

U L T R A L O A D E R

ULTRA LOADER TECHNICAL MANUAL

*A complete reference to the loader, all settings, the input services,
per game firmware signing and protection, troubleshooting tips and the arcade boards it supports.*

Technical Manual · Resource Guide · Settings Descriptions

V1.0.0

Contents

About Ultra Loader.....	3
Ultra Loader's Three Mechanisms.....	6
The Ultra "Omni" Input Services.....	6
Installation and Folder Layout.....	7
Quick Start.....	8
Reading the Log.....	8
The Main UI Window.....	9
Display and Rendering Options.....	10
Audio Settings	12
Game Settings	13
Controllers and Modules.....	14
Configure Controls Dialog (Chapter 8 Cont'd).....	18
The Omni Input Services (In Detail).....	19
Ultra Omni-Play (gamepad).....	19
Ultra Omni-Steer (flight stick).....	20
Ultra Omni-Wheel (racing wheel).....	21
Ultra Omni-Track (lightgun).....	22
Ultra Omni-Bridge (keyboard/mouse).....	24
Credits and Coins.....	25
HOTKEY & BASE COIN	25
AUTO-INSERT ON GAME BOOT.....	26
ON-SCREEN COIN PILL.....	26
Boards and Systems.....	27
Module Signing and Protection.....	29
Ultra.ini Reference (The Settings File).....	30
Environment Variables (For the devs).....	35
Troubleshooting.....	40

About Ultra Loader

What Ultra Loader is, what it runs, its overall purpose, and the concept of native execution.

To begin, Ultra Loader is, at its core, a *native* launcher for modern arcade games. This is, first and foremost, a very *important* distinction. Ultra Loader was not built to emulate in a standard emulator sense. It was built to run modern arcade games in their native environment, running them as if they were built for the machine, in a clean and optimized setting, as if it were sitting in its original home cabinet. This distinction alone, separates Ultra Loader from mostly everything that has come before it.

Arcade launchers like tekNOParrot and a few others, devise mechanisms to play many of these games at the surface level; giving the game what it needs to run through modified profiles and other tactics that mostly *do work*, but Ultra Loader's style and premise is not to do this. For all intents and purposes, the loader does things the *hard* way, and when I say the *hard* way, I truly mean it. To get some of these modern arcade games to run, I had to literally go function by function, and bridge the gap between the game and the running environment, all to keep to the concept of *native arcade execution*.

To be fair, the Linux Loader (formerly Lindbergh Loader) guys are seeking to do the same thing as well, and knowing this, I would happily love to combine efforts if our paths do cross one day. I have already personally figured out Lindbergh emulation in Windows, but would love to combine our work to the betterment of both projects, as I find their goal of preserving native Lindbergh cabinets to be admirable. I think this is personally the right way to do things at this point, and this is truly the only way that these games can run and play in their original form and be made available for the whole community.

Over time, we will also discuss more of the underlying motivations behind Ultra Loader (and there is certainly a lot), but I wanted to instead talk more about what it does. So at the bare minimum, Ultra Loader takes a game that was written for a specific arcade board, let's say, Taito Type X, or Sega Lindbergh, or Namco ES3 and so on, and gets it running on an ordinary PC, supplying the pieces of the cabinet that are missing: the I/O board, the coin mechanism, the dongle or key chip, the display mode, and of course, the controls. This is fairly easy to do if you know what you're doing.

However, what is not easy, (as some boards will quickly show you), is the concept of native emulation for games that were *not designed* for ordinary PC's, such as games designed for specific custom hardware setups, or perhaps games that run in other operating systems. One such popular board, is one we already mentioned above, which is Sega Lindbergh. Lindbergh was built to run in Linux, which is why the Linux Loader guys have been so successful in getting it to run rather smoothly, and any frontend systems that use Linux also benefit from running their core as well, such as Batocera, etc.

However, the challenge is getting the loader to run properly in Windows, which has proven to be, for a very long time, a hassle. There are many differences in Linux vs Windows, and it is not so simple to emulate Lindbergh in Windows, much less so, to *natively execute* linux ELF's. However, I have done it. With a lot of painstaking work over many nights, I have figured out how to natively load linux ELF's in Windows. The process was long and tedious, but it does work, and in truth, I would not have released Ultra Loader if I couldn't figure it out, as this was also the key to unlocking other cool games that run only on Linux.

Thus I was able to keep to the original premise of native execution, even for boards that do not work natively in Windows or on common PC hardware. Ultra Loader, in this sense, is very strict. But this is good, as it is a big leap and jump for the community. Prior to this, all we had were certain types of pseudo hacks and compatibility layers in order to get these games working. But now we don't need any of that. All we now need, is time and community dev work on fine tuning per game code in the board modules, so that each game runs clean and fast. As it stands now, Ultra Loader launches all games natively, using its custom-built firmware modules without any other barriers, and the loader even uses the accompanying software it was designed to use. What I mean by this, is essentially the concept that some games need special bypasses to make them believe they are talking to their parent servers that unfortunately no longer exist. Thus I had to take their software, study it and reverse engineer it to

insert code to make the game behave as if it is still connected. The common term for this is “spoofing”. We can talk about that later when we have discussions on the technical details in real-time.

Now as stated above, Ultra Loader is not an emulator in the traditional sense, and I want this to be clear. If you define an emulator as a piece of software that emulates the function of the game by replicating its code in clean-room concepts (much of what MAME does) and runs a rom or software package, then perhaps a case can be made to say that this is then, the same thing, and therefore I am only arguing semantics. However, as I stated before, the mechanism of Ultra Loader is true native execution, and especially for the PC-based arcade boards, many of the games are normal Windows executables, and Ultra Loader runs it directly, injecting a small hook that stands in for the hardware the game expects to find. This is, for all intents and purposes, ***Native Execution***. This is the concept behind the loader’s full mechanism, and for all games that run on Ultra Loader, this is the standard and bar it must pass. For older non-PC based boards, it hosts a core instead, but both paths are driven from the same window and the same settings. The reason I chose to develop the loader in this way, was to establish a baseline of clean-room execution (which is fully defensible in court if it comes down to it) and to bypass all the myriad workarounds and hacks the community has had to deal with and endure.

People just want things to be easy to use, and to simply *work*. Anyone who plays these games know of the horrors of modern arcade emulation setups, wrestling with controls and drivers and accompanying warez and all that kind of stuff, just to get a single game to boot...which may very well still break later. Teknoparrot’s launcher suffers greatly from this. The setup is no less than hell, and a game that works today may not work tomorrow. Indeed, many pieces of software can be accused of this at one time or another in this space, but it need not be this way. We can certainly do better. Then coupled onto that, we have the wonderfully glorious concept of *gatekeeping*, which you will hear me talk a lot about. Quite a few of these games are indeed available and can be played, but of course we have to talk about and discuss the teknoelephant in the room. Lol, *teknoelephant*.

This, I refer to, is the decade-long establishment of the false gods. What has been happening with the community over the last ten years has not been good. The ones who run teknoparrot have profited off of the community in ways it should not have. Say what you will of capitalism and its claims, you should not gatekeep games you do not own. This is simple and clear. I am also a business-minded person, and I currently have a few side hustles going on, all in addition to my studies and my professional career, and I even have quite a few ideas, some of which I am now currently developing, to make money and be successful, and this is what everyone should aim to do if that is your dream and goal. But it cannot be done by profiting off of, and gatekeeping games that people want to play, especially when you did not create them. These games were developed for arcades, where people can play them with a coin or a ticket. How then, have we allowed one person to monopolize this right under our noses for so long, and we have done little to nothing about it? As it is right now, to play some of these games, you must pay someone who did not make them. This is fundamentally wrong and can *never* be justified. It may not be an immensely large amount of games, but over time, the method itself has now become something that the community has rightfully developed ill-will over, and remains a point of contention for many people. Inevitably, it was bound to happen that *someone* would make Ultra Loader.

If, you are like some, who actively support and defend teknoparrot’s gatekeeping methods, then you might argue that *someone* had to do it, and in spite of their gatekeeping and profiteering, it funded the development of methods to play these games, and thus it was a necessary evil. But that is a false gods lie. Look at myself. I needed no funding or even access to these physical boards in order to develop Ultra Loader, and as stated before, the modules are entirely clean-room, and I did not take source from any other project. This was one of my hard-set rules, just so I could prove that it could be done. All I used was the documented board info and specs available to the public, submitted by hardware preservationists who loved tinkering and playing around with technology, and then I did a massive deep dive into the world of emulation and simply reverse-engineered the necessary components. This could have been easily done by anyone. Therefore, the argument that we needed the false gods to do this at the community’s expense was all a lie. Many do not know this, but the one called Reaver (the head/owner/creator of teknoparrot) actually took founding work from someone else, and used that to develop the early days of his “arcade drive loader” that launched some of these games in the early days.

The original work was not his. So even then, no one can make the case that they should have done things how they did it, and they had a *right* to charge. And even if that *was* the case, and we needed to pay or “fund” people to do this work, then why not state it as such, and let the community choose a few well-seasoned, and well-trusted devs to explore and dig into these boards, and publish their results to the community at-large? The community would have happily funded this pursuit, as long as everyone benefited.

A perfect example of this, is the amazing guy that goes by the now famous moniker, “Glorious Eggroll”. For those who don’t know him, his work with the Proton Compatibility layer, forked specifically to enhance compatibility with many PC games has earned him legendary status, and people happily send him funds and support him, so much so, that he now does it as his full time job, and all his findings go downstream directly to the community with ZERO gatekeeping. So the model works, and could have been implemented for modern arcade games the same way. But no, greed was allowed to hold sway, and many people simply went along with it, as they felt they had no other choice. This, in a sense, is what we allowed, and we shouldn’t have.

However, I won’t wallow in the past any more than that, because people genuinely felt that these things were a mystery, and only the oldest and wisest amongst us could do it. We truly felt there was no other option. But, as of this new release, that has all changed, and now finally we have another viable and sustainable option, and one that will last. I am just sad that it took ten years for someone to finally build something, and my sorrow is not that we now have an alternative, but that if we had only come together back then and established a better way, shunning this greedy practice we let go on for so long, we would be so much farther along by now. I think that if we had done things differently back then, we would have modern arcade emulation as easy and as smooth as Genesis or SNES emulation today.

Hence my personal stance is this. No one should have the sole authority to gatekeep games to the public, and only a game developer should have that right, as it’s their game. I will not drone on about this much longer at this present moment, as we need to get back to the manual, but suffice it to say, the community is eager, ready and calling for a change. Ultra Loader is that change. Ultra Loader will be free, and will always be available to the community. The work done for the loader will benefit all. No one will be charged to play a game, and the only thing preventing a popular game from being played will be the speed at which we can get them running natively in their custom arcade environments. The amazing work that my predecessors did, like the folks who developed Jconfig and the many devs on places like Emuline and other sites, along with the work that many others have put in, have been invaluable to the community, and this has allowed people to play these amazing games outside of teknoparrot’s domain.

I am, no, *we* are all deeply grateful for this.

Now while I did not take from Jconfig or any other source (all firmware modules were built from scratch) it is because of their separate work that I was confident enough to do this. I simply told myself that if they could, then I could at least try, and look what resulted from it. I am also fully aware that Jconfig also has elements of openparrot work, but the fact remains. Ultra Loader has chosen to do things differently, and I am proud to say that the games that are currently supported, run natively, smoothly and fast, with high levels of stability. The loader also offers convenience to the standard of what even a professional would need for their arcade cabinets, and I am very proud of the work that I have done. It’s truly time for a change. The community wants an easy to use tool that just works and won’t cost several nights of blood, tears and other bodily fluids just to play a single game. We can do better, and I am happy to be a part of this community. I truly hope you enjoy Ultra Loader, and that you come along with me, as we make this thing even more awesome, and more enjoyable and accessible for everyone. First and foremost, and before anything else, this was and remains my *ultimate* goal.

Now my friends, without further ado, let’s dive into the manual.

Ultra Loader's Three Mechanisms

PIECE	WHAT IT IS
The loader itself	The program you interact with: ultra-loader.exe. It picks a game, works out which board that game belongs to, applies your settings as environment variables, and hands off to a module.
A custom game firmware module	One DLL per arcade board, installed under %LOCALAPPDATA%\Ultra Loader\modules\. It launches the game suspended, injects the board's hook, and waits. Each board owns its own module and its own hook source, so work on one board cannot disturb another.
A custom arcade hook (packed with the firmware)	A DLL injected into the running game. It answers the game's requests for JVS I/O, DirectInput, DirectSound, Direct3D and the dongle, essentially all of the hardware that isn't there.

The Ultra “Omni” Input Services

This is by far my most favorite feature of Ultra Loader, and I loved creating it. There are five “omni” services that run inside the loader and publish a single, shared picture of what the player is doing. Every game's hook reads from that same shared block, which is why one controller setup applies everywhere rather than having to be repeated per game. This is the *solution* to per game setups.

Furthermore, each service comes with a smart “auto-calibration” setup, that handles mostly all controllers and gamepads and other HID compliant USB or Bluetooth gameplay devices. I built it specifically to help users simply get into the game and play. No more complicated setups. No more annoying workarounds to setup your controllers, racing wheels or lightguns. Mostly all hardware will be recognized and will work, even *clone* controllers, lol. The ultra omni services does it all for you.

The five services are outlined below:

SERVICE	HANDLES
<i>Ultra Omni-Play</i>	All Gamepads, Arcade Sticks, Xinput and DirectInput, up to ten slots.
<i>Ultra Omni-Steer</i>	Flight sticks and HOTAS setups, including throttle and rudder.
<i>Ultra Omni-Wheel</i>	Racing wheels, Racing pedals and shifters, with full force feedback.
<i>Ultra Omni-Track</i>	Lightguns, plus the full aiming and calibration that goes with them.
<i>Ultra Omni-Bridge</i>	Keyboards and Mouses. Many games only work with HID peripherals.

As mentioned above, these services auto-calibrate almost *everything*. This includes all types of Zero Delay arcade setups and mostly all arcade peripherals and devices. Within the Ultra Omni-Track Lightgun service especially, you will find some super useful options in there, that will effectively eliminate 99% of the problems people currently face when using Lightguns and zero delay encoders for many games. I won't spoil the surprise though. You will have to dive in and see for yourself...

Personal Notes:

The goal here was to effectively eliminate the nightmare of these types of loaders having insufficient or inadequate controls that barely work. The five Omni services cover almost ANYTHING a player would need in order to play mostly any modern arcade game, including in a 2-player or 4-player arcade cabinet. Also, keep in mind, that anything you set in the main window is a global default. But anything a board module sets is a board default, but it gives way to an explicit setting of yours from the main window. If a game behaves differently from what the main window says, a board default override is usually the reason, and a quick look at the ultra.log should tell you everything.

Installation and Folder Layout

Where everything lives in the main folder and on the backend.

PATH	WHAT IT HOLDS
ultra-loader.exe	The loader itself. Self-contained.
art\, fonts\	Splash art and the fonts used by the boot console. Ultra Loader will bundle the fonts used for the loader. As for the art folder, Ultra Loader will also bundle all supported games logos in a .pak file and you can then install them directly from the main ui menu, and each logo will appear per game. You may also source your own logos if you prefer. Make sure your custom logos are labelled correctly per game, then copy them into the art folder by system. For Ex: <i>Ultra Loader/art/Namco ES3/Mario_Kart_GP_DX.png</i>
ultra.ini	Sits in the same folder as ultra-loader.exe. It contains your settings. <i>Chapter 12</i> lists every key.
ultra.log	Also sits in the same folder as ultra-loader.exe. It shows what happened on the last run. This is the first thing to read when something does not work.
%LOCALAPPDATA%\Ultra Loader\modules\	Installed board modules, one folder each, plus their signatures and hooks.
%LOCALAPPDATA%\Ultra Loader\gamedata\	Per-game saved data the loader keeps out of the game folder.

Your game dumps are never written to, or modified, ever.

*Ultra Loader treats game folders as read-only source material. This is to ensure all dumps keep their full integrity, and also detach reliance on internal file modification. Where a game genuinely must be modified, such as a language switch in its **own** config, the original is copied alongside as <file>.ultra-lang-bak before the first edit, and the change is always reversible.*

Quick Start

Getting one game running, and knowing whether it worked.

- 1. Browse to your game collection, and select a game.** Use the game selector in the main window by hitting “Browse” and navigating to your folder where you hold your arcade game dumps, and search for and choose the actual game executable, (game.exe) not a launcher bat script or a shortcut. Shortcuts may or may not work, so prefer you to use the actual executable. Make sure the matching ultra firmware files are copied into the firmware folder by system, so Ultra Loader can find them.
- 2. It will quickly identify the board. (on the backend)** Ultra Loader reads the path and the executable from the firmware to decide which board the game belongs to, and it will say so in the log as [PREFLIGHT]. It can never guess wrong, as the firmware file contains the game’s board information. So even if you were to mistakenly mislabel a game’s firmware file or put it in another arcade system’s folder, it will still find the correct board. This is done in the background, so you won’t have to do anything here.
- 3. 3. Check or modify your display settings.** Most arcade games were written for a fixed resolution. See Chapter 5 for what the game upscaler does, and when to leave it alone for best results.
- 4. Launch the game.** Hit “Save & Launch” near the bottom of the UI to launch the game. The Ultra Loader blue init screen appears first, then the game’s boot console, (if any) and then the game should load. For most games, this is almost instant.

Reading the Log

When a game does not behave, the *ultra.log* answers most questions on its own. These are the lines worth knowing:

LOG LINE	MEANING
[PREFLIGHT] auto-detected system: X	The board Ultra Loader decided on. If this is wrong, nothing after it will be right.
[SIGCHECK] X: signature OK	The firmware module passed its signature check and is allowed to load.
[DISPATCH] X -> ...dll	The module that actually ran.
[FALLBACK] N candidate module(s)	The first-choice module could not load, so others were tried.
game exited with code 0	A clean exit.
game exited with code 3221225477	0xC0000005 an access violation. The game crashed; it did not quit. This is important for troubleshooting.

The Main UI Window

Choosing a game, and the main window buttons.

OPTION	WHAT IT DOES
Browse... ID_BROWSE	Pick the game file to launch.
Game Library ID_LIBRARY	Open the Game Library: a visual grid of your games' artwork. Double-click a tile to launch it.
Scan folder for games ID_SCAN_GAMES	Scan a folder for supported games and pick from the list.
Save & Launch ID_LAUNCH	Save all settings and launch the selected game now. If that game has an Ultra Firmware container, the settings it carries are applied over yours just before launch, so a value the container pins stays pinned no matter what you set here. Your own ultra.ini is not rewritten - the override lives only in the running game, so anything the container does not pin is still yours. Containers are only consulted for the native PC arcade boards.
Save ID_SAVE	Save all settings to ultra.ini without launching the game. You should get a confirmation that the setting was saved to ultra.ini
Cancel ID_CANCEL	Discard changes and close. Warning: This will exit the loader.
Exit ID_EXIT	Close Ultra Loader without launching anything. Settings you have not saved will be discarded; make sure you hit “Save” first if you want to have them persist to the next launch.

Display and Rendering Options

Resolution, upscaling, frame pacing and the screen itself. Every control in the DISPLAY section.

Arcade games are fixed-resolution in a way PC games are not: many ask for exactly 640×480 and will not negotiate with you. Thus in order to render fullscreen or at a higher resolution, we must force the games to do it. The options here work specifically for that, either by asking the game for a larger buffer where it can cope, or by stretching what it produces to fill your screen.

If a game renders black, start here

The upscaler and sharpening run as an extra pass over the game's own output. On some titles and some drivers that pass produces a stable black screen while the game itself runs perfectly; audio, inputs and logic are all correct, but no pixels. If that happens, just go and turn the upscaler off and set sharpening to zero before suspecting anything else.

OPTION	WHAT IT DOES
Output resolution ID_RES	Output (window/back-buffer) resolution. The game renders at its native size and is upscaled to this. 4K looks best with FSR.
Display mode ID_DMODE	Windowed = resizable window. Borderless = full-screen window (best for alt-tab). Fullscreen is acted on by only two boards - Namco ES3 (D3D11) and Taito Type X2 - where it gives the same borderless window; on every other board it leaves the game's own display request alone, so a title that asks for an exclusive device gets one. Output resolution still matters in Windowed: boards with an upscaler resize the window to it, and the retro cores use it as their windowed size.
Upscaler ID_UPSCALE	How the image is scaled to the window. Bilinear = plain. CAS = contrast-adaptive sharpen. FSR = AMD FidelityFX (EASU+RCAS), the sharpest. (Yes, I built it into the loader. Cool, eh?)
Display fit ID_ASPECT	Stretch fills the screen (but may distort). Pillarbox keeps the 4:3 profile. Zoom fills and crops. Fit to Screen scales with no cutoff. How much of that survives depends on the board: the Lindbergh OpenGL path gets your exact choice, while the injected D3D8 and D3D9 hooks are only told keep or fill - Pillarbox becomes keep, and Stretch, Zoom and Fit to Screen all become fill - and they apply it only while rotation is running, with 90 or 270 sending keep whatever you picked. The D3D11 hooks read neither value, so on a D3D11 title this setting does nothing.
Sharpness ID_SHARP	Upscaler sharpness, 0-100. Affects CAS and FSR's RCAS pass. Higher = crisper edges, too high = ringing.
FXAA ID_AA	FXAA post anti-aliasing: smooths jagged edges cheaply. A light blur is normal.
Screen rotation ID_ROTATE	Rotate the whole display: 90/180/270. Use 270 (TATE) for vertical-monitor cabs. You can also pair this with Ultra Omni-Track's Swap X/Y for lightgun games. I know Shmups fans will appreciate this.
FPS cap ID_FPS	Frame-rate cap. 0 = uncapped. Set to your refresh (e.g. 60) to save power and reduce tearing when V-Sync is off.

OPTION	WHAT IT DOES
GL driver ID_GLDRIVER	Which OpenGL driver arcade PC titles use. Native = your GPU vendor's driver (fastest). Mesa options are portable software or hardware-accelerated alternatives; picking one downloads a small graphics pack the first time.
Frame gen ID_FRAMEGEN	Frame Generation for planned retro cores (PS3, Supermodel, Flycast, MAME, ...). Inserts blended in-between frames so a 30fps or 60fps game runs at 60/90/120/180/240fps. Costs ~1 sim-frame of latency. 2x is a safe universal pick; 3x/4x would need a high-refresh display.
V-Sync (cap to refresh) ID_VSYNC	Synchronize presentation to the monitor refresh to remove tearing. Adds up to one frame of latency. Standard for most games.
Mute splash sound ID_SPLASH	Silence the Ultra Loader boot jingle. The splash screen still shows.
Hide console window ID_CONSOLE	Hide the diagnostic console window while a game runs. Default is on. Turn off to watch the boot log.
Easy-Config (auto per-title) ID_EASY	Easy-Config auto-applies a known-good preset (resolution, skips, fixes) for each recognized title on launch.
UI font ID_FONT	Typeface for this settings UI. Custom arcade fonts ship in the fonts\ folder; Consolas and Segoe UI are built in.
Always show menu on launch ID_BOOTMENU	On = always open this menu at launch. Off = boot straight into the remembered game. You must hold the CTRL key on launch to return to the main window.
Theme ID_THEME	Color theme for the loader UI. Switches live so you can preview each one. Small feature, but fun. My favorite is Tan with Consolas font.
Cabinet Mode: Boot directly to game (hold SHIFT to see menu) ID_BOOTGAME	Cabinet/attract mode: ALWAYS launch the saved game and never show the menu. Takes precedence over the menu setting. Hold SHIFT while the loader starts to override and open this menu (escape hatch). Useful for dedicated cabinets running one title.

For some people, they will use this in an arcade cabinet in attract mode. So I created these functions, so you can simulate a real cab from plug in and turn on, directly to game. You would just automate your OS to boot directly to the desktop, then launch Ultra Loader, and then the game will launch as you saved it.

To help you visualize the settings and how to implement them: here is a brief table:

Setting: <i>Always Show Menu</i>	Setting: <i>Boot Straight to Game</i>	This Boots To:
ON	OFF	Menu (Boots to main ui window)
ON	ON	Game (SHIFT brings back menu)
OFF	ON	Game (SHIFT brings back menu)
OFF	OFF	Game (SHIFT brings back menu)

*If the “Always Show Menu” checkbox is mistakenly unticked (set to off) along with the “Boot Straight to Game” checkbox also set to off, it will force the loader to boot and launch the last known game that was played. If there is no game selected and no game ever run, it will then override and produce the game menu. This will occur until a game is either selected, or the “Boot Straight to Game” checkbox is ticked.

Audio Settings

The AUDIO section. Simple, straightforward, cool and normal.

OPTION	WHAT IT DOES
Volume (0-100) ID_VOL	Master volume, Set it from 0-100. No muss, no fuss. This controls the main game audio, but you may also change in-game audio for more loudness or quietness in the game's service menu or settings.
Menu Sounds	Small but common feature. Mostly cosmetic. As you browse the pause menu, you will hear a navigation tone and a special auditory tone for accept or enter, and one for back or cancel.
Pause audio during pause menu ID_PAUSEAUD	Silences the game audio while the in-game pause menu is open. The pause menu's audio options page shows the same switch as 'Pause in Menu', and its Save Settings rewrites that one line in ultra.ini and leaves the rest of the file alone, so a change you make in-game is what the loader reads next time it starts. Keep in mind, this only mutes the game audio during the Pause menu screen, so it isn't distracting while you make settings changes, but the arcade game will still continue running in the background. If I were to stop the game process while the pause menu remained up, this would cause memory overload issues that would crash the game. I'm working on a way to do this, but as of right now, the game continues to run in the background.

Game Settings

Per-title behavior: free play, language, attract mode and save states.

OPTION	WHAT IT DOES
Skip attract movie ID_SMOVIE	Skip the attract/intro movie middleware so games boot straight to playable. Useful for some Lindbergh titles and other boards. Also useful if you want to skip the intro movie and just go straight to the title screen or the coin insert screen.
Skip audio initialization ID_SAUDIO	Skip sound-hardware init. A troubleshooting switch for titles that crash in audio setup.
Skip JVS I/O enumeration ID_SJVS	Skip JVS I/O enumeration. Only for titles that hang when probing for arcade I/O you don't have. Leave this off normally.
Free Play (no coins needed) ID_FREEPLAY	Enable the game's native Free Play so no coins are needed to start. Many people enjoy this feature, but some games won't work with it.
Snapshots / save-states ID_SNAPSHOTS	Enable in-session save-states so you can roll back to any earlier moment in play. Requires a relaunch and a small performance cost. Also, the feature can be buggy. The loader is doing top-level memory control in games that do not support it, in order to perform snapshots, so it could potentially result in a crash. The feature works, but use at own discretion.
English (If game supports it) ID_ENGLISH	Use the game's own English text <i>if</i> it ships it. Two ways, both reversible: flip its language/region config switch in-game (Sega Rally 3, Ford Racing, Haunted Museum, Wacky Races), or load its English asset folder instead of the Japanese one (Mario Kart GP DX). Nothing is translated and no game files are modified; any config the loader does change is kept alongside as .ultra-lang-bak. If the game ships only in Japanese, then it stays in Japanese, and titles with no English at all are left completely alone. Untick to go fully back to the shipped language. Note: If you modify or patch your game with an English patch or mod, then make sure to turn this option OFF (unticked).
Enable audio ID_AUDIO	Master audio enable for the running game. If off, the game may still output sound, but the loader will not send the sound to your speakers. Check this option first if you experience no sound across all your games.
Boot to console system menu ID_BOOT_XMB	Boot the console's built-in system menu instead of the selected game. Available for Retro systems like: PS3 (XMB), PS2 (Browser), Dreamcast (BIOS menu), Chihiro (test menu), Vita (LiveArea), Xbox 360 (NXE). Non-console titles are unaffected. *Requires the matching BIOS/firmware to be installed.
Enable credit injector ID_CI_ENABLE	Universal Insert Coin: press a hotkey (or click the on-screen pill) to add a credit function to a game. Works for arcade titles across every supported system (Lindbergh, Type X, in-host libretro cores). Free Play, when turned on, silently no-ops the injector.
Configure... ID_CI_CONFIG	Open the Credit Injector sub-panel: choose the insert-coin hotkey, credits per coin, and whether to show the on-screen COIN pill. A highly requested feature for games that don't have it.

Controllers and Firmware

Why I designed the controls this way, and what per game, “ultra firmware” (.ufw) does.

For Ultra Loader, there are three distinct device software layers that decide what happens when you press a button, and each owns its own role. The Omni input services in the loader decides WHICH DEVICES EXIST, how they present, and what state they are in. The Omni services are responsible for enumerating all devices plugged in and connected to your PC, examining and confirming which gamepads, wheels, flight sticks, mouse and keyboard and lightguns are connected, along with which slot each one occupies, as well as how its coordinates are scaled and where its dead zones sit. That work is the same for every game and every board, which is why it lives in the loader and is configured universally.

As an indispensable part of Ultra Loader, and why each game runs as smoothly as it does, each game possesses a custom-built firmware “profile” or container, and that firmware container contains important runtime files and things needed to get each game working. It also decides WHAT EACH CONTROL MEANS on that game’s so called “cabinet”, and which JVS switch bit a button drives. It also decides which analog channel a pedal feeds, and the calibration window the game’s own saved settings expect. That map travels inside the signed, protected firmware container, so it belongs to that game alone and no other game can see it or use it.

The actual system board module (contained within the game’s firmware profile) is the translator between the two. It takes the container’s map and renders it onto the JVS wire the game reads. It holds no knowledge of any particular game, and that’s the whole point of it. The rule that stops each part from fighting each other, (which is common in such setups) is that where the container has an opinion, the container *always* wins, and this governs the whole input sequence, rather than part of it. The loader itself has its own mapping as a fallback for games that possess no internal controls (which is rare), and this applies only to titles that ship no map of their own.

If I had been lazy and done things the easy way, it would be chaos. No game would work properly, and it would take ten years (as the false gods did) to properly accumulate enough communal data in order to write individual profiles for each game. Honestly speaking, if I didn’t do things in this way, you would end up with the hot mess that teknoparrot currently enjoys. Controls would not work for many games, and at best, they would be unstable, as many can now attest, or would require super complicated or convoluted setups just to get a certain game running. Setups would constantly fail. Games would refuse to load, and it would be right back to square one.

This is because one game reads its buttons at the JVS positions that normal convention expects, and the next one puts START on the eighth switch and its VIEW button on the seventh, and its steering on analog on channel two rather than channel zero. This is what causes all of the problems. Each game wants things done their own way, and the solution was never to starve one and ingratiate the other. Each game is unique, and each game is right about themselves. Since we currently don’t have a universal layer for all games (which is nothing short of nigh impossible with current means) I have found that only the game can settle what it wants, and so the answer is to simply ship what the game wants for itself.

It seems that this answer was a most simple one to reach, but not so with these types of arcade games. However, experiencing failure after failure to develop a universal hook (and trust me, I tried, even losing all my hard won controls and setting me back by weeks) I realized that this is the only answer. As from the beginning, one of my main goals was to ensure that Ultra Loader was stable and could NOT break on simple, everyday setups. This was primarily the motivation and drive, and this is why all game’s controls will live in its own custom firmware, and not in the loader itself. In addition to this, my other rule (which is solved by the first) is that no dump ever gets written to. *Ever*. The arcade dump must stay clean (for preservation and future purposes), and as such, all contents and areas for

modification must be dumped to a cache from the custom firmware included for the game, and then read back from there.

This also solves many other problems as well. Now, using this method, two cabinets on the same board can never disagree or fight each other for controls. No game can grab the lightgun for itself, and then hold it, and another game complains or breaks. All games will co-exist within the loader. All games running natively and taking only from what the omni services provide for them and saying thank you after, lol. This is the way. I also wanted to unify things where I could, as many people, myself included, have found it quite difficult to access certain things when needed. So for all games, the thing I did set is that the TEST menu for all games will always be **L3** (on gamepads) and the SERVICE menu will always be **R3** on gamepads.

This will ensure that whenever you need to enter the service menu or test menu for any game to calibrate controls or fiddle around, you can always get to it. I did this because I think there should be one standard, enforced across all arcade games as much as possible, where you can reach its internals no matter where you are. My philosophy is that operator controls should not move between titles, and both are deliberate presses on a gamepad that nothing normally uses for most if not all gameplay. Note, I also set credits/coins insert as the SELECT button, and nothing else should insert a credit, unless you set it that way from within the loader.

Now, as for the custom firmware system I employed for each game (and we will talk more about it in Chapter 12), the idea was simply that each game receive its own tiny little environment for gameplay. This pattern is not novel or new, and I didn't come up with it first to be honest. Even tekparrot has its game .xml profiles, and I believe Jconfig uses .json files. My goal was broader and much more ambitious, in that I wanted to provide a mechanism whereby the community could help "bring up" and improve these games, and I needed to do so in a manner that provided protection and safety for everyone, while still allowing community devs and tinkerers to play.

Thus I developed the individual game firmware system, (for example: Mario Kart.ufw) which the loader loads alongside the game, and it contains the game's micro environment for play, but also in practice, much, much more. The technical details I will leave for the devs and my fellow nerds, but each firmware package is tiny, and it comes in the form of a top and bottom layer. The bottom layer is what boots the game and runs it. It is read-only and never writable. The loader will only accept this layer. The top layer, is the play/tinker/dev area. This area can only be enabled one time per session in the loader, and can never be graduated to the bottom layer.

Devs and tinkerers and experienced coders better than myself may play here and work to improve all games for the community, if they so wish and want to help. Once done, there will be an upstream pipeline, where all firmware packages will need to be properly signed before the loader will accept it again. We will talk more about this in Chapter 12, but this is what I came up with in order to ensure that the community is protected, and we never deal with the gatekeeping and otherwise preventable problems that we are now facing today. More on this later. For now, please take a look at the next page to do a deep dive into some more of the loader's buttons and options. This will give you more insight into each function.

OPTION	WHAT IT DOES
Configure Controls ID_INPUTCFG	Open Controller Configuration: launchers for the Ultra Omni-Play (gamepad), Ultra Omni-Track (lightgun), Ultra Omni-Wheel (racing), Ultra Omni-Steer (flight stick) and Ultra Omni-Bridge (keyboard and mouse) panels.
View Controls ID_CONTROLS	Show the control map/button layout for the selected game. The first time you open it for a game the loader probes that game's own executable for its real button names, which takes a second or three - the wait is the scan, not a hang - and the result is remembered until you pick a different game. If the probe finds nothing, you get Ultra Loader's hand-written layout for that title instead.
Install Game Arts ID_INSTALL_ART	Install game arts/system logos (.pak). Used on the boot console and game tiles. Ultra Loader will include most logos, but you may also source your own if you wish to display custom logos in the title area. Just make sure to label the logos to match the name of the game folder in your game collection, so the loader can recognize it and display the correct logo. Please see Chapter 3 for the art folder. A pack is unpacked into art\<system>\ rather than copied in whole, so you can open the folder and see exactly which logos you have. You are asked Replace or Merge first - Replace empties every system folder before extracting, your own logos included, while Merge keeps what is there and only fills the gaps. Case and punctuation are ignored when matching, so pokken_tournament.png and Pokken Tournament.png are the same name, but the name is cut at the FIRST dot - keep dots out of it, or Ridge Racer V.2.png and Ridge Racer V.3.png collide.
Add System BIOS Files ID_BIOS_PACK	Pick a folder of BIOS files. Recognized files are sorted into the right module folders. PS3UPDAT.PUP is auto-unpacked and copied into dev_flash automatically by the bundled pup_extract.exe - note this covers PS3 only, as PS Vita firmware uses different keys and cannot be unpacked this way. No need for a third-party install of any emulator software to install the bios files. Ultra Loader will do it for you. This is a part of the commitment to making things as easy and as simple as possible for users. An easy way to cover everything, is to download a mini-bios pack, (many online) and just install that.
View Firmware ID_MODULES	Open the Ultra Firmware container for the game selected in the main window: its board, identity, signature, what is packed inside it, and whether it is clean or carries an unblessed test layer, which shows as EDITED. With no game selected, or one that has no container, you get the list of everything installed instead. Browse Firmware reaches any other .ufw on disk, including one you have not installed yet, and Enable Test Mode arms that one container to run unsigned off its test layer for this session - every other game stays fully verified.
Install Firmware ID_INSTALL_CORE	Install Ultra Firmware containers - pick one or more .ufw files, not a DLL. Each one is signature-checked before anything is copied, so a container that fails the check is refused here with the reason, instead of failing later at launch where it would look like a broken game. Accepted ones are filed under firmware\<board>\ next to the loader; restart the loader to pick them up.

OPTION	WHAT IT DOES
Uninstall Firmware ID_UNINSTALL_CORE	Shows the firmware you have installed, then opens the loader's firmware folder in Explorer so you can remove a container by deleting its .ufw file. Deleting it is what stops that game running; the board module travels inside the container, so there is no separate module left behind to fall back on. To play that game again you will need to install its container again, so keep a copy.
Uninstall BIOS Files ID_UNINSTALL_BIOS	Delete BIOS and console firmware files from the installed system folders - the retro and console systems that keep a BIOS, since a native arcade board carries what it needs inside its own firmware container. You are shown every file and its size before anything is deleted. Only files whose names are on the loader's known-BIOS list are touched: module DLLs, your own files and anything it does not recognize are left exactly where they are. Only bios files are removed.

Configure Controls Dialog (Chapter 8 Cont'd)

OPTION	WHAT IT DOES
Enable gamepad input ID_PLENABLE	Enable Ultra Omni-Play gamepad input. Same master switch as inside the Ultra Omni-Play panel.
Ultra Omni-Play » ID_PLAY	Open the Ultra Omni-Play panel: up to 10 gamepads, with per-slot dead zone/curve/rumble test, master switches (universal support, rumble, auto-cal, trigger threshold), and auto-detect for Xbox, PlayStation, Switch Pro, Steam, 8BitDo, Hori, Qanba, Razer, MadCatz, PDD, Power A and Logitech controllers.
Enable lightgun input ID_LGEN	Enable Ultra Omni-Track lightgun input. Same master switch as inside the Ultra Omni-Track panel.
Ultra Omni-Track » ID_OMNITRACK	Open the full Ultra Omni-Track lightgun panel: per-gun brands, 1-8 players, calibration, TATE, off-screen reload, and one-click Auto-Config. Also, stress-free calibration.
Enable wheel input ID_WHENABLE	Enable Ultra Omni-Wheel racing input. Same master switch as inside the Ultra Omni-Wheel panel.
Ultra Omni-Wheel » ID_WHEEL	Open the Ultra Omni-Wheel panel: up to 4 racing wheels, per-slot dead zone/sensitivity/curve/invert, FFB test buttons, and auto-detect for Logitech, Thrustmaster, Fanatec, Moza and Simagic racing wheels.
Enable flight-stick input ID_FLENABLE	Enable Ultra Omni-Steer flight-stick input. Same master switch as inside the Ultra Omni-Steer panel.
Ultra Omni-Steer » ID_FLIGHT	Open the Ultra Omni-Steer panel: up to 4 flight sticks/HOTAS setups, HOTAS pairing (stick + throttle + rudder), per-slot dead zone/sensitivity/curve/invert X/Y/Z, throttle reverse detent, FFB, and auto-detect for Thrustmaster, Logitech, Saitek, VKB, Virpil, CH, WinWing, Honeycomb, Turtle Beach, Microsoft and PXN flight sticks.
Enable Keyboard Bridge ID_BRENABLE	For native PC-arcade dumps that only read raw keyboard+mouse inputs (no JVS/DirectInput/gun API).
Ultra Omni-Bridge » ID_BRIDGE	Open the Ultra Omni-Bridge panel: cursor source (gamepad stick left/right), dead zone, sensitivity, P1/P2 click-button mapping, and START/COIN/SERVICE/TEST key taps. Ultra Omni-Bridge synthesizes real cursor moves, clicks, and keyboard taps for a connected gamepad or lightgun so those titles are playable, even without a physical mouse and keyboard. It fully maps all controls to your gamepad automatically, without having to custom-select the letters on the keypad. However, you may also enter the settings to adjust your custom preference as you like. This is safe to leave on, and does nothing when no gamepad/lightgun is present.
OK ID_OK	Apply all controller changes and close.
Cancel ID_CANCEL	Discard all controller changes and close.

The Omni Input Services (In Detail)

Gamepads, wheels, flight sticks and lightguns all configured once, then applied to every game.

The following pages will go into each omni service in detail, along with its per-slot settings. This is the immense level of work I put into making sure that people NEVER have to worry about controller setup or compatibility again. I aimed for something like 99% compatibility, and for most people, this should hopefully be enough.

Each service has a master switch plus a row of settings for every slot. The per-slot settings are the same for each slot, so they are listed once per service below with the slot number written as N.

Ultra Omni-Play (gamepad)

OPTION	WHAT IT DOES
Enable Ultra Omni-Play gamepad input s_op[IOP_ENABLE]	Master switch. When on, Ultra Omni-Play routes every connected gamepad (Xinput slots 0-3 and DirectInput slots 4-9, up to 10 players) into the running game.
Auto-detect pads (zero setup) s_op[IOP_AUTOPROF]	Zero-setup: read each pad's USB VID/PID, identify the brand (Xbox / PlayStation / Switch Pro / 8BitDo / Hori / Qanba / Razer / MadCatz / PDP / Power A / Logitech / Brook / Xin-Mo / generic HID / zero delay encoders), and pick sensible dead zone + curve + rumble defaults per slot. Zero delay USB encoder boards, Sanwa and Seimitsu arcade sticks, and Brook fighting boards are all recognized and work plug-and-play with no additional setup.
Universal gamepad support (Xbox / PlayStation / Switch / arcade sticks / generic HID) s_op[IOP_UNIVERSAL]	Accept ALL gamepad families at once: Xinput, DirectInput, generic HID pads, arcade sticks, Xbox / PlayStation / Switch Pro / 8BitDo / Hori / Qanba / Razer / MadCatz / PDP / Power A / Logitech. Leave on for plug-and-play.
Rumble / force-feedback (Xbox motors + Xinput) s_op[IOP_RUMBLE]	Global rumble/force-feedback gate. On = the service sends motor effects to any pad that supports them (Xbox motors, DualShock/DualSense actuators, Switch Pro HD Rumble). Off = every pad is silent.
Intelligent stick auto-calibration (corrects drift + centre offset) s_op[IOP_AUTOCAL]	One of my favorite options in Ultra Omni-Play. This setting will continuously correct stick drift and center-point offset. This is for all those who can't afford or don't want to purchase those new fancy hall-effect sticks, or still use good, solid older controllers, like myself. Turn this off only for new controllers, and/or for premium pads that already calibrate onboard.
Trigger threshold (1-100%) s_op[IOP_TRIG]	How far an analog trigger (LT / RT / L2 / R2 / ZL / ZR) must be pulled before it registers as a digital press. Range 1-100%. Lower = hair-trigger, higher = firm pull.
Save s_op[IOP_OK]	Apply and persist Ultra Omni-Play settings to ultra.ini, then close.
Cancel s_op[IOP_CANCEL]	Discard changes and close.

Per-slot settings. One row per slot; N is the slot number.

OPTION	WHAT IT DOES
Enable slot IOP_SLOT0_PLR + slot	Enable pad slot N (Xinput for slots 1-4, DirectInput for slots 5-10).
Brand / status IOP_SLOT0_BRAND + slot	Detected pad in slot N, brand + connection status. Auto-detect fills this in.
Dead zone % IOP_SLOT0_DZ + slot	Dead zone % (0-50) for slot N. Ignores tiny stick movement around center; kills drift.
Response curve % IOP_SLOT0_CV + slot	Response curve % (50-200) for slot N. 100 = linear, <100 = softer near center, >100 = sharper at the edges.
Rumble IOP_SLOT0_RMB + slot	Rumble on/off for slot N, overrides the master rumble gate for this pad only.
Test IOP_SLOT0_TEST + slot	Test: fire a short rumble burst on slot N so you can confirm the pad is alive and mapped to the right slot.

Ultra Omni-Steer (flight stick)

OPTION	WHAT IT DOES
Enable Ultra Omni-Steer flight-stick input s_os[IOS_ENABLE]	Master switch. When on, Ultra Omni-Steer feeds roll/pitch/yaw/throttle + trigger + cabinet buttons into every flight-aware system (Lindbergh JVS analog for After Burner Climax, Type X flight cabs, libretro flight cores).
Auto-detect sticks + HOTAS pairing s_os[IOS_AUTOPROF]	Zero-setup: reads each connected stick's USB VID/PID, identifies the brand (Thrustmaster / Logitech / Saitek / VKB / Virpil / CH / WinWing / Honeycomb / Turtle Beach / Microsoft / PXN), and pairs HOTAS splits (stick + throttle + rudder) into one logical slot.
Enable force-feedback (FFB) s_os[IOS_FFBON]	Global FFB gate. On = the service sends effects to FFB-capable sticks (Microsoft Sidewinder FFB2, Thrustmaster T.16000M FORCE, VKB MCG Ultimate, Virpil MongoosT-50 FFB). Off = every stick is silent.
Save s_os[IOS_OK]	Apply and persist Ultra Omni-Steer settings to ultra.ini, then close.
Cancel s_os[IOS_CANCEL]	Discard changes and close.

Per-slot settings. One row per slot; N is the slot number.

OPTION	WHAT IT DOES
Enable slot IOS_SLOT0_PLR + slot	Enable flight-stick slot N (HOTAS: stick + throttle + rudder paired here).
Brand / status IOS_SLOT0_BRAND + slot	Detected stick in slot N, brand + connection status. Auto-detect fills this in.
Dead zone % IOS_SLOT0_DZ + slot	Dead zone % (0-50) for slot N. Ignores tiny stick movement around centre.
Sensitivity % IOS_SLOT0_SN + slot	Sensitivity % (50-200) for slot N. 100 = 1:1 axis.
Response curve % IOS_SLOT0_CV + slot	Response curve % (50-200) for slot N. 100 = linear, >100 = expo (sharp at edges).
Invert X IOS_SLOT0_IX + slot	Invert X (roll) for slot N.
Invert Y IOS_SLOT0_IY + slot	Invert Y (pitch) for slot N. On = pull-back climbs (classic flight-sim).
Invert Z IOS_SLOT0_IZ + slot	Invert Z (yaw / rudder) for slot N.
Throttle reverse IOS_SLOT0_TREV + slot	Throttle reverse detent for slot N, jet-style throttles that pull below idle for reverse thrust. Off = throttle is 0..+1. On = throttle is -1..+1 with idle at 0.
FFB test IOS_SLOT0_FFB + slot	Test: fire a short FFB burst on slot N so you can confirm the stick is alive and mapped to the right slot.

Ultra Omni-Wheel (racing wheel)

OPTION	WHAT IT DOES
Enable Ultra Omni-Wheel racing input s_ow[IOW_ENABLE]	Master switch. When on, Ultra Omni-Wheel feeds wheel steer + pedals + shift + cabinet buttons into every wheel-aware system (Type X/X2 JVS, Sega Europa-R/RingEdge, libretro racing cores).
Auto-detect wheels (zero setup) s_ow[IOW_AUTOPROF]	Zero-setup: reads each connected wheel's USB VID/PID + HID product string, identifies the brand (Logitech G29 / G920 / G923 / DFGT, Thrustmaster T300 / T500 / T150, Fanatec CSL / CSW / DD, Moza, Simagic) and loads its calibration.
Enable force-feedback (FFB) s_ow[IOW_FFBON]	Send road-feel + rumble to wheels that support FFB. Turn off for a quiet wheel or when a broken driver keeps engaging the motor at zero speed.
Save s_ow[IOW_OK]	Apply and persist Ultra Omni-Wheel settings to ultra.ini, then close.
Cancel s_ow[IOW_CANCEL]	Discard changes and close.

Per-slot settings. One row per slot; N is the slot number.

OPTION	WHAT IT DOES
Enable slot IOW_SLOT0_PLR + slot	Enable wheel slot N.
Dead zone % IOW_SLOT0_DZ + slot	Dead zone % (0-50). Ignores tiny wheel movement around center.
Sensitivity % IOW_SLOT0_SN + slot	Sensitivity % (50-200). 100 = 1:1 steering.
Response curve % IOW_SLOT0_CV + slot	Response curve % (50-200). 100 = linear, >100 = expo (sharp at edges).
Invert IOW_SLOT0_INV + slot	Invert steering axis for slot N.
Brand / status IOW_SLOT0_BRAND + slot	Detected wheel in slot N, brand + connection status. Auto-detect fills this in.

Ultra Omni-Track (lightgun)

OPTION	WHAT IT DOES
Enable Ultra Omni-Track lightgun input s_ot[IOT_ENABLE]	Master switch. When on, Ultra Omni-Track feeds gun aim + buttons into the running game (Lindbergh JVS, Type X / X2 and every libretro core). When off, the loader falls back to pad/keyboard.
Horizontal sensitivity s_ot[IOT_XSENS]	Horizontal aim gain, 10-300%. 100% = exact 1:1 tracking. Applied around screen center so it never shifts your zero.
Vertical sensitivity s_ot[IOT_YSENS]	Vertical aim gain, 10-300%. 100% = exact 1:1 tracking. Applied around screen center.
X calibration offset s_ot[IOT_XOFF]	Horizontal trim, -50..+50. Negative shifts aim left, positive shifts right. Fixes a constant left/right bias.
Y calibration offset s_ot[IOT_YOFF]	Vertical trim, -50..+50. Negative shifts aim up, positive shifts down.
Mirror horizontal axis (invert X) s_ot[IOT_INVX]	Flip aim left/right. Use for mirror-mounted cabs or a gun whose camera is upside-down on the X axis.
Mirror vertical axis (invert Y) s_ot[IOT_INVY]	Flip aim up/down. Use if up and down are reversed (common on some IR mounts).
Swap X / Y axes : TATE mode s_ot[IOT_SWAP]	Transpose the aim axes for portrait / rotated (TATE) cabinets so the gun still tracks correctly after the screen is turned 90 degrees.
Off-screen auto-reload s_ot[IOT_OFFSCR]	The classic arcade reload: pull the trigger while pointing OFF the screen to reload. Turn off if a game uses a dedicated reload button instead. Right-click always reloads regardless.
Auto-detect gun & load its profile (zero setup) s_ot[IOT_AUTOPROF]	Zero setup: on launch Ultra Omni-Track reads each connected device's USB VID/PID + product name, matches the gun maker, and auto-loads its profile (recoil, border, button map). No manual picking needed. If you have a lot of guns, you will love this.

OPTION	WHAT IT DOES
Aim border (thin white, auto-fit) s_ot[IOT_BORDER]	Yes, my dear Sinden fans, I made this one especially for you. This setting will draw Ultra Omni-Track's <i>own</i> thin white border around the screen edge: a present-time overlay that auto-fits any resolution. This COMPLETELY replaces Sinden's thick border for camera guns. You no longer need Sinden's buggy border software in order to play lightgun games. You can not only draw a thin white border, I also allow you to customize how thin or thick you want it. This is also harmless for IR guns.
Select Gun Hardware	This will allow you to select your lightgun manufacturer profile. Selecting this helps Ultra Loader apply the correct settings to your guns. I included profiles for almost every lightgun manufacturer out there, and should work for mostly everyone.
Px size s_ot[IOT_BORDERPX]	Aim-border thickness, 1-40 px. Thin = a few px; it auto-scales to your screen resolution. No more adjusting your screen resolution and games to adhere to the border. The border adheres to you.
Auto-Config (smart setup) s_ot[IOT_AUTOCFG]	This setting is made for simple and easy setup. Don't want to fiddle with your lightguns? Click this and Ultra Omni-Track detects your attached pointing devices, turns on the lightgun, creates one player per device set to Auto-detect, and restores sane aim defaults. You won't ever run into USB device collisions or errors ever again. Each gun gets its own distinct ID.
Save s_ot[IOT_OK]	Apply all Ultra Omni-Track settings to ultra.ini, then close.
Cancel s_ot[IOT_CANCEL]	Discard changes made in this panel and close.

Per-slot settings. One row per slot; N is the slot number.

OPTION	WHAT IT DOES
Enable player slot IOT_PLR0 + slot	Enable Player N's gun slot.
Assigned gun IOT_ASG0 + slot	Choose which gun drives Player N. Auto-detect binds the next gun that moves; or pin a specific brand profile to this slot.

Ultra Omni-Bridge (gamepad to keyboard/mouse)

OPTION	WHAT IT DOES
Enable Keyboard Bridge s_ob[IOB_ENABLE]	Master switch. When on, a connected gamepad or lightgun drives synthetic mouse cursor moves, clicks, and key taps for games that only read raw OS keyboard+mouse input (no JVS/DirectInput/gun API). Safe to leave on: does nothing when no gamepad/lightgun is present.
Drive cursor from gamepad stick when no lightgun pose is present s_ob[IOB_GAMEPAD_SRC]	A real lightgun pose (presence/position) always wins. When no lightgun is present, this lets a connected gamepad's analog stick move the synthetic cursor instead. You can now play your favorite lightgun games with ease using your gamepad.
Aim with right stick (default: left stick) s_ob[IOB_STICK_RIGHT]	Which analog stick moves the synthetic cursor. Off = left stick (default). On = right stick.
Stick deadzone (0-50%) s_ob[IOB_DEADZONE]	Ignores tiny stick movement around center before it moves the cursor.
Cursor sensitivity (10-300%) s_ob[IOB_SENS]	Stick-to-cursor speed multiplier. 100% = normal speed.
Player 2's trigger is the RIGHT mouse button (off = left) s_ob[IOB_P2_RIGHT]	Player 1's trigger is always left-click. This picks which physical button P2's trigger maps to. On (default).
Save s_ob[IOB_OK]	Apply Omni-Bridge settings to ultra.ini, then close.
Cancel s_ob[IOB_CANCEL]	Discard changes and close.

Credits and Coins

Free play, the on-screen coin pill, and automatic coin and credit insertion.

Most arcade games expect a coin mechanism. Ultra Loader can supply one by three ways:

1. By switching the game into its own free-play mode where it has one,
2. By giving you a coin button, or
3. By inserting credits automatically when the game boots.

HOTKEY & BASE COIN

OPTION	WHAT IT DOES
Insert-coin hotkey ID_HK	You can set the keyboard key that drops a coin while a game is running. The loader watches for this key globally, so it works even though the game owns the keyboard. Just be sure to pick a key the game itself does not use.
Credits per coin (1-9) ID_CPC	Essentially, how many credits one coin is worth, ranging all the way from 1 to 9. Some games will make you insert 2 coins to get 1 credit. Pokken Tournament is one. There are many others. Some may even ask for up to 4 or more coins for one single gameplay credit (greedy, I know). This setting in fact, mirrors the cabinet operator setting, and a machine set to "2 coins = 1 credit" can also be configured in the game's own test/service menu if it allows it. Some games don't have that option though. So the idea is that if you don't want to do a deep dive into the game's coin/credit mechanism or play around in the test or service menu, you can adjust it here. For most games though, I recommend you leave it at 1 unless you know the game expects otherwise. For instance, in a scenario where you put 2 coins as base instead of 1, for a game that requires 2 coins to establish a gameplay credit, you will have just basically saved yourself a single coin insertion. Minor, but the fun is in the details. If you are establishing a proper arcade cabinet, this can make the difference between 2 clicks of the coin button, or 1. Plus if you so happen to charge per coin, (a super niche scenario) then you might like this option.
Player (which slot receives) ID_PLAYER	Which player slot the coin is credited to, P1 through P4. On a two-player cabinet each side has its own coin slot, and some games will only start the side that was paid for. Again, for most users, this may not be needed, but I wanted to be thorough.
JVS pulse duration (ms) ID_HOLD	Yes, I went there. I told you, this is the level of detail I threw into this thing. This is an option to adjust how long the simulated coin switch is held closed, in milliseconds (range is 30-200). A normal real-life coin mechanism holds the switch for roughly 60ms, which is the default. You can raise the number if a game misses coins on insertion, or lower it if one press (or real life coin insert) registers as two coins. If we are making arcade software that will sit in authentic cabinets, then we gotta do it right, and we gotta do it tight.

AUTO-INSERT ON GAME BOOT

OPTION	WHAT IT DOES
Automatically insert credits when a game starts ID_AUTO	This is a cool feature I thought people might like. This setting drops credits automatically as soon as the game boots, so you never see the insert-coin screen. Similar to free-play, but this does not disable the insert-coin mechanism. It just speeds past it without your input. This is super convenient for a game you are testing repeatedly. You can just turn it off if you want to have normal cabinet behavior.
How many credits (1-9) ID_AUTO_N	How many credits to insert automatically at boot, 1 to 9. Only used when automatic insertion is switched on. Mirrors the Credits Per Coin setting, but specific to the above setting only.

ON-SCREEN COIN PILL

OPTION	WHAT IT DOES
Show clickable COIN pill during gameplay ID_SHOW	Draw a small clickable COIN button on top of the running game, so you can add credits with the mouse instead of the hotkey. Useful on a desktop; but turn it off on a real cabinet.
Pill position ID_POS	Which corner the COIN pill sits in: bottom-right (default), bottom-left, top-right or top-left. Move it if it covers something the game draws or renders in that corner.
Show live credit count inside the pill ID_SHOWCNT	Print the current credit count inside the pill, so you can see at a glance how many credits the game has banked.
Play cabinet coin-drop sound on insert ID_SOUND	Play a coin-drop sound when a credit is inserted, the way a real cabinet clunks. Cosmetic only; it does not change what the game receives. It's just a auditory indication/confirmation.
Save ID_OK	Apply all controller changes and close.
Cancel ID_CANCEL	Discard controller changes and close.

Boards and Systems

Every arcade board Ultra Loader supports, and what each one needs.

Ultra Loader identifies the board from the game's path and executable, then loads the matching firmware container. The table below is the loader's full catalogue.

Board detection is built-in

Board keywords are matched against the game's path, using the longest and most specific name first. Please be sure to name your game collections accurately (Taito Type X, Sega RingWide, etc.) however, the loader has been built with smart logic to recognize games by its firmware container internal header, so in truth, even if you label a game wrong, the loader should still be able to find it. That being said, you still need to label your games properly, so that your art/icons can display.

MODULE ID	BOARD (USE THESE NAMES)	I/O	GRAPHICS
1. sega_lindbergh	Sega Lindbergh	JVS	GeForce 7
2. taito_typex	Taito Type X	JVS	GeForce
3. taito_typex2	Taito Type X2	JVS	Radeon
4. taito_typex3	Taito Type X3	JVS	GPU
5. taito_typex4	Taito Type X4	JVS	GPU
6. taito_typezero	Taito Type X Zero	JVS	GPU
7. taito_typexplus	Taito Type X+	JVS	GPU
8. taito_typexzero	Taito Type X Zero	JVS	GPU
9. taito_nesica	Taito NESiCAXLive	USB	GPU
10. taito_nesica2	Taito NESiCAXLive2	USB	GPU
11. sega_ringedge	Sega RingEdge	JVS	GPU
12. sega_ringwide	Sega RingWide	JVS	GPU
13. sega_ringedge2	Sega RingEdge 2	JVS	GPU
14. sega_nu	Sega Nu	USB	GPU
15. sega_nu2	Sega Nu 2	USB	GPU
16. sega_nuc	Sega Nu SX	USB	GPU
17. sega_all	Sega ALLS	USB	GPU
18. sega_all_ux	Sega ALLS UX	USB	GPU
19. sega_europar	Sega Europa-R	JVS	GPU
20. namco_n2	Namco System N2	JVS	GPU
21. namco_es1	Namco System ES1	JVS	GPU
22. namco_es2	Namco System ES2	JVS	GPU
23. namco_es2plus	Namco System ES2+	JVS	GPU
24. namco_es3	Namco System ES3	JVS	GPU
25. namco_es4	Namco System ES4	JVS	GPU
26. namco_bna1	Namco System BNA1	JVS	GPU

MODULE ID	BOARD (USE THESE NAMES)	I/O	GRAPHICS
27.konami_pc	Konami Arcade PC	I/O	GPU
28.konami_bemani	Konami Bemani PC	ACIO	GPU
29.konami_python	Konami PYTHON	JVS	PS2 GS
30.cave_pc	Cave Arcade PC	COM	GPU
31.exa_arcadia	exA-Arcadia	JVS	GPU
32.unico_pc	UNICO Arcade PC	JVS	GPU
33.rawthrills_pc	Raw Thrills Arcade PC	JVS	GPU
34.rawthrills_g5	Raw Thrills g5	LPT/FTDI	OpenGL
35.wahlap_wpc	Wahlap WPC	JVS	GPU
36.andamiro_pcplus	Andamiro PC+	JVS	GPU
37.global_vr	Global VR System	JVS	GPU
38.igs_pc	IGS Arcade PC	JVS	GPU
39.incredible_technologies	Incredible Technologies PC	JVS	GPU
40.play_mechanix	Play Mechanix	JVS	GPU
41.examu_exboard	Examu eX-Board	JVS	GPU
42.generic_pc	Arcade PC	I/O	GPU

Firmware Signing and Protection

The most important chapter of this entire manual. Please read this to understand what I did and why.

Given the problem of gatekeeping and profiteering by certain individuals using these modern arcade games on the community, along with the development and potential of this software, I realized that the responsibility fell on me to deliver some form of protection against something like this happening again. My reasoning and sentiments for doing this was not merely to prevent or stop the gatekeeping and profiteering, but also very importantly, to protect the community. Many people love playing these games (myself included), and the types of software that launch them usually live on the edge of what most operating systems (looking at you, Windows) would report as exploits and viruses, which as anyone knows, is not true, but it is still a major problem facing the community. Malware exists, and many ill-intentioned people still use it constantly.

There are also very malicious actors out there, people who will write code and deliberately target you and your important files, and the risk of that happening is even more prevalent now, with the rise of AI and other tools made available to the public. I also use AI to correct and enhance my work, and I have come to see how easy it would be for bad actors to write malicious code. As such, I thought deeply about it, and I decided that I must take steps to ensure that this software is safe, clean and stable for everyone to use, regardless of whether you have knowledge about software or none at all. Everyone should be able to use this software with ease and without fear, knowing that it is fully verified. As such, I developed a firmware *signing* system for all games, permanently locked against the loader.

What this means, is that every game's firmware is signed, and the loader verifies that signature before it will load the game. The check is also fully enforced, and any module that has been edited or tampered with, even in the slightest of ways will be refused, and the log will show a [SIGCHECK] error with the reason. This matters greatly if the community is to build upon these modules to run on Ultra Loader. All firmware will be released to the public as open source, and anyone can take them and begin editing, improving or cultivating them, however, the loader itself will, for the time being, remain closed-source. This is to ensure that the upstream flow of modules edited and improved by the community can be checked against a clean, sandboxed instance of the loader, and can be scanned for malicious code and viruses, and also verified, before pushing it out to the general public. Essentially, a fully rebuilt module must be re-signed before it will load the game again. Rebuilding the firmware container alone will leave you with a module that the loader will not touch. This will also prevent random, unsigned modules from being loaded by people who will happily download an untrusted, unverified firmware, and load it.

That being said, I don't intend to do this alone, as one guy working on all these games will take forever to work through the full catalog. I think that the only way forward is to trust the community to do the right thing. So as time goes on, we will select (based on community recommendation) a few trustworthy folks, who I will provide with the signing *keygen*, and they will have the full authority to verify, build and re-sign modules for the general public. My plan is not to keep the authority to myself, as I know that this will be the first complaint people will lobby against me. I fully know and realize that open source is the best way to garner trust, and as such, I release the firmware modules for community editing, but for now, we must do something to protect this new tool, or people will take it, improve upon it, and then create another teknoparrot scenario, or worse, hoard it all for themselves, where they once again monetize access to these games, and people will once more be forced to "pay to play", something I am strongly against.

Therefore, the plan (for now) is to have all improved firmwares run upstream to me through perhaps GitHub, and as community devs make improvements, I will test and sign the modules and then release them back to the public. This is the best plan I could think of for now, and I am fully open and amenable to any idea or plan of action that anyone has that will offer both protection and assurance of long-term community benefit. I hope everyone understands my reasoning behind why I did this, and why for now, the loader itself will remain, mostly closed-source.

Ultra.ini Reference (The Settings File)

Every key the loader reads and writes, in the order it stores them.

Settings are saved to the ultra.ini file sitting in beside the loader. You can edit it by hand while the loader is closed, and you can also save to it directly from the Pause Menu. It is also fully rewritten atomically on save, so a crash won't leave a half-written file. Please see the next page for the full .ini settings table with all 110+ functions.

KEY	Default
[display]	
width	1920
height	1080
display_mode	1
aspect	3
vsync	1
fps_cap	60
rotate	0
upscaler	2
sharpness	50
antialias	1
anisotropy	1
cutscene_fill	1
framegen	0
[loader]	
show_splash	1
mute_splash_sound	0
hide_console	1
easy_config	1
boot_to_menu	1
boot_to_game	0
boot_console_menu	0
skip_movie	1
skip_jvs	0
skip_audio	0
freeplay	0
snapshots	0
force_english	0
allnet	1
allnet_elevate	1
[credit_injector]	
credit_injector_enable	0
credit_injector_hotkey	45
credit_injector_credits_per_coin	1
credit_injector_show_button	1
credit_injector_auto_insert	0
credit_injector_auto_insert_count	1
credit_injector_player	0
credit_injector_hold_ms	60
credit_injector_pill_position	0
credit_injector_show_count	1

credit_injector_sound_feedback	1
[vita]	
vita_homebrew_mode	1
vita_verbose_missing_modules	1
[audio]	
audio_enable	1
audio_volume	50
pause_game_audio	1
menu_sounds	1
[controllers]	
rumble	1
deadzone	12
gun_cabinet	0
stick_sensitivity	100
stick_curve	0
stick_smooth	40
stick_invert_x	0
stick_invert_y	0
trigger_threshold	25
autocal	1
analog_cal	1
lightgun_enable	1
lightgun_x_sens	100
lightgun_y_sens	100
lightgun_x_offset	0
lightgun_y_offset	0
lightgun_invert_x	0
lightgun_invert_y	0
lightgun_swap_xy	0
lightgun_offscreen_reload	1
lightgun_brands	256
lightgun_players	1
lightgun_assign	0,0,0,0,0,0,0,0
lightgun_auto_profile	1
lightgun_border	0
lightgun_border_thickness	3
lightgun_show_experimental	0
universal_gamepad	1
[wheel]	
wheel_enable	0
wheel_auto_profile	1
wheel_ffb_enable	1

wheel_players	1
wheel_deadzone	8,8,8,8
wheel_sensitivity	100,100,100,100
wheel_curve	100,100,100,100
wheel_invert_steer	0,0,0,0
[flight]	
flight_enable	0
flight_auto_profile	1
flight_ffb_enable	1
flight_players	1
flight_deadzone	6,6,6,6
flight_sensitivity	100,100,100,100
flight_curve	100,100,100,100
flight_invert_x	0,0,0,0
flight_invert_y	0,0,0,0
flight_invert_z	0,0,0,0
flight_throttle_deadzone	4,4,4,4
flight_throttle_curve	100,100,100,100
flight_throttle_reverse	0,0,0,0
[play]	
play_enable	1
play_auto_profile	1
play_players	1
play_deadzone	10,10,...
play_curve	100,100,...
play_rumble	1,1,...
[bridge]	
bridge_enable	0
bridge_gamepad_source	1
bridge_stick_right	0
bridge_deadzone	15
bridge_sensitivity	100
bridge_p2_button_right	1
bridge_vk_start	49
bridge_vk_coin	53
bridge_vk_service	0
bridge_vk_test	0
[game]	
game_path	(last launched game)
library_path	(empty)
library_compact_console_tiles	0
font_name	Segoe UI

theme_name	Grey
gl_driver	(empty)

Environment Variables (For the devs)

Overriding any setting from outside, for scripting and for testing.

Every setting reaches the game as an environment variable, and an explicit variable set before launch wins over both the saved settings and any board default. This is the supported way to script the loader or to test one change without disturbing your configuration.

The loader itself publishes or reads the 150+ variables below. Individual board hooks add many more. Those are documented in the board's own source code and are not user settings.

VARIABLE	REFERENCED IN
ULTRA_AA	d3d8_sharpen.c, d3d8_sharpen.h, d3d8_throttle.c
ULTRA_ALLNET_NETID	allnet_setup.c
ULTRA_ALLNET_SERIAL	allnet_setup.c
ULTRA_ANISO	glx_emu.c, glx_emu.c, glx_emu.c
ULTRA_ART_TRACE	settings.c
ULTRA_ASPECT	d3d8_throttle.c, d3d9_throttle.c, d3d9_throttle_examu.c
ULTRA_AUDIO_VOL	audio_engine.c, audio_engine.c, audio_engine.c
ULTRA_AUTOCAL	omni_client.h, main.c, settings.c
ULTRA_BOOT_AUDIO	lindbergh_main.c, lindbergh_main.c, main.c
ULTRA_BOOT_CONSOLE_H	boot_console.h
ULTRA_BOOT_CONSOLE_MENU	main.c
ULTRA_BOOT_GPU	lindbergh_main.c, lindbergh_main.c, main.c
ULTRA_BOOT_IO	lindbergh_main.c, lindbergh_main.c, main.c
ULTRA_BOOT_SYSTEM	lindbergh_main.c, inject_main.c, inject_main.c
ULTRA_BOOT_TITLE	lindbergh_main.c, inject_main.c, inject_main.c
ULTRA_BOOT_XMB	main.c
ULTRA_BORDERLESS	d3d9_throttle_pcplus.c, d3d8_throttle.c, d3d9_throttle.c
ULTRA_BRIDGE_DEADZONE	hook_dllmain_unico.c, hook_dllmain_unico.c, omni_client.h
ULTRA_BRIDGE_ENABLE	hook_dllmain_pcplus.c, omni_bridge.c, hook_dllmain.c
ULTRA_BRIDGE_GAMEPAD	hook_dllmain_unico.c, hook_dllmain_unico.c, omni_client.h
ULTRA_BRIDGE_P2_RIGHT	hook_dllmain_unico.c, hook_dllmain_unico.c, omni_client.h
ULTRA_BRIDGE_SENS	hook_dllmain_unico.c, hook_dllmain_unico.c, omni_client.h
ULTRA_BRIDGE_STICK_RIGHT	hook_dllmain_unico.c, hook_dllmain_unico.c, omni_client.h
ULTRA_BRIDGE_VK_COIN	omni_client.h, main.c
ULTRA_BRIDGE_VK_SERVICE	omni_client.h, main.c
ULTRA_BRIDGE_VK_START	hook_dllmain_unico.c, hook_dllmain_unico.c, omni_client.h
ULTRA_BRIDGE_VK_TEST	omni_client.h, main.c

VARIABLE	REFERENCED IN
ULTRA_CAP_GL_PRESENT	lindbergh_module.c, ultra_module.h, settings.h
ULTRA_CAS_SHARP	glx_emu.c, glx_emu.c, glx_emu.c
ULTRA_CATALOG_H	ultra_catalog.h
ULTRA_CI_PULSE	credit_injector_service.c
ULTRA_CI_PULSE_SLOT	credit_injector_service.c, main.c
ULTRA_CURRENT_ABI_VERSION	sign_module.c, module_manifest.h, module_sig_verify.c
ULTRA_CUT_FILL	glx_emu.c, glx_emu.c, glx_emu.c
ULTRA_DISP_MODE	glx_emu.c, glx_emu.c, glx_emu.c
ULTRA_DI_AXIS_	di_range.h, omniplay_service.c, omniwheel_service.c
ULTRA_DI_AXIS_RX	di_range.h, di_range.c, omniwheel_service.c
ULTRA_DI_AXIS_RY	di_range.h, di_range.c
ULTRA_DI_AXIS_RZ	di_range.h, di_range.c, omniplay_service.c
ULTRA_DI_AXIS_SLIDER	di_range.h, di_range.c
ULTRA_DI_AXIS_X	di_range.h, di_range.c, omniwheel_service.c
ULTRA_DI_AXIS_Y	di_range.h, di_range.c, omniwheel_service.c
ULTRA_DI_AXIS_Z	di_range.h, di_range.c, omniplay_service.c
ULTRA_FLIGHT_AUTO_PROFILE	omni_client.h, main.c, settings.c
ULTRA_FLIGHT_ENABLE	namco_es3_main.c, omni_client.h, main.c
ULTRA_FLIGHT_FFB	omni_client.h, main.c, settings.c
ULTRA_FLIGHT_PLAYERS	main.c, omnisteer_service.c
ULTRA_FPS_CAP	andamiro_pcplus_main.c, d3d9_throttle_pcplus.c, cave_pc_main.c
ULTRA_FREEPLAY	ftdi_shim.h, ftdi_shim.h, ftdi_shim.h
ULTRA_FSR_SHARP	glx_emu.c, glx_emu.c, glx_emu.c
ULTRA_FULLSCREEN	d3d11_throttle_es3.c, d3d9_throttle.c, settings.c
ULTRA_GAMEDATA	jvs_serial_hook_adr.c, jvs_serial_hook_pcplus.c, jvs_serial_hook.c
ULTRA_GAME_CAL	jvs_input.c, jvs_input.c, jvs_input.c
ULTRA_GL_DRIVER	glx_emu.c, libgl_stub.c, lindbergh_main.c
ULTRA_HASH_BYTES	sign_module.c, verify_module.c, module_manifest.h
ULTRA_HOST_FULLSCREEN	main.c
ULTRA_HOST_HEIGHT	main.c
ULTRA_HOST_WIDTH	main.c
ULTRA_INPUT_PUMP	ultra_input_pump.h, main.c, ultra_input_pump.c
ULTRA_IO_ANALOG	hook_dllmain_pcplus.c, ultra_io_shared_pcplus.h, hook_dllmain.c
ULTRA_IO_ANALOG_OWNED	ultra_io_shared.h, ultra_io_shared.h, ultra_io_shared.h
ULTRA_IO_COINS	hook_dllmain_pcplus.c, ultra_io_shared_pcplus.h, hook_dllmain.c
ULTRA_IO_DIAG	ultra_io_publish.c

VARIABLE	REFERENCED IN
ULTRA_IO_ENVVAR	hook_dllmain_pcplus.c, ultra_io_shared_pcplus.h, hook_dllmain.c
ULTRA_IO_FLAG_HOOK_ATTACHED	hook_dllmain_pcplus.c, ultra_io_shared_pcplus.h, hook_dllmain.c
ULTRA_IO_FLAG_LOADER_ALIVE	ultra_io_shared_pcplus.h, input_src_gun.c, typex2_gun.c
ULTRA_IO_MAGIC	hook_dllmain_pcplus.c, ultra_io_shared_pcplus.h, hook_dllmain.c
ULTRA_IO_PLAYERS	hook_dllmain_pcplus.c, ultra_io_shared_pcplus.h, hook_dllmain.c
ULTRA_IO_PUBLISH	ultra_io_publish.h, ultra_io_publish.c
ULTRA_IO_SHM	hook_dllmain_pcplus.c, ultra_io_shared_pcplus.h, hook_dllmain.c
ULTRA_IO_VERSION	ultra_io_shared_pcplus.h, ultra_io_shared.h, ultra_io_shared_generic.h
ULTRA_ISHM_PADS	ultra_input_shm.h, ultra_input_pump.c, ultra_transport.c
ULTRA_ISHM_STICKS	ultra_input_shm.h, ultra_input_pump.c, ultra_transport.c
ULTRA_ISHM_WHEELS	ultra_input_shm.h, ultra_input_pump.c, ultra_transport.c
ULTRA_JVS_LOG	d3d9_throttle_pcplus.c, hook_dllmain_pcplus.c, jvs_serial_hook_pcplus.c
ULTRA_LIGHTGUN_ENABLE	lindbergh_main.c, hook_dllmain_generic.c, lindbergh_main.c
ULTRA_LIGHTGUN_XOFF	lindbergh_main.c, lindbergh_main.c, omni_client.h
ULTRA_LIGHTGUN_XSENS	lindbergh_main.c, lindbergh_main.c, omni_client.h
ULTRA_LIGHTGUN_YOFF	lindbergh_main.c, lindbergh_main.c, omni_client.h
ULTRA_LIGHTGUN_YSENS	lindbergh_main.c, lindbergh_main.c, omni_client.h
ULTRA_LOADER_LOG	ultra_io_publish.c
ULTRA_LOGO	resource_loader.h, settings.c
ULTRA_MAX_TRUSTED_KEYS	trusted_keys.h
ULTRA_MODEL2_CORE	main.c
ULTRA_MODULE	cave_pc_main.c, examu_exboard_main.c, lindbergh_main.c
ULTRA_MODULE_FALLBACK	main.c
ULTRA_MODULE_ID_MAX	sign_module.c, verify_module.c, module_manifest.h
ULTRA_MUTE_SPLASH	main.c, settings.c, splash.c
ULTRA_NAMCO_ENGLISH	jvs_serial_hook_es3.c, settings.c
ULTRA_NO_LOGFILE	main.c
ULTRA_NO_SPLASH	lindbergh_main.c, inject_main.c, inject_main.c
ULTRA_OV_AUTOCLOSE	pause_menu_service.c
ULTRA_OV_PAGE	pause_menu_service.c
ULTRA_OV_SHOT	pause_menu_service.c

VARIABLE	REFERENCED IN
ULTRA_PAD	omni_client.h, main.c, settings.c
ULTRA_PARALLEL_LOAD	main.c
ULTRA_PAUSE_AUDIO	settings.c
ULTRA_PLAY_AUTO_PROFILE	omni_client.h, main.c, settings.c
ULTRA_PLAY_ENABLE	omni_client.h, main.c, settings.c
ULTRA_PLAY_PLAYERS	main.c, omniplay_service.c
ULTRA_PRESENT_FILTER	glx_emu.c, glx_emu.c, glx_emu.c
ULTRA_PRESENT_HZ	settings.h, main.c, settings.c
ULTRA_PS3_FW_INSTALLER	settings.c
ULTRA_RES	d3d9_throttle_pcplus.c, d3d8_throttle.c, d3d9_throttle.c
ULTRA_RES_H	d3d9_throttle_pcplus.c, d3d8_throttle.c, d3d9_throttle.c
ULTRA_RES_W	d3d9_throttle_pcplus.c, d3d8_throttle.c, d3d9_throttle.c
ULTRA_ROTATE	board_profile.c, d3d8_throttle.c, d3d9_throttle.c
ULTRA_RUMBLE	omni_client.h, main.c, settings.c
ULTRA_SCENE_SCALE	lindbergh_main.c, maintrace.c, maintrace.c
ULTRA_SEGA_CONTENT_DIR	truecrypt_bypass.c, truecrypt_bypass_nu2.c, truecrypt_bypass.c
ULTRA_SHARPEN	d3d8_sharpen.c, d3d8_sharpen.h, d3d8_throttle.c
ULTRA_SIG_FORMAT_VERSION	sign_module.c, verify_module.c, module_manifest.h
ULTRA_SIG_MAGIC	sign_module.c, verify_module.c, module_manifest.h
ULTRA_SKIP_AUDIO	patcher.c, segaapi_stub.c, patcher.c
ULTRA_SKIP_JVS	patcher.c, patcher.c, patcher.c
ULTRA_SKIP_MOVIE	patcher.c, patcher.c, patcher.c
ULTRA_SM_FRAMEGEN	main.c, settings.c
ULTRA_SM_SHARPEN	settings.c
ULTRA_SM_SHARPNESS	settings.c
ULTRA_STICK	omni_client.h, main.c, settings.c
ULTRA_STICK_DIR_OFF	ultra_io_publish.c
ULTRA_STICK_DIR_ON	ultra_io_publish.c
ULTRA_TOUCH	hook_dllmain_typex3.c, ultra_touch.h, ultra_touch.c
ULTRA_TOUCH_BTN	ultra_touch.h, ultra_touch.c
ULTRA_TOUCH_CURSOR	ultra_touch.c
ULTRA_TOUCH_DEADZONE	ultra_touch.c
ULTRA_TOUCH_HZ	ultra_touch.h, ultra_touch.c
ULTRA_TOUCH_LOG	ultra_touch.h, ultra_touch.c
ULTRA_TOUCH_LX	ultra_touch.c
ULTRA_TOUCH_LY	ultra_touch.c
ULTRA_TOUCH_MOUSE	ultra_touch.c
ULTRA_TOUCH_PAD	ultra_touch.c

VARIABLE	REFERENCED IN
ULTRA_TOUCH_PAD_POINTER	ultra_touch.c
ULTRA_TOUCH_RX	ultra_touch.c
ULTRA_TOUCH_RY	ultra_touch.c
ULTRA_TOUCH_SLIDE	ultra_touch.c
ULTRA_TOUCH_SPEED	ultra_touch.c
ULTRA_TRACE	hook_dllmain_adr.c, hook_dllmain.c, hook_dllmain_examu.c
ULTRA_TRACE_ERRTAB	ultra_tracer.h, ultra_tracer.c
ULTRA_TRACE_LOG	ultra_tracer.h, ultra_touch.c, ultra_tracer.c
ULTRA_TRACE_MAXEXC	ultra_tracer.h, ultra_tracer.c
ULTRA_TRACE_VERBOSE	ultra_tracer.h, ultra_tracer.c
ULTRA_TRIGGER_THRESHOLD	omni_client.h, main.c, settings.c
ULTRA_UNIVERSAL_GAMEPAD	omni_client.h, main.c, settings.c
ULTRA_VITA_FW_REPORT	vita_firmware_check.h, main.c
ULTRA_VITA_HOMEBREW_MODE	main.c
ULTRA_VITA_VERBOSE_MISSING_MODULES	main.c
ULTRA_VSYNC	settings.c
ULTRA_WHEEL	omni_client.h, main.c, settings.c
ULTRA_WHEEL_AUTO_PROFILE	omni_client.h, main.c, settings.c
ULTRA_WHEEL_ENABLE	omni_client.h, main.c, settings.c
ULTRA_WHEEL_FFB	omni_client.h, main.c, settings.c
ULTRA_WHEEL_PLAYERS	main.c, omniwheel_service.c
ULTRA_WINDOWED	d3d9_throttle_pcplus.c, d3d8_throttle.c, d3d9_throttle.c

Troubleshooting

The failures that come up the most, and what each one usually means.

SYMPTOM	CAUSE AND FIX
Game starts, then exits with code 3221225477	That is 0xC0000005, an access violation. This is a <i>crash</i> , not a quit. Check [PREFLIGHT] first: a game running under the wrong board is the most common cause, because a board hook may carry per-game code that is meaningless against another executable.
Game runs perfectly but has no controls at all	Before adjusting any mapping, confirm the game's input path exists. A missing dependency DLL, or an input API the hook does not intercept (a game using DirectInput 7 where the hook only handles DirectInput 8), produces exactly this: everything else correct, no input.
Game renders black but audio is heard and game logic runs	Turn OFF the upscaler and set sharpening to 0. That post-process pass is the usual cause for issues on boards like Taito Type X and Taito Type X2 hardware. Also, if you have the “use English if supported” setting ON, that could potentially throw off a game, so you can also try to turn that off to see if it helps. Some games that do not have English language may not know what to do if the loader is asking for it to render in English, and the game may panic.
Game window is small in a corner	The game asked for a fixed backbuffer smaller than the window. Enable the upscaler, or set the output resolution explicitly.
remote LoadLibraryA timed out (wait=0x102)	Injection was blocked, almost always caused by your pesky antivirus behavior shielding. The reason this happens is because writing a DLL and injecting it is the same thing a virus or dropper does. Just add an exception for the Ultra Loader folder and the game folder to your antivirus and that should fix it.
[SIGCHECK] refuses a module	The module does not match its signature. If you rebuilt or edited a module, you must send it to me, or to a designated signer to get it re-signed before it can be loaded again. This is to protect the community and prevent gatekeeping and exploitation.
A board module is reported as not installed or the module refuses to load	The loader will say so and fall back to another module, which may run the wrong board's code. Install the correct module rather than relying on the fallback. Also make sure your arcade system folders are labelled correctly. Even a misspelling of Sega RingEdge as Sega Ringege (missed the “d”) can cause the loader not to find the right module, rendering your games unable to load.
Game reports itself OFFLINE	Expected for titles with network authentication when the network files are absent from the dump. Most such games are fully playable offline. I may or may not be working on a solution for true <i>universal</i> online multiplayer...but for now, most arcade games play offline just fine.