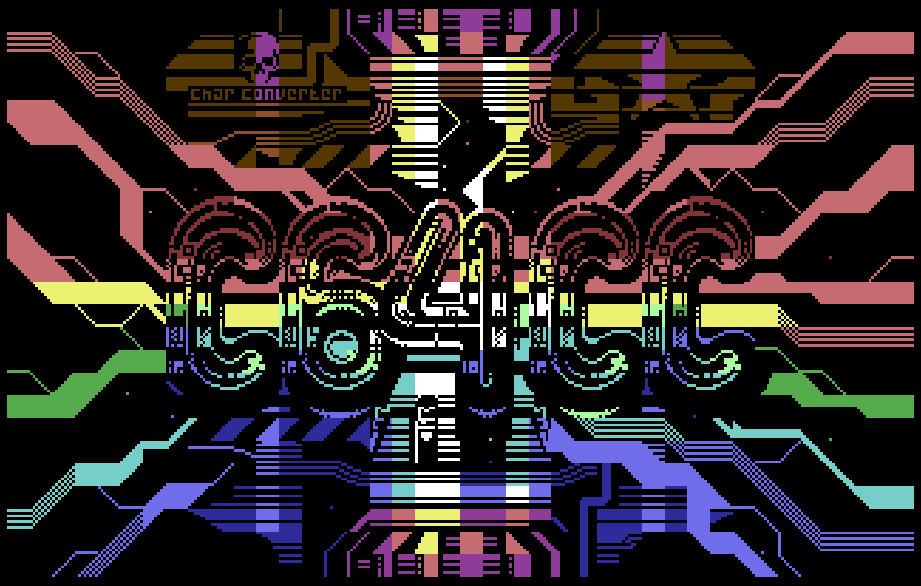


C64CC

Commodore 64 Character Set Converter

---

version 1.0

Manual

www.elder0010.com

cover image and program icon by Sulevi / Extend

C64CC v1.0 - Commodore 64 Character Set Converter — Manual

Index

Introduction

C64CC (Commodore 64 Character Set Converter) turns PNG images into ready-to-use Commodore 64 character set data — Hi Res, Multi Colour, or ECM — including the screen (character) map, colour RAM (D800) map, and an optional ready-to-run `.prg` executable.

It was built around one idea: **coders and graphicians should be able to work from the same tool**, instead of one side having to hand-convert assets for the other. A graphician can sit down with the GUI, iterate visually on colours, dithering, and cropping, and export finished assets directly. A coder can drive the exact same conversion engine from the command line, wired straight into a build script or Makefile, so a project's charset regenerates automatically from its source art on every build — no manual export step, and no drift between what the artist made and what ships.

This also makes the tool handy for fast prototyping: a coder can set a maximum character count that fits their demo or game's budget (**Max chars** in the GUI, `-nchars` on the CLI) and drop in almost any source image — the tool automatically adapts it to that limit, reusing characters once the budget runs out, instead of requiring hand-tuned source art from the start.

Feature summary

- **Three video modes** — Hi Res, Multi Colour, and ECM, each with mode-appropriate colour handling.
- **Automatic colour guessing** — analyses a full-colour source image and picks the best C64 palette combination, so artists never need to pre-quantize their artwork by hand.
- **Five dithering algorithms** — None, Floyd-Steinberg, Atkinson, Diffuse (Bayer), and Levenshtein-based character reuse.
- **Any image size** — up to 320×200 maps directly onto a C64 screen; larger images convert too (with a scrollable preview), they just can't be turned into a runnable `.prg`. Smaller or non-block-aligned images are padded automatically.
- **Cropping** — export just a sub-region, with export-specific behaviour for the C64 executable (in-place zero-fill, no repositioning) versus the other export types (extract and reposition).
- **Multiple export formats** — raw binary, KickAssembler `.asm` source, PNG previews, and a self-contained `.prg` that runs immediately on a real C64 or emulator.
- **Two interfaces, one engine** — a full GUI for interactive, visual work, and a scriptable CLI with the same feature set for automated build pipelines.

Platforms

- **Windows** — two executables, both the same program: `C64CC.exe` for the GUI (double-click it; no console window ever appears) and `C64CC-cli.exe` for terminals and build scripts. Windows fixes at build time whether a binary owns a console, so one file can't cover both without flashing a console up on every GUI launch.
- **Linux** — a single executable doubles as both: launch it with no arguments for the GUI, or from a terminal with arguments for the CLI.

- **macOS** — the zip holds the GUI app bundle `C64CC.app` and, in a separate folder, `cli/C64CC-cli` for terminals and build scripts. A `.app` can't take command-line arguments, hence the second binary; it lives in its own folder rather than beside the app because a Unix executable sitting next to one invites being double-clicked from Finder.

---

1. Video modes

Mode	Bits/pixel	Colours per 8×8 block	Notes
Hi Res	1bpp	Shared background + 1 per-block foreground	
Multi Colour (MC)	2bpp	Shared BG/MC0/MC1 + 1 per-block D800 colour	D800 stored 0–7 internally, exported +8
ECM	1bpp	4 global backgrounds (BG0–BG3) + 1 per-block foreground	Hard-capped at 64 unique characters

2. Image requirements

- **PNG only.** The file's real content is checked rather than its extension, so a JPEG or BMP renamed to `.png` is rejected, as is any file that isn't an image at all. Both the GUI and the CLI say which format they actually found.
- **Any size.** Images up to **320×200** correspond to a real C64 screen and can be exported as a runnable `.prg`. Larger images convert normally and every other export still works — only the C64 executable is unavailable (see §8).
- Images smaller than the C64 screen, or not a multiple of 8, are automatically padded up to the nearest 8×8 block boundary using the selected background colour, so the padding converts into empty/background characters.
- **Transparency becomes background.** Fully transparent pixels are composited onto the selected background colour, so they convert into empty background characters; semi-transparent pixels are blended against it. The alpha channel is never simply discarded — doing that would expose whatever hidden RGB the source editor happened to leave underneath, which is invisible while drawing and differs between tools, so the same transparent background could come out white, black or anything else.

---

3. Source image colours & palette mapping

On import, every pixel is compared against the fixed table below and reassigned to whichever of the 16 entries is numerically closest (smallest squared RGB distance) — a simple nearest-colour match, not an exact lookup. A pixel of `#6A3A2C` and a pixel of `#68372B` both simply become

palette index 2 ("red"); there's no meaningful difference between "close enough" and "exact" as far as the converter is concerned.

Index	Name	Hex (as used by this tool)
0	black	`#000000`
1	white	`#FFFFFF`
2	red	`#68372B`
3	cyan	`#70A4B2`
4	purple	`#6F3D86`
5	green	`#588D43`
6	blue	`#352879`
7	yellow	`#B8C76F`
8	orange	`#6F4F25`
9	brown	`#433900`
10	light red	`#9A6759`
11	dark gray	`#444444`
12	gray	`#6C6C6C`
13	light green	`#9AD284`
14	light blue	`#6C5EB5`
15	light gray	`#959595`

This nearest-colour mapping is applied at two different points, for two different purposes:

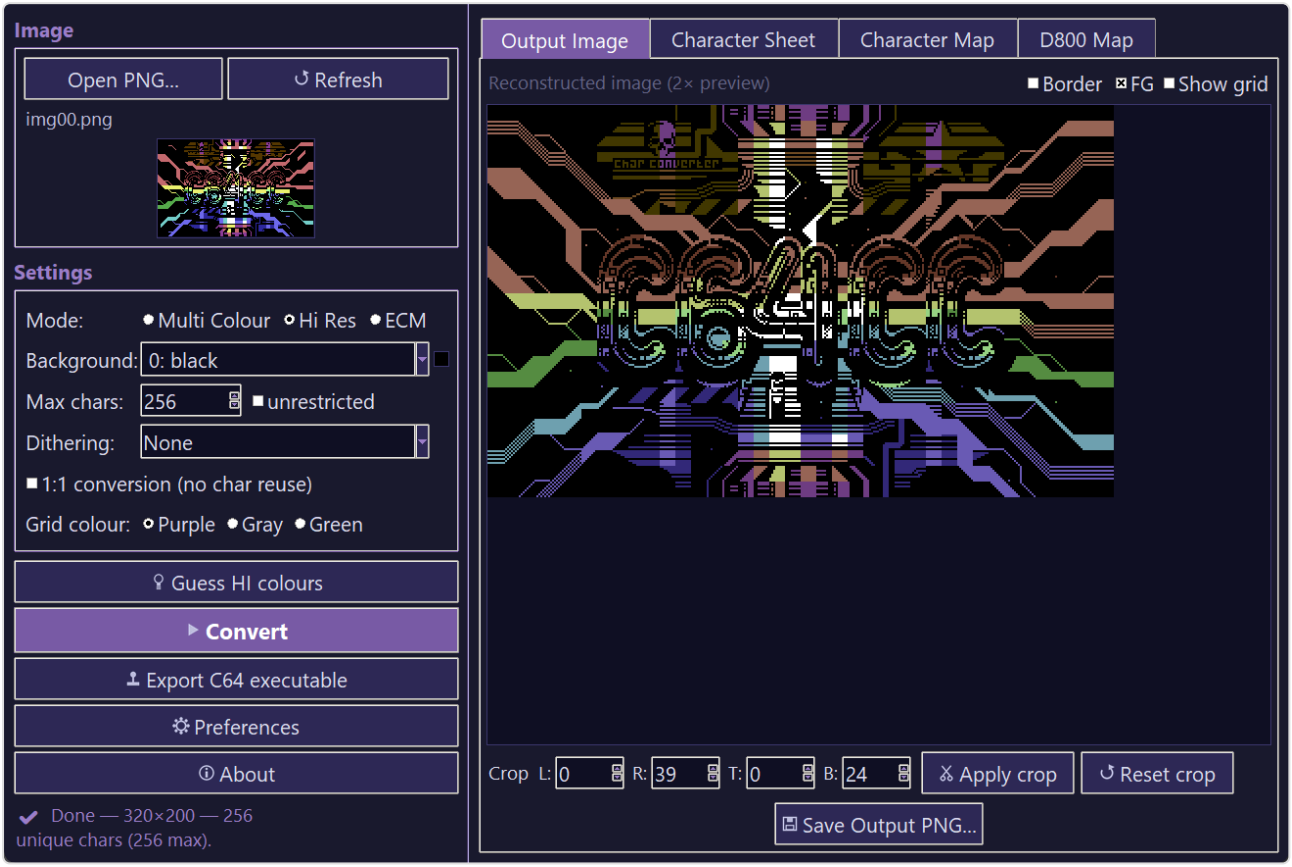
- **During colour guessing** (§5) — the whole image is quantized once, purely to work out which combination of *shared* global colours (BG/MC0/MC1, BG, or BG0–BG3 depending on mode) covers the image best.
- **During Convert** — each block is matched again, this time constrained to whatever colours are actually allowed for that block (e.g. {BG, FG}, or {BG, MC0, MC1, D800}), and optionally spread out with dithering to approximate colours the fixed palette can't represent directly.

Practical implication: any reasonably-coloured source image will work. A similar palette — same rough hues, roughly the right brightness — is enough; it does not need to match the table above exactly. The closer your source image's colours already are to these 16 values, the higher the guess "score" reported by the tool and the less a block's detail gets lost in the process, but there is no hard requirement to pre-match them.

---

4. GUI usage

Launch the application with no arguments to open the GUI.



4.1 Left panel

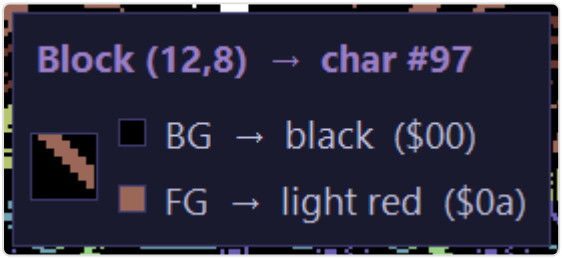
- **Open PNG…** — load a source image from disk. You can also simply **drag & drop** an image file from your file manager anywhere onto the window — it behaves exactly the same as opening it through the dialog.
- **Refresh** — reloads the current source image from disk (same file path) and, if a conversion has already been run, re-converts it immediately. This is meant for working alongside an external image editor: a graphician can edit and save the PNG in their tool of choice, switch back here, and just press **Refresh** to see the updated output — no need to re-open the file or re-enter any settings.
- **Mode** — HI / MC / ECM selector.
- **Colour pickers** — background, MC0/MC1 (MC mode) or BG0–BG3 (ECM mode). Each has a **guess colours** button that analyses the image and picks the best palette combination — see §5 for how it works and when it runs.
- **Max chars** — cap on unique characters generated, 1–2048 (ECM is hard-capped at 64). 2048 is eight C64 charsets of 256; asking for more stops with an error. Going **above 256** is fine for every export except the C64 executable — see §8.3.
- **unrestricted** — ticking this ignores the Max chars value (the field greys out) and lets the conversion use as many characters as the image needs. The hard limits still apply: 2048

overall, 64 in ECM mode — so "unrestricted" means "don't impose a budget of my own", not "no limit". Handy when you want to see how many characters an image really takes before deciding on a budget. The CLI has the same thing as `-unrestricted`.

- **Dithering** — None, Floyd-Steinberg, Atkinson, Diffuse (Bayer), or Levenshtein — see §6 for how each one works.
- **1:1 conversion (no char reuse)** — normally, blocks with identical bitmap patterns share a single charset entry. With this checked, every block gets its **own** charset slot — a blunt one-character-per-block copy with no deduplication, so the character map is simply sequential (block *N* uses character *N*). Useful when the charset will be post-processed or animated per screen cell and shared characters would get in the way. The image's block count must fit within **Max chars**, otherwise the conversion stops with an error.
- **Convert** — runs the conversion. It runs in the background, so the window stays responsive throughout; a progress bar appears below the buttons tracking the blocks processed, and disappears when the result is ready. The Convert and export buttons are disabled while a conversion is in flight.
- **Export C64 executable** — see §8.
- **Preferences** — see §4.5.

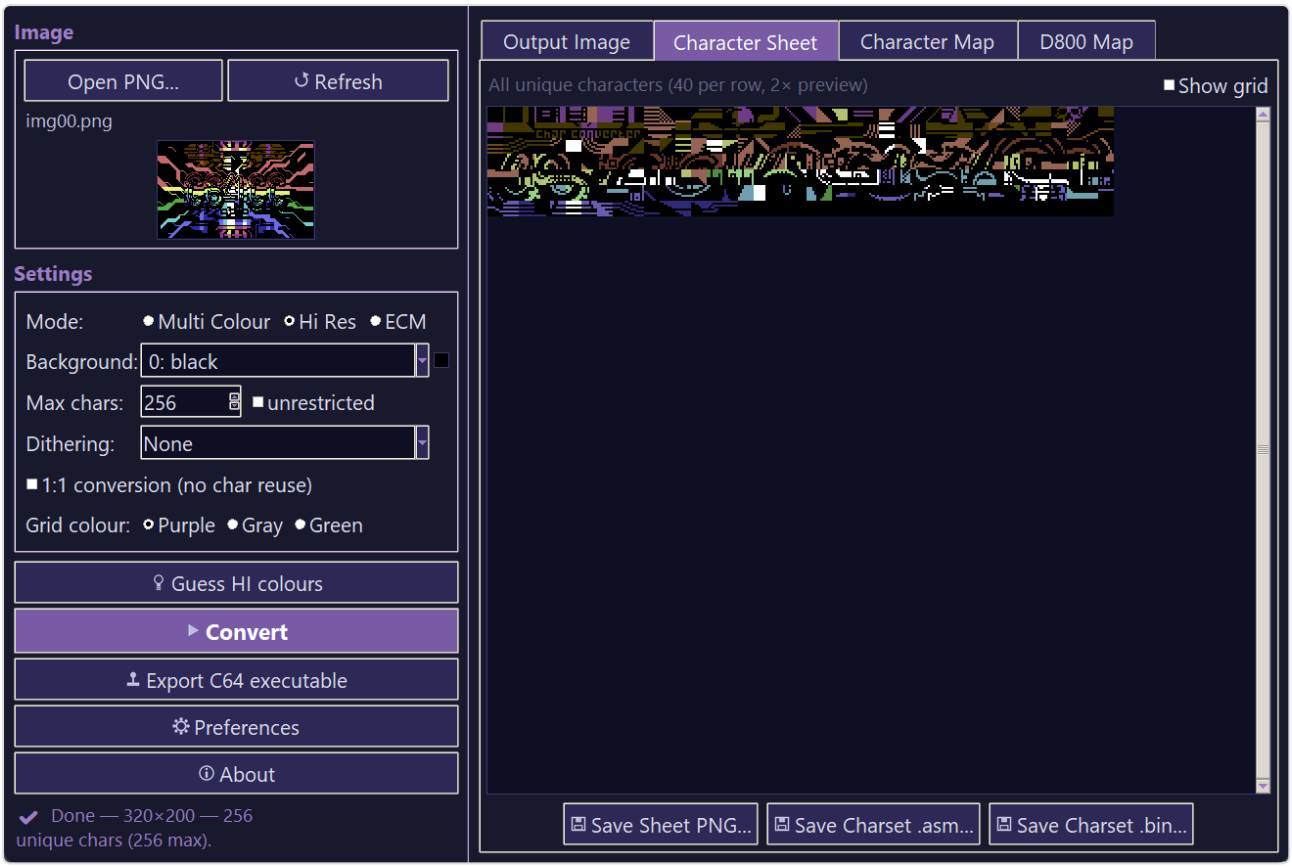
4.2 Output Image tab

- 2× zoomed preview of the reconstructed image.
- **Colour role toggles** — show/hide MC0, MC1, D800 (MC mode); FG (HI mode); FG, BG1–BG3 (ECM mode) to inspect individual colour layers.
- **Show Grid** checkbox + grid colour (Purple / Gray / Green).
- **Crop toolbar** — L / R / T / B spinboxes (in block units), plus:
- **Apply crop** — commits the crop; affects the Character Sheet, Character Map, D800 Map tabs, and all exports (see below for how it behaves with the C64 executable export).
- **Reset crop** — restores the full image.
- The Output Image preview itself always shows the full image; a dimmed overlay marks the excluded region.
- **Hover tooltip** — hovering a block shows a 4× zoomed preview plus its colour swatches, with the block highlighted on the canvas.
- **Save Output PNG…**



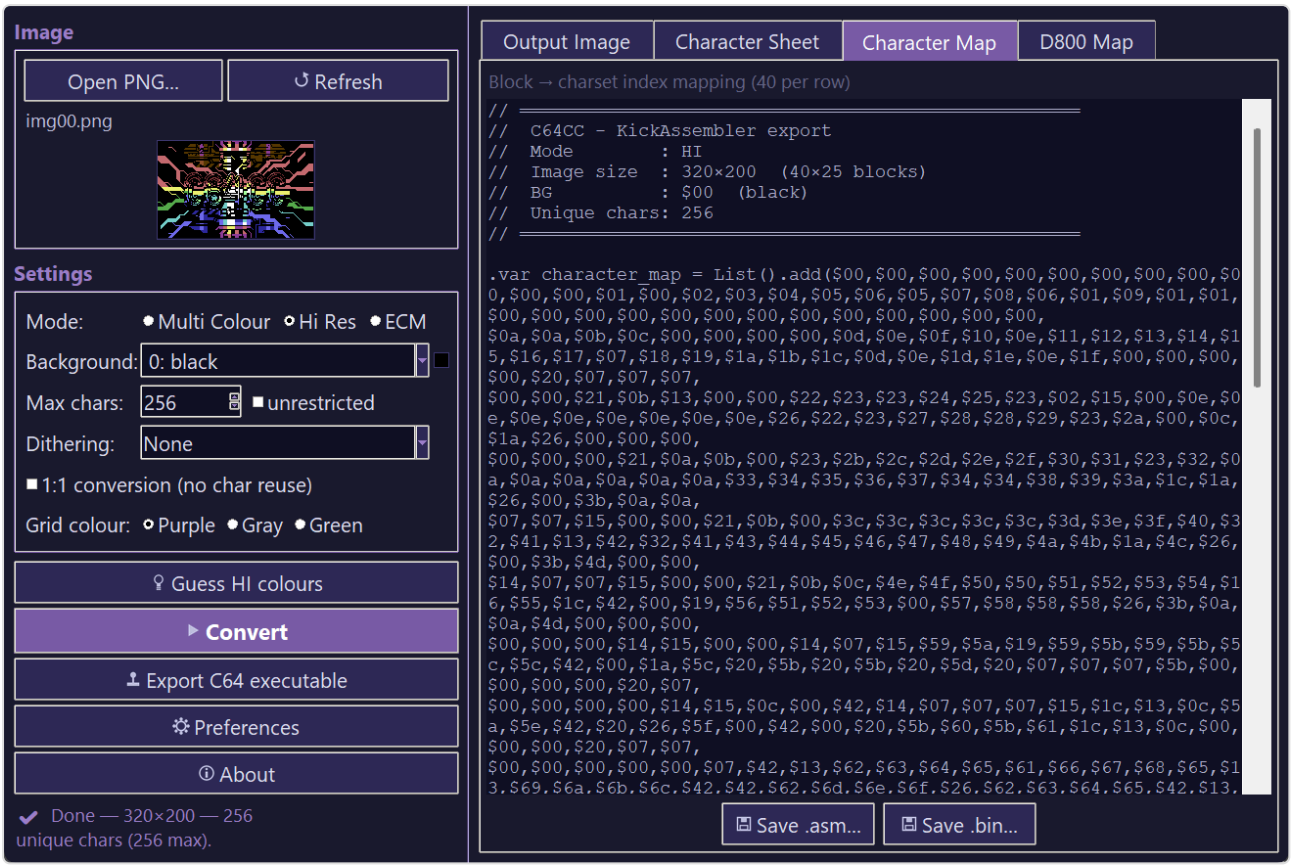
4.3 Character Sheet tab

- Scrollable 2× preview, 40 characters per row; reflects the current crop.
- **Save Sheet PNG...** (inline layout respected per preference)
- **Save Charset .asm** (KickAssembler)
- **Save Charset .bin** (raw binary)



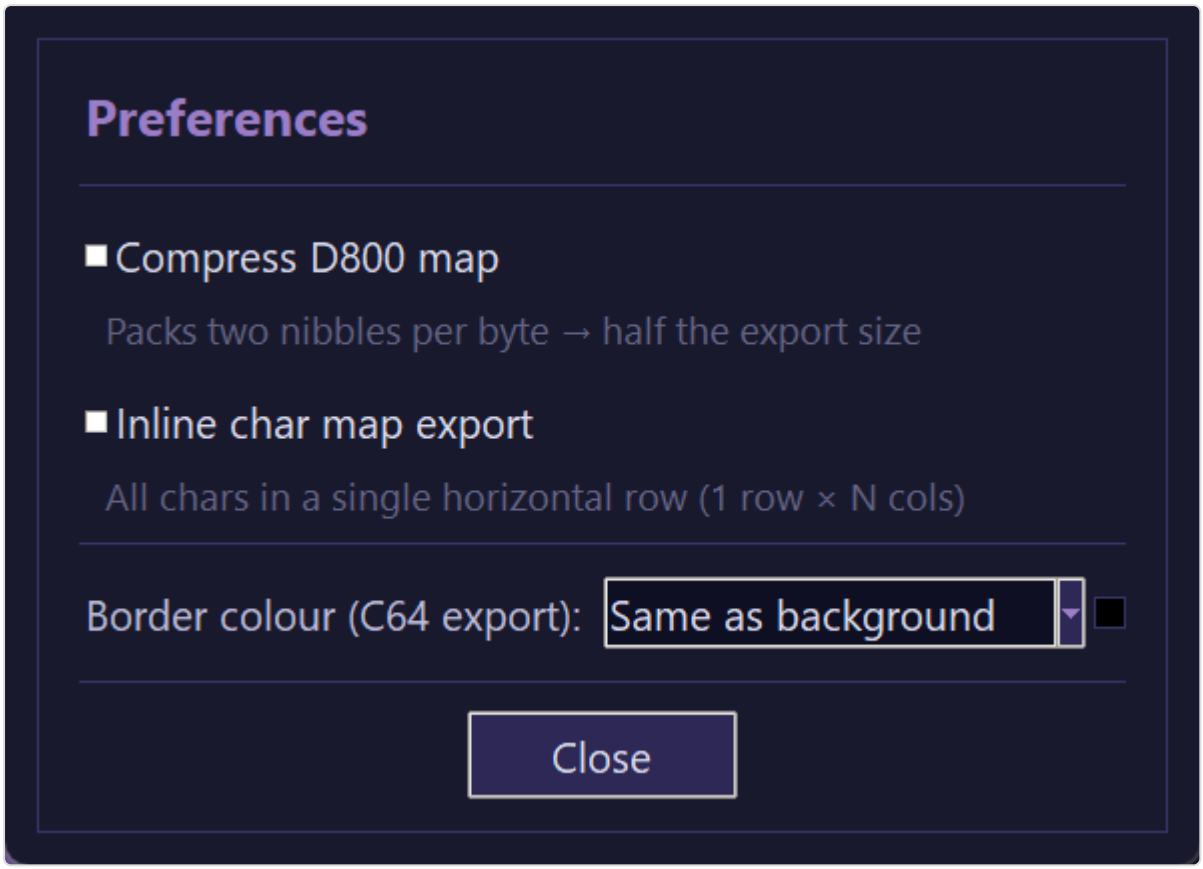
4.4 Character Map / D800 Map tabs

- KickAssembler-format text preview of the per-block character indices / colour-RAM values; both reflect the current crop.
- **Save .asm** and **Save .bin** for each.



4.5 Preferences

- **Compress D800 map** — nibble-packs the D800 map to half its size for the standalone `.asm` / `.bin` exports (never applied to the `.prg`, which always uses the uncompressed 1000-byte colour RAM layout).
- **Border colour** — used by the C64 export and the bordered preview; default is "Same as background".
- **Inline char map export** — export the character sheet as a single row.



---

5. How colour guessing works

Guessing is a distinct, explicit step — it never runs on its own:

- **GUI:** click the mode's **Guess colours** button. It only fills in the global colour dropdowns; it does **not** convert the image.
- **CLI:** runs automatically before conversion when `-g true` (the default); skipped entirely when `-g false`, in which case `-bg` / `-mc0` / `-mc1` are used as given.

There are two separate stages, and it's worth keeping them apart:

1. **Importing / guessing (this section)** — analyses the *whole* image once to pick the **shared, global** colours only: BG/MC0/MC1 for MC, BG for Hi Res, or BG0–BG3 for ECM. Nothing per-block is decided yet.
2. **Converting (§5.4)** — once the global colours are fixed (guessed or hand-picked), pressing Convert builds the actual charset: each 8×8 block gets its own best per-block colour (D800 for MC, FG for HI/ECM) on top of whatever global colours are currently set. Convert never re-runs the guesser itself — it just uses whatever is in the colour dropdowns / CLI flags at that moment.

5.1 Multi Colour guess

- Every pixel is snapped to its nearest of the 16 C64 palette colours (§3).

- For each 8×8 block, the set of distinct colours it contains is recorded.
- Every combination of 1–3 colours that actually appear anywhere in the image is tried as a candidate (BG, MC0, MC1) triple.
- A triple is scored by how many blocks it can represent: a block counts as valid if every pixel belongs to the triple, or if exactly one colour is left over *and* that colour is in the 0–7 range — the only values a per-block D800 colour can hold before the MC-enable bit is added (see §10).
- The highest-scoring triple wins; its hit rate is shown in the status bar, e.g. "BG=0 MC0=2 MC1=10 — 980/1000 blocks valid (98%)".

5.2 Hi Res guess

- Pixels are snapped to the palette as above.
- BG is simply the single most frequent palette colour across the whole image, by raw pixel count.
- The reported count is the number of blocks that already have ≤2 distinct colours — Hi Res can only represent BG plus one FG per block, so blocks with more than 2 colours will lose detail regardless of dithering. **Read it as "how well does this image suit Hi Res", not as a mark for the guess**: it is measured from the source image alone and does not change with the background picked. A low number means the artwork is busy for Hi Res, not that the guess went wrong.
- FG itself isn't decided here — it's chosen independently per block during conversion.

5.3 ECM guess

- Pixels are snapped to the palette as above.
- The 4 most frequent palette colours **by block presence** (how many blocks contain that colour at least once, not raw pixel count) become BG0–BG3.
- The reported score is the number of blocks whose pixels all belong to one of those 4 colours plus at most one extra colour — that extra colour becomes the block's FG (and its active background slot) during conversion.

5.4 What happens during Convert (per block)

Once the shared colours are fixed, Convert decides each block's own colour(s):

- **Hi Res** — a quick full-palette nearest-neighbour pass picks the block's most common non-BG colour as FG, then the block is re-dithered with the selected dithering method, constrained to just {BG, FG}.
- **ECM** — each of the 4 global backgrounds is tried in turn (each paired with its own best FG), and the BG/FG pair with the lowest reconstruction error is kept before the final dither pass.
- **MC** — the D800 colour (0–7, excluding whichever of BG/MC0/MC1 already overlap it) that minimises the block's quantisation error is picked, then the block is dithered constrained to

{BG, MC0, MC1, D800}. If the chosen D800 colour ends up unused after dithering, 0 is stored instead — better compression, no visual difference.

---

6. How dithering works

The **Dithering** dropdown controls how a block's pixels are quantised down to the small set of colours actually allowed for that block (e.g. {BG, FG} for Hi Res, {BG, MC0, MC1, D800} for MC). Each 8×8 block is dithered **independently** — error never carries over from one character to the next, since each block is its own self-contained bitmap on real C64 hardware.

Four of the five options change *how a pixel's colour is chosen*; the fifth (Levenshtein) doesn't touch pixel quantisation at all — see §6.5.

Option	Best for
None	flat, clean artwork; fastest and most predictable
Floyd-Steinberg	photos and smooth shading
Atkinson	sprites and line art; keeps contrast
Diffuse	retro-looking, evenly textured art
Levenshtein	busy images pushed against the character limit

6.1 None (nearest neighbour)

Each pixel simply takes the closest allowed colour, with no correction. Fast and predictable, but gradients break into visible bands.

6.2 Floyd-Steinberg

The classic error-diffusion method: the colour accuracy lost on each pixel is passed on to its neighbours, so the average colour over an area stays close to the original. Usually the best choice for photographic or smoothly shaded sources. Very flat colours can pick up a faint diagonal streak.

6.3 Atkinson

Error diffusion too (the method the original Macintosh used), but it deliberately throws part of the error away instead of passing all of it on. Slightly less colour-accurate than Floyd-Steinberg, though flat areas stay cleaner and contrast holds up better — a good pick for sprite-like or line art.

6.4 Diffuse (Bayer / ordered dithering)

Applies a fixed 8×8 pattern rather than passing error between pixels. That pattern is exactly one character wide, so it repeats identically on every block and gives a regular cross-hatch texture

instead of organic noise. This is close to how C64 art was dithered by hand, so it often looks the most authentic; less accurate than Floyd-Steinberg on smooth gradients.

6.5 Levenshtein

This one doesn't affect pixel colours at all — there it behaves exactly like **None**. It only changes what happens once the character budget (**Max chars**) runs out and new blocks have to reuse existing characters: instead of reusing the character that looks closest in colour, it reuses the one whose pattern is closest in *shape*.

Below the character cap it makes no difference at all. Worth trying when a busy image is heavily character-limited and you would rather keep the shape of repeated patterns than their exact colour.

---

7. Cropping — behaviour by export type

Setting a crop (L/R/T/B, in block units) and clicking **Apply crop** affects exports differently depending on the target:

- **Charset sheet, .asm/.bin map/D800 exports, Output PNG:** the crop *extracts and repositions* — only the selected block rectangle is kept, and it becomes the new top-left origin of the exported data (optional left/top padding blocks, via the pad spinboxes, can add blank rows/ columns before it).
- **C64 executable (.prg):** the crop behaves differently — see below.

When the source image changes. A crop is expressed in block coordinates, so it can only be trusted against the image it was drawn on:

- **Loading a new image** (Open PNG..., or drag & drop) clears the crop entirely — coordinates from the previous image mean nothing in the new one.
- **Refresh** keeps the crop if it still fits. If the file changed size on disk, the crop is clamped to the new block grid instead, and dropped altogether if clamping leaves it covering the whole image.

Either way a crop can never silently trim an image it was never drawn on, while the edit-and-refresh workflow keeps a crop you set on purpose.

---

8. C64 executable (.prg) export

Available for all three modes. Requirements:

- Image no larger than 320×200 (any size up to that is fine) — this is the size of a real C64 screen, so there is nowhere to put the extra blocks. For a larger image the GUI greys out **Export C64 executable** and the status bar says why; on the CLI, `-c64` reports an error while the other exports still run.
- ≤ 256 unique characters — a `.prg` runs off a single charset. Over that, the GUI greys the button out and the CLI's `-c64` reports an error; **all the other exports still work** (see §8.3).

8.1 Images smaller than 320×200

The C64 screen is always a fixed 40×25 character grid. If the source image is smaller, its block grid is placed at the **top-left** of that 40×25 grid; every remaining screen cell is pointed at a genuinely blank (all-zero bitmap) character, so it renders as flat background colour. This filler character is a real, otherwise-unused charset slot (or an existing all-zero pattern if the charset is completely full) — never character index 0, since that slot has no guarantee of being blank.

8.2 Crop + C64 executable

Unlike the other export types, applying a crop **does not reposition** the image for the `.prg` export: the kept block rectangle stays exactly where it was on the 320×200 screen, and every block *outside* the crop rectangle is zero-filled (same guaranteed-blank-character mechanism as §8.1) rather than being cut out and shifted to the top-left. Left/top padding (the pad spinboxes) is ignored for this export, since it only makes sense for the repositioning exports.

This means you can use crop purely as an "erase everything outside this region" tool for the C64 executable, without disturbing where the remaining picture appears on screen.

8.3 More than 256 characters

Max chars (`-nchars`) goes up to **2048** — eight C64 charsets of 256. Asking for more stops with an error rather than producing something unusable.

Above 256 characters, only the C64 executable becomes unavailable. Every other export keeps working, with one change to the binary character map:

Charset size	Binary char map
≤ 256 characters	1 byte per block, as before
> 256 characters	2 bytes per block, little-endian

In the 2-byte form the **low byte is the character** within its charset and the **high byte is which charset** it lives in (0–7), so a coder can split it straight into a charset bank plus a character index. The GUI status bar and the CLI both say when this wider form is being written, so it can't happen silently. The KickAssembler `.asm` map is unaffected — it always writes plain values (`$1ff` and so on).

---

9. CLI usage

Any argument triggers CLI mode:

```
<app> -i <input.png> [options]
```

`<app>` is the executable for your platform — on **Windows** that's `C64CC-cli.exe` (the console build; `C64CC.exe` is the GUI one and has no console to print to), on **macOS** `cli/C64CC-cli` from the zip, and on **Linux** the single `C64CC` binary, which serves both roles.

The `C64CC-cli` builds never open a window — run one with no arguments and it prints this help. (On Linux, where one binary does both, no arguments opens the GUI instead.)

Flag	Description
`-i <path>`	Input PNG file (required)
`-s <mode>`	Screen mode: `MC` | `HI` | `ECM` (default: `MC`)
`-g <bool>`	Guess colours: `true` | `false` (default: `true`) — see §5
`-bg <0-15>`	Background colour index (ignored if `-g true`)
`-mc0 <0-15>`	MC0 colour index (ignored if `-g true`)
`-mc1 <0-15>`	MC1 colour index (ignored if `-g true`)
`-nchars <n>`	Max unique characters, 1–2048 (default: 256, or 64 for ECM). Above 256 the char map is written 2 bytes per block and `-c64` is unavailable — see §8.3
`-unrestricted`	Use as many characters as the image needs (i.e. `-nchars 2048`; ECM still caps at 64). Overrides `-nchars`. Same as the GUI's **unrestricted** tickbox
`-d <method>`	Dithering: `none` | `floyd` | `atkinson` | `diffuse` | `levenshtein` (default: `none`) — see §6
`-f <fmt>`	Export format: `b` (binary) | `t` (text/KickAssembler) (default: `b`)
`-o <path>`	Output path prefix (default: same as input)
`-crop <L,R,T,B>`	Crop to block range (0-indexed, e.g. `2,37,1,23`). For `-c64`, cropped-out blocks are zero-filled in place rather than repositioning the image.
`-1to1`	1:1 conversion — no character reuse: every block gets its own charset slot even if its pattern duplicates an earlier one, so the character map is sequential. The image's block count must fit in `-nchars`.
`-inline`	Inline char map export (one row)
`-compress`	Compress D800 map (pack 2 nibbles per byte)
`-c64`	Export C64 executable (`.prg`) — needs an image ≤ 320×200 and ≤ 256 chars, see §8

Flag	Description
`-c64-border <n>`	Border colour index for C64 export (default: same as BG)
`-img`	Export output image and character sheet as PNG
`-h`	Show help

Exported files

(`<prefix>` = output path or input filename without extension)

File	Produced when
`<prefix>_charset.bin` / `.asm`	Always
`<prefix>_charmap.bin` / `.asm`	Always
`<prefix>_d800.bin` / `.asm`	Always
`<prefix>.prg`	`-c64`
`<prefix>_output.png`	`-img`
`<prefix>_sheet.png`	`-img`

Examples

```
# Convert to MC mode with auto colour guessing, export a .prg
<app> -i photo.png -s MC -c64

# Hi Res, explicit colours, Floyd-Steinberg dithering, KickAssembler output
<app> -i logo.png -s HI -g false -bg 0 -d floyd -f t

# Crop before exporting the C64 executable — kept region stays in place,
# everything outside blocks (5,3)-(34,21) is zero-filled
<app> -i photo.png -c64 -crop 5,34,3,21
```

---

10. Colour-RAM (D800) encoding notes

- Stored internally as 0–7; exported with **+8** added for MC mode (bit 3 is the MC-enable flag in real C64 colour RAM). HI/ECM export the raw palette index with no offset.
- `-compress` / the "Compress D800 map" preference nibble-packs two values per byte for the standalone export files — this is **never** applied to the `.prg`, which always writes the full uncompressed 1000-byte colour RAM layout (required by the hardware).