press it — never by itself (chapter 9).

ROMs and operating systems you bring yourself

Studio's Amiga emulator downloads include free AROS replacement ROMs. Studio does not include original Kickstart, AmigaOS or MEGA65 ROMs. Those are yours to buy, and they are cheap and legal to get:

- **AmigaOS 3.2 and its Kickstart ROMs:** AmigaOS 3.2 is sold by [Hyperion Entertainment](#), as a downloadable CD-ROM image or as a boxed set from their authorized dealers. The CD holds the Kickstart ROM images for every classic Amiga model plus the install disks, which is everything an emulator needs. If you also want a physical ROM chip for real hardware, Hyperion's dealers sell one against proof of your AmigaOS 3.2 purchase.
- **Older Kickstarts (1.3, 2.x, 3.1):** Cloanto's [Amiga Forever](#) is the licensed source. Its encrypted ROMs need the `rom.key` file that comes with them, so keep it in the same folder.
- **Without buying anything:** Amiberry and FS-UAE include AROS replacement ROMs. They boot many programs, but they are not AmigaOS, so expect differences.
- **C64:** nothing to buy. VICE includes its ROMs.

MEGA65 ROMs and cores. The MEGA65 ROM contains original Commodore 65 code, so it is licensed to MEGA65 owners only and is not a public download. Here is how to get it:

1. Create an account on the [MEGA65 Filehost](#). That is the official home of cores, ROMs, tools and software.

2. In the user menu, choose **Redeem Code** and enter the nine-character code printed on the slip of paper that came with your MEGA65 (it looks like `123-ABC-456`). This makes you a registered owner.
3. Download the **MEGA65 Core Release Package**. The owner version contains the ROM, and the core `.cor` is for flashing the real machine.
4. For Xemu, the ROM file is usually named `911001.bin`. Rename it to `MEGA65.ROM` and put it in Xemu's data folder `~/Library/Application Support/xemu-lgb/mega65/`. Then, in the running emulator, right-click and choose **Disks > SD-card > Update files on SD image**. When Xemu offers to use the previously installed ROM, answer no.

Without a MEGA65, Xemu can run the free, open-source [Open ROMs](#). Their own authors call them work in progress, so they are fine for curiosity but not a real replacement. Step-by-step with screenshots: [Dan Sanderson's welcome guide](#) and [Xemu's ROM tutorial](#).

Finding a real MEGA65

Automatic discovery is the default when deploying on the same network. **Look for my MEGA65** listens for announcements for three seconds; it does not upload, reset or start a program. Silence is inconclusive, not an error, and an announcement does not prove that a subsequent transfer will succeed.

Use this address explicitly saves a heard address. For **Set the address by hand...**, use the vendor tools' scoped IPv6 form, such as `fe80::1234%en0` — not a DNS name or IPv4 address. Studio saves the text without validating that syntax. Clear it and press **Save** to restore automatic discovery. This address is separate from the local transport executables.

A C64 Ultimate or an Apollo

Turn the machine on and connect it to the same Wi-Fi or network as this Mac. For the Apollo, **ApolloExplorer** must be running on the Amiga; Studio sends your program through it when you deploy.

Type the machine's host into the field and press **Save** (or **Enter**). This edits the default named machine of that kind in **Settings ▶ Machine**. Creating one if it can select a different named machine. For the C64 Ultimate use its

network name, such as `C64-Ultimate-Founders.local`. The name survives the router handing out a new address.

Test connection saves what you typed, then checks the machine. Nothing is uploaded or started.

- **C64 Ultimate:** asks the Ultimate API for its version `GET /v1/version`
- **Apollo:** sends three pings and accepts any reply. This shows the machine is on the network, not that ApolloExplorer is running.

Result	Meaning
answered (green)	Something answered at that host. Only a deploy shows whether a program runs.
No answer	Off, on another network, a name that does not resolve — or Studio is not allowed on the local network. Not an error.
answered, but said no (red)	Something is there and refused; on the Ultimate, usually Web Remote Control Service switched off, or a wrong network password.
No host saved	There is nothing to test yet.
The test itself could not run (red)	Neither a yes nor a no.

The step's mark is checked once a host is saved; whether the machine answers today is the test's sentence, not the mark.

The Apollo's deploy drawer. Below the host, the Apollo step has a **Deploy** drawer where Studio puts your program and `deploy.txt` to where `deployd` must watch. Empty means `Personal:Sources` is the Apollo's own drawer — change it only if your Apollo is set up differently (for example, `Personal:deploy/`). It is the drawer of the default Apollo in **Settings ► Machine** in the field showing `Personal:Sources`. A target can choose another machine or override its drawer in **Settings ► Target**.

DEV · laptop

optional and needs ApolloExplorer running. It first saves the drawer locally, then asks the Apollo for its volumes and whether the

drawer exists. Nothing is written on the Apollo. A missing-volume result lists available volumes; a missing-drawer result names the drawer. It can also report that ApolloExplorer is not listening.

The sample projects

Six small programs, one per machine and language, each teaching one thing:

- **c64-basic-hello** — C64 BASIC. Shows how readable BASIC text becomes a program the C64 can load (that is what `petcat` does).
- **c64-asm-hello** — C64 assembly. Shows why a machine-code program starts with a tiny BASIC line `BasicUpstart2` so that `RUN` can find it.
- **mega65-asm-hello** — MEGA65 assembly. The same idea done by hand at `$2001` with a guard that keeps it honest.
- **mega65-basic-hello** — MEGA65 BASIC 65. `petcat -w65` makes `$2001` PIG.
- **amiga-asm-hello** — Amiga assembly. A program with no fixed load address ("hunks"), and console output through operating-system library calls.
- **amiga-c-hello** — Amiga C. `vbcc` links the startup code and C library into an executable you run from the Amiga Shell.

Clicking one on the start page copies it into your projects folder, or reopens the existing sample there when its manifest name matches. The wizard uses the same rule. Conflicting paths get `-2` `-3` ; nothing is overwritten.

The projects folder

New projects and sample copies are created in one folder, named in **Settings** ► **General** ► **Projects folder**. It starts at `~/Documents/Rebound Retro Studio/`. **Choose...** points it somewhere else. **Reveal** opens it, and **Default** appears when a custom folder is set. Press **Save** to keep changes made in Settings. The setup wizard's last step names the same folder and has **Change projects folder...** so your first sample copy can already land where your projects live. A folder chosen there is saved right away, into the same setting. Projects that live elsewhere keep working exactly as before. This setting only decides where Studio puts new ones, and it is also the folder the New Project dialog opens with.

Running the wizard again

`File ► Set Up Machines...` reopens the machine list, already set up means the configured tool paths or managed installations exist, not that a build has been proved. A marked machine can still be ticked and walked again — that is how you add an emulator or a real machine later. `File ► Set Up Commodore 64...` `Set Up MEGA65...` and `Set Up Amiga...` open one machine's journey directly. Nothing either does depends on being the first run. Studio offers the setup row after build/preflight failures containing `not configured` or `does not point at a file`: the banner reads "a tool is missing — `File ► Set Up Commodore 64...`". Other failures may not offer it.

3. Setting up the tools

This is the chapter to read when a build says a tool is not configured, or when the wizard could not find something. It answers three questions: which tools does each machine need, where do the vendors hide them, and which Settings page owns which field. The tables at the end are generated from the same data the wizard uses, so they cannot disagree with it.

Rows in Settings and in the setup wizard keep their explanations behind a  beside the label. Hover it to read; click it to keep it open while you work. Click the  again or outside the help card to close it.

Three kinds of tool

Every machine uses tools for different jobs. One language is enough to start:

- A **build tool** turns your source text into a program the old machine can load. Assemblers (KickAssembler, ACME, vasm) do this for assembly; `petcat` does it for Commodore BASIC, and `vbcc` compiles Amiga C. Without a build tool, nothing can be built -- the only genuinely blocking case.
- A **run tool** is an emulator: VICE for the C64, Xemu for the MEGA65, Amiberry or FS-UAE for the Amiga. Without it, builds still work; they just have nowhere to run on this Mac.
- A **hardware tool** puts the program on the real machine -- `etherload` and `mega65_ftp` for a MEGA65 over Ethernet. Most people evaluating Studio do not own the hardware, so this is never blocking.

Two real-hardware paths need no local command-line tool: the C64 Ultimate and the Apollo Amiga. Add named machines in `Settings ▶ Machine`, enter **Machine name** and **Address** (MEGA65: optional), then **Save row**. **Make default** selects the fallback; each hardware target's **Machine** picker can select a named machine instead. For Ultimate passwords, use **New password** and **Set**. ApolloExplorer must run on the Amiga, with `deployd` watching the deploy drawer (normally `Personal:Sources`).

Where the vendors hide them

- KickAssembler is a Java `.jar` (not an app). It comes as a zip from `theweb.dk` (the file you point Studio at is `KickAss.jar`). It needs a Java runtime on your Mac; Studio requires Java 21 or newer.
- VICE ships as a disk image. Inside it, `bin/` holds both `x64sc` (the C64 emulator) and `petcat` (the BASIC tool). Studio wants each one's path.
- ACME has a **Download & build** route using pinned 0.97 source and Apple's installed developer tools. Its old official Mac binary is Intel-only; `vasm` has the same route — its vendor also ships no macOS binary — and 2.0f is built from source here. Both end up as a single executable; the wizard also searches standard local binary locations.
- `vbcc` needs more than the `vc` driver; the `vbccm68k` compiler, `vasm`, `vlink`, and the **AmigaOS 68k target package** (configuration, C headers and runtime libraries). **Download & build the Amiga C toolchain** collects all of it — five archives from the vendor's own server, each verified against a pinned SHA-256 — and assembles a single directory with `bin/`, `config/` and `targets/` (which is also the layout described below if you install it yourself). Nothing is bundled with Studio, and no prebuilt macOS compiler exists to download; the sources come from the official `vbcc` site. In **Settings ▸ Machine**, point `vbcc driver (vc)` at `bin/vc` and `vbcc directory (VBCC)` at the root containing `config/` and `targets/`. A driver at `<root>/bin/vc` lets Studio derive that root. Keep the compiler root free of spaces and shell metacharacters; the vendor configuration expands it unquoted. Your project folder may contain spaces; keep `vasmm68k_mot` and `vlink` at `<root>/bin` alongside the compiler, so a Finder launch needs no shell setup. For AmigaOS APIs, set the NDK `Include_H` directory, or let the setup's NDK row download it. Studio also finds a sibling `NDK3.2/Include_H` automatically. The standard C sample needs only the target package's C header. **Settings ▸ Target ▸ Link libraries** accepts library names without `-l` (for example, `amiga`). Defines accept `DEBUG`, `NTSC=1`.
- Xemu for the MEGA65 may be `xmega65` inside `xmega65.app` or `Xemu.app`. Studio's download installs a standalone `xmega65` in its tools directory. It still needs a MEGA65 ROM; see chapter 2 and the wizard's ROM

etherload and **mega65 ftp** live inside **M65Connect** `/Applications/M65Connect.app/Contents/Resources/etherload.osx` and `mega65 ftp.osx` beside it. The wizard's download puts the bundle in `~/Applications/m65connect-2.5/` instead.

- **Amiberry** and **FS-UAE** have verified downloads in the wizard. Their complete app bundles are retained, including libraries and resources; the executable is in `Contents/MacOS`. Installation is separate from choosing the emulator's machine configuration and mounted host folder.

Three places settings live

Studio has one Settings dialog: **File ▶ Settings...** `⌘,` with five pages:

- **General and Editor** — how Studio looks and behaves. Theme and most Editor changes preview live and are restored by **Dismiss** (`Esc`). **BASIC** line numbers is an exception: it saves immediately and **Dismiss** does not undo it. **Projects** folder and update switches wait for **Save**.
- **Machine** — named machines and tool locations, shared with File Manager in `~/.config/rebound-labs/config.json`. Machine actions save immediately. **Dismiss** (`Esc`) does not undo them. Tool-location drafts wait for **Save**. With a project open it shows only the tools that project uses; otherwise all.
- **Project** — the project's name and its list of targets (chapter 4).
- **Target** — the selected target: its entry file, assets, include search paths, toolchain, run destination, and the project-scoped settings of those adapters. "Written to this project's `rebound-project.json`, which is committed with your code."

Machine and Project/Target are deliberately separate: a friend who opens your project gets your targets, not your machine's tool locations. An absolute include path is an explicit exception to portability: prefer relative paths.

Theme (General), Syntax (Editor), Toolchain and Run (Target) are dropdowns: open the list, then click the value you want. Long lists scroll; click outside to close one without choosing. `Esc` dismisses Settings, subject to the exceptions above.

Short choices stay side by side; on/off preferences use toggles, and num/enc/font

DEV · laptop keep their `- / +` controls.

Updates

Settings > **General** > **Updates** has two switches. **Check for updates** (on by default) asks our server at each start whether a newer Studio is out; each check shows the server your IP address. Turning it off stops automatic Studio update checks, not the separate download-manifest request when you open setup. **Also offer beta versions** (off by default) adds the beta channel; you are then offered the newer of the stable and the beta release. Both switches are saved with **Save**.

A newer version appears as a green line at the right of the bottom bar, "Rebound Retro Studio 1.4.1 is available — release notes", and in **Help** ▶ **About Rebound Retro Studio** which also says when the check is off, still running, or could not reach the server. Pressing the line (or **About's Release Notes**) opens the release notes in your browser. Studio never downloads or installs an update itself; you download the new DMG and replace the app as you installed it.

Shared includes

In **Settings** > **Target** > **Include search paths**, choose **Add path**, then type `../libs` or use **Browse...** to choose that directory. Browse stores a path relative to the project folder where possible, including sibling folders. Absolute paths are accepted too. Each row is one path; spaces and commas do not separate paths.

Up / **Down** controls the order of these additional search directories. **Remove** removes a search entry; never a folder on disk. **Save** writes the selected target's `source_roots`. **Dismiss** discards the draft. Blank rows must be filled or removed before **Save**. A path may name a folder that is not currently available on this machine.

ACME, `vasm` and `ybcc` use these paths. Other toolchains currently ignore them; the **Target** page states this when one is selected. With `../libs` configured, `vasm` can resolve `include "gfx/macros.i"` from the sibling library directory. The assembler's own default search locations still apply; this list orders the additional paths, not every location the assembler checks.

These paths also extend **Search in Files**. They do not add a folder to the Explorer, copy library code or switch the active project.

The tables

Below, per machine, in the order the wizard installs them. "Tested with" names the version Studio's error parsing was captured against; a newer one almost always works, and the wizard names the difference rather than refusing.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

As of 2023-07-20, the tables below are empty. The wizard has not yet installed any of the tools listed below.

4. Projects and targets

A **project** is a folder with a file named `rebound-project.json` in it. That file is the whole definition: the project's name, its targets, and which target opens by default. Everything else in the folder is yours.

Creating one

File ▶ New Project... `↑%N` opens the New project dialog: a Name, a Create in choice, a folder (choose it with **Browse...**), and a Template dropdown:

C64 Assembly (KickAssembler)

C64 BASIC (petcat)

MEGA65 Assembly (ACME)

MEGA65 BASIC (petcat)

Amiga Assembly (vasm)

Amiga C (vbcc)

Create in decides what the folder means:

- **New folder** — the folder is the parent; Studio creates `<Name>` under it, writes a starter source file and the project file, and opens it.
- **Existing folder** — the folder is the project; sources you already have, say a batch of programs pulled off a disk image. Studio writes the project file beside them and leaves every file as it is; the starter is only written if the template's entry file (`main.bas` `main.asm` ...) is not already there. Picking the folder fills an empty Name with the folder's name.

A line under the folder spells out exactly which folder Create will write into. Studio refuses to create a project inside a folder that already has one.

Opening one

File ▶ Open Project... `↑%O` asks for the folder containing `rebound-project.json`. Its picker selects folders, not the manifest file. The start page lists recent projects. **File ▶ Open Recent Project...** lists them except the current one. `↑` `↓` to choose. `Enter` to open. Entries whose folder has been removed quietly drop off.

Studio will not leave a project with unsaved changes: whichever way you open another one, it says so and stays where it is until you save.

File ▶ Close Project returns to the start page.

What a target is

A target is one way of building and running this project. It names:

- the machine `c64` `mega65` `amiga` and the language
- the toolchain — the build tool `kickassembler` `petcat` `acme` `vasm` `vbcc`
- the run destination — an emulator `vice-x64sc` `xemu-xmega65` `amiberry` `fs-uae` or real hardware `c64-ultimate` `mega65-etherload` `apollo`
- the entry file — the source file the build starts from
- optionally, assets to ship beside the program, and per-adapter settings such as KickAssembler's `Defines`

A project can have several targets, and that is the everyday reason to have two: the same code with **Run** pointing at the emulator in one target and at the real machine in the other. **File ▶ Switch Target** `⇧⌘T` flips between them and remembers your choice locally for this project. If the project has only one, Studio says so: "project defines only one target".

Many mains, one project

A folder of small programs — even `.bas` has its own program — is one project whose entry changes with what you are reading. **Build ▶ Use Active File as Entry** `⇧⌘B` points the active target's entry at the file in the editor's active tab and writes the project file `⌘B` builds it from then on. The row is enabled only while that tab is a file inside the project, and Studio refuses while a build is running. Doing it twice for the same file says "already the entry".

Editing targets

Settings ▶ Project lists the targets. The `★` marks the default — what a fresh copy of the project opens with. `⇧⌘T` remembers your choice locally without saving it as default. Click a displayed target name to open its full settings, or

click **Rename** to edit the name in place. Press **Enter** or click **Save** to commit a name change. Non-default rows offer **Default**. Each row offers **Duplicate** and, when there is more than one, **Remove**. Duplicate is the quick way to make a full hardware twin of an emulator target.

Add Target creates a clean target with the project default's machine, language, dialect, toolchain, run destination and entry, but without its include paths, distribution contents, adapter settings, run arguments or custom command. Studio gives it a unique name and opens its **Target** page immediately.

Settings ► Target edits the selected one: **Name**, **Entry** (required, red while empty), **Include search paths**, the **Toolchain** and **Run** dropdowns, the chosen adapters' project-scoped fields, and **Command line**. Hardware targets have a **Machine** picker for the default or a named shared machine. Commodore targets also have **Distribution contents** (files or immediate folder contents); Amiga targets cannot build a disk image yet. **Save** writes the project file; **Dismiss** (**Esc**) discards the draft.

The command line

Every toolchain runs from one line, and the **Command line** field shows it as a

template: `{program} -quiet -Fhunkexe -o {output} {includes} {defines} {entry}` for **vasm**, `{java} -jar {jar} {entry} -o {output} {defines}` for **kick assembler** and so on. Each `{placeholder}`

filled in at build time from the settings around it — the **?** beside **Command line** explains each one for this toolchain — so an edited line keeps following your include paths and defines. You may change anything: add `-m68020` for a listing option, put a full path in place of `{program}` or reorder the arguments. Quotes group a word that contains spaces. **Runs:** beneath the field previews the expanded command; **Cannot run:** reports planning errors such as an unknown placeholder. A preview is not proof that an executable exists or will run, and **Save** is not a tool test.

Only an edited line is stored in the project file `command`; the factory line is not, so a future Studio can improve its defaults without a copy in every project pinning the old ones. **Reset to default** brings the factory line back, and is dimmed while the line already is it. One rule the field cannot bend: the line must produce the artifact `{output}` because that is what **Run** stages — a build that exits successfully. **DEV · laptop** This is reported as a failure that names the file.

Amiga run arguments

An Amiberry, FS-UAE or Apollo target also shows Run arguments. This is raw Amiga Shell syntax appended to the executable name in `deploy.txt`. Studio preserves quotes and spacing inside the value and previews the exact line the Amiga receives. The complete command is limited to 118 bytes so it fits deployd's 120-byte `FGETS` buffer with its newline and terminating zero. Arguments belong to the target, so emulator and hardware targets may use different maps or options without changing the machine-wide emulator setup.

The project file

You can read it. It is small JSON: `version`, `name`, `default_target`, `targets` list where each target carries `name`, `machine`, `language`, `dialect`, `toolchain`, `run_target`, `entry` and optional `run_arguments`, `source_roots`, `assets`, `settings` and `command` and, once you have attached a folder, `folders` list (below). Studio validates it on open: target names must be unique, there must be at least one, and `default_target` must name one of them. Commit it with your code. Absolute include paths, attached folders, adapter overrides or paths typed into a custom command can make it specific to your Mac; prefer relative paths where possible.

The Explorer panel

The sidebar's Explorer panel (`⇧⌘E`) shows the project folder and attachments. Its toolbar and the File menu share the same actions: New File..., New Folder..., Rename..., `F2` Move to Trash..., `Del` and Refresh Explorer. Deleting moves to the Trash; nothing is destroyed outright.

Folders beside the project

A library you share between projects – `libs/` next to `game/` – belongs in the panel without becoming part of the project. `File ► Add Folder to Explorer...` (the toolbar icon with tooltip Add Folder to Explorer) asks for a folder and adds it as a second top-level row under the project's own. The picker opens beside the project, because a sibling folder is the usual case. Inside that row, the folder works as it does in the project: open, open in a new tab, new file, new folder, rename, move to Trash, drag between its own folders. What it does not do is

cross the line: a file cannot be dragged from the project into the library or back, and the row itself cannot be renamed, moved or trashed.

The attachment is remembered in the project file's `folders` list, relative to the project where the disk allows, so it survives reopening and travels with a clone. Studio refuses a folder that is already in the panel, the project folder itself, or a folder inside or around one that is — two rows over the same files would be two answers to "which one did I edit".

What an attached folder is not. It is not a second project. `%B` `%R` staging/commit and Use Active File as Entry keep using the primary project. Source Control comparisons can use an attached repository; with several, choose one in Compare in which repository. A library's `rebound-project.json` is just a file in the tree. Nor is it an include search path: attaching `libs/` shows you its sources, it does not tell the assembler where to find them. For that, add the folder under Settings • Target • Include search paths (chapter 3) — the two are independent, and a folder can be either, or both.

Remove Folder from Explorer (the icon with that tooltip, or the File menu row) takes the selected top-level row out of the panel and nothing else: the folder and every file in it stay on disk, open tabs stay open, and Add Folder brings it back. It is deliberately a different action, button and menu row from **Move to Trash...**, which is the one that removes files. Only an attached row can be removed this way; the project folder cannot.

A folder that has moved or been renamed shows a `name (unavailable)` dimmed, at the top level. It cannot be opened or written into; **Refresh Explorer** picks it up again once it is back and **Remove Folder from Explorer** lets it go.

5. Build, Run, Deploy

Two keys do most of the work: **Build** **⌘B** turns the active target's entry file into a program, and **Run** **⌘R** sends that program wherever the target points. **Build and Run** **⇧⌘R** does both in one go: it saves, builds, and runs the fresh program only if the build succeeds — a failed build opens its errors instead. **Build** also saves all unsaved files before building. All three live in the **Build** menu. **Run** does not save or build: it uses a successful build from this session, or an existing artifact on disk before this session's first build. A failed or running build cannot run. The menu's **Run** needs a successful session build even when **⌘R** can use the disk artifact. With File Manager focused, **⌘R** runs its selected file instead.

What a build produces

Studio launches the build tool directly and captures everything it prints. Compiler drivers such as `ybcc` then invoke their own compiler, assembler and linker stages. On success the result lands in the project's `build/` folder:

- C64 (Kick Assembler or petcat): `build/<entry stem>.prg` — a PRG, the file format a C64 loads.
- MEGA65 (ACME or petcat): `build/<entry stem>.prg` — also a PRG, but a MEGA65 one. Run adapters check format compatibility; output paths are not separated by target.
- Amiga (yasm or ybcc): `build/<entry stem>` with no extension — an Amiga-executable, built from "hunks" rather than loaded at one fixed address. The stem of `main.asm` is `main`.

The dock's status line reads `no build yet` then `● building gen 1...` then `✓ gen 1 · main.prg` `✗ gen 1 · 2 error(s)` `gen` is the build's generation number: each build counts up, so you can tell a fresh result from a stale one.

Reading errors

Errors and warnings appear in the dock's **Build** tab. **Window ▶ Build Output** opens per-line, as `main.asm:12:5 error: ...` followed by raw tool output. **DEV · laptop** docks and reveals **Build**, so the result is in view when it arrives.

Click a row to jump to that line in the editor. **Next Diagnostic** **⌘E** walks through them from the keyboard. A line that has no file position — the tool complaining about itself, say — shows as **[kickassembler] error: ...**

Some checks happen before the tool runs. **petcat** for instance, silently accepts BASIC it will corrupt, so Studio checks line numbers and a few other things first and reports those as ordinary errors.

Run: emulators

With an emulator as the target's run destination, **Run** uses the available built program. Plain process runs replace the previous Studio-launched process. Amiberry and FS-UAE staged runs reuse an already running emulator of that name, including one you launched outside Studio.

- VICE **x64sc** for the C64.
- Xemu **xmega65** for the MEGA65.
- Amiberry or FS-UAE for the Amiga. These need two more settings: the emulator's own config file **.uae** **.fs-uae** and a deploy directory — a folder on your Mac the emulated Amiga sees as a drive. Studio copies the executable there, then atomically publishes **deploy.txt** **Run** **deployd** inside that same Amiga directory; it consumes the marker and launches the fresh program without a second emulator or a manual Shell command.

Create a distribution image

Build ► Create Distribution Image always builds the active target first, then creates a complete new Commodore disk image in **build/**. C64 targets default to D64, MEGA65 targets default to D81. Choose either format and edit the disk name, two-character disk ID and executable name in **Settings ► Target**.

The executable is always the first file. Add the remaining ordered contents with **Add Files...** or **Add Folder Contents...**. A folder source is read again for **Create Distribution Image** and **Put Assets on MEGA65**, not ordinary **Build**. Newly added or removed immediate files are reflected without editing the project. Nested folders are refused rather than silently flattened.

Studio owns the generated image. Each action creates it from scratch inside the **DEV · laptop** replaces the previous image only after the new one is complete.

Files removed from the target therefore cannot survive as state disk contents, and a failed generation leaves the previous image untouched.

Create an empty disk image

With a project open, choose **Window ► File Manager**, browse to a local folder, then choose **File ► New Disk Image...** Enter the **Filename** (including its extension), choose **D64**, **D81** or **ADF**, and enter the separate **Disk name** shown when the disk is mounted. **D64/D81** also have a two-character **Disk ID**. The new image is selected in the pane; an existing destination is never overwritten. Standalone File Manager offers the same dialog.

ADF creates a formatted 880 KiB data disk, with **OFS** selected by default or **FFS** as an option. It does not install boot code. File Manager still reads ADF contents without modifying them; mount the new image in an Amiga emulator and populate it there.

FS-UAE writeback matters: by default, FS-UAE saves floppy changes in overlays, not in the original ADF. To produce a distributable ADF, enable

writable_floppy_images = 1 in your emulator configuration. Studio does not change this setting for you.

Run: real hardware

- **C64 Ultimate**. No cable, no local tool: Studio talks over your network. Add a named C64 Ultimate in **Settings ► Machine**, enter its address (such as **C64-Ultimate-Founders.local**) and press **Save row**. Use **New**, **password** and **Set** if needed. **Make default** selects the fallback; the target's **Machine** picker can choose another. The wizard's **Real hardware** step edits the default machine. **Run** sends and starts the PRG.
- **MEGA65 over Ethernet**. Needs **etherload** (Chapter 3) and the MEGA65 on the same network. **Run** transfers the PRG and starts it. In **Settings ► Machine** the named MEGA65's optional **Address (MEGA65: optional)** can hold an IPv6 link-local address with its interface, e.g. **fe80::...%en0**. The target's **Machine** picker selects which machine to use. **Build ► Put Assets on MEGA65** copies the target's **Distribution** contents onto the MEGA65's SD card with **mega65_ftp**. Files land in the card's root using

their destination names, including **On-disk name overrides**. Duplicate names are refused case-insensitively.

- **Apollo (Amiga)**. Studio sends the executable to the deploy drawer, then send `deploy.txt` `deployd`. Watching that drawer launches the fresh program on CofinOS or ApolloOS; Studio reports "deployd notified" after both transfers are confirmed. **ApolloExplorer must be running on the Amiga**. Configure a named Apollo's address and drawer in `Settings ► Machine`. The target's **Machine** picker selects it or the default, a target drawer overrides that machine's drawer. Otherwise the fallback is `Personal:Sources`. The wizard edits the default Apollo; **Test connection** pings it and **Check drawer** asks ApolloExplorer whether its drawer exists.

One honest warning: a successful transfer is not a successful program. The C64 Ultimate reports the upload worked; whether your code then crashes is something only the machine's screen knows. When in doubt, make the program show a visible sign of life first — a border colour is the classic.

The C64 Ultimate remote

When the active target runs on a C64 Ultimate, the **Machine** menu comes alive and a small strip of the same buttons appears in the bottom panel's header, beside the line that reports what happened. They use the same resolved machine as `Run`, including a named machine selected by the target.

- **Reset** `^%R` — resets the C64. Your RAM survives, as with the machine's own reset button.
- **Reboot** `^%B` — the Ultimate's own reboot, its harder restart. RAM survives this one too.
- **Menu Button** `^%U` — the Ultimate's own menu, on its screen. Press it again to leave.
- **Pause / Resume** `^%P` — stops and restarts the C64's processor. The label follows what Studio did; the machine cannot be asked, so a pause made on the machine itself is not reflected. **Reset**, **Reboot** and **Clean Start** clear Studio's pause state. **Power Off...** also returns the label to **Pause**, but leaves the machine off.
- **Clean Start** `^%C` — the switch-off-and-on-again you usually wanted. Studio `Power Off...` the machine, writes the pattern a cold C64's memory holds into `$0002-$0FFF` and `$E000-$FFFF` reads it back, and only then resets. A

leftover from the last run cannot make this one look like it works. The read-back covers `$0002-$9FFF` and `$C000-$CFFF` under BASIC and the KERNAL a read shows the ROM, so the pattern is sent there but cannot be checked `$D000-$DFFF` is left alone (a write there would hit the video and sound chips), and so are `$0000/$0001`. If the read-back still differs after a second try, Studio refuses, names the address, and leaves the machine paused rather than starting it on half-written memory — Reset or Clean Start again.

- **Power Off...** `^%0` — asks first, naming the machine. Switching it back on always takes the rocker switch; the network goes down with the machine.

Every action reports in the status line: green when it happened, red with the reason when it did not.

The MEGA65 remote

When the active target runs on a real MEGA65 (over `etherload`) the Machine menu's two MEGA65 rows come alive, the C64 Ultimate's rows dim, and the strip shows the MEGA65's buttons instead. They use the same `etherload` and resolved named-machine address as `Run`. The MEGA65 must be listening: DIP switch 2 ON, and `SHIFT + £` pressed so the power LED blinks green/yellow.

- **Reset → MEGA65 Mode** `^%R` — resets the MEGA65 into MEGA65 mode, the mode every MEGA65 target builds for. `^%R` means Reset on whichever machine the target deploys to.
- **Reset → C64 Mode** `^%4` — resets it into C64 mode, as `G064` would.

The MEGA65 stays listening after either reset (the LED keeps blinking), so you do not press `SHIFT + £` again. A remote reset skips your autoboot program: the one a power-on or the reset button starts from the SD card does not run. Ethernet offers nothing more than these two resets: pause, typing into the machine, screenshots and reloading the core need the serial monitor and a UART/JTAG adapter, which Studio does not use yet.

Success is green; a refusal is red and says what to do. If no MEGA65 answers, the line names every fix: DIP switch 2 ON, `SHIFT + £` until the LED blinks green/yellow, and UDP port 4510 let through the firewall. Studio does not trust

`etherload`’s exit code alone: it reports a reset only when the tool says it sent one.

When Studio says no

Many refusals appear in the bottom status line and name a key, field or action. Project-creation errors appear in their dialog. File Manager has its own notices. Examples:

- "open a project first (⌘O)"
- "no runnable artifact — build first (⌘B)"
- "unsaved changes — save first (⌘S)"
- "a tool is missing — File • Set Up Commodore 64..." — a path is not set; the wizard fixes it.
- "no C64 Ultimate host configured — add a default C64 Ultimate in the Settings • Machine machine list, e.g. C64-Ultimate-Founders.local". Missing `etherload` and `mega65 ftp` instead need their tool paths configured.
- "Machine • Reboot needs a C64 Ultimate target — switch to one with Switch Target (⌘T)" — the remote only talks to the machine the active target deploys to. `Reset` asks for "a C64 Ultimate or MEGA65 hardware target".
- "the active target does not deploy to a real MEGA65 — switch targets (⌘T)" and "no assets declared — add files in Settings • Target under Distribution contents", both from `Put Assets`
- "build in progress — switch after it finishes" — targets cannot change mid-build.

The terminal

The band below the editor is a real shell `$SHELL` or `/bin/zsh`. A new session starts in the current project folder; existing sessions keep their directory across project changes. `⌘J` shows and hides it, `⌘2` focuses it, `⌘1` returns to the editor. Ordinary keys go to the shell, including Vim letters; the shell decides what they do. Studio handles its reserved chords first. Other app chords go to the editor with a project open, except terminal Copy/Paste. `Window ► Maximize Terminal` grows it to its allowed maximum, leaving an editor floor and the other bottom panel. `Show Terminal Manager` lists the sessions and lets you open more.

6. The editor

The editor in the middle of Studio is the Rebound Editor, embedded. When an Editor tab is active, its **File**, **Edit**, **Selection**, **Find**, **View**, **Tools** menus join Studio's own in the one menu bar. The active surface owns its editing commands; chapter 7 lists Editor chords.

File Manager in the workbench

With a project open, choose **Window** > **File Manager**. It opens one tab in the same strip as your files; choosing it again selects that tab rather than creating another. Group, reorder and color it like an Editor tab. First open starts at the project folder on the left and your home folder on the right. Pane locations and bookmarks are remembered per project, including when you close and later reopen the File Manager tab.

Double-click local editable text to open it in an Editor tab. Images keep their file-manager browsing behavior; programs keep their emulator and **Run on Machine** actions. Other local files use their default application, with the same executable/firmware refusals as standalone File Manager. Opening a file never consumes the File Manager tab.

File Manager's menus replace the Editor editing menus while it is active. Studio's project, build, theme and window commands remain available, as do the Explorer and terminal. Studio owns the theme; **Help** > **File Manager Preferences...** retains File Manager's font and list preferences without replacing the standalone application's theme or pane session.

Run  (with File Manager focused) runs the selected file, even before any project build. File Manager prepares the emulator/media/hardware plan; Studio executes it and shows the result in its output dock. **Set up...** opens **Settings** > **Machine** for the required adapter, even if the current project targets a different machine.

Transfers keep running behind other tabs. Closing File Manager, switching or closing the project, and quitting show **Transfers running** while transfers are active. **Cancel transfers and close** cancels them, rather than waiting for success. **Keep**

`open` cancels the requested transition, not the transfer. Already-confirmed filesystem operations finish before closing.

Moving an open file carries its Editor buffer to the new path, including unsaved edits. Deleting it keeps the text unsaved in the Editor. Copying over a clean open file reloads its new contents; overwriting a dirty destination, or moving onto another open document's path, is refused before disk changes. Read [Notices](#) for the reason and Save/close guidance.

Two keyboard profiles

The editor has two ways of reading your keys, and you choose in [Settings](#) ► [Editor](#) how Keymap, by selecting **VIM** or **STANDARD**:

- **VIM** (the default) — modal editing: you are in normal mode until you press [i](#) to insert text and [Esc](#) to leave again. [:w](#) saves, [:q](#) closes a pane, [u](#) undoes. If you have never used Vim, the Standard profile is the honest recommendation until you feel like learning it.
- **STANDARD** — what most editors do: type to type. [%S](#) saves, [%Z](#) undoes, [%F](#) finds. Line and mark commands open small, initially empty prompts; other menu commands can execute directly. [⌘M](#) then a letter and [Enter](#) remembers the caret, [⌘J](#) then the same letter, jumps back.

The menu bar is a cheat sheet for whichever profile is active; the same item shows [:w](#) in one and [%S](#) in the other.

What "Vim" means here

The Vim profile is Vim-flavoured, not Vim. It has Normal, Insert, Visual and visual Block modes, counts, operators, text objects, marks, registers, [:](#) commands and [/](#) search; chapter 7 lists the bindings. Visual Block does not support [y](#) or [c](#). Block deletion stores joined text in a charwise register, so pasting inserts it linearly, not as a rectangle. There is no `.vimrc` plugin model. [:set](#) Commands such as [gd](#), [:preview](#), [:compare](#) and [:gdiff](#), plus hover cards, are additions.

One deliberate editing-key departure is:

In Normal mode, an editing key does what it does in Insert mode and leaves you in Insert mode.

Enter splits the line at the caret (auto-indent included) and you are typing. In Vim it moves down one line, which **j** already does.

Backspace deletes the character to the left and you are typing. In Vim it moves left, which **h** already does.

Delete deletes the character under the caret and you are typing. In Vim it is **x** and you stay in Normal mode.

At a line's edge they join lines exactly as in Insert mode. **Backspace** in column 1

joins with the line above. **Delete** at the end joins with the one below. A count repeats the key — **3x** deletes three characters. The deletion and whatever you type after it undo together, as one step; in Vim they are two. The one habit this can trip: if you press **Delete** to delete and then reach for **j** you type **j** — press

Esc first, as you would after **i**.

If that reads as heresy, it is decided and documented, and the Standard profile is one setting away. If it reads as an improvement, you are the person it was made for.

Tabs, panes and groups

A file opens in a tab; an Explorer preview can replace a clean, un-paired preview

tab. **⌘]** and **⌘[** move between tabs. A window can be split — **View ▶ Split**

Left/Right and **Split Top/Bottom** **:vsp** **:sp** in Vim **⌘** **↑⌘** in

Standard — and **Next Pane** moves the focus between the halves. Tabs can be

grouped **Tabs ▶ Group with Previous Tab** **Group with Next Tab**,

collapsed, renamed and given a colour, which is how a large project stays

navigable. When tabs overflow the bar, opening or selecting a file brings its tab into view automatically, including files selected in the Explorer. The bar moves only as far as needed; selecting an already-visible tab leaves it in place. For a Markdown

file, **View ▶ Markdown Preview** **:preview** switches that same pane between

editable source and rendered Markdown. It does not create a split. Choose **Markdown Preview** again to return to the source with its caret, contents and undo history intact.

Folding

Any line whose following lines are indented deeper can be folded: a label with its instructions, a C function body, a BASIC loop you indented by hand. A small ▼ in the gutter marks such a line; click it to collapse the block to its header, which then reads

```
start: ... 12 lines
```



```
za
```

toggles the fold at the caret. `zc` `zo` close and open it. `zM` folds everything. `zR`

unfolds everything, and `zj` `zk` jump to the next / previous header. View offers **Toggle Fold**, **Fold All**, **Unfold All**, **Next Fold** and **Previous Fold**, not separate close/open items. Folds nest with indentation and are remembered per file across sessions. A collapsed block never hides the caret; jumping into one — from a

search hit, a build error `gd` — opens it. Folding follows indentation, except that a closing-only

```
}
```

```
)
```

```
]
```

line (optionally followed by `;` or `,`) folds with its block. A brace on its own line still becomes the header rather than the line above.

Finding things

Find ► Find...

Find Next

Find Previous

Step through matches. `Replace...` adds a replace row. Searches are literal unless you switch the bar to regular expressions. `Go to Line...` does what it says.

What the editor knows about your machine

Hover an instruction mnemonic — `lda` `sta` `jsr` — for a summary and flags.

Legal 6502 instructions also show addressing modes and cycles; 45GS02 and 68000 cards show CPU provenance instead. Hover a hardware address like

```
$D020
```

and the card names the register. In a BASIC file, hover a keyword —

```
print
```

```
poke
```

```
chr$
```

— and the card says what it does. Both come from the same tables the Docs panel's reference pages are generated from, so the card and the page cannot disagree.

BASIC is coloured the way the Co4 itself reads it: the machine looks for keywords anywhere outside quotes and needs no spaces, so `forI=1T010` colours

completely — and `printer` now `print` is coloured inside it, because typing that on a real Co4 really does print the variable `er`

C source and header files `.c` `.h` highlight keywords, preprocessor directives,

DEV · laptop `#` character literals and comments. Block comments remain

comments across lines. This is syntax colouring, not a C language server: it does not provide type-aware completion or compiler diagnostics while you type. Build with `⌘B` for vbcc's diagnostics.

Line numbers, handled for you

Two things a BASIC listing needs, and neither is worth typing by hand.

Enter numbers the next line. Press it at the end of a numbered line and the new line starts with its number, following the step your listing already uses — tens if you write in tens, fives if you write in fives. Between `10` and `20` you get `15` where the gap is full `10` `11` you get a plain line and the number is yours to choose, because the alternative would be renumbering your program behind a keystroke. Switch it off in **Settings ▸ Editor ▸ BASIC line numbers** if you would rather type them.

Edit ▸ Renumber BASIC Lines `⌘␣:renumber` rennumbers the whole listing from 10 in tens, and rewrites every `GOTO` `GOSUB` `ON ... GOTO` `THEN` `<line>` and `RUN` that pointed at the old numbers. `⌘␣:renumber 100 5` starts at 100 and counts in fives. It is one undo step.

Two things it deliberately does not do. `␣GOTO` inside a string, after `REM` in a `DATA` payload is text, not a jump, and stays exactly as written. And a jump to a line that does not exist is left alone and reported in the status bar — renumbering it would point it at a different line than it does now, which is a change to what your program means.

The Outline panel `⌘␣L` lists labels for 6502/KiCk Assembler syntax, not 45G502 or 68000 assembly. It also lists Markdown headings, including this manual. Clicking one jumps there.

Edit ▸ Complete Word `Ctrl+P`, `Ctrl+N` to cycle the other way) completes identifiers from the current buffer in Standard or Vim Insert mode. It has no popup or semantic knowledge: the assembler still checks whether a label exists.

Search in Files

`⌘␣F` **Window ▸ Search in Files...** opens the Search panel in the dock with `DEV · laptop` field. Type a term and press **Enter** or click **Search**, each hit is one

row, `file:line` and the line itself. Clicking a row opens a different file in a new nailed tab at that line; a hit in the file already open reuses its tab. The buffer you searched from — including unsaved edits — stays open. The three switches beside the field are the find bar's: **Aa** match case, **ab** whole word, **/** regular expression.

It searches, in this order: the project folder, every attached folder, the target's include search paths, and — for C — the compiler's own headers (ybcc's `targets/m68k-amigaos/include` and the NDK's `Include_H`). Hits outside the project are prefixed with their root's name, so `Include_H/exec/types.h` and a `types.h` of your own do not read alike. `build/` `target/` and dot-entries are skipped, as are binaries; the panel's header says how many hits in how many files, and says so when it stopped at the first 2000.

It is a walk, not an index: every Enter reads the files again, which on a project plus the NDK takes well under a second.

Go to Definition

⌘-click an identifier, or put the caret on it and press **gd** in Vim Normal mode. On Linux, use **Ctrl-click**. Studio searches the same roots as Search in Files, looking for declarations in the active target's language. One match opens its file; several open the Search panel under **definitions of...** where you choose a row. No match leaves you in place and shows a notice.

This is declaration matching, not a compiler's symbol resolver. It recognises C types, macros and function declarations, assembly labels and equates, and BASIC line numbers and variable declarations. It cannot resolve scope or conditional compilation; a prototype and an implementation can both appear. The search reads files on disk, not unsaved edits. A successful single jump leaves your previous Search query untouched.

Git, without leaving

Two surfaces, and they stay small on purpose:

• **Source Control** in the dock (**Window ► Source Control**) shows **Staged Changes** and **Changes**. Click a file to mark it (the row inverts), click again to unmark; clicking a list's heading marks or unmarks the whole list. The

DEV · laptop



stages the marked files and



unstages them. Double-click a

file to open it.  in a row discards its changes after asking (an untracked file goes to the Trash). A Commit message field and Commit button, plus Pull (fast-forward only) and Push. If the folder is not a repository it says so: "This project is not in a git repository."

- **Git Diff** **View ▶ Git Diff** or **:gdiff** compares HEAD with the live, editable working buffer. **unstaged** compares the index with that buffer; **staged** compares two read-only snapshots. In Studio, **⌘D** opens Docs, not Git Diff. Inside a diff, **Next Change / Previous Change** **]c** **[c** or **F7** **Shift+F7** step between hunks, and **Revert Change** **dc** **Alt+F7** or the  (in the gutter) puts one hunk back the way it was.
- **Compare Against...**  in the Source Control toolbar **:compare** first asks **Compare in which repository** if several are available. Then choose a branch or revision — type to filter, click one to highlight it, double-click (or Return) to compare; click the mode to switch it. It then lists what differs from that revision; double-click a file to open it side by side. The current side is your real file — edit it, revert a hunk into it, and save it as usual. The revision side stays read-only.

Read-only views

Reference pages and this manual open as **read-only preview tabs**, not automatic side-by-side panes. Scroll and drag-select/copy rendered text; ordinary **⌘F** and  do not search the rendered page. Use the Docs search to find a page or section. The old side of a diff is also read-only.

Saving and reloading

Save writes the file. **Reload from Disk** is enabled only for files without unsaved edits; it rereads changes made by a tool or git command. Closing a tab or pane that holds the last view of an unsaved file asks **Save / Discard / Cancel** first. **:q!** (in Vim discards without asking), and quitting Studio with unsaved changes asks too.

Restored files load in the background, retaining their saved caret and scroll positions while they wait. A pane says **Loading ...** until its contents arrive. If a read fails, it shows the reason rather than looking like an empty file. **Reload from Disk** again and restores the retained position if it succeeds. A genuinely

empty file still looks empty. Docs pages restore through the Docs catalog and keep their own rendered scroll positions.

Save As changes the document's name and path only after the write succeeds. A refused or failed write leaves the original identity intact; an unknown outcome is reported as unknown, not as success. A file that has not loaded cannot be saved. While a write is still running, closing its window or replacing the project is refused—try again once the save has finished.

7. Keyboard shortcuts

Everything below is generated from the same tables the menus are drawn from, and spelled for the computer this Studio is running on — on a Mac you see `⌘` on Linux the `Ctrl` chords that mean the same thing. If a key here does not do what it says, that is a bug, not a typo.

Two things to know about how Studio hands out keys:

- **Studio chords precede normal surface input.** `⌘B` builds from the editor, terminal or sidebar, but open dialogs, menus and rename fields can own the keyboard first. Off macOS these avoid plain `Ctrl` which the shell needs.
- **Other keys follow focus, with an app-chord exception.** Ordinary terminal keys go to the shell. Terminal Copy/Paste stays there; other application chords go to the editor when a project is open. Sidebar app chords also reach the editor.

8. Docs, themes and zoom

The Docs panel

 shows the sidebar's **Docs** panel. Its pages are grouped by subject: the active machine, Assembly or BASIC, the active toolchain, and Studio itself. The compact groups keep unlike references apart without hiding them behind different windows. Clicking a page opens a read-only preview tab, not a split.

The search field above the groups searches page titles and full text as you type. **Search** runs the visible query explicitly. Results name their group, page and matching section; clicking one opens the page at that section. Clear the field to return to the grouped catalog.

Three kinds of bundled page live here. The **instructionsets** and **hardware registers** are generated from the tables the editor's hover cards read, so they are always in step with the editor. The **quick reference** and **screen memory** pages for the C64 are written by hand, checked against the machine's programming manual. And each toolchain — KickAssembler, petcat, ACME, vasm and vbcc — brings a **Reference** (its directives, macros, includes, conditionals, expressions and output formats) and a **Command Line** page (options and what Studio passes), condensed from the tool's own manual; each opens with the source and version it was read against, and the official manual stays the authority. Switch the target's toolchain and that group switches with it.

An Amiga target also offers a collapsed **NDK Autodocs** group, when Studio can find **Autodocs/** beside the **Include_H** configured under **Settings ► Machine ► AmigaOS NDK headers (Include_H)**. These are the NDK's original exhaustive library, device, class and datatype references, converted from their fixed-width text format when Studio reads them. They are never bundled with Studio; without a local NDK the group is simply absent. The Docs search includes them.

With a project open, the **Studio** group contains this manual, **Writing Your Own Theme**, and **Third-Party Licences** — the notices of every open-source library, font and icon set Studio is built with, also reachable from **Help ► About Rebound**.

DEV · laptop  The Docs search finds a library or licence there by name.

Opening a page activates its tab and focuses the editor area. Switch the sidebar to **Outline** `⇧⌘L` — the active page's headings become a table of contents — click one to jump. That is how this manual and the converted Autodocs are meant to be read.

Themes

Every colour in Studio — editor, terminal, menus, panels — comes from one theme.

Nine are built in: `neon-retro` `cyberpunk` `amber` `green` `typewriter` `c64` `workbench-1.2` `workbench` and `workbench-3.3`.

Settings ▸ General ▸ Theme lets you choose a built-in or user theme directly from a dropdown. **Window ▸ Cycle Theme** `^⌘T` still steps through them without opening Settings.

- Your own themes are small TOML files in `~/.config/rebound-retro-studio/themes/` **Window ▸ Reveal Themes Folder...** opens that folder in the Finder. A theme can `extends` a built-in and change three lines. Restart Studio to discover a newly added theme. The page [Writing Your Own Theme](#) is the full guide, including colour names and which ones move together.
- A misspelled colour name is reported, not ignored — staring at a colour that never changes is the worse outcome.

Zoom

View ▸ Zoom In `⌘+` **Zoom Out** `⌘-` and **Reset Zoom** `⌘0` change the editor's font size between 8 and 40 points (15 is the default), and the whole window follows: menus, sidebar, dock and dialogs scale with the text, so a larger font is a larger Studio, not a large editor inside small chrome. The setting is remembered between launches.

One consequence worth knowing: after zooming a lot and back with `⌘0` the terminal band may stay at its minimum height, exactly as it would after shrinking the window. Drag it back or **Window ▸ Maximize Terminal** and restore.

9. When something goes wrong

Studio tries to say what is wrong in one line and name the key that fixes it. This chapter is for the times that line is not enough.

"a tool is missing"

A build stopped because a path in **Settings ► Machine** is empty or points at a file that is not there. The banner names the wizard row, e.g. **File ► Set Up Commodore 64...** Run it; moving between wizard steps saves discovered paths into empty settings without overwriting paths you chose. Check the tools builds a throwaway sample. **Not set up yet** means the tool is missing; use **Choose installed tool...** or a download button when offered. Chapter 3 explains the vendor locations.

The wizard found it, the build still fails

- **That file ran, and it is not this tool. Choose another one.** — point the field at the real binary.
- **Found it, but it would not run: ...** — the file is there but will not start. Read the reason. **Waiting for the Java runtime below**, or **Java is not installed on this Mac**, points to the Java dependency row.

macOS blocked this tool because it was downloaded in a web browser

— a tool you downloaded with Safari or another browser (M65Connect, typically) still carries macOS's download flag, and macOS stops it the moment Studio runs it. Open Anyway in System Settings approves the app, not the tools inside it. Press **Allow M65Connect's tools** (the button names the app) on that row; Studio removes the flag from the app — the same as `xattr -dr com.apple.quarantine /Applications/M65Connect.app` in a terminal — and looks again. It does this only when you press the button. While Studio looks, macOS may show "M65Connect" Not Opened; choose **Done**, never **Move to Trash**. If allowing fails (an app another user installed, for example), the row says why; the wizard's own M65Connect download is never flagged.

Java 8 is older than Java 21, the version Studio is

DEV · laptop **with** — Studio runs Kick Assembler with Java 21 or newer.

Download Temurin 21 on the wizard's Java row, or choose a newer JDK. If you set the old Java's path in **Settings ► Machine** yourself, clear it: a path you typed always wins.

- **Version differs** — You have {found}; Studio was tested with {tested}. That usually works. If error lines in Build Output point at plainly wrong places, the tool's message format changed; the tested version is the one to install.

"nothing happens" when I press a key

Check which surface has focus. Studio chords normally precede surface input, but dialogs, menus and rename fields can own the keyboard first. In the terminal, ordinary Vim letters go to the shell; Copy/Paste stays there, while other app chords go to the editor with a project open. **%1** focuses the editor, **%2** the terminal. On a German or other non-US layout, a chord that needs a shifted character can be unreachable; the menu item still works with the mouse — and that is worth reporting, because such chords get re-picked.

The program built but does not run on the real machine

The transfer succeeded and the program crashed a moment later — the machine knows, Studio cannot. Start from the sample project for that machine; if the sample runs and yours does not, the difference is in your code, most often the start address or the BASIC upstart. Make the first instructions change the border colour, so that "did it start at all?" has a visible answer.

The MEGA65 is not reached

etherload needs the MEGA65 on the same network. Press **SHIFT + E** on the MEGA65 to enable network access. In setup, **Look for my MEGA65** listens for it. **Set the address by hand...** reveals **Manual address** if needed. Use a scoped IPv6 address such as **fe80::...%en0**. Named MEGA65 addresses also live in **Settings ► Machine**; the target can select a named machine. Put **Assets** on MEGA65 additionally needs **mega65_ftp** configured and the target's **Distribution** contents filled.

Studio stopped seeing a tool after I rebuilt or re-signed something

Not Studio's doing, but worth knowing: macOS ties some permissions to the exact binary. A rebuilt or re-signed app or tool loses grants like Accessibility and has to be granted again in System Settings.

Where things are

What	Where
Studio's preferences	<code>~/.config/rebound-retro-studio/config.json</code>
Shared tool locations and named machines	<code>~/.config/rebound-labs/config.json</code>
Machine passwords	System keychain
Your themes	<code>~/.config/rebound-retro-studio/themes/</code>
Compilers and assemblers Studio downloaded	<code>~/.local/share/rebound-retro-studio/tools/</code>
VICE, Amiberry and FS-UAE downloads	<code>~/Applications/<bundle>-<version>/</code>
xemu download	Standalone <code>xmega65</code> <code>index</code> <code>~/.local/share/rebound-retro-studio/tools/</code>
Sample projects you opened	your projects folder <code>~/Documents/Rebound Retro Studio/</code> <code>index</code> changed <code>Settings ▶ General</code> <code>open with</code> <code>File ▶ Settings...</code>
	<code><project>/rebound-project.json</code>

What	Where
A project's definition	
Build results	<code><project>/build/</code>

Reporting a bug

Choose `Help ► Report a Problem...` Studio gathers its version, OS and machine information (macOS version and Mac model on a Mac), tool discovery and the end of the last build's output, and shows you all of it first. **Open Mail Draft** puts that text into a new message in your mail program — edit it, add what happened, and nothing is sent until you press **Send** there. No mail program set up? **Copy Text** and mail it from anywhere. Your home folder appears as `~`

Quote the exact banner text too; it is written to be quoted. The same tool report is printed by the `rebound-retro-studio` binary inside the app bundle when run with `--discover` in a terminal.