MAME Documentation

Release 0.289

MAMEdev Team

Oct 08, 2026

CONTENTS

1 What is MAME **3**
- 1.1 I. Purpose . . . 3
- 1.2 II. Cost . . . 4
- 1.3 III. Software Image Files . . . 4
- 1.4 IV. Derivative Works . . . 4
- 1.5 V. Official Contact Information . . . 4

2 Health Warnings **5**
- 2.1 Epilepsy Warning . . . 5

3 Getting MAME prepared **7**
- 3.1 An Introduction to MAME . . . 7
- 3.2 Purpose of MAME . . . 7
- 3.3 Systems Emulated by MAME . . . 7
- 3.4 Supported OS . . . 8
- 3.5 System Requirements . . . 8
- 3.6 BIOS Dumps and Software . . . 9
- 3.7 Installing MAME . . . 9
- 3.8 Compiling MAME . . . 9
- 3.9 Configuring MAME . . . 24

4 Basic MAME Usage and Configuration **27**
- 4.1 Using MAME . . . 27
- 4.2 MAME's User Interface . . . 28
- 4.3 Default Keyboard Controls . . . 36
- 4.4 MAME Menus . . . 51
- 4.5 How does MAME look for files? . . . 58
- 4.6 Front-ends . . . 64
- 4.7 About ROMs and Sets . . . 65
- 4.8 Common Issues and Questions (FAQ) . . . 67

5 MAME Command-line Usage and OS-Specific Configuration **75**
- 5.1 Universal Command-line Options . . . 75
- 5.2 Windows-Specific Command-line Options . . . 130
- 5.3 SDL-Specific Command-line Options . . . 131
- 5.4 Command-line Index . . . 133

6 Plugins **143**
- 6.1 Introduction . . . 143
- 6.2 Using plugins . . . 143
- 6.3 Included plugins . . . 144

7 Advanced configuration **155**
7.1 Multiple Configuration Files . . . 155
7.2 MAME Path Handling . . . 157
7.3 Shifter Toggle Disable . . . 158
7.4 BGFX Effects for (nearly) Everyone . . . 159
7.5 HLSL Effects for Windows . . . 162
7.6 GLSL Effects for *nix, OS X, and Windows . . . 170
7.7 Controller Configuration Files . . . 172
7.8 Stable Controller IDs . . . 177
7.9 Linux Lightguns . . . 179

8 MAME Debugger **181**
8.1 Introduction . . . 181
8.2 Debugger commands . . . 182
8.3 Specifying devices and address spaces . . . 219
8.4 Debugger expression syntax . . . 221

9 Lua Scripting Interface **227**
9.1 Introduction . . . 227
9.2 Features . . . 227
9.3 API reference . . . 228
9.4 Interactive Lua console tutorial . . . 309

10 MAME External Tools **313**
10.1 chdman – CHD (Compressed Hunks of Data) File Manager . . . 313
10.2 Imgtool - *A generic image manipulation tool for MAME* . . . 322
10.3 Imgtool Format Info . . . 325
10.4 Castool - *A generic cassette image manipulation tool for MAME* . . . 347
10.5 Floptool - *A generic floppy image manipulation tool for MAME* . . . 352
10.6 Other tools included with MAME . . . 355
10.7 Developer-focused tools included with MAME . . . 355

11 Contributing to MAME **357**
11.1 Testing and reporting bugs . . . 357
11.2 Contributing to MAME's source code . . . 358
11.3 Use of AI . . . 369

12 Technical Specifications **371**
12.1 MAME Naming Conventions . . . 371
12.2 MAME Layout Files . . . 372
12.3 MAME Layout Scripting . . . 394
12.4 Object Finders . . . 407
12.5 Input System . . . 424
12.6 The device_memory_interface . . . 431
12.7 The device_rom_interface . . . 434
12.8 The device_disasm_interface and the disassemblers . . . 435
12.9 The device_sound_interface . . . 439
12.10 Emulated system memory and address spaces management . . . 444
12.11 CPU devices . . . 459
12.12 The new floppy subsystem . . . 463
12.13 The new SCSI subsystem . . . 473
12.14 The new 6502 family implementation . . . 477
12.15 UML Instruction Reference . . . 485
12.16 Software 3D Rendering in MAME . . . 565
12.17 Audio effects . . . 584

12.18 OSD audio support 586
12.19 BGFX vector CRT renderer 591

13 MAME and security concerns **603**

14 The MAME License **605**

15 Contribute **607**



Note

This documentation is a work in progress. You can track the status of these topics through MAME's issue tracker. Learn how you can *contribute*.

CHAPTER

ONE

WHAT IS MAME

MAME is a multi-purpose emulation framework.

MAME's purpose is to preserve decades of software history. As electronic technology continues to rush forward, MAME prevents this important "vintage" software from being lost and forgotten. This is achieved by documenting the hardware and how it functions. The source code to MAME serves as this documentation. The fact that the software is usable serves primarily to validate the accuracy of the documentation (how else can you prove that you have recreated the hardware faithfully?). MAME, originally the Multiple Arcade Machine Emulator, absorbed sister-projects, including MESS (Multi-Emulator Super System) and AGEMAME (Arcade Gambling Extensions for MAME), so MAME now documents a wide variety of (mostly vintage) computers, video game consoles, calculators and gambling machines, in addition to the arcade video games that were its initial focus.

MAME®
Copyright © 1997-2026 MAMEdev and contributors
MAME is a registered trademark of Gregory Ember

1.1 I. Purpose

MAME's main purpose is to be a reference to the inner workings of the emulated machines. This is done both for educational purposes and for preservation purposes, in order to prevent historical software from disappearing forever once the hardware it runs on stops working. Of course, in order to preserve the software and demonstrate that the emulated behavior matches the original, one must also be able to actually use the software. This is considered a nice side effect, and is not MAME's primary focus.

It is not our intention to infringe on any copyrights or patents on the original software and systems. All of MAME's source code is either our own or freely available. To operate, the emulator requires images of the original ROMs, CDs, hard disks or other media from the machines, which must be provided by the user. No portions of the original software are included in the MAME executable.

1.2 II. Cost

MAME is made available at no cost. Its source code is freely available. The project as whole is distributed under the terms of the GNU General Public License, version 2 or later (GPL-2.0+), but most of code, including core functionality, is also available under the terms of the more permissive 3-clause BSD license (BSD-3-clause).

1.3 III. Software Image Files

ROM, CD, hard disk and other media images are all copyrighted material. They may not be lawfully distributed without the explicit permission of the copyright holder(s). They are not "abandonware", nor has copyright expired on any of the software supported by MAME.

MAME is not intended to be used as a tool for mass copyright infringement. Therefore, it is strongly against the authors' wishes to sell, advertise, or link to resources that provide illegal copies of ROM, CD, hard disk or other media images.

1.4 IV. Derivative Works

Because the name MAME is trademarked, you must abide by the rules set out for trademark usage if you wish to use "MAME" as part of the name your derivative work. In general, this means you must request permission, which requires that you follow the guidelines above.

The version number of any derivative work should reflect the version number of the MAME release from which it was derived.

1.5 V. Official Contact Information

For questions regarding the MAME license, trademark, or other usage, see https://www.mamedev.org/legal.html

CHAPTER

TWO

HEALTH WARNINGS

2.1 Epilepsy Warning

A very small percentage of individuals may experience epileptic seizures when exposed to certain light patterns or flashing lights. Exposure to certain patterns or backgrounds on a television screen or computer monitor, or while playing video games may induce an epileptic seizure in these individuals.

Certain conditions may induce previously undetected epileptic symptoms even in persons who have no history of prior seizures or epilepsy. These conditions can include emulation accuracy or inaccuracy, computer performance at the time of running MAME, video card drivers, your monitor, and a lot of other factors. If you, or anyone in your family, has an epileptic condition, consult your physician prior to using MAME.

If you experience any of the following while using MAME, **IMMEDIATELY** discontinue use and consult your physician before resuming use of MAME.

- Dizziness
- Altered vision
- Eye or muscle twitches
- Loss of awareness
- Disorientation
- Any involuntary movement
- Convulsions

CHAPTER

THREE

GETTING MAME PREPARED

This section covers initial preparation work needed to use MAME, including downloading MAME, compiling MAME from source, and configuring MAME.

3.1 An Introduction to MAME

MAME, formerly an acronym which stood for Multi Arcade Machine Emulator, documents and reproduces through emulation the inner components of arcade machines, computers, consoles, chess computers, calculators, and many other types of electronic amusement machines. As a nice side-effect, MAME allows to use on a modern PC those programs and games which were originally developed for the emulated machines.

At one point there were actually two separate projects, MAME and MESS. MAME covered arcade video games, while MESS covered home and business systems. They are now merged into the one MAME.

MAME is written in C++ and can currently emulate over 32,000 individual systems from the last five decades.

3.2 Purpose of MAME

The primary purpose of MAME is to preserve decades of arcade, computer, and console history. As technology continues to rush forward, MAME prevents these important "vintage" systems from being lost and forgotten.

3.3 Systems Emulated by MAME

The Arcade Database contains a complete list of the systems currently emulated. As you will notice, being supported does not always mean that the status of the emulation is perfect. You may want

1. to check the status of the emulation in the wiki pages of each system, accessible from the drivers page (e.g. for Apple Macintosh, from the page for the mac128.cpp driver you can reach the pages for both **macplus** and **macse**),
2. to read the corresponding **sysinfo.dat** entry in order to better understand which issues you may encounter while running a system in MAME (again, for Apple Macintosh Plus you have to check this entry).

Alternatively, you can simply see the status by yourself, launching the system emulation and taking a look at the red or yellow warning screen which appears before the emulation starts, if any. Notice that if you have information which can help to improve the emulation of a supported system, or if you can directly contribute fixes and/or addition to the current source, you can do any of the following:

- Send in a pull request (for code) or open an issue (information) on our GitHub page

- Post the information or code on the MAME Forums
- Follow the instructions on our contact page

3.4 Supported OS

The current source code can be directly compiled under all the main operating systems: Microsoft Windows (both with DirectX/BGFX native support or with SDL support), Linux, FreeBSD, and macOS.

3.5 System Requirements

MAME is written in C++, and has been ported to numerous platforms. Over time, as computer hardware has evolved, the MAME code has evolved as well to take advantage of the greater processing power and hardware capabilities offered.

The official MAME binary packages are compiled and designed to run on standard Windows-based systems. The minimum requirements are:

- An x86-64 CPU implementing the x86-64v2 feature set (16-byte compare/exchange, `lahf`/`sahf` instructions in long mode, population count instruction and SSE 4.2), or
- An Arm CPU implementing the ARMv8.2-A feature set
- A 64-bit edition of Windows 10 or later
- 4 GB RAM
- DirectX 9.0c for Windows
- A Direct3D or OpenGL capable GPU with support for non-power-of-two texture sizes

In general, any x86-64 CPU from 2015 onwards or the vast majority of 64-bit Arm CPUs from 2018 onwards should be suitable. It is possible to compile MAME yourself with support for older CPUs at the cost of some performance.

Of course, the minimum requirements are just that: minimal. You may not get optimal performance from such a system, but MAME should run. Modern versions of MAME require more power than older versions, so if you have a less-capable PC, you may find that using an older version of MAME may get you better performance, at the cost of greatly lowered accuracy and fewer supported systems.

MAME will take advantage of 3D hardware for compositing artwork and scaling displayed software to full screen. To make use of this, you should have at least a semi-modern computer with semi-modern 3D hardware made within the last five to ten years.

HLSL or GLSL special effects such as CRT simulation will put a very heavy load on your video card, especially at higher resolutions. You will need a fairly powerful modern video card, and the load on your video card goes up exponentially as your resolution increases. If HLSL or GLSL are too intensive, try reducing your output resolution.

Keep in mind that even on the fastest computers available, MAME is still incapable of playing some systems at full speed. The goal of the project isn't to make all system run speedy on your system; the goal is to document the hardware and reproduce the behavior of the hardware as faithfully as possible.

3.6 BIOS Dumps and Software

Most of the systems emulated by MAME requires a dump of the internal chips of the original system. These can be obtained by extracting the data from an original unit, or finding them (at your own risk) on various place on the Internet. Being copyrighted material, MAME does not come with any of these.

Also, you may want to find some software to be run on the emulated machine where it does not have internal software (e.g. some computers will need a disk to boot to an operating system).

Again, Google and other search engines are your best friends. MAME does not provide any software in the MAME package to be run on the emulated machines because it is very often (almost always, in the case of console software) protected by copyright.

The MAME team has been permitted to redistribute some old software, which can be found in the ROMS section of the MAME site.

3.7 Installing MAME

3.7.1 Microsoft Windows

You simply have to download the latest binary archive available from http://www.mamedev.org and to extract its content to a folder. You will end up with many files (below you will find explanations about some of these), and in particular **mame.exe**. This is a command line program. The installation procedure ends here. Easy, isn't it?

3.7.2 Other Operating Systems

In this case, you can either look for pre-compiled (SDL)MAME binaries (e.g. in the repositories of your favorite Linux distro) which should simply extract all the needed files in a folder you choose, or compile the source code by yourself. In the latter case, see our section on *compiling MAME*.

3.8 Compiling MAME

- *All Platforms*
  - *Compilation database*
- *Microsoft Windows*
  - *Using a standard MSYS2 installation*
  - *Building with Microsoft Visual Studio*
  - *Some notes about the MSYS2 environment*
- *Fedora Linux*
- *Debian and Ubuntu (including Raspberry Pi and ODROID devices)*
- *Arch Linux*
- *Apple macOS*
- *Emscripten Javascript and HTML*

- *Compiling the Documentation*
  - *Compiling the Documentation on Microsoft Windows*
  - *Compiling the Documentation on Debian and Ubuntu*
- *Useful Options*
  - *Overall build options*
  - *Tool locations*
  - *Including subsets of supported systems*
  - *Optional features*
  - *Compilation options*
  - *Library/framework locations*
- *Known Issues*
  - *GNU C Library fortify source feature*
  - *Issues affecting MinGW clang*
  - *Issues affecting Microsoft Visual Studio*
- *Unusual Build Configurations*
  - *Linking using the LLVM linker*
  - *Creating static libraries using the LLVM archiver*
  - *Cross-compiling MAME*
  - *Using libc++ on Linux*
  - *Using a GCC/GNU libstdc++ installation in a non-standard location on Linux*

3.8.1 All Platforms

- To compile MAME, you need a C++20 compiler and runtime library. We support building with GCC version 11 or later and clang version 13 or later. MAME should run with GNU libstdc++ version 11 or later or libc++ version 11 or later. The initial release of any major version of GCC should be avoided. For example, if you want to compile MAME with GCC 12, you should use version 12.1 or later.
- Whenever you are changing build parameters, (for example changing optimisation settings, or adding tools to the compile list), or system drivers sources are added, removed, or renamed, the project files need to be regenerated. You can do this by adding **REGENIE=1** to the make arguments, or updating the modification time of the makefile (for example using the **touch** command). Failure to do this may cause difficult to troubleshoot problems.
- If you want to add various additional tools to the compile, such as *chdman*, add a **TOOLS=1** to your make command, like **make REGENIE=1 TOOLS=1**
- You can build an emulator for a subset of the systems supported by MAME by using *SOURCES=<driver>,...* in your make command. For example **make SUBTARGET=pacem SOURCES=src/mame/pacman/pacman.cpp REGENIE=1** would build an emulator called *pacem* including the system drivers from the source file pacman.cpp (*REGENIE=1* is specified to ensure project files are regenerated). You can specify folders to include their entire contents, and you can separate multiple files/folders with commas. You can also omit the *src/mame/* prefix in many cases.

If you encounter linking errors after changing the included sources, delete the static libraries for the subtarget from the build folder. For the previous example on Windows using GCC, these would be in *build/mingw-gcc/bin/x64/Release/mame_pacem* by default.

- On a system with multiple CPU cores, compilation can be sped up by compiling multiple source files in parallel. This is done with the **-j** parameter. For instance, **make -j5** is a good starting point on a system with a quad-core CPU.

  Note: a good number to start with is the total number of CPU cores in your system plus one. An excessive number of concurrent jobs will increase compilation time, particularly if the compiler jobs exhaust available memory. The optimal number depends on many factors, including number of CPU cores, available RAM, disk and filesystem performance, and memory bandwidth.

- Debugging information can be added to a compile using *SYMBOLS=1* though most users will not want or need to use this. This increases compile time and disk space used. Note that a full build of MAME including internal debugging symbols will exceed the maximum size for an executable on Windows, and will not be possible to run without first stripping the symbols.

Putting all of these together, we get a couple of examples:

Rebuilding MAME on a dual-core (e.g. i3 or laptop i5) machine:

```
make -j3
```

Rebuilding MAME for just the Pac-Man and Galaxian families of systems, with tools, on a quad-core (e.g. i5 or i7) machine:

```
make SUBTARGET=pacem SOURCES=src/mame/pacman,src/mame/galaxian TOOLS=1 REGENIE=1 -j5
```

Rebuilding MAME for just the Apple II systems, compiling up to six sources in parallel:

```
make SUBTARGET=appulator SOURCES=apple/apple2.cpp,apple/apple2e.cpp,apple/apple2gs.cpp␣
↪REGENIE=1 -j6
```

Compilation database

To generate a `compile_commands.json` file for editors and tools such as clangd, add the **jcdb** target to your usual make command:

```
make OVERRIDE_CC=clang OVERRIDE_CXX=clang++ TOOLS=1 jcdb
```

This runs GENie's compilation database generator without compiling MAME. The database is written to the MAME source root, using the selected compiler, architecture and debug/release configuration. Build options such as **SOURCES**, **SUBTARGET**, **TOOLS**, **DEBUG**, and **PTR64** apply as usual. The build dependencies needed to generate project files must be installed.

Run the command again after changing build options or adding or removing source files. Each invocation replaces the database with the selected configuration; **REGENIE=1** is not required. Generated source files and headers still need to be produced by a normal build before tools can analyse files that use them.

3.8.2 Microsoft Windows

The information here is very detailed, and assumes you're aware of the options available and what they mean. As an alternative, we also provide a tutorial for compiling MAME on Windows on our web site.

MAME for Windows is built using the MSYS2 environment. You will need a 64-bit version of Windows 10 1809 or later and a reasonably up-to-date MSYS2 installation. Building for 64-bit ARM (AArch64) requires a 64-bit ARM system running Windows 11 or later.

- By default, MAME will be built using native Windows OS interfaces for window management, audio/video output, font rendering, etc. If you want to use the portable SDL (Simple DirectMedia Layer) interfaces instead, you can add **OSD=sdl** to the make options. The main emulator binary will have an `sdl` prefix prepended (e.g. `sdlmame.exe`). You will need to install the MSYS2 packages for SDL 2 version 2.0.14 or later.
- By default, MAME will include the native Windows debugger. To also include the portable Qt debugger, add **USE_QTDEBUG=1** to the make options. You will need to install the MSYS2 packages for Qt 6.

Using a standard MSYS2 installation

MAME is built for Windows using a standard MSYS2 installation with additional packages installed to provide the development tools needed for building MAME. These instructions assume you have some familiarity with MSYS2 and the **pacman** package manager.

- Install the MSYS2 environment from the MSYS2 homepage.
- Install packages necessary to build MAME. At the very least, you'll need `bash`, `git` and `make`.
- To generate API documentation from source, you'll need `doxygen`.
- If you plan to rebuild bgfx shaders and you want to rebuild the GLSL parser, you'll need `bison`.

The additional packages you'll need depend on the CPU architecture you're building for.

64-bit x86-64 (libstdc++/UCRT)

- You'll need `mingw-w64-ucrt-x86_64-gcc` and `mingw-w64-ucrt-x86_64-python`.
- To compile using the clang compiler, you'll also need `mingw-w64-ucrt-x86_64-clang`.
- To use the LLVM linker and archiver (generally much faster than the GNU linker and archiver), you'll need `mingw-w64-ucrt-x86_64-lld`, `mingw-w64-ucrt-x86_64-llvm-tools`, `mingw-w64-ucrt-x86_64-llvm` and `mingw-w64-ucrt-x86_64-libc++`.
- For debugging, you may want to install `mingw-w64-ucrt-x86_64-gdb`.
- To build against the portable SDL interfaces, you'll need `mingw-w64-ucrt-x86_64-SDL2` and `mingw-w64-ucrt-x86_64-SDL2_ttf`.
- To build the Qt debugger, you'll need `mingw-w64-ucrt-x86_64-qt6-base`.
- To build the HTML user/developer documentation, you'll need `mingw-w64-ucrt-x86_64-librsvg`, `mingw-w64-ucrt-x86_64-python-sphinx`, `mingw-w64-ucrt-x86_64-python-sphinx_rtd_theme` and `mingw-w64-ucrt-x86_64-python-sphinxcontrib-svg2pdfconverter`.
- To build the PDF documentation, you'll additionally need `mingw-w64-ucrt-x86_64-texlive-latex-extra` and `mingw-w64-ucrt-x86_64-texlive-fonts-recommended`.
- Open the **ucrt64.exe** helper from the **msys64** installation folder or the **MSYS2 UCRT64** shortcut from the start menu to start a Bash shell configured with the correct paths and environment variables.

64-bit x86-64 (libc++/UCRT)

- You'll need `mingw-w64-clang-x86_64-clang`, `mingw-w64-clang-x86_64-python` and `mingw-w64-clang-x86_64-gcc-compat`.
- To use the LLVM linker and archiver (generally much faster than the GNU linker and archiver), you'll need `mingw-w64-clang-x86_64-lld`, `mingw-w64-clang-x86_64-llvm-tools`, `mingw-w64-clang-x86_64-llvm` and `mingw-w64-clang-x86_64-libc++`.
- For debugging, you may want to install `mingw-w64-clang-x86_64-gdb`.
- To build against the portable SDL interfaces, you'll need `mingw-w64-clang-x86_64-SDL2` and `mingw-w64-clang-x86_64-SDL2_ttf`.
- To build the Qt debugger, you'll need `mingw-w64-clang-x86_64-qt6-base`.
- To build the HTML user/developer documentation, you'll need `mingw-w64-clang-x86_64-librsvg`, `mingw-w64-clang-x86_64-python-sphinx`, `mingw-w64-clang-x86_64-python-sphinx_rtd_theme` and `mingw-w64-clang-x86_64-python-sphinxcontrib-svg2pdfconverter`.
- To build the PDF documentation, you'll additionally need `mingw-w64-clang-x86_64-texlive-latex-extra` and `mingw-w64-clang-x86_64-texlive-fonts-recommended`.
- Open the **clang64.exe** helper from the **msys64** installation folder or use the **MSYS2 CLANG64** shortcut to start a Bash shell configured with the correct paths and environment variables.

64-bit ARM (libc++/UCRT)

- You'll need `mingw-w64-clang-aarch64-clang`, `mingw-w64-clang-aarch64-python` and `mingw-w64-clang-aarch64-gcc-compat`.
- To use the LLVM linker and archiver (generally much faster than the GNU linker and archiver), you'll need `mingw-w64-clang-aarch64-lld`, `mingw-w64-clang-aarch64-llvm-tools`, `mingw-w64-clang-aarch64-llvm` and `mingw-w64-clang-aarch64-libc++`.
- For debugging, you may want to install `mingw-w64-clang-aarch64-lldb`.
- To build against the portable SDL interfaces, you'll need `mingw-w64-clang-aarch64-SDL2` and `mingw-w64-clang-aarch64-SDL2_ttf`.
- To build the Qt debugger, you'll need `mingw-w64-clang-aarch64-qt6-base`.
- To build the HTML user/developer documentation, you'll need `mingw-w64-clangarm64-x86_64-librsvg`, `mingw-w64-clangarm64-x86_64-python-sphinx`, `mingw-w64-clangarm64-x86_64-python-sphinx_rtd_theme` and `mingw-w64-clangarm64-x86_64-python-sphinxcontrib-svg2pdfconverter`.
- It is not currently possible to build the PDF documentation using the CLANGARM64 environment as the TeX Live tools do not work. You may be able to use the UCRT64 or CLANG64 environment to build the PDF documentation on a 64-bit ARM system.
- Open the **clangarm64.exe** helper from the **msys64** installation folder or use the **MSYS2 CLANGARM64** shortcut to start a Bash shell configured with the correct paths and environment variables.

32-bit x86

It is no longer possible to compile 32-bit x86 MAME build using an MSYS2 environment. Up-to-date 32-bit versions of GCC exceed the available address space when compiling MAME's address space classes and Lua bindings. MSYS2 no longer provides 32-bit x86 LLVM and clang packages. To build MAME for 32-bit x86 systems, you must use an environment that allows cross-compiling for a 32-bit target using 64-bit tools (e.g. using a MinGW environment on a Linux system).

Building with Microsoft Visual Studio

- You can generate Visual Studio 2022 projects using **make vs2022**. The solution and project files will be created in `build/projects/windows/mame/vs2022` by default (the name of the `build` folder can be changed using the `BUILDDIR` option). This will always regenerate the projects, so **REGENIE=1** is *not* needed.
- Adding **MSBUILD=1** to the make options will build the solution using the Microsoft Build Engine after generating the project files. Note that this requires paths and environment variables to be configured so the correct Visual Studio tools can be located; please refer to the Microsoft-provided instructions on using the Microsoft C++ toolset from the command line. You may find it easier to not use **MSBUILD=1** and load the project file into Visual Studio's GUI for compilation.
- The MSYS2 environment is still required to generate the project files, convert built-in layouts, compile UI translations, etc.

Some notes about the MSYS2 environment

MSYS2 uses the pacman tool from Arch Linux for package management. There is a page on the Arch Linux wiki with helpful information on using the pacman package management tool.

The MSYS2 environment includes two kinds of tools: MSYS2 tools designed to work in a UNIX-like environment on top of Windows, and MinGW tools designed to work in a more Windows-like environment. The MSYS2 tools are installed in `/usr/bin` while the MinGW tools are installed in `/ucrt64/bin`, `/clang64/bin` and/or `/clangarm64/bin` (relative to the MSYS2 installation directory). MSYS2 tools work best in an MSYS2 terminal, while MinGW tools work best in a Microsoft command prompt.

The most obvious symptom of this is that arrow keys don't work in interactive programs if you run them in the wrong kind of terminal. If you run MinGW GDB or python from an MSYS2 terminal window, command history won't work and it may not be possible to interrupt an attached program with gdb. Similarly it may be very difficult to edit using MSYS2 vim in a Microsoft command prompt window.

MAME is built using the MinGW compilers, so the MinGW directories are included earlier in the `PATH` environment variable for the build environments. If you want to use an interactive MSYS2 program from an MSYS2 shell, you may need to type the absolute path to avoid using the MinGW equivalent instead.

MSYS2 GDB has issues debugging MinGW programs like MAME. You will get better results by installing the MinGW version of GDB or LLDB and running it from a Windows Command Prompt window to debug MAME.

GNU Make supports both POSIX-style shells (e.g. bash) and the Microsoft cmd.exe shell. When using the cmd.exe shell, the `copy` command doesn't provide a useful exit status, so file copy tasks can fail silently. This may cause your build to appear to succeed while producing incorrect results. The command to link MAME exceeds the maximum command length supported by the cmd.exe shell, so you will not be able to produce a complete build of MAME. As such, we recommend that you *do not* install MinGW versions of GNU Make, and use the MSYS2 version of GNU Make from the `make` package.

A complete build of MAME including line number symbols exceeds the size limit imposed by the PE file format and cannot be run. Workarounds include including only a subset of the systems supported by MAME, extracting symbols to a separate file and stripping excess symbols from the MAME executable, or compiling MAME using clang with the option to place symbols in external PDB files.

3.8.3 Fedora Linux

You'll need a few prerequisites from your Linux distribution. Make sure you get SDL 2 version 2.0.14 or later as earlier versions lack required functionality:

```
sudo dnf install git gcc gcc-c++ SDL2-devel SDL2_ttf-devel libXi-devel libXinerama-devel␣
↪qt6-qtbase-devel expat-devel fontconfig-devel alsa-lib-devel pulseaudio-libs-devel
```

If you want to use the more efficient LLVM tools for archiving static libraries and linking, you'll need to install the corresponding packages:

```
sudo dnf install lld llvm
```

Compilation is exactly as described above in All Platforms.

To build the HTML user/developer documentation, you'll need Sphinx, as well as the theme and the SVG converter:

```
sudo dnf install python3-sphinx python3-sphinx_rtd_theme python3-sphinxcontrib-
↪rsvgconverter
```

The HTML documentation can be built with this command:

```
make -C docs html
```

3.8.4 Debian and Ubuntu (including Raspberry Pi and ODROID devices)

You'll need a few prerequisites from your Linux distribution. Make sure you get SDL 2 version 2.0.14 or later as earlier versions lack required functionality:

```
sudo apt-get install git build-essential python3 libsdl2-dev libsdl2-ttf-dev␣
↪libfontconfig-dev libpulse-dev qt6-base-dev qt6-base-dev-tools qtchooser
```

Compilation is exactly as described above in All Platforms. Note the Ubuntu Linux modifies GCC to enable the GNU C Library "fortify source" feature by default, which may cause issues compiling MAME (see *Known Issues*).

3.8.5 Arch Linux

You'll need a few prerequisites from your distro:

```
sudo pacman -S base-devel git sdl2_ttf python libxinerama libpulse alsa-lib qt6-base
```

Compilation is exactly as described above in All Platforms.

3.8.6 Apple macOS

You'll need a few prerequisites to get started. Make sure you're on macOS 14.5 "Sonoma" or later and Xcode 16.2 or later. You will need SDL 2 version 2.0.14 or later. You'll also need to install Python 3 – it's currently included with the Xcode command line tools, but you can also install a stand-alone version or get it via the Homebrew package manager.

- Install **Xcode** from the Mac App Store or ADC (AppleID required).
- To find the corresponding Xcode for your MacOS release please visit xcodereleases.com to find the latest version of Xcode available to you.

- Launch **Xcode**. It will download a few additional prerequisites. Let this run through before proceeding.
- Once that's done, quit **Xcode** and open a **Terminal** window.
- Type **xcode-select --install** to install additional tools necessary for MAME (also available as a package on ADC).

Next you'll need to get SDL 2 installed.

- Go to this site and download the *macOS* .dmg file
- If the .dmg doesn't open automatically, open it
- Click "Macintosh HD" (or whatever your Mac's hard disk is named) in the left pane of a **Finder** window, then open the **Library** folder and drag the **SDL2.framework** folder from the SDL disk image into the **Frameworks** folder. You will have to authenticate with your user password.

If you don't already have it, get Python 3 set up:

- Go to the official Python site, navigate to the releases for macOS, and click the link to download the installer for the latest stable release (this was Python 3.10.4 at the time of writing).
- Scroll down to the "Files" section, and download the macOS version (called "macOS 64-bit universal2 installer" or similar).
- Once the package downloads, open it and follow the standard installation process.

Finally to begin compiling, use Terminal to navigate to where you have the MAME source tree (*cd* command) and follow the normal compilation instructions from above in All Platforms.

3.8.7 Emscripten Javascript and HTML

First, download and install Emscripten 6.0.2 or later by following the instructions at the official site.

Once Emscripten has been installed, use **source emsdk_env.sh** or **emsdk_env.bat** to set environment variables. Since MAME requires SDL libraries, prepare them with

```
embuilder build sdl3 sdl3_ttf
```

After dependencies are compiled, it should be possible to compile MAME out-of-the-box using Emscripten's **emmake** tool. Because a full MAME compile is too large to load into a web browser at once, you will want to use the SOURCES parameter to compile only a subset of the project, e.g. (in the MAME directory):

```
emmake make SUBTARGET=pacmantest SOURCES=src/mame/pacman/pacman.cpp
```

The **SOURCES** parameter should have the path to at least one driver **.cpp** file. The make process will attempt to locate and include all dependencies necessary to produce a complete build including the specified driver(s). However, sometimes it is necessary to manually specify additional files (using commas) if this process misses something. e.g.

```
emmake make SUBTARGET=apple2e SOURCES=src/mame/apple/apple2e.cpp,src/devices/machine/
↪applefdc.cpp
```

The value of the **SUBTARGET** parameter serves only to differentiate multiple builds and need not be set to any specific value.

Other make parameters can also be used, e.g. **-j** for multithreaded compilation as described earlier.

When the compilation reaches the emcc phase, you may see a number of *"unresolved symbol"* warnings. At the moment, this is expected for OpenGL-related functions such as glPointSize. Any others may indicate that an additional dependency file needs to be specified in the **SOURCES** list. Unfortunately this process is not automated and you will need to search the source tree to locate the files supplying the missing symbols. You may also be able to get away with ignoring the warnings if the code path referencing them is not used at run-time.

If all goes well, a **.js** file will be output to the current directory. This file cannot be run by itself, but requires an HTML loader to provide it with a canvas to draw to and to pass in command-line parameters. The Emularity project provides such a loader.

There are example **.html** files in that repository which can be edited to point to your newly compiled MAME **.js** file and pass in whatever parameters you desire. You will then need to place all of the following on a web server:

- The compiled MAME **.js** file
- The compiled MAME **.wasm** file if using WebAssembly
- The **.js** files from the Emularity package (**loader.js**, **browserfs.js**, etc.)
- A **.zip** file with the ROMs for the MAME driver you would like to run (if any)
- Any software files you would like to run with the MAME driver
- An Emularity loader **.html** modified to point to all of the above

You need to use a web server instead of opening the local files directly due to security restrictions in modern web browsers.

If the result fails to run, you can open the Web Console in your browser to see any error output which may have been produced (e.g. missing or incorrect ROM files). A "ReferenceError: foo is not defined" error most likely indicates that a needed source file was omitted from the **SOURCES** list.

3.8.8 Compiling the Documentation

Compiling the documentation will require you to install several packages depending on your operating system.

Compiling the Documentation on Microsoft Windows

On Windows, you'll need a couple of packages from the MSYS2 environment. You can install these packages with

```
pacman -S mingw-w64-x86_64-librsvg mingw-w64-x86_64-python-sphinx mingw-w64-x86_64-
↪python-sphinxcontrib-svg2pdfconverter
```

If you intend to make a PDF via LaTeX, you'll need to install a LaTeX distribution such as TeX Live:

```
pacman -S mingw-w64-x86_64-texlive-fonts-recommended mingw-w64-x86_64-texlive-latex-extra
```

Compiling the Documentation on Debian and Ubuntu

On Debian/Ubuntu flavors of Linux, you'll need **python3-sphinx/python-sphinx** and the **python3-pip/python-pip** packages:

```
sudo apt-get install python3-sphinx python3-pip
pip3 install sphinxcontrib-svg2pdfconverter
```

On Debian, you'll need to install the **librsvg2-bin** package:

```
sudo apt-get install librsvg2-bin
```

If you intend to make a PDF via LaTeX, you'll need to install a LaTeX distribution such as TeX Live:

```
sudo apt-get install librsvg2-bin latexmk texlive texlive-science texlive-formats-extra
```

From this point you can do `make html` or `make latexpdf` from the **docs** folder to generate the output of your choice. Typing `make` by itself will tell you all available formats. The output will be in the docs/build folder in a subfolder based on the type chosen (e.g. `make html` will create *docs/build/html* with the output.)

3.8.9 Useful Options

This section summarises some of the more useful options recognised by the main makefile. You use these options by appending them to the **make** command, setting them as environment variables, or adding them to your prefix makefile. Note that in order to apply many of these settings when rebuilding, you need to set **REGENIE=1** the first time you build after changing the option(s). Also note that GENie *does not* automatically rebuild affected files when you change an option that affects compiler settings.

Overall build options

PREFIX_MAKEFILE
: Name of a makefile to include for additional options if found (defaults to **useroptions.mak**). May be useful if you want to quickly switch between different build configurations.

BUILDDIR
: Set to change the name of the subfolder used for project files, generated sources, object files, and intermediate libraries (defaults to **build**).

REGENIE
: Set to **1** to force project files to be regenerated.

VERBOSE
: Set to **1** to show full commands when using GNU Make as the build tool. This option applies immediately without needing regenerate project files.

IGNORE_GIT
: Set to **1** to skip the working tree scan and not attempt to embed a Git revision description in the version string.

Tool locations

OVERRIDE_CC
: Set the C/Objective-C compiler command. (This sets the target C compiler command when cross-compiling.)

OVERRIDE_CXX
: Set the C++/Objective-C++ compiler command. (This sets the target C++ compiler command when cross-compiling.)

OVERRIDE_LD
: Set the linker command. This is often not necessary or useful because the C or C++ compiler command is used to invoke the linker. (This sets the target linker command when cross-compiling.)

OVERRIDE_AR
: Set static library archiver command.

PYTHON_EXECUTABLE
: Set the Python interpreter command. You need Python 3.2 or later to build MAME.

CROSS_BUILD
: Set to **1** to use separate host and target compilers and linkers, as required for cross-compilation. In this case, **OVERRIDE_CC**, **OVERRIDE_CXX** and **OVERRIDE_LD** set the target C compiler, C++ compiler and linker commands, while **CC**, **CXX** and **LD** set the host C compiler, C++ compiler and linker commands.

Including subsets of supported systems

SUBTARGET
Set emulator subtarget to build. Some pre-defined subtargets are provided, using Lua scripts in *scripts/target/mame* and system driver filter files in *src/mame*. User-defined subtargets can be created using the **SOURCES** or **SOURCEFILTER** option.

SOURCES
Specify system driver source files and/or folders to include. Usually used in conjunction with the **SUBTARGET** option. Separate multiple files/folders with commas.

SOURCEFILTER
Specify a system driver filter file. Usually used in conjunction with the **SUBTARGET** option. The filter file can specify source files to include system drivers from, and individual system drivers to include or exclude. There are some example system driver filter files in the *src/mame* folder.

Optional features

TOOLS
Set to **1** to build additional tools along with the emulator, including **unidasm**, **chdman**, **romcmp**, and **srcclean**.

EMULATOR
When set to **0**, the main emulator target will not be created. This is intended to be used in conjunction with setting **TOOLS** to **1** to build the additional tools without building the emulator.

NO_OPENGL
Set to **1** to disable building the OpenGL video output module.

NO_USE_PORTAUDIO
Set to **1** to disable building the PortAudio sound output module and the PortAudio library.

NO_USE_PULSEAUDIO
Set to **1** to disable building the PulseAudio sound output module on Linux.

USE_WAYLAND
Set to **1** to include support for bgfx video output with the Wayland display server.

USE_TAPTUN
Set to **1** to include the tap/tun network module, or set to **0** to disable building the tap/tun network module. The tap/tun network module is included by default on Windows and Linux.

USE_PCAP
Set to **1** to include the pcap network module, or set to **0** to disable building the pcap network module. The pcap network module is included by default on macOS and NetBSD.

USE_QTDEBUG
Set to **1** to include the Qt debugger on platforms where it's not built by default (e.g. Windows or macOS), or to **0** to disable it. You'll need to install Qt development libraries and tools to build the Qt debugger. The process depends on the platform.

Compilation options

NOWERROR
Set to **1** to disable treating compiler warnings as errors. This may be needed in marginally supported configurations.

DEPRECATED
Set to **0** to disable deprecation warnings (note that deprecation warnings are not treated as errors).

DEBUG
Set to **1** to enable runtime assertion checks and additional diagnostics. Note that this has a performance cost, and is most useful for developers.

OPTIMIZE
Set optimisation level. The default is **3** to favour performance at the expense of larger executable size. Set to **0** to disable optimisation (can make debugging easier), **1** for basic optimisation that doesn't have a space/speed trade-off and doesn't have a large impact on compile time, **2** to enable most optimisation that improves performance and reduces size, or **s** to enable only optimisations that generally don't increase executable size. The exact set of supported values depends on your compiler.

SYMBOLS
Set to **1** to include additional debugging symbols over the default for the target platform (many target platforms include function name symbols by default).

SYMLEVEL
Numeric value that controls the level of detail in debugging symbols. Higher numbers make debugging easier at the cost of increased build time and executable size. The supported values depend on your compiler. For GCC and similar compilers, **1** includes line number tables and external variables, **2** also includes local variables, and **3** also includes macro definitions.

PDB_SYMBOLS
Set to **1** to generate CodeView format symbols in separate PDB files, allowing source-level debugging using Microsoft Visual Studio or WinDbg. It can also be used with other tools that can load symbols from PDB files, e.g. the Intel VTune and AMD μProf performance analysis tools. This option is only supported for MinGW builds using the clang compiler and the LLVM linker (lld). This option only takes effect if the **SYMBOLS** option is set to a non-zero value.

ARCHOPTS
Additional command-line options to pass to the compiler and linker. This is useful for supplying code generation or ABI options, for example to enable support for optional CPU features.

ARCHOPTS_C
Additional command-line options to pass to the compiler when compiling C source files.

ARCHOPTS_CXX
Additional command-line options to pass to the compiler when compiling C++ source files.

ARCHOPTS_OBJC
Additional command-line options to pass to the compiler when compiling Objective-C source files.

ARCHOPTS_OBJCXX
Additional command-line options to pass to the compiler when compiling Objective-C++ source files.

Library/framework locations

SDL_INSTALL_ROOT
SDL installation root directory for shared library style SDL.

SDL_FRAMEWORK_PATH
Search path for SDL framework.

USE_LIBSDL
Set to **1** to use shared library style SDL on targets where framework is default.

USE_SYSTEM_LIB_ASIO
Set to **1** to prefer the system installation of the Asio C++ asynchronous I/O library over the version provided with the MAME source.

USE_SYSTEM_LIB_EXPAT
Set to **1** to prefer the system installation of the Expat XML parser library over the version provided with the MAME source.

USE_SYSTEM_LIB_ZLIB
Set to **1** to prefer the system installation of the zlib data compression library over the version provided with the MAME source.

USE_SYSTEM_LIB_ZSTD
Set to **1** to prefer the system installation of the Zstandard data compression library over the version provided with the MAME source.

USE_SYSTEM_LIB_JPEG
Set to **1** to prefer the system installation of the libjpeg image compression library over the version provided with the MAME source.

USE_SYSTEM_LIB_FLAC
Set to **1** to prefer the system installation of the libFLAC audio compression library over the version provided with the MAME source.

USE_SYSTEM_LIB_LUA
Set to **1** to prefer the system installation of the embedded Lua interpreter over the version provided with the MAME source.

USE_SYSTEM_LIB_SQLITE3
Set to **1** to prefer the system installation of the SQLITE embedded database engine over the version provided with the MAME source.

USE_SYSTEM_LIB_PORTMIDI
Set to **1** to prefer the system installation of the PortMidi library over the version provided with the MAME source.

USE_SYSTEM_LIB_PORTAUDIO
Set to **1** to prefer the system installation of the PortAudio library over the version provided with the MAME source.

USE_SYSTEM_LIB_UTF8PROC
Set to **1** to prefer the system installation of the Julia utf8proc library over the version provided with the MAME source.

USE_SYSTEM_LIB_GLM
Set to **1** to prefer the system installation of the GLM OpenGL Mathematics library over the version provided with the MAME source.

USE_SYSTEM_LIB_RAPIDJSON
Set to **1** to prefer the system installation of the Tencent RapidJSON library over the version provided with the MAME source.

USE_SYSTEM_LIB_PUGIXML
Set to **1** to prefer the system installation of the pugixml library over the version provided with the MAME source.

3.8.10 Known Issues

GNU C Library fortify source feature

The GNU C Library has options to perform additional compile- and run-time checks on string operations, enabled by defining the `_FORTIFY_SOURCE` preprocessor macro. This is intended to improve security at the cost of a small amount of overhead. MAME is not secure software, and we do not support building with `_FORTIFY_SOURCE` defined.

Some Linux distributions (including Gentoo and Ubuntu) have patched GCC to define `_FORTIFY_SOURCE` to `1` as a built-in macro. This is problematic for more projects than just MAME, as it makes it hard to disable the additional checks (e.g. if you don't want the performance impact of the run-time checks), and it also makes it hard to define `_FORTIFY_SOURCE` to `2` if you want to enable stricter checks. You should really take it up with the distribution maintainers, and make it clear you don't want non-standard GCC behaviour. It would be better if these distributions defined this macro by default in their packaging environments if they think it's important, rather than trying to force it on everything compiled on their distributions. (This is what Red Hat does: the `_FORTIFY_SOURCE` macro is set in the RPM build environment, and not by distributing a modified version of GCC.)

If you get compilation errors in `bits/string_fortified.h` you should first ensure that the `_FORTIFY_SOURCE` macro is defined via the environment (e.g. a **CFLAGS** or **CXXFLAGS** environment variable). You can check to see whether the `_FORTIFY_SOURCE` macro is a built-in macro with your version of GCC with a command like this:

gcc -dM -E - < /dev/null | grep _FORTIFY_SOURCE

If `_FORTIFY_SOURCE` is defined to a non-zero value by default, you can work around it by adding **-U_FORTIFY_SOURCE** to the compiler flags (e.g. by using the **ARCHOPTS** setting, or setting the **CFLAGS** and **CXXFLAGS** environment variables.

Issues affecting MinGW clang

MinGW clang and llvm can give spurious undefined symbol errors when linking using CodeView format symbols with high symbol detail levels. If you encounter undefined symbol errors when linking with **PDB_SYMBOLS=1** to produce CodeView format symbols, try setting **SYMLEVEL=1** to reduce the symbol detail level (line number tables will still be included, but local variables will be omitted).

Issues affecting Microsoft Visual Studio

The MSVC compiler produces spurious warnings about potentially uninitialised local variables. You currently need to add `NOWERROR=1` to the options passed to make when generating the Visual Studio project files. This stops warnings from being treated as errors. (MSVC seems to lack options to control which specific warnings are treated as errors, which other compilers support.)

3.8.11 Unusual Build Configurations

Linking using the LLVM linker

The LLVM linker is generally faster than the GNU linker that GCC uses by default. This is more pronounced on systems with a high overhead for file system operations (e.g. Microsoft Windows, or when compiling on a disk mounted over a network). To use the LLVM linker with GCC, ensure the LLVM linker is installed and add `-fuse-ld=lld` to the linker options (e.g. in the **LDFLAGS** environment variable or in the **ARCHOPTS** setting).

Creating static libraries using the LLVM archiver

The LLVM archiver is generally faster than the GNU archiver that is used by default. This is more pronounced on systems with a high overhead for file system operations (e.g. Microsoft Windows, or when compiling on a disk mounted over a network). To use the LLVM archiver, ensure the LLVM archiver is installed and add `OVERRIDE_AR=llvm-ar` to the options passed to make when generating the project files.

Cross-compiling MAME

MAME's build system has basic support for cross-compilation. Set **CROSS_BUILD=1** to enable separate host and target compilers, set **OVERRIDE_CC** and **OVERRIDE_CXX** to the target C/C++ compiler commands, and if necessary set **CC** and **CXX** to the host C/C++ compiler commands. If the target OS is different to the host OS, set it with **TARGETOS**. For example it may be possible to build a MinGW32 x64 build on a Linux host using a command like this:

```
make TARGETOS=windows PTR64=1 OVERRIDE_CC=x86_64-w64-mingw32-gcc OVERRIDE_CXX=x86_64-w64-
↪mingw32-g++ OVERRIDE_LD=x86_64-w64-mingw32-ld MINGW64=/usr**
```

(The additional packages required for producing a standard MinGW32 x64 build on a Fedora Linux host are `mingw64-gcc-c++`, `mingw64-winpthreads-static` and their dependencies. Non-standard builds may require additional packages.)

Using libc++ on Linux

MAME may be built using the LLVM project's "libc++" C++ Standard Library. The prerequisites are a working clang/LLVM installation, and the libc++ development libraries. On Fedora Linux, the necessary packages are **libcxx**, **libcxx-devel**, **libcxxabi** and **libcxxabi-devel**. Set the C and C++ compiler commands to use clang, and add **-stdlib=libc++** to the C++ compiler and linker options. You could use a command like this:

```
env LDFLAGS=-stdlib=libc++ make OVERRIDE_CC=clang OVERRIDE_CXX=clang++ ARCHOPTS_CXX=-
↪stdlib=libc++ ARCHOPTS_OBJCXX=-stdlib=libc++
```

The options following the **make** command may be placed in a prefix makefile if you want to use this configuration regularly, but **LDFLAGS** needs to be be set in the environment.

Using a GCC/GNU libstdc++ installation in a non-standard location on Linux

GCC may be built and installed to a custom location, typically by supplying the **--prefix=** option to the **configure** command. This may be useful if you want to build MAME on a Linux distribution that still uses a version of GNU libstdc++ that predates C++20 support. To use an alternate GCC installation to build MAME, set the C and C++ compilers to the full paths to the **gcc** and **g++** commands, and add the library path to the run-time search path. If you installed GCC in /opt/local/gcc72, you might use a command like this:

```
make OVERRIDE_CC=/opt/local/gcc72/bin/gcc OVERRIDE_CXX=/opt/local/gcc72/bin/g++␣
↪ARCHOPTS=-Wl,-R,/opt/local/gcc72/lib64
```

You can add these options to a prefix makefile if you plan to use this configuration regularly.

3.9 Configuring MAME

- *Getting Started: A Quick Preface*
- *Initial Setup: Creating mame.ini From Command Line on Windows*
- *Initial Setup: Creating mame.ini From Command Line on Linux or MacOS*
- *Initial Setup: Graphical Setup*

3.9.1 Getting Started: A Quick Preface

Once you have MAME installed, the next step is to configure it. There are several ways to do this, and each will be covered in turn.

If you are on Windows, the MAME executable will be called `mame.exe`.

If you are on Linux or MacOS, the MAME executable will be called `mame`.

3.9.2 Initial Setup: Creating mame.ini From Command Line on Windows

First, you will need to *cd* to the directory where you installed MAME into. If, for instance, you have MAME installed in `C:\Users\Public\MAME` you will need to type `cd C:\Users\Public\MAME` into the command prompt.

Then you have MAME create the config file by typing `mame -createconfig`. MAME will then create the `mame.ini` file in the MAME installation folder. This file contains the default configuration settings for a new MAME installation.

3.9.3 Initial Setup: Creating mame.ini From Command Line on Linux or MacOS

The steps for Linux and MacOS are similar to those of Windows. If you installed MAME using the package manager that came from a Linux distro, you will type `mame -createconfig` into your terminal of choice.

If you have compiled from source or downloaded a binary package of MAME, you will `cd` into the directory you put the MAME files into.

For instance, `cd /home/myusername/mame`

Then you will type `./mame -createconfig` into your terminal of choice.

You can then need to edit the `mame.ini` file in your favorite text editor, e.g. *Notepad* on Windows or *vi* on Linux/MacOS, or you can change settings from inside of MAME.

3.9.4 Initial Setup: Graphical Setup

This is the easiest way to get started. Start MAME by opening the MAME icon in the location where you installed it. This will be `mame.exe` on Windows, `mame` on Linux and macOS.

Once MAME has started, you can either use your mouse to click on the **Configure Options** menu selection at the bottom center of your screen, or you can switch panes to the bottom one (default key is Tab), then press the menu accept button (default key is Return/Enter) to go into the Configuration menu.

Choose **Save Configuration** to create the `mame.ini` file with default settings. From here, you can either continue to configure things from the graphical user interface or edit the `mame.ini` file in your favorite text editor.

CHAPTER

FOUR

BASIC MAME USAGE AND CONFIGURATION

This section describes general usage information about MAME. It is intended to cover aspects of using and configuring MAME that are common across all operating systems. For additional OS-specific options, please see the separate documentation for your platform of choice.

4.1 Using MAME

If you want to dive right in and skip the command line, there's a nice graphical way to use MAME without the need to download and set up a front end. Simply start MAME with no parameters, by double-clicking the **mame.exe** file or running it directly from the command line. If you're looking to harness the full power of MAME, keep reading further.

On macOS and *nix-based platforms, please be sure to set your font up to match your locale before starting, otherwise you may not be able to read the text due to missing glyphs.

If you are a new MAME user, you could find this emulator a bit complex at first. Let's take a moment to talk about software lists, as they can simplify matters quite a bit. If the content you are trying to play is a documented entry on one of the MAME software lists, starting the content is as easy as

mame.exe <*system*> <*software*>

For instance:

mame.exe nes metroidu

will load the USA version of Metroid for the Nintendo Entertainment System.

Alternatively, you could start MAME with

mame.exe nes

and choose the *software list* from the cartridge slot. From there, you could pick any software list-compatible software you have in your roms folders. Please note that many older dumps of cartridges and discs may either be bad or require renaming to match up to the software list in order to work in this way.

If you are loading an arcade board or other non-software list content, things are only a little more complicated:

The basic usage, from command line, is

mame.exe <*system*> <*media*> <*software*> <*options*>

where

- <*system*> is the short name of the system you want to emulate (e.g. nes, c64, etc.)
- <*media*> is the switch for the media you want to load (if it's a cartridge, try **-cart** or **-cart1**; if it's a floppy disk, try **-flop** or **-flop1**; if it's a CD-ROM, try **-cdrom**)

- <*software*> is the program / game you want to load (and it can be given either as the fullpath to the file to load, or as the shortname of the file in our software lists)
- <*options*> is any additional command line option for controllers, video, sound, etc.

Remember that if you type a <*system*> name which does not correspond to any emulated system, MAME will suggest some possible choices which are close to what you typed; and if you don't know which <*media*> switch are available, you can always launch

mame.exe <*system*> **-listmedia**

If you don't know what <*options*> are available, there are a few things you can do. First of all, you can check the command line options section of this manual. You can also try one of the many *Front-ends* available for MAME.

Alternatively, you should keep in mind the following command line options, which might be very useful on occasion:

mame.exe -help

gives a basic summary of command line options for MAME, as explained above.

mame.exe -showusage

gives you the (quite long) list of available command line options for MAME. The main options are described, in the *Universal Command-line Options* section of this manual.

mame.exe -showconfig

gives you a (quite long) list of available configuration options for MAME. These options can always be modified at command line, or by editing them in mame.ini which is the main configuration file for MAME. You can find a description of some configuration options in the *Universal Command-line Options* section of the manual (in most cases, each configuration option has a corresponding command line option to configure and modify it).

mame.exe -createconfig

creates a brand new **mame.ini** file, with default configuration settings. Notice that mame.ini is basically a plain text file, so you can open it with any text editor (e.g. Notepad, Emacs or TextEdit) and configure every option you need. However, no particular tweaks are needed to start, so you can leave most of the options unaltered.

If you execute **mame -createconfig** when you already have an existing mame.ini from a previous MAME version, MAME automatically updates the pre-existing mame.ini by copying changed options into it.

Once you are more confident with MAME options, you may want to adjust the configuration of your setup a bit more. In this case, keep in mind the order in which options are read; see *Order of Configurattion File Loading* for details.

4.2 MAME's User Interface

- *Introduction*
- *Navigating menus*
  - *Using a game controller*
  - *Using a mouse or trackball*
  - *Using a touch screen*
- *Configuring inputs*
  - *Digital input settings*
  - *Analog input settings*

- *The system and software selection menus*
  - *Navigation controls*
- *The simple system selection menu*

4.2.1 Introduction

MAME provides a simple user interface for selecting the system and software to run and changing settings while running an emulated system. MAME's user interface is designed to be usable with a keyboard, game controller, or pointing device, but will require a keyboard for initial configuration.

The default settings for the most important controls to know when running an emulated system, and the settings they correspond to in case you want to change them, are as follows:

Scroll Lock, or Forward Delete on macOS (Toggle UI Controls)
: For emulated systems with keyboard inputs, enable or disable UI controls. (MAME starts with UI controls disabled for systems with keyboard inputs unless the *ui_active option* is on.)

Tab (Show/Hide Menu)
: Show or hide the menu during emulation.

Escape (UI Back and UI Cancel)
: Return to the system selection menu, or exit if MAME was started with a system specified (from the command line or using an *external front-end*).

4.2.2 Navigating menus

By default, MAME menus can be navigated using the keyboard cursor keys. All the UI controls can be changed by going to the **General Inputs** menu and then selecting **User Interface**. The default keyboard controls on a US ANSI QWERTY layout keyboard, and the settings they correspond to, are as follows:

Up Arrow (UI Up)
: Highlight the previous menu item, or the last item if the first item is highlighted.

Down Arrow (UI Down)
: Highlight the next menu item, or the first item if the last item is highlighted.

Left Arrow (UI Left)
: For menu items that are adjustable settings, reduce the value or select the previous setting (these menu items show left- and right-facing triangles beside the value).

Right Arrow (UI Left)
: For menu items that are adjustable settings, increase the value or select the next setting (these menu items show left- and right-facing triangles beside the value).

Return/Enter keypad Enter (UI Select)
: Select the highlighted menu item.

Forward Delete, or Fn+Delete on some compact keyboards (UI Clear)
: Clear setting or reset to default value.

Escape (UI Back and UI Cancel)
: Clear the search if searching the menu, otherwise close the menu, returning to the previous menu, or returning to the emulated system in the case of the main menu (there's usually an item at the bottom of the menu for the same purpose).

Home (UI Home)
Highlight the first menu item and scroll to the top of the menu.

End (UI End)
Highlight the last menu item and scroll to the bottom of the menu.

Page Up (UI Page Up)
Scroll the menu up by one screen.

Page Down (UI Page Down)
Scroll the menu down by one screen.

[(UI Previous Group)
Move to the previous group of items (not used by all menus).

] (UI Next Group)
Move to the next group of items (not used by all menus).

Using a game controller

MAME supports navigating menus with a game controller or joystick, but only the most important UI controls have joystick assignments by default:

- Move the first joystick up or down in the Y axis to highlight the previous or next menu item.
- Move the first joystick left or right in the X axis to adjust settings.
- Press the first button on the first joystick to select the highlighted menu item.
- If the first joystick has at least three buttons, press the second button on the first joystick to close the menu, returning to the previous menu, or returning to the emulated system in the case of the main menu (there's usually an item at the bottom of the menu for the same purpose).

For gamepad-style controllers, the left analog thumb stick and directional pad usually control UI navigation. Depending on the controller, the right analog thumb stick, triggers and additional buttons may automatically be assigned to UI inputs. Check the **User Interface** input assignments menu to see how controls are assigned.

If you want to be able to use MAME with a game controller without needing a keyboard, you'll need to assign joystick buttons (or combinations of buttons) to these controls as well:

- **Show/Hide Menu** to show or hide the menu during emulation
- **UI Back** to close menus
- **UI Cancel** to return to the system selection menu or exit MAME
- **UI Clear** isn't essential for basic emulation, but it's used to clear or reset some settings to defaults
- **UI Home**, **UI End**, **UI Page Up**, **UI Page Down**, **UI Previous Group** and **UI Next Group** are not essential, but make navigating some menus easier

If you're not using an external front-end to launch systems in MAME, you should assign joystick buttons (or combinations of buttons) to these controls to make full use of the system and software selection menus:

- **UI Focus Next/UI Focus Previous** to navigate between panes
- **UI Add/Remove favorite**, **UI Export List** and **UI Audit Media** if you want access to these features without using a keyboard or pointing device

Using a mouse or trackball

MAME supports navigating menus using a mouse or trackball that works as a system pointing device:

- Click menu items to highlight them.
- Double-click menu items to select them.
- Click the left- or right-pointing triangle to adjust settings.
- For menus or text boxes with too many items or lines to fit on the screen, press on the upward- or downward-pointing triangle at the top or bottom to scroll up or down.
- Use vertical scrolling gestures to scroll menus or text boxes with too many items or lines to fit on the screen.
- Click toolbar items to select them, or hover over them to see a description.

If you have enough additional mouse buttons, you may want to assign button combinations to the **Show/Hide Menu**, **Pause**, **UI Back** and/or **UI Cancel** inputs to make it possible to use MAME without a keyboard.

Using a touch screen

MAME has basic support for navigating menus using a touch screen:

- Tap menu items to highlight them.
- Double-tap menu items to select them.
- Swipe left or right (horizontally) on the highlighted menu item to adjust the setting if applicable.
- Swipe up or down (vertically) to scroll menus or text boxes with too many items to fit on the screen.
- For menus or text boxes with too many items or lines to fit on the screen, press on the upward- or downward-pointing triangle at the top or bottom to scroll up or down.

Note that for SDL-based MAME, the *enable_touch* option must be switched on to use touch screen support.

4.2.3 Configuring inputs

MAME needs a flexible input system to support the control schemes of the vast array of systems it emulates. In MAME, inputs that only have two distinct states, on and off or active and inactive, are called *digital inputs*, and all other inputs are called *analog inputs*, even if this is not strictly true (for example multi-position switches are called analog inputs in MAME).

To assign MAME's user interface controls or the default inputs for all systems, select **Input Settings** from the main menu during emulation and then select **Input Assignments (general)** from the Input Settings menu, or select **General Settings** from the system selection menu and then select **Input Assignments** from the General Settings menu. From there, select a category.

To assign inputs for the currently running system, select **Input Settings** from the main menu during emulation and then select **Input Assignments (this system)** from the Input Settings menu. Inputs are grouped by device and sorted by type. You can move between devices with the next group and previous group keys/buttons (opening/closing brackets **[** and **]** on the keyboard by default).

The input assignment menus show the name of the emulated input or user interface control on the left, and the controls (or combination of controls) assigned to it on the right.

To adjust the sensitivity, auto-centre speed and inversion settings, or to see how emulated analog controls react to your inputs, select **Input Settings** from the main menu during emulation, and then select **Analog Input Adjustments** from the Input Settings Menu (this item only appears on the Input Settings menu for systems with analog controls).

Digital input settings

Each emulated digital input has a single assignment setting. For flexibility, MAME can combine controls (keys, buttons and joystick axes) using logical **and**, **not** and **or** operations. This is best illustrated with some examples:

Kbd 1
: In this simple case, pressing the **1** key on the keyboard activates the emulated input or user interface control.

Kbd Down or Joy 1 Down
: Pressing the down arrow on the keyboard or moving the first joystick down activates the emulated input or user interface control.

Kbd P not Kbd Shift not Kbd Right Shift
: Pressing the **P** key on the keyboard while not pressing either **Shift** key activates the emulated input or user interface control. MAME does not show the implicit **and** operations.

Kbd P Kbd Shift or Kbd P Kbd Right Shift
: Pressing the **P** key while also pressing either of the **Shift** keys activates the emulated input or user interface control. Once again, the implicit **and** operations are not shown.

(In technical terms, MAME uses Boolean sum of products logic to combine inputs.)

When a digital input setting is highlighted, the prompt below the menu shows whether selecting it will replace the current assignment or append an **or** operation to it. Press **UI Left/Right** before selecting the setting to switch between replacing the assignment or appending an **or** operation to it. Press **UI Clear** (**Delete** or **Forward Delete** by default) to clear the highlighted setting, or restore the default assignment if it is currently cleared.

When you select a digital input setting, MAME will wait for you to enter an input or a combination of inputs for a logical **and** operation:

- Press a key or button or move an analog control once to add it to the **and** operation.
- Press a key or button or move an analog control twice to add a **not** item to the **and** operation. Pressing the same key or button or moving the same analog control additional times toggles the **not** on and off.
- Press **UI Cancel** (**Escape** by default) to leave the setting unchanged.
- The new setting is shown below the menu. Wait one second after activating an input to accept the new setting.

Here's how to produce some example settings:

Kbd 1
: Press the **1** key on the keyboard once, then wait one second to accept the setting.

Kbd F12 Kbd Shift Keyboard Alt
: Press the **F12** key on the keyboard once, press the left **Shift** key once, press the left **Alt** key once, then wait one second to accept the setting.

Kbd P not Kbd Shift not Kbd Right Shift
: Press the **P** key on the keyboard once, press the left **Shift** key twice, press the right **Shift** key twice, then wait one second to accept the setting.

Analog input settings

Each emulated analog input has three assignment settings:

- Use the *axis setting* to assign an analog axis to control the emulated analog input. The axis setting uses the name of the input with the suffix "Analog". For example the axis setting for the steering wheel in Ridge Racer is called **Steering Wheel Analog**.
- Use the *increment setting* assign a control (or combination of controls) to increase the value of the emulated analog input. The increment setting uses the name of the input with the suffix "Analog Inc". For example the increment setting for the steering wheel in Ridge Racer is called **Steering Wheel Analog Inc**. This is a digital input setting – if an analog axis is assigned to it, MAME will not increase the emulated input value at a proportional speed.
- Use the *decrement setting* assign a control (or combination of controls) to decrease the value of the emulated analog input. The decrement setting uses the name of the input with the suffix "Analog Dec". For example the decrement setting for the steering wheel in Ridge Racer is called **Steering Wheel Analog Dec**. This is a digital input setting – if an analog axis is assigned to it, MAME will not decrease the emulated input value at a proportional speed.

The increment and decrement settings are most useful for controlling an emulated analog input using digital controls (for example keyboard keys, joystick buttons, or a directional pad). They are configured in the same way as emulated digital inputs (*see above*). **It's important that you don't assign the same control to the axis setting as well as the increment and/or decrement settings for the same emulated input at the same time.** For example if you assign Ridge Racer's **Steering Wheel Analog** setting to the X axis of the left analog stick on your controller, you *should not* assign either the **Steering Wheel Analog Inc** or **Steering Wheel Analog Dec** setting to the X axis of the same analog stick.

You can assign one or more analog axes to the axis setting for an emulated analog input. When multiple axes are assigned to an axis setting, they will be added together, but absolute position controls will override relative position controls. For example suppose for Arkanoid you assign the **Dial Analog** axis setting to **Mouse X or Joy 1 LSX or Joy 1 RSX** on a mouse and Xbox-style controller. You will be able to control the paddle with the mouse or either analog stick, but the mouse will only take effect if both analog sticks are in the neutral position (centred) on the X axis. If either analog stick is *not* centred on the X axis, the mouse will have no effect, because a mouse is a relative position control while joysticks are absolute position controls.

For absolute position controls like joysticks and pedals, MAME allows you to assign either the full range of an axis or the range on one side of the neutral position (a *half axis*) to an axis setting. Assigning a half axis is usually used for pedals or other absolute inputs where the neutral position is at one end of the input range. For example suppose for **Ridge Racer** you assign the **Brake Pedal Analog** setting to the portion of a vertical joystick axis below the neutral position. If the joystick is at or above the neutral position vertically, the brake pedal will be released; if the joystick is below the neutral position vertically, the brake pedal will be applied proportionally. Half axes are displayed as the name of the axis followed by a plus or minus sign (**+** or **-**). Plus refers to the portion of the axis below or to the right of the neutral position; minus refers to the portion of the axis above or to the left of the neutral position. For pedal or analog trigger controls, the active range is treated as being above the neutral position (the half axis indicated by a minus sign).

When keys or buttons are assigned to an axis setting, they conditionally enable analog controls assigned to the setting. This can be used in conjunction with an absolute position control to create a "sticky" control.

Here are some examples of some possible axis setting assignments, assuming an Xbox-style controller and a mouse are used:

Joy 1 RSY
: Use vertical movement of the right analog stick to control the emulated input.

Mouse X or Joy 1 LT or Joy 1 RT Reverse
: Use horizontal mouse movement, or the left and right triggers to control the emulated input. The right trigger is reversed so it acts in the opposite direction to the left trigger.

Joy 1 LB Joy 1 LSX
: Use horizontal movement of the left analog stick to control the emulated input, but *only* while holding the left shoulder button. If the left shoulder button is released while the left analog stick is not centred horizontally, the emulated input will hold its value until the left shoulder button is pressed again (a "sticky" control).

not Joy 1 RB Joy 1 RSX or Joy 1 RB Joy 1 RSX Reverse
: Use horizontal movement of the right analog stick to control the emulated input, but invert the control if the right shoulder button is held.

When you select an axis setting, MAME will wait for you to enter an input:

- Move an analog control to assign it to the axis setting.
- Press a key or button (or a combination of keys or buttons) *before* moving an analog control to conditionally enable the analog control.
- When appending to a setting, if the last assigned control is an absolute position control, move the same control again to cycle between the full range of the axis, the portion of the axis on either side of the neutral position, and the full range of the axis reversed.
- When appending to a setting, if the last assigned control is a relative position control, move the same control again to toggle reversing the direction of the control on or off.
- When appending to a setting, move an analog control other than the last assigned control or press a key or button to add an **or** operation.
- Pressing **UI Cancel** (**Escape** by default) leaves the setting unchanged.
- The new setting is shown below the menu. Wait one second after moving an analog control to accept the new setting.

To adjust sensitivity, auto-centring speed and inversion settings for emulated analog inputs, or to see how they respond to controls with your settings, select **Input Settings** from the main menu during emulation, and then select **Analog Input Adjustments** from the Input Settings Menu. Settings for emulated analog inputs are grouped by device and sorted by type. You can move between devices with the next group and previous group keys/buttons (opening/closing brackets **[** and **]** on the keyboard by default). The state of the emulated analog inputs is shown below the menu, and reacts in real time. Press the **On Screen Display** key or button (the backtick/tilde key by default on a US ANSI QWERTY keyboard) to hide the menu to make it easier to test without changing settings. Press the same key or button to show the menu again.

Each emulated input has four settings on the **Analog Controls** menu:

- The *increment/decrement speed* setting controls how fast the input value increases or decreases in response to the controls assigned to the increment/decrement settings.
- The *auto-centering speed* setting controls how fast the input value returns to the neutral state when the controls assigned to the increment/decrement settings are released. Setting it to zero (**0**) will result in the value not automatically returning to the neutral position.
- The *reverse* setting allows the direction of the emulated input's response to controls to be inverted. This applies to controls assigned to the axis setting *and* the increment/decrement settings.
- The *sensitivity* setting adjusts the input value's response to the control assigned to the axis setting.

Use the UI left/right keys or buttons to adjust the highlighted setting. Selecting a setting or pressing the UI clear key/button (**Forward Delete** by default) restores its default value.

The units for the increment/decrement speed, auto-centering speed and sensitivity settings are tied to the driver/device implementation. The increment/decrement speed and auto-centering speed settings are also tied to the frame rate of the first emulated screen in the system. The response to controls assigned to the increment/decrement settings will change if the system changes the frame rate of this screen.

4.2.4 The system and software selection menus

If you start MAME without specifying a system on the command line, the system selection menu will be shown (assuming the *ui option* is set to **cabinet**). The system selection menu is also shown if you select **Select New System** from the main menu during emulation. Selecting a system that uses software lists shows the similar software selection menu.

The system and software selection menus have the following parts:

- The heading area at the top, showing the emulator name and version, the number of systems or software items in the menu, and the current search text. The software selection menu also shows the name of the selected system.
- The toolbar immediately below the heading area. The exact toolbar buttons shown depend on the menu. Hover the mouse pointer over a button to see a description. Click a button to select it.

  Toolbar buttons are add/remove highlighted system/software from favourites (star), export displayed list to file (diskette), audit media (magnifying glass), show info viewer ("i" emblazoned on blue circle), return to previous menu (bent arrow on blue), and exit (cross on red).
- The list of systems or software in the centre. For the system selection menu, there are configuration options below the list of systems. Clones are shown with a different text colour (grey by default). You can right-click a system name as a shortcut to show the System Settings menu for the system.

  Systems or software items are sorted by full name or description, keeping clones immediately below their parents. This may appear confusing if your filter settings cause a parent system or software item to be hidden while one or more of its clones are visible.
- The info panel at the bottom, showing summary information about the highlighted system or software. The background colour changes depending on the emulation status: green for working, amber for imperfectly emulated features or known issues, or red for more serious issues.

  A yellow star is show at the top left of the info panel if the highlighted system or software is in your favourites list.
- The collapsible list of filter options on the left. Click a filter to apply it to the list of systems/software. Some filters show a menu with additional options (e.g. specifying the manufacturer for the **Manufacturer** filter, or specifying a file and group for the **Category** filter).

  Click **Unfiltered** to display all items. Click **Custom Filter** to combine multiple filters. Click the strip between the list of filters and the list of systems/software to show or hide the list of filters. Be aware that filters still apply when the list of filters is hidden.
- The collapsible info viewer on the right. This has two tabs for showing images and information. Click a tab to switch tabs; click the left- or right-facing triangles next to the image/info title to switch between images or information sources.

  Emulation information is automatically shown for systems, and information from the software list is shown for software items. Additional information from external files can be shown using the *Data plugin*.

You can type to search the displayed list of systems or software. Systems are searched by full name, manufacturer and full name, and short name. If you are using localised system names, phonetic names will also be searched if present. Software items are searched by description, alternate titles (`alt_title` info elements in the software lists), and short name. **UI Cancel** (Escape by default) will clear the search if currently searching.

Navigation controls

In addition to the usual *menu navigation controls*, the system and software selection menus have additional configurable controls for navigating the multi-pane layout, and providing alternatives to toolbar buttons if you don't want to use a pointing device. The default additional controls (with a US ANSI QWERTY keyboard), and the settings they correspond to, are:

Tab (UI Focus Next)
: Move focus to the next area. The order is system/software list, configuration options (if visible), filter list (if visible), info/image tabs (if visible), info/image source (if visible).

Shift+Tab (UI Focus Previous)
: Move focus to the previous area.

Alt+D (UI External DAT View)
: Show the full-size info viewer.

Alt+F (UI Add/Remove favorite)
: Add or remove the highlighted system or software item from the favourites list.

F1 (UI Audit Media)
: Audit ROMs and/or disk images for systems. The results are saved for use with the **Available** and **Unavailable** filters.

When focus is on the filter list, you can use the menu navigation controls (up, down, home and end) to highlight a filter, and **UI Select** (Return/Enter by default) apply it.

When focus is on any area besides the info/image tabs, you can change the image or info source with left/right. When focus is on the info/image tabs, left/right switch between tabs. When focus is on the image/info tabs or source, you can scroll the info using up, down, page up, page down, home and end.

You can move focus to an area by clicking on it with the middle mouse button.

4.2.5 The simple system selection menu

If you start MAME without specifying a system on the command line (or choose **Select New System** from the main menu during emulation) with the *ui option* set to **simple**, the simple system selection menu will be shown. The simple system selection menu shows fifteen randomly selected systems that have ROM sets present in your configured *ROM folder(s)*. You can type to search for a system. Clearing the search causes fifteen systems to be randomly selected again.

The info panel below the menu shows summary information about the highlighted system. The background colour changes depending on the emulation status: green for working, amber for imperfectly emulated features or known issues, or red for more serious issues.

4.3 Default Keyboard Controls

- *Controls Foreword*
- *MAME User Interface Controls*
  - *System and software selection menus*
- *Default Arcade Machine Controls*
- *Default Arcade Game Controls*

  - *Player 1 Controls*
  - *Player 2 Controls*
  - *Player 3 Controls*
  - *Player 4 Controls*
- *Default Mahjong and Hanafuda Keys*
- *Default Gambling Keys*
  - *Default Blackjack Keys*
  - *Default Poker Keys*
  - *Default Slots Keys*
- *Default Computer Keys*
- *Other Machines*

4.3.1 Controls Foreword

MAME supports a vast array of different types of machines, with a significantly different array of inputs across them. This means that some keyboard keys, mouse buttons, and joystick buttons will be used for multiple functions. As a result, the control charts below are separated by machine-types to make it easier to find what you're looking for.

All of the controls below are fully configurable in the user interface. These charts show the default configuration.

Note that the defaults shown here are arranged by US ANSI key positioning. If you are using a different layout, the keys will vary.

4.3.2 MAME User Interface Controls

The controls here cover MAME functions such as MAME's menus, machine pause, and saving/loading save states.

Tab
: Toggles the configuration menu.

`/~ (backtick/tilde key)
: Toggles the On-Screen Display.

  If you are running with -debug, this key sends a 'break' in emulation.

  When a slider control is visible, the keyboard controls are the same as in the Slider Controls menu:

  - **Up** - select previous parameter to modify.
  - **Down** - select next parameter to modify.
  - **Left** - decrease the value of the selected parameter.
  - **Right** - increase the value of the selected parameter.
  - **Delete** - reset parameter value to its default.
  - **Alt+Left** - decrease the value by the largest amount.
  - **Control+Left** - decrease the value by 10x.
  - **Shift+Left** - decrease the value by 0.1x.

- **Shift+Alt+Left** - decrease the value by the smallest amount.
- **Alt+Right** - increase the value by the largest amount.
- **Control+Right** - increase the value by 10x.
- **Shift+Right** - increase the value by 0.1x.
- **Shift+Alt+Right** - increase the value by the smallest amount.

Up Arrow
Highlight previous UI menu option.

Down Arrow
Highlight next UI menu option.

Left Arrow
Change current UI option setting when an arrow is present on it.

Right Arrow
Change current UI option setting when an arrow is present on it.

Home/End
Highlight first or last UI menu option.

[ ]
Move to previous or next group in UI menus that support it (e.g. move to the inputs for the previous or next device in the **Input Assignments (this System)** menu).

Enter/Joystick 1 Button 1
Select currently highlighted UI menu option.

Space
Show comment on currently highlighted UI menu option.

Delete
Clear/reset to default when highlighting an entry on the input configuration, cheat options, and plugin options pages.

F1
Power the machine on for machines that have specific power button behavior.

F2
Power the machine off for machines that have specific power button behavior.

F3
Soft resets the machine.

Left Shift+F3
Performs a "hard reset", which tears everything down and re-creates it from scratch. This is a more thorough and complete reset than the reset you get from hitting F3.

F4
Shows the game palette, decoded graphics tiles/characters and any tilemaps.

Use the Enter key to switch between the three modes (palette, graphics, and tilemaps).

Press F4 again to turn off the viewer. The key controls in each mode vary slightly:

Palette/colortable mode:

- **[ ]** - switch between palette devices.
- **Up/Down** - scroll up/down one line at a time.
- **Shift+Up/Down** - scroll right/left one cell at a time.

- **Page Up/Page Down** - scroll up/down one page at a time (hold Control or Alt to scroll 10 or 100 pages).
- **Home/End** - move to top/bottom of list.
- **-/+** - increase/decrease the number of colors per row.
- **0** - restore the default number of colors per row.
- **Enter** - switch to graphics viewer.

Graphics mode:

- **[ ]** - switch between different graphics sets.
- **Up/Down** - scroll up/down one line at a time.
- **Shift+Up/Down** - scroll right/left one cell at a time.
- **Page Up/Page Down** - scroll up/down one page at a time (hold Control or Alt to scroll 10 or 100 pages).
- **Home/End** - move to top/bottom of list.
- **Left/Right** - change color displayed (hold Control or Alt for wider control).
- **R** - rotate tiles 90 degrees clockwise.
- **-/+** - increase/decrease the number of tiles per row (hold Shift to restrict to integer scale factors).
- **0** - restore the default number of tiles per row (hold Shift to restrict to integer scale factors).
- **Enter** - switch to tilemap viewer.

Tilemap mode:

- **[ ]** - switch between different tilemaps.
- **Up/Down/Left/Right** - scroll 8 pixels at a time.
- **Shift+Up/Down/Left/Right** - scroll 1 pixel at a time.
- **Control+Up/Down/Left/Right** - scroll 64 pixels at a time.
- **R** - rotate tilemap view 90 degrees clockwise.
- **-/+** - decrease/increase the zoom factor.
- **0** - expand small tilemaps to fill the display.
- **Enter** - switch to palette/colortable mode.

Note: Not all systems have decoded graphics and/or tilemaps.

Left Shift+F4
: While paused, loads the most recent rewind save state.

F5
: Pauses the emulated machine.

Left Shift+F5
: While paused, advances to next frame. If rewind is enabled, a new rewind save state is also captured.

F6
: Create a save state. Requires an additional keypress to identify the state, similar to the load option above. If an existing save state is present, it will also appear in the selection menu to allow overwriting of that save state.

Left Shift+F6
: Create a quick save state.

F7
Load a save state. You will be prompted to press a key or select from the menu to determine which save state you wish to load.

Note that the save state feature is not supported for a large number of drivers. If a given driver is not known to work perfectly, you will receive a warning that the save state may not be valid when attempting to save or load.

Left Shift+F7
Load a quick save state.

F8
Decrease frame skipping on the fly.

Left Shift+F8
Toggle cheat mode. (if started with "-cheat")

Left Alt+F8
Decrease Prescaling. (*SDL MAME only*)

F9
Increase frame skipping on the fly.

Left Alt+F9
Increase Prescaling. (*SDL MAME only*)

F10
Toggle speed throttling.

Left Alt+F10
Toggle HLSL Post-Processing. (*Windows non-SDL MAME only*)

Left Alt+F10
Toggle Filter. (*SDL MAME only*)

F11
Toggles speed display.

Left Shift+F11
Toggles internal profiler display (if compiled in).

Left Alt+F11
Record HLSL Rendered Video.

F12
Saves a screen snapshot.

Left Shift+F12
Begin recording MNG video.

Left Control+Left Shift+F12
Begin recording AVI video.

Left Alt+F12
Take HLSL Rendered Snapshot.

Insert (Windows non-SDL MAME)/Page Down (SDL MAME)
Fast forward. While held, runs game with throttling disabled and with the maximum frameskip.

Left Alt+Enter
Toggles between full-screen and windowed mode.

Scroll Lock/Forward Delete (Mac Desktop)/fn-Delete (Mac Laptop)
Default mapping for the **uimodekey**.

This key toggles MAME's response to user interface keys such as the (by default) **Tab** key being used for menus. All emulated machines which require emulated keyboards will start with UI controls disabled by default and you can only access the internal UI by first hitting this **uimodekey** key. You can change the initial status of the emulated keyboard as presented upon start by using *-uimodekey*

Escape
Exit emulator, return to the previous menu, or cancel the current UI option.

System and software selection menus

The system and software selection menus use additional controls

Tab
Moves keyboard/controller focus to the next UI panel.

Shift+Tab
Moves keyboard/controller focus to the previous UI panel.

Left Alt+F
Adds or removes the selected system or software list item from the favorites list.

Left Alt+E
Exports the currently displayed list of systems.

Left Alt+D
Shows the full-size info viewer if info is available for the selected system or software list item. (Shows information loaded by the data plugin from external files, including history.xml and mameinfo.dat.)

F1
Audits system ROMs and disk images.

4.3.3 Default Arcade Machine Controls

This section covers controls that are applicable to most kinds of arcade machines. Note that not all machines will have all of these controls. All the controls below are fully configurable in the user interface. This list shows the standard keyboard configuration.

5 ***(not numeric keypad)***
Coin slot 1

6 ***(not numeric keypad)***
Coin slot 2

7 ***(not numeric keypad)***
Coin slot 3

8 ***(not numeric keypad)***
Coin slot 4

Backspace
Bill 1 (For machines that have a bill receptor/note reader)

T
Tilt

Usually a tilt switch or shock sensor that will end the current game, reset credits and/or reset the machine if the machine is knocked excessively hard or moved. Most commonly found on pinball machines.

O
Door Interlock (A coin door for example)

- *(not numeric keypad)*
Volume Down

For machines that have an electronic volume control.

= *(not numeric keypad)*
Volume Up

For machines that have an electronic volume control.

F1
Memory Reset

This resets high scores, credits/winnings, statistics, and/or operator settings on machines that support it.

F2
Service Mode

This is a momentary push-button on some machines, while it is a toggle switch or DIP switch on others.

9 *(not numeric keypad)*
Service 1

Service buttons are typically used to give free credits or to navigate the operator service menus.

0 *(not numeric keypad)*
Service 2

- *(not numeric keypad)*
Service 3

= *(not numeric keypad)*
Service 4

4.3.4 Default Arcade Game Controls

This section covers controls for arcade games using common joystick/button control schemes. All the controls below are fully configurable in the user interface. This list shows the standard keyboard configuration.

5 *(not numeric keypad)*
Coin slot 1

6 *(not numeric keypad)*
Coin slot 2

7 *(not numeric keypad)*
Coin slot 3

8 *(not numeric keypad)*
Coin slot 4

1 *(not numeric keypad)*
Player 1 start or 1 player mode

2 *(not numeric keypad)*
Player 2 start or 2 players mode

3 *(not numeric keypad)*
Player 3 start or 3 players mode

4 *(not numeric keypad)*
Player 4 start or 4 players mode

Player 1 Controls

Up Arrow
Player 1 Up

Down Arrow
Player 1 Down

Left Arrow
Player 1 Left

Right Arrow
Player 1 Right

E
Player 1 Up on Left Stick for dual-stick machines (e.g. Robotron)

D
Player 1 Down on Left Stick for dual-stick machines (e.g. Robotron)

S
Player 1 Left on Left Stick for dual-stick machines (e.g. Robotron)

F
Player 1 Right on Left Stick for dual-stick machines (e.g. Robotron)

I
Player 1 Up on Right Stick for dual-stick machines (e.g. Robotron)

K
Player 1 Down on Right Stick for dual-stick machines (e.g. Robotron)

J
Player 1 Left on Right Stick for dual-stick machines (e.g. Robotron)

L
Player 1 Right on Right Stick for dual-stick machines (e.g. Robotron)

Left Ctrl/Mouse B0/Gun 1 Button 0
Player 1 Button 1

Left Alt/Mouse B2/Gun 1 Button 1
Player 1 Button 2

Spacebar/Mouse B1/Joystick 1 Button 1 or B
Player 1 Button 3

Left Shift
Player 1 Button 4

Z
Player 1 Button 5

X
Player 1 Button 6

C
Player 1 Button 7

V
Player 1 Button 8

B
Player 1 Button 9

N
Player 1 Button 10

M
Player 1 Button 11

,
Player 1 Button 12

.
Player 1 Button 13

/
Player 1 Button 14

Right Shift
Player 1 Button 15

Player 2 Controls

R
Player 2 Up

F
Player 2 Down

D
Player 2 Left

G
Player 2 Right

A
Player 2 Button 1

S
Player 2 Button 2

Q
Player 2 Button 3

W
Player 2 Button 4

E
Player 2 Button 5

Player 3 Controls

I
Player 3 Up

K
Player 3 Down

J
Player 3 Left

L
Player 3 Right

Right Control
Player 3 Button 1

Right Shift
Player 3 Button 2

Enter ***(not numeric keypad)***
Player 3 Button 3

Player 4 Controls

8 ***(on numeric keypad)***
Player 4 Up

2 ***(on numeric keypad)***
Player 4 Down

4 ***(on numeric keypad)***
Player 4 Left

6 ***(on numeric keypad)***
Player 4 Right

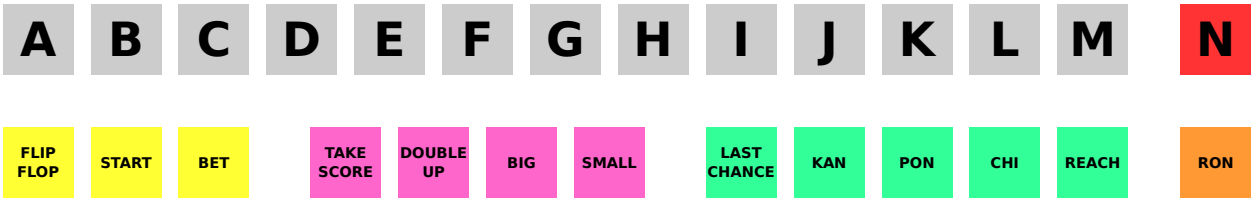
0 ***(on numeric keypad)***
Player 4 Button 1

. ***(on numeric keypad)***
Player 4 Button 2

Enter ***(on numeric keypad)***
Player 4 Button 3

4.3.5 Default Mahjong and Hanafuda Keys

Most mahjong and hanafuda games use a standard control panel layout. Some keys may not be present, depending on the kind of game. For example games without a bonus game feature may lack the Take Score, Double Up, Big and Small keys, and games without gambling features may also lack the Bet key. Some games may not use all keys that are present. For example many games do not use the Flip Flop and Last Chance keys.



Due to the large number of keys, MAME only provides default input configuration for a single set of player controls. For multi-player mahjong/hanafuda games, or mahjong/hanafuda games with multiple player positions, manual configuration is required. All the keys below are fully configurable in the user interface. This list shows the standard keyboard configuration.

5 ***(not numeric keypad)***
Coin slot 1

6 ***(not numeric keypad)***
Coin slot 2

7 ***(not numeric keypad)***
Coin slot 3

8 ***(not numeric keypad)***
Coin slot 4

Y
Player 1 Mahjong/Hanafuda Flip Flop

1 ***(not numeric keypad)***
Player 1 start or 1 player mode

2 ***(not numeric keypad)***
Player 2 start or 2 players mode

3 ***(not numeric keypad)***
Player 3 start or 3 players mode

Mahjong Bet

4 ***(not numeric keypad)***
Player 4 start or 4 players mode

Right Ctrl
Player 1 Mahjong/Hanafuda Take Score

Right Shift
Player 1 Mahjong/Hanafuda Double Up

Enter
Player 1 Mahjong/Hanafuda Big

Backspace
Player 1 Mahjong/Hanafuda Small

Right Alt
Player 1 Mahjong/Hanafuda Last Chance

Ctrl
Mahjong Kan

Alt
Mahjong Pon

Spacebar
Mahjong Chi

Shift
Mahjong Reach

Z
Mahjong Ron

A
Player 1 Mahjong/Hanafuda A

B
Player 1 Mahjong/Hanafuda B

C
Player 1 Mahjong/Hanafuda C

D
Player 1 Mahjong/Hanafuda D

E
Player 1 Mahjong/Hanafuda E

F
Player 1 Mahjong/Hanafuda F

G
Player 1 Mahjong/Hanafuda G

H
Player 1 Mahjong/Hanafuda H

I
Player 1 Mahjong I

J
Player 1 Mahjong J

K
Player 1 Mahjong K

L
Player 1 Mahjong L

M
Player 1 Mahjong M

Player 1 Hanafuda Yes

N
Player 1 Mahjong N

Player 1 Hanafuda No

O
Player 1 Taiwanese Mahjong O

P
Player 1 Taiwanese Mahjong P

Q
Player 1 Taiwanese Mahjong Q

4.3.6 Default Gambling Keys

All the keys below are fully configurable in the user interface. This list shows the standard keyboard configuration.

Note that many gambling games use buttons for multiple functions. For example a slots game may use the Start button to stop all reels, lacking a dedicated Stop All Reels button, or a poker game may use the hold buttons to control the double-up bonus game, lacking dedicated Take Score, Double Up, High and Low buttons.

5
Coin slot 1

6
Coin slot 2

7
Coin slot 3

8
Coin slot 4

Backspace
Bill 1 (For machines that have a bill receptor/note reader)

I
: Payout

Q
: Key In

W
: Key Out

F1
: Memory Reset

9 ***(not numeric keypad)***
: Service 1 (Service buttons are typically used to give free credits or to navigate the internal operator service menus)

0 ***(not numeric keypad)***
: Service 2

  Book-Keeping (for machines that have this functionality)

- ***(not numeric keypad)***
: Service 3

= ***(not numeric keypad)***
: Service 4

M
: Bet

1 ***(not numeric keypad)***
: Player 1 start or 1 player mode

2 ***(not numeric keypad)***
: Deal

L
: Stand

D
: Double Up

  For games that allow gambling winnings in a double-or-nothing bonus game, this gambles the winnings from the main game in the bonus game.

F
: Half Gamble

  Used by games that allow gambling half or all of the winnings from the main game in the bonus game, this stakes half the winnings from the main game.

G
: Take Score

  For games that allow gambling winnings in a double-or-nothing bonus game, this stakes the winnings from the main game.

A
: High/big

S
: Low/small

Default Blackjack Keys

All the keys below are fully configurable in the user interface. This list shows the standard keyboard configuration.

1
: Player 1 start or 1 player mode

  Used to deal a new hand for games that have separate buttons to deal a new hand and draw an additional card.

2
: Deal (hit)

  Used to draw an additional card, and to deal a new hand in games that don't use separate buttons to deal a new hand and draw an additional card.

L
: Stand

Default Poker Keys

All the keys below are fully configurable in the user interface. This list shows the standard keyboard configuration.

1
: Player 1 start or 1 player mode

  Used to deal a new hand for games that have separate buttons to deal a new hand and draw replacement cards.

2
: Deal

  Used to draw replacement cards, and to deal a new hand in games that don't use separate buttons to deal a new hand and draw replacement cards.

Z
: Hold 1/discard 1

X
: Hold 2/discard 2

C
: Hold 3/discard 3

V
: Hold 4/discard 4

B
: Hold 5/discard 5

N
: Cancel

  Used by some games to cancel current selection for cards to hold/discard.

Default Slots Keys

All the keys below are fully configurable in the user interface. This list shows the standard keyboard configuration.

1
: Player 1 start or 1 player mode

X
: Stop Reel 1

C
: Stop Reel 2

V
: Stop Reel 3

B
: Stop Reel 4

Z
: Stop All Reels

4.3.7 Default Computer Keys

All the keys below are fully configurable in the user interface. This list shows the standard keyboard configuration.

Note that controls can vary widely by computer type, so not all keys are shown here. See the **Input Assignments (this system)** section of MAME's Input Settings menu for details for the machine you are currently using.

Tab
: Toggles the configuration menu.

Scroll Lock/Forward Delete (Mac Desktop)/fn-Delete (Mac Laptop)
: Default mapping for the **uimodekey**.

  This key toggles MAME's response to user interface keys such as the (by default) **Tab** key being used for menus. All emulated machines which require emulated keyboards will start with UI controls disabled by default and you can only access the internal UI by first hitting this **uimodekey** key. You can change the initial status of the emulated keyboard as presented upon start by using *-uimodekey*

F2
: Start tape for machines that have cassette tape drives.

Shift+F2
: Stop tape for machines that have cassette tape drives.

Left Shift+Scroll Lock
: Pastes from system clipboard into the emulated machine.

Alphanumeric Keys
: These keys are mapped to their equivalents in the emulated machine by default.

4.3.8 Other Machines

All the keys are fully configurable in the user interface.

Note that controls can vary widely by machine type, so default keys are not shown here and defaults will vary considerably based on the manufacturer and style. See the **Input Assignments (this system)** section of MAME's Input Settings menu for details for the machine you are currently using.

4.4 MAME Menus

- *Introduction*
- *Main menu*
- *Input Settings menu*
- *Toggle Inputs menu*
- *Keyboard Selection menu*
- *Input Devices menu*
- *Audio Mixer menu*
- *Audio Effects menu*
  - *Filter effect*
  - *Compressor effect*
  - *Reverb effect*
  - *Equalizer effect*

4.4.1 Introduction

To show the *main menu* while running an emulated system in MAME, press the **Show/Hide Menu** key or button (**Tab** by default). If the emulated system has keyboard inputs, you may need to press the **Toggle UI Controls** key or button (**Scroll Lock**, or **Forward Delete** on macOS, by default) to enable user interface controls first. You can dismiss a menu by pressing the **UI Back** key or button (**Escape** by default). Dismissing a menu will return to its parent menu, or to the running system in the case of the main menu.

You can hide a menu and return to the running system by pressing the **Show/Hide Menu** key or button. Pressing the **Show/Hide Menu** key or button again will jump back to the same menu. This is useful when testing changes to settings.

Emulated system inputs are ignored while menus are displayed. You can still pause or resume the running system while most menus are displayed by pressing the **Pause** key or button (**F5** on the keyboard by default).

If you start MAME without specifying a system on the command line, the system selection menu will be shown (assuming the *ui option* is set to **cabinet**). The system selection menu is also shown if you select **Select New System** from the main menu during emulation.

For more information on navigating menus, *see the relevant section*.

4.4.2 Main menu

The main menu is shown when you press the **Show/Hide Menu** key or button while running an emulated system or while the system information screen is displayed. It provides access to menus used to change settings, control various features, and show information about the running system and MAME itself.

If you press the **Show/Hide Menu** key or button to show the main menu while the system information screen is displayed, the emulated system will not start until the main menu is dismissed (either by selecting **Start System**, pressing the **UI Back** key or button, or pressing the **Show/Hide Menu** key or button). This can be useful for mounting media images or changing DIP switches and machine configuration settings before the emulated system starts.

Input Settings
: Shows the *Input Settings* menu, where you can assign controls to emulated inputs, adjust analog control settings, control toggle inputs, and test input devices.

DIP Switches
: Shows the DIP Switches menu, where configuration switches for the running system can be changed. This item is not shown if the running system has no DIP switches.

Machine Configuration
: Shows the Machine Configuration menu, where various settings specific to the emulated system can be changed. This item is not shown if the running system has no configuration settings.

Bookkeeping
: Shows uptime, coin counter and ticket dispenser statistics (if relevant) for the running system.

System Information
: Shows information about the running system as emulated in MAME, including CPU, sound and video devices.

Warning Information
: Shows information about imperfectly emulated features of the running system. This item is not shown if there are no relevant warnings.

Media Image Information
: Shows information about mounted media images (if any). This item is only shown if the running system has one or more media devices (e.g. floppy disk drives or memory card slots).

File Manager
: Shows the File Manager menu, where you can mount new or existing media image files, or unmount currently mounted media images. This item is only shown if the running system has one or more media devices (e.g. floppy disk drives or memory card slots).

Tape Control
: Shows the Tape Control menu, where you can control emulated cassette tape mechanisms. This item is only shown for systems that use cassette tape media.

Pseudo Terminals
: Shows the status of any pseudo terminal devices in the running system (used to connect the emulated system to host pseudo terminals, for example via emulated serial ports). This item is not shown if there are no pseudo terminal devices in the running system.

BIOS Selection
: Shows the BIOS Selection menu, where you can select the BIOS/boot ROM/firmware for the system and slot cards it contains. This item is not shown if no BIOS options are available.

Slot Devices
: Shows the Slot Devices menu, where you can choose between emulated peripherals. This item is not shown for systems that have no slot devices.

Barcode Reader
Shows the Barcode Reader menu, where you can simulate scanning barcodes with emulated barcode readers. This item is not shown if there are no barcode readers in the running system.

Network Devices
Shows the Network Devices menu, where you can set up emulated network adapters that support bridging to a host network. This item is not shown if there are no network adaptors that support bridging in the running system.

Audio Mixer
Shows the *Audio Mixer* menu, where you configure how MAME routes sound from the emulated system to host system sound outputs, and from host system sound inputs to the emulated system.

Audio Effects
Shows the *Audio Effects* menu, where you can configure audio effects applied to emulated sound output.

Slider Controls
Shows the Slider Controls menu, where you can adjust various settings, including video adjustments and individual sound channel levels.

Video Options
Shows the Video Options menu, where you can change the view for each screen/window, as well as for screenshots.

Crosshair Options
Shows the Crosshair Options menu, where you can adjust the appearance of crosshairs used to show the location of emulated light guns and other absolute pointer inputs. This item is not shown if the emulated system has no absolute pointer inputs.

Cheat
Shows the Cheat menu, for controlling the built-in cheat engine. This item is only shown if the built-in chat engine is enabled. Note that the cheat plugin's menu is accessed via the Plugin Options menu.

Plugin Options
Shows the Plugin Options menu, where you can access settings for enabled plugins. This item is not shown if no plugins are enabled, or if the main menu is shown before the emulated system starts (by pressing the Show/Hide Menu key/button while the system information screen is displayed).

External DAT View
Shows the info viewer, which displays information loaded from various external support files. This item is not shown if the *data plugin* is not enabled, or if the main menu is shown before the emulated system starts (by pressing the Show/Hide Menu key/button while the system information screen is displayed).

Add To Favorites/Remove From Favorites
Adds the running system to the favourites list, or removes it if it's already in the favourites list. The favourites list can be used as a filter for the system selection menu.

About MAME
Shows the emulator version, data model, and copyright license information.

Select New System
Shows the system selection menu, where you can select a system to start a new emulation session. This item is not shown if the main menu is shown before the emulated system starts (by pressing the Show/Hide Menu key/button while the system information screen is displayed).

Close Menu/Start System
Closes the main menu, returning control of the running system. Shows **Start System** if the main menu is shown before the emulated system starts (by pressing the Show/Hide Menu key/button while the system information screen is displayed).

4.4.3 Input Settings menu

The Input Settings provides options for assigning controls to emulated inputs, adjusting analog control settings, controlling toggle inputs, and testing input devices. You can reach the Input Settings menu by selecting **Input Settings** from the *main menu*. The items shown on this menu depend on available emulated inputs for the running system. Available emulated inputs may depend on slot options, machine configuration settings and DIP switch settings.

Input Assignments (this system)
: Lets you select assign controls to emulated inputs for the running system. See the section on *configuring inputs* for more details. This item is not shown if the running system has no enabled inputs that can be assigned controls.

Analog Input Adjustments
: Shows the Analog Input Adjustments menu, where you can adjust sensitivity, auto-centring speed and inversion settings for emulated analog inputs, and see how the emulated analog inputs respond to controls with your settings. For more details, see the *analog input settings* section for more details. This item is not shown if the running system has no enabled analog inputs.

Keyboard Selection
: Shows the *Keyboard Selection menu*, where you can select between emulated and natural keyboard modes, and enable and disable keyboard and keypad inputs for individual emulated devices. This item is not shown if the running system has no keyboard or keypad inputs.

Toggle Inputs
: Shows the *Toggle Inputs menu*, where you can view and adjust the state of multi-position or toggle inputs. This item is not shown if the running system has no enabled toggle inputs.

Input Assignments (general)
: Lets you select assign user interface controls, or assign default controls for all emulated systems. See the section on *configuring inputs* for more details.

Input Devices
: Shows the *Input Devices menu*, which lists the input devices recognised by MAME.

4.4.4 Toggle Inputs menu

The Toggle Inputs menu shows the current state of multi-position or toggle inputs. Common examples include mechanically locking Caps Lock keys on computers, and two-position gear shit levers on driving games. You can reach the Toggle Inputs menu by selecting **Toggle Inputs** from the *Input Settings menu*. Note that available emulated inputs may depend on slot options, machine configuration settings and DIP switch settings.

Inputs are grouped by the emulated device they belong to. You can move between devices using the **Next Group** and **Previous Group** keys or buttons. Names of inputs are shown on the left, and the current settings are shown on the right.

To change the state of an input, highlight it and use the **UI Left** and **UI Right** keys or buttons, or click the arrows beside the current setting.

4.4.5 Keyboard Selection menu

The Keyboard Selection menu lets your switch between emulated and natural keyboard modes, and enable or disable keyboard inputs for individual emulated devices. You can reach the Keyboard Selection menu by selecting **Keyboard Selection** from the *Input Settings menu*.

In emulated keyboard mode, keyboard and keypad inputs behave like any other digital inputs, responding to their assigned controls. In natural keyboard mode, MAME attempts to translate typed characters to emulated keystrokes. The initial keyboard mode is set using the *natural option*.

There are a number of unavoidable limitations in natural keyboard mode:

- The emulated system must to support it.
- The selected keyboard *must* match the keyboard layout selected in the emulated software.
- Keystrokes that don't produce characters can't be translated. (e.g. pressing a modifier key on its own, such as **Shift** or **Control**).
- Holding a key until the character repeats will cause the emulated key to be pressed repeatedly as opposed to being held down.
- Dead key sequences are cumbersome to use at best.
- Complex input methods will not work at all (e.g. for Chinese/Japanese/Korean).

Each emulated device in the system that has keyboard and/or keypad inputs is listed on the menu, allowing keyboard/keypad inputs to be enabled or disabled for individual devices. By default, keyboard/keypad inputs are enabled for the first device with keyboard inputs (if any), and for all other devices that have keypad inputs but no keyboard inputs. The enabled keyboard/keypad inputs are automatically saved to the configuration file for the system when the emulation session ends.

4.4.6 Input Devices menu

The Input Devices menu lists input devices recognised by MAME and enabled with your current settings. Recognised input devices depend on the *keyboardprovider*, *mouseprovider*, *lightgunprovider* and *joystickprovider* options. Classes of input devices can be enabled or disabled using the *mouse*, *lightgun* and *joystick* options. You can reach the Input Devices menu by selecting **Input Devices** from the *Input Settings menu* or the General Settings menu.

Input devices are grouped by device class (for example keyboards or light guns). You can move between device classes using the **Next Group** and **Previous Group** keys or buttons. For each device, the device number (within its class) is shown on the left, and the name is shown on the right.

Select a device to show the supported controls for the device. The name of each control is displayed on the left and its current state is shown on the right. When an analog axis control is highlighted, its state is also shown in graphical form below the menu. Digital control states are either zero (inactive) or one (active). Analog axis input states range from -65,536 to 65,536 with the neutral position at zero. You can also select **Copy Device ID** to copy the device's ID to the clipboard. This is useful for setting up *stable controller IDs* in *controller configuration files*.

4.4.7 Audio Mixer menu

The Audio Mixer menu allows you to configure how MAME routes sound from emulated speakers to system sound outputs, and from system sound inputs to emulated microphones. There are two kinds of routes: *full routes*, and *channel routes*:

- A full output route sends sound from all channels of an emulated sound output device to a host sound output. MAME automatically decides how to assign emulated channels (typically speakers) to output channels, based on speaker position information.
- Similarly, a full input route sends sound from a host sound input to all channels of an emulated sound input device. MAME automatically decides how to assign input channels to emulated channels (typically microphones), based on microphone position information.
- A channel route sends sound from a single emulated sound output channel to a single host sound output channel, or from a single host sound input channel to a single emulated sound input channel.

Only one full route is allowed for each combination of an emulated sound output or input device and host sound output or input. Only one channel route is allowed between an individual emulated channel and an individual host sound channel.

Routes are grouped by emulated device. For each device, full routes are listed before channel routes. For each route, you can select the system sound output or input and adjust the volume from -96 dB (quietest) to +12 dB (loudest). For channel routes, you can also select the individual emulated channel and host channel. Select **Remove this route** to remove a route.

Select **Add new full route** to add a new full route to that group. If possible, it will be added and the menu highlight will move to the newly added route. If routes between the highlighted device and every host output/input already exist, no route will be added.

Select **Add new channel route** to add a new channel route to that group. If possible, it will be added and the menu highlight will move to the newly added route. If routes between all channels for the highlighted device and every host output/input channel already exist, no route will be added.

Some sound modules allow channel assignments and volumes to be controlled using an external mixer interface (for example the PipeWire module for Linux has this capability). In these cases, MAME does its best to follow the changes you make in the external mixer interface and save changes in its configuration.

The audio routes are saved in the system configuration file.

4.4.8 Audio Effects menu

The Audio Effects menu allows you to configure audio effects that are applied to emulated sound output before it's routed to host sound outputs. An independent effect chain is applied for each emulated sound output device.

The effect chain itself is not configurable. It always consists of these four effects, in this order:

- Filters
- Compressor
- Reverb
- Equalizer

When editing parameters for an output device's effect chain, inherited default parameter values are showing dimmed, while parameter values set for that chain are shown in the normal text colour. Press the UI Clear key (Del/Delete/Forward Delete on the keyboard by default) to reset a parameter to use the inherited default value.

Edit the **Default** chain to set default parameter value that can be inherited by output device chains. When editing the **Default** chain, you can restore the built-in default value for a parameter by pressing the UI Clear key (Del/Delete/Forward Delete on the keyboard by default).

By default, the high-pass filter is enabled, with minimal cutoff frequency for DC offset removal. All other effects are bypassed (technically, the equalizer effect is active too, but all bands are set to 0 dB so it's still turned off).

The Audio Effects menu also allows you to configure the algorithm used for audio sample rate conversion. The default **LoFi** algorithm has modest CPU requirements. The recommended **HQ** algorithm provides higher quality sample rate conversion at the expense of requiring substantially higher CPU performance.

The **HQ** algorithm has additional parameters. Increasing the **HQ latency** can improve quality. If it's increased too much and multiple sound chips are used, the latencies will stack up and you will end up with too much lag at the end. When decreasing the latency below 1 ms, the resampler will lose its potential (in fact, it will sound similar to MAME's lower quality resampler from before version 0.278). Increasing the **HQ filter max size** or **HQ filter max phases** can improve quality at the expense of higher CPU performance requirements.

Filter effect

This effect implements a second-order high-pass filter and a second-order low-pass filter. The high-pass filter allows DC offsets to be removed. The low-pass filter can simulate the poor high-frequency response typical of many arcade cabinets and television sets.

The Q factor controls how sharp the transition from the stop band to the passband is. Higher factors provide a sharper transition. Values over 0.71 cause the filter to amplify frequencies close to the cutoff frequency, which may be surprising or undesirable.

Compressor effect

This effect provides dynamic range compression (it is based on a reimplementation of Alain Paul's Versatile Compressor). Dynamic range compression reduces the difference in volume between the softest and loudest sounds. It's useful in a variety of situations, for example it can help make quiet sounds more audible over background noise.

The parameters are:

Threshold
: The level at which the amplification fully stops.

Ratio
: The maximum amplification.

Attack
: The reaction time to loud sounds to reduce the amplification.

Release
: The reaction time to allow the amplification to go back up.

Input gain
: The amplification level at the input.

Output gain
: The amplification level at the output.

Convexity
: The shape of the relationship between distance to the threshold and ratio value. Higher values give a steeper shape.

Channel link
: At 100%, all channels of an output device are amplified identically, while at 0% they are fully independent. Intermediate values give intermediate behaviour.

Feedback
: Allows some of the output to be fed back to the input.

Inertia
: Higher values make the ratio change more slowly.

Inertia decay
: Tweaks the impact of the Inertia.

Ceiling
: The maximum level allowed just before the output amplification. Causes soft clipping at that level.

By setting **Attack** to 0 ms, **Release** to Infinite, and **Ratio** to Infinity:1, the compressor will turn into a brickwall limiter (leave the advanced settings to default). If you increase **Input gain** on top of that, with a **Threshold** of eg. -3 dB, it will act like a dynamic normalizer.

Reverb effect

Not documented yet.

Equalizer effect

A five-band parametric equalizer, allowing to amplify or attenuate specific frequency bands.

The three middle filters are bandpass/bandreject filters, meaning they amplify or attenuate frequencies around the configured centre frequency. The first and last filters can also be configured as bandpass/bandreject filter by setting the mode to **Peak**. Setting the mode to **Shelf** causes the filter to amplify or attenuate all frequencies below (for the first filter) or above (for the last filter) the configured cutoff frequency.

The Q factor controls the sharpness of the peak or trough in frequency response for bandpass/bandreject filters (the Q factor is not adjustable for **Shelf** mode). Higher Q factors give a sharper shape, affecting a narrower range of frequencies.

4.5 How does MAME look for files?

- *Introduction*
  - *Terminology*
- *Search path options*
- *Archive files*
- *How does MAME search for media?*
  - *System ROMs*
  - *Device ROMs*
  - *Software Item ROMs*
  - *CHD format disk images*

- *Loose software*
- *Diagnosing missing media*

4.5.1 Introduction

Unlike typical desktop applications where you browse your disk and select a file to open or a location to save to, MAME has settings to tell it where to look for the files it needs. You can change these settings by starting MAME without specifying a system, selecting **Configure Options** from the system selection menu, and then selecting **Configure Directories** (remember to select **Save Configuration** if you want to keep your changes). You can also change settings by editing your mame.ini and ui.ini files directly, or specify settings on the command line. For information on available options for controlling where MAME searches for files, see *Core Search Path Options*.

Terminology

It's necessary to understand some MAME-specific terminology used in the explanations here:

System
: A system is a complete machine that can be emulated by MAME. Some systems run fixed software, while others can load software from software list items and/or media files.

Device
: An emulated component that can be used by multiple systems, or by other devices. Some devices require ROM dumps, and some devices allow software from additional software lists to be used with a system.

Parent system
: MAME uses so-called parent/clone relationships to group related systems. One system in the group is chosen to be the *parent* and the others are called *clones*. (The choice of the parent system is somewhat arbitrary. It is not necessarily the original or definitive variant.)

BIOS system
: A system configured with no software. This is mostly applicable for arcade systems that used interchangeable game cartridges or ROM boards. Note that this is *not* the same as the BIOS selection settings that allow you to select system boot ROMs or device firmware.

Software item
: A software package described in a software list. Software items may consist of multiple *parts* that can be mounted independently. Due to the large variety of media supported by MAME, software parts may use different *loaders*. These include the *ROM loader*, typically used for cartridge media, and the *image file loader*, used for software parts consisting of a single media image (including floppy disk and cassette media).

Parent software item
: Related software items are grouped using parent/clone relationships, in a similar way to related systems. This is usually used to group different versions or releases of the same piece of software. If a software item has a parent item, it will always be in the same software list.

Short name
: MAME uses *short names* to uniquely identify systems and devices, to uniquely identify software lists, to uniquely identify software items within a software list, and to uniquely identify software parts within a software item.

  You can see the short name for a system by highlighting it in the system selection menu, ensuring the info panel is visible on the right, and showing the **General Info** in the **Infos** tab. For example the short name for the Nintendo Virtual Boy is `vboy`. System and device short names can also be seen in the output of various command line verbs, including `-listxml`, `-listfull`, `-listroms` and `-listcrc`.

You can see the short names for a software item and the software list it belongs to by highlighting it in the software selection menu, ensuring the info panel is visible on the right, and showing the **Software List Info** in the **Infos** tab. For example the short name for Macintosh System Software 6.0.3 is `sys603` and the short name of the software list it belongs to is `mac_flop`. Software list short names match their file names (for example the Sega Mega Drive/Genesis cartridge software list is called **megadriv.xml** and its short name is `megadriv`). You can also see the short names software lists, software items and parts by finding the `name` attributes in the XML software list files.

4.5.2 Search path options

Most options for specifying locations to search allow multiple directories to be specified, separated by semicolon (`;`) characters. Environment variables are expanded, using CMD shell syntax on Windows, or Bourne shell syntax on UNIX-like systems.

Relative paths are interpreted relative to the current working directory at the time of use. If you start MAME by double-clicking it in Windows Explorer, the working directory is set to the folder containing the MAME executable. If you start MAME by double-clicking it in the macOS Finder or from most Linux desktop environments, the working directory will be set to your home directory.

4.5.3 Archive files

MAME can load files from PKZIP and 7-Zip archives (these must have `.zip` and `.7z` file name extensions, respectively). A number of extensions to the PKZIP format are supported, including Zip64 for large archives, NTFS timestamps, and LZMA compression. Only ASCII or UTF-8 filenames are supported in PKZIP archives (7-Zip archives always use UTF-16 filenames).

MAME *does not* load files from nested archives. MAME will not load files stored in a PKZIP or 7-Zip archive which is itself contained within a PKZIP or 7-Zip archive. Multi-segment archives and encrypted archives are not supported. The legacy "implode" compression method in PKZIP archives is not supported.

MAME may perform poorly with archives containing large numbers of files. Files compressed using the LZMA compression algorithm are inherently more CPU-intensive to decompress than files compressed using simpler algorithms. MAME does not take the archive layout into consideration when loading files from archives, so using "solid compression" often results in MAME decompressing the same data repeatedly when loading media.

4.5.4 How does MAME search for media?

Use the *rompath* option sets the folders where searches for ROM dumps, disk images, and other media. By default MAME looks for media in a folder called **roms** in the working directory. For the purpose of this discussion, floppy disk, cassette, paper tape and other media images that are not stored in CHD format are treated as ROM dumps.

When searching for system, device and software ROM dumps, MAME treats folders and archives inside the folders configured in you `rompath` setting as equivalent, but remember the limitation that MAME cannot load files from an archive contained within another archive. MAME looks for a folder first, then a PKZIP archive, and finally a 7-Zip archive. When searching for a ROM dump in an archive, MAME first looks for a file with the expected name and CRC. If no matching file is found, MAME looks for a file with the expected CRC ignoring the name. If no matching file is found, MAME finally looks for a file with the expected name, ignoring the CRC.

While MAME can load disk images in CHD format from inside archives, this is not recommended. CHD files contain compressed data stored in a format allowing random access. If a CHD format disk image is stored in a PKZIP or 7-Zip archive, MAME needs to load the entire file into memory in order to use it. For hard disk or LaserDisc images in particular, this will likely use an excessive amount of swap file space, hurting performance and possibly reducing the life expectancy of your disks or SSDs. It's best to keep CHD format disk images in folders.

System ROMs

For each folder configured in your `rompath` setting, MAME looks for system ROMs in the following locations:

- A folder or archive matching the short name of the system itself.
- A folder or archive matching the short name of the system's parent system, if applicable.
- A folder or archive matching the short name of the corresponding BIOS system, if applicable.

Using Shiritsu Justice Gakuen as an example, MAME will search for system ROMs as follows:

- The short name of the system is `jgakuen`, so MAME will look for a folder called **jgakuen**, a PKZIP archive called **jgakuen.zip**, or a 7-Zip archive called **jgakuen.7z**.
- The parent system is the European version of Rival Schools, which has the short name `rvschool`, so MAME will look for a folder called **rvschool**, a PKZIP archive called **rvschool.zip**, or a 7-Zip archive called **rvschool.7z**.
- The corresponding BIOS system is the Capcom ZN2 board, which has the short name `coh3002c`, so MAME will look for a folder called **coh3002c**, a PKZIP archive called **coh3002c.zip**, or a 7-Zip archive called **coh3002c.7z**.

Device ROMs

For each folder configured in your `rompath` setting, MAME looks for device ROMs in the following locations:

- A folder or archive matching the short name of the device.
- A folder or archive matching the short name of the device's parent ROM device, if applicable.
- A folder or archive matching the short name of the system.
- A folder or archive matching the short name of the system's parent system, if applicable.
- A folder or archive matching the short name of the corresponding BIOS system, if applicable.

Using a Unitron 1024 Macintosh clone with a French Macintosh Plus keyboard with integrated numeric keypad attached as an example, MAME will look for the keyboard microcontroller ROM as follows:

- The short name of the French Macintosh Plus keyboard is `mackbd_m0110a_f`, so MAME will look for a folder called **mackbd_m0110a_f**, a PKZIP archive called **mackbd_m0110a_f.zip**, or a 7-Zip archive called **mackbd_m0110a_f.7z**.
- The parent ROM device is the U.S. Macintosh Plus keyboard with integrated numeric keypad, which has the short name `mackbd_m0110a`, so MAME will look for a folder called **mackbd_m0110a**, a PKZIP archive called **mackbd_m0110a.zip**, or a 7-Zip archive called **mackbd_m0110a.7z**.
- The short name of the Unitron 1024 system is `utrn1024`, so MAME will look for a folder called **utrn1024**, a PKZIP archive called **utrn1024.zip**, or a 7-Zip archive called **utrn1024.7z**.
- The parent system of the Unitron 1024 is the Macintosh Plus, which has the short name `macplus`, so MAME will look for a folder called **macplus**, a PKZIP archive called **macplus.zip**, or a 7-Zip archive called **macplus.7z**.
- There is no corresponding BIOS system, so MAME will not search in any further locations.

Software Item ROMs

For each folder configured in your `rompath` setting, MAME looks for software item ROMs in the following locations:

- A folder or archive matching the short name of the software item inside a folder matching the short name of the software list (or a folder matching the short name of the software item inside an archive matching the name of the software list).
- A folder or archive matching the short name of the parent software item inside a folder matching the short name of the software list, if applicable (or a folder matching the short name of the parent software item in an archive matching the name of the software list).
- A folder or archive matching the short name of the software item. (This is for convenience for software items that also run as stand-alone systems with the same short name, such as Neo Geo games.)
- A folder or archive matching the short name of the parent software item, if applicable. (This is for convenience for software items that also run as stand-alone systems with the same short name, such as Neo Geo games.)

If you load the German version of Dune II from the Mega Drive/Genesis cartridge software list in the PAL Mega Drive console, MAME will look for the cartridge ROM as follows:

- The short name of the software item for the German version of Dune II is `dune2g` and the short name of the Mega Drive/Genesis cartridge software list is `megadriv`, so MAME will look for a folder called **dune2g**, a PKZIP archive called **dune2g.zip** or a 7-Zip archive called **dune2g.7z** inside a folder called **megadriv** (or a folder called **dune2g** inside a PKZIP archive called **megadriv.zip** or a 7-Zip archive called **megadriv.7z**).
- The parent software item is the general European PAL version of Dune II in the same software list, which has the short name `dune2`, so MAME will look for a folder called **dune2**, a PKZIP archive called **dune2.zip** or a 7-Zip archive called **dune2.7z** inside a folder called **megadriv** (or a folder called **dune2** inside a PKZIP archive called **megadriv.zip** or a 7-Zip archive called **megadriv.7z**).
- Next MAME will ignore the short name of the software list and use the short name of the software item only, looking for a folder called **dune2g**, a PKZIP archive called **dune2g.zip** or a 7-Zip archive called **dune2g.7z**.
- Still ignoring the short name of the software list, MAME will use the short name of the parent software item only, looking for a folder called **dune2**, a PKZIP archive called **dune2.zip** or a 7-Zip archive called **dune2.7z**.

CHD format disk images

MAME searches for system, device and software item CHD format disk images in almost the same way it searches for ROMs, with just a few differences:

- For systems and software items, MAME will check the parent system or software item if applicable for alternate names for a disk image with the same content digest. This allows you to keep a single copy of a CHD format disk image for a parent system or software item and any clones that expect a disk image with the same content, irrespective of the name the clones expect.
- For software items, MAME will look for CHD format disk images in a folder matching the short name of the software list. This is for convenience when all items in a software list only contain a single CHD format disk image each.
- We recommend that you *do not* store CHD format disk images inside PKZIP or 7-Zip archives. However, if you do decide to do this, MAME will only find CHD format disk images inside archives with an expected name. This is because MAME uses the content digest from the CHD header, not the checksum of the CHD file itself. The checksum of the CHD file itself can vary depending on compression options.

To save space, MAME allows delta CHD files to be used for clone systems, devices with parent ROM devices and clone software items. The delta CHD file must use a CHD format disk image from the parent system, parent ROM device or parent software item as its parent CHD file. The space saved depends on how much content can be reused from the parent CHD file. MAME searches the same locations for parent CHD files that it would search for the disk image itself.

Loose software

Many systems support loading media from a file by supplying the path on the command line for one of the media options. Relative paths are interpreted relative to the current working directory.

You can specify a path to a file inside a PKZIP or 7-Zip archive similarly to specifying a path to a file in a folder (keep in mind that you can have at most a single archive file in a path, as MAME does not support loading files from archives contained within other archives). If you specify a path to a PKZIP or 7-Zip archive, MAME will use the first file found in the archive (this depends on the order that files are stored in the archive – it's most useful for archives containing a single file).

Start the Nintendo Entertainment System/Famicom system with the file **amazon_diet_EN.nes** mounted in the cartridge slot:

```
mame nes -cart amazon_diet_EN.nes
```

Start the Osborne-1 system with the first file in the archive **os1xutls.zip** mounted in the first floppy disk drive:

```
mame osborne1 -flop1 os1xutils.zip
```

Start the Macintosh Plus system with the file **system tools.img** in the archive **sys603.zip** mounted in the first floppy disk drive:

```
mame macplus -flop1 "sys603.zip/system tools.img"
```

Diagnosing missing media

When starting a system from MAME's system selection menu or software selection menu, MAME will list any missing system or device ROM dumps or disk images, as long as at least one ROM dump or disk image for the system is present. For clone systems, at least one ROM dump or disk image *unique to the clone* must be present for MAME to list missing ROM dumps and disk images.

If all system and device ROM dump and disk images are present and the system is being started with a software item, MAME will check that ROM dumps and disk images for the software item are present. If at least one ROM dump or disk image for the software item is present, MAME will list any missing ROM dumps or disk images.

For example if you try to start the Macintosh Plus system and the keyboard microcontroller ROM dump is missing, MAME displays the following error message:

> Required ROM/disk images for the selected system are missing or incorrect. Please acquire the correct files or select a different system.
>
> 341-0332-a.bin (mackbd_m0110a) - not found
>
> Press any key to continue.

The name of the missing ROM dump is shown (**341-0332-a.bin**), as well as the short name of the device it belongs to (`mackbd_m0110a`). When a missing ROM dump or disk image is not specific to the selected system, the short name of the system or device it belongs to is shown.

If you start a system in MAME from a command prompt, MAME will show where it searched for any ROM dumps or disk images that were not found.

Using the example of a Unitron 1024 Macintosh clone with a French keyboard connected, MAME will show the following error messages if no ROMs are present:

```
mame utrn1024 -kbd frp
342-0341-a.u6d NOT FOUND (tried in utrn1024 macplus)
342-0342-a.u8d NOT FOUND (tried in utrn1024 macplus)
341-0332-a.bin NOT FOUND (tried in mackbd_m0110a_f mackbd_m0110a utrn1024 macplus)
```

MAME used the system short name `utrn1024` and the parent system short name `macplus` when searching for system ROMs. When searching for the keyboard microcontroller ROM, MAME used the device short name `mackbd_m0110a_f`, the parent ROM device short name `mackbd_m0110a`, the system short name `utrn1024`, and the parent system short name `macplus`.

Software parts that use the ROM loader (typically cartridge media) show similar messages when ROM dumps are not found. Using the example of the German version of Dune II on a PAL Mega Drive, MAME will show the following error messages if no ROMs are present:

```
mame megadriv dune2g
mpr-16838-f.u1 NOT FOUND (tried in megadriv\dune2g megadriv\dune2 dune2g dune2 megadriv␣
↪genesis)
Fatal error: Required files are missing, the machine cannot be run.
```

MAME searched for the cartridge ROM using:

- The software list short name `megadriv` and the software item short name `dune2g`.
- The software list short name `megadriv` and the parent software item short name `dune2`.
- The software item short name `dune2g` only.
- The parent software item short name `dune2` only.
- The locations that would be searched for the PAL Mega Drive system (the system short name `megadriv` and the parent system short name `genesis`).

Software parts that use the image file loader (including floppy disk and cassette media) only check for media after ROM images are loaded, and missing media files are shown differently. Using the example of Macintosh System 6.0.3, MAME will show these error messages if the software is missing:

```
mame macplus -flop1 sys603:flop1
:fdc:0:35dd: error opening image file system tools.img: No such file or directory␣
↪(generic:2) (tried in mac_flop\sys603 sys603 macplus)
Fatal error: Device Apple/Sony 3.5 DD (400/800K GCR) load (-floppydisk1 sys603:flop1)␣
↪failed: No such file or directory
```

The error messages show where MAME searched for the image file in the same format. In this case, it used the software list short name `mac_flop` and the software short name `sys603`, the software short name `sys603` only, and the locations that would be searched for system ROMs.

4.6 Front-ends

A number of third party tools for MAME to make system and software selection simpler are available. These tools are called “front-ends”, and there are far too many to list exhaustively here. Some are free, some are commercial – caveat emptor. Some older front-ends predate the merging of MAME and MESS and do not support the additional console, hand-held, and computer functionality inherited from MESS.

This following list is not an endorsement of any of these front-ends by the MAME team. It simply shows a number of commonly used free front-ends to provide a starting point.

QMC2 (multiple platforms)
Provides a graphical interface for configuring many of MAME's settings and features. Also includes ROM management and media auditing features. Written in C++ using the Qt toolkit, the source code is on SourceForge.

Negatron (multiple platforms)
Negatron emphasises features for configuring emulated computers and consoles. Written in Java, the source code is on GitHub.

BletchMAME (multiple platforms)
BletchMAME takes advantage of MAME's Lua scripting interface to integrate tightly and effectively replace MAME's internal user interface. It has many useful features for home computer emulation. Written in C++, the source code is on GitHub.

IV/Play (Microsoft Windows)
IV/Play has a customisable interface designed to be navigated with a keyboard. It supports software lists and has flexible filtering.

Retrofire (Japanese, Microsoft Windows)
Provides a Japanese-language graphical interface for launching systems or software in MAME.

The MAME team will not provide support for issues with front-ends. For support, we suggest contacting the front-end author or asking on one of the popular MAME-friendly forums on the Internet.

4.7 About ROMs and Sets

Handling and updating of ROMs and Sets used in MAME is probably the biggest area of confusion and frustration that MAME users will run into. This section aims to clear up a lot of the most common questions and cover simple details you'll need to know to use MAME effectively.

Let's start with a simple definition of what a ROM is.

4.7.1 What is a ROM image?

For arcade games, a ROM image or file is a copy of all of the data inside a given chip on the arcade motherboard. For most consoles and handhelds, the individual chips are frequently (but not always) merged into a single file. As arcade machines are much more complicated in their design, you'll typically need the data from a number of different chips on the board. Grouping all of the files from Puckman together will get you a **ROM set** of Puckman.

An example ROM image would be the file **pm1_prg1.6e** stored in the **Puckman** ROM set.

4.7.2 Why ROM and not some other name?

ROM stands for Read-Only Memory. The chips used to store the game data were not rewritable and were permanent (as long as the chip wasn't damaged or aged to death!)

As such, a copy of the data necessary to reconstitute and replace a dead data chip on a board became known as a "ROM image" or ROMs for short.

4.7.3 Parents, Clones, Splitting, and Merging

As the MAME developers received their third or fourth revision of Pac-Man, with bugfixes and other code changes, they quickly discovered that nearly all of the board and chips were identical to the previously dumped version. In order to save space, MAME was adjusted to use a parent/clone set system.

A given set, usually (but not necessarily) the most recent bugfixed World revision of a game, will be designated as the parent. All sets that use mostly the same chips (e.g. Japanese Puckman and USA/World Pac-Man) will be clones that contain only the changed data compared to the parent set.

This typically comes up as an error message to the user when trying to run a Clone set without having the Parent set handy. Using the above example, trying to play the USA version of Pac-Man without having the **PUCKMAN.ZIP** parent set will result in an error message that there are missing files.

Now we add the final pieces of the puzzle: non-merged, split, and merged sets.

MAME is extremely versatile about where ROM data is located and is quite intelligent about looking for what it needs. This allows us to do some magic with how we store these ROM sets to save further space.

A **non-merged set** is one that contains absolutely everything necessary for a given game to run in one ZIP file. This is ordinarily very space-inefficient, but is a good way to go if you want to have very few sets and want everything self-contained and easy to work with. We do not recommend this for most users.

A **split set** is one where the parent set contains all of the normal data it should, and the clone sets contain *only* what has changed as compared to the parent set. This saves some space, but isn't quite as efficient as

A **merged set** takes the parent set and one or more clone sets and puts them all inside the parent set's storage. For instance, if we combine the Puckman sets, Midway Pac-Man (USA) sets, and various other related official and bootleg sets all into **PUCKMAN.ZIP**, the result would be a **merged set**. A complete merged set with the parent and all clones uses less disk space than a split set.

With those basic principles, there are two other kinds of set that will come up in MAME use from time to time.

First, the **BIOS set**: Some arcade machines shared a common hardware platform, such as the Neo-Geo arcade hardware. As the main board had data necessary to start up and self-test the hardware before passing it off to the game cartridge, it's not really appropriate to store that data as part of the game ROM sets. Instead, it is stored as a BIOS image for the system itself (e.g. **NEOGEO.ZIP** for Neo-Geo games)

Secondly, the **device set**. Frequently the arcade manufacturers would reuse pieces of their designs multiple times in order to save on costs and time. Some of these smaller circuits would reappear in later boards that had minimal common ground with the previous boards that used the circuit, so you couldn't just have them share the circuit/ROM data through a normal parent/clone relationship. Instead, these re-used designs and ROM data are categorized as a *Device*, with the data stored as a *Device set*. For instance, Namco used the *Namco 51xx* custom I/O chip to handle the joystick and DIP switches for Galaga and other games, and as such you'll also need the **NAMCO51.ZIP** device set as well as any needed for the game.

4.7.4 Troubleshooting your ROM sets and the history of ROMs

A lot of the frustration users feel towards MAME can be directly tied to what may feel like pointless ROM changes that seem to only serve to make life more difficult for end-users. Understanding the source of these changes and why they are necessary will help you to avoid being blindsided by change and to know what you need to do to keep your sets running.

A large chunk of arcade ROMs and sets existed before emulation did. These early sets were created by arcade owners and used to repair broken boards by replacing damaged chips. Unfortunately, these sets eventually proved to be missing critical information. Many of the early dumps missed a new type of chip that contained, for instance, color palette information for the screen. The earliest emulators approximated colors until the authors discovered the existence of these missing chips. This resulted in a need to go back and get the missing data and update the sets to add the new dumps as needed.

It wouldn't be much longer before it would be discovered that many of the existing sets had bad data for one or more chips. These, too, would need to be re-dumped, and many sets would need complete overhauls.

Occasionally games would be discovered to be completely wrongly documented. Some games thought to be legitimate ended up being bootleg copies from pirate manufacturers. Some games thought to be bootlegs ended up being legit. Some games were completely mistaken as to which region the board was actually from (e.g. World as compared to Japan) and this too would require adjustments and renaming.

Even now, occasional miracle finds occur that change our understanding of these games. As accurate documentation is critical to detailing the history of the arcades, MAME will change sets as needed to keep things as accurate as possible within what the team knows at the time of each release.

This results in very spotty compatibility for ROM sets designated for older versions of MAME. Some games may not have changed much within 20-30 revisions of MAME, and others may have drastically changed multiple times.

If you hit problems with a set not working, there are several things to check-- are you trying to run a set meant for an older version of MAME? Do you have any necessary BIOS or Device ROMs? Is this a Clone set that would need to have the Parent as well? MAME will tell you what files are missing as well as where it looked for these files. Use that to determine which set(s) may be missing files.

4.7.5 ROMs and CHDs

ROM chip data tends to be relatively small and are loaded into system memory in their entirety. Some games also used additional storage media such as hard disks, CD-ROMs, DVDs, and LaserDiscs. Those storage media are, for multiple technical reasons, not well-suited to being stored the same way as ROM data and won't fully fit in memory in some cases.

Thus, a new format was created for these in the CHD file. **Compressed Hunks of Data** files, or CHD files for short, are designed very specifically around the needs of mass storage media. Some arcade games, consoles, and PCs will require one or more CHD files to run. As CHD files are already compressed, they **should not** be stored PKZIP or 7-Zip archives as ROM images would be.

To save space when multiple variants of a system or software item are present, MAME supports *delta CHD* files. A delta CHD file only stores the parts of the data that differ from its *parent CHD* file. This allows large space savings when different variants share a lot of data. Delta CHD files can only be used for clone systems, devices with a parent ROM device, and clone software items. A delta CHD file must use a (non-delta) CHD file from the parent system, parent ROM device or parent software item as its parent CHD file. The parent CHD file must be present to use a delta CHD file, or MAME will not be able to read the shared data from it.

4.8 Common Issues and Questions (FAQ)

Disclaimer: The following information is not legal advice and was not written by a lawyer.

1. *Why does my game show an error screen if I insert coins rapidly?*
2. *Why is my non-official MAME package (e.g. EmuCR build) broken? Why is my official update broken?*
3. *Why does MAME support console games and dumb terminals? Wouldn't it be faster if MAME had just the arcade games? Wouldn't it take less RAM? Wouldn't MAME be faster if you just X?*
4. *Why do my Neo Geo ROMs no longer work? How do I get the Humble Bundle Neo Geo sets working?*
5. *How can I use the Sega Genesis & Mega Drive Classics collection from Steam with MAME?*
6. *Why does MAME report "missing files" even if I have the ROMs?*
7. *How can I be sure I have the right ROMs?*

8. *Why is it that some games have the US version as the main set, some have Japanese, and some are the World?*
9. *How do I legally obtain ROMs or disk images to run on MAME?*
10. *Isn't copying ROMs a legal gray area?*
11. *Can't game ROMs be considered abandonware?*
12. *I had ROMs that worked with an old version of MAME and now they don't. What happened?*
13. *What about those arcade cabinets on eBay that come with all the ROMs?*
14. *What about those guys who burn DVDs of ROMs for the price of the media?*
15. *But isn't there a special DMCA exemption that makes ROM copying legal?*
16. *But isn't it OK to download and "try" ROMs for 24 hours?*
17. *If I buy a cabinet with legitimate ROMs, can I set it up in a public place to make money?*
18. *But I've seen Ultracade and Global VR Classics cabinets out in public places? Why can they do it?*
19. *HELP! I'm getting a black screen or an error message in regards to DirectX on Windows!*
20. *I have a controller that doesn't want to work with the standard Microsoft Windows version of MAME, what can I do?*
21. *What happened to the MAME support for external OPL2-carrying soundcards?*
22. *What happened to the MAME support for autofire?*
23. *Does MAME support G-Sync or FreeSync? How do I configure MAME to use them?*

4.8.1 Why does my game show an error screen if I insert coins rapidly?

This is not a bug in MAME. On original arcade hardware, you simply could not insert coins as fast as you can mash the button. The only ways you could feed credits at that kind of pace was if the coin mech hardware was defective or if you were physically trying to cheat the coin mech.

In either case, the game would display an error for the operator to look into the situation to prevent cheating them out of their hard-earned cash. Keep a slow, coin-insert-ish pace and you won't trigger this.

4.8.2 Why is my non-official MAME package (e.g. EmuCR build) broken? Why is my official update broken?

Many MAME features, such as software lists, HLSL or BGFX shaders, Lua plugins and UI translations, use external files. Updates to these features often require the external files to be updated along with MAME. Unfortunately, builds provided by third parties may only include the main MAME executable, or may include outdated external files. Using an updated MAME executable with outdated external files causes issues with features reliant on the external files. Despite repeated requests that they distribute MAME complete with matching external files, some of these third parties persist in distributing incomplete or broken MAME updates.

As we have no control over how third parties distribute MAME, all we really can do is recommend against obtaining MAME from sites like EmuCR. We cannot provide any support for packages we didn't build ourselves. You can completely avoid these issues by compiling MAME yourself, or using an official package we provide.

You may also encounter this problem if you do not update the contents of the `hlsl`, `bgfx` or `plugins` folders when updating your MAME installation with a new official build.

4.8.3 Why does MAME support console games and dumb terminals? Wouldn't it be faster if MAME had just the arcade games? Wouldn't it take less RAM? Wouldn't MAME be faster if you just X?

This is a common misconception. The actual size of the MAME file doesn't affect the speed of it; only the parts that are actively being used are in memory at any given time.

In truth, the additional supported devices are a good thing for MAME as they allow us to stress test sections of the various CPU cores and other parts of the emulation that don't normally see heavy utilization. While a computer and an arcade machine may use the exact same CPU, how they use that CPU can differ pretty dramatically.

No part of MAME is a second-class citizen to any other part. Video poker machines are just as important to document and preserve as arcade games.

There's still room for improvements in MAME's speed, but chances are that if you're not already a skilled programmer any ideas you have will have already been covered. Don't let that discourage you-- MAME is open source, and improvements are always welcome.

4.8.4 Why do my Neo Geo ROMs no longer work? How do I get the Humble Bundle Neo Geo sets working?

Recently the Neo Geo BIOS was updated to add a new version of the Universe BIOS. This was done between 0.171 and 0.172, and results in an error trying to load Neo Geo games with an un-updated **neogeo.zip** set.

This also affects the Humble Bundle set: the games themselves are correct and up to date as of MAME 0.173 (and most likely will remain so) though you'll have to pull the ROM set .ZIP files out of the package somehow yourself. However, the Neo Geo BIOS set (**neogeo.zip**) included in the Humble Bundle set is incomplete as of the 0.172 release of MAME.

We suggest you contact the provider of your sets (Humble Bundle and DotEmu) and ask them to update their content to the newest revision. If enough people ask nicely, maybe they'll update the package.

4.8.5 How can I use the Sega Genesis & Mega Drive Classics collection from Steam with MAME?

As of the April 2016 update to the program, the ROM images included in the set are now 100% compatible with MAME and other Genesis/Mega Drive emulators. The ROMs are contained in the **steamapps\Sega Classics\uncompressed ROMs** folder as a series of *.68K* and *.SGD* images that can be loaded directly into MAME. PDF manuals for the games can be found in **steamapps\Sega Classics\manuals** as well.

4.8.6 Why does MAME report "missing files" even if I have the ROMs?

There can be several reasons for this:

- It is not unusual for the ROMs to change for a game between releases of MAME. Why would this happen? Oftentimes, better or more complete ROM dumps are made, or errors are found in the way the ROMs were previously defined. Early versions of MAME were not as meticulous about this issue, but more recent MAME builds are. Additionally, there can be more features of a game emulated in a later release of MAME than an earlier release, requiring more ROM code to run.
- You may find that some games require CHD files. A CHD file is a compressed representation of a game's hard disk, CD-ROM, or laserdisc, and is generally not included as part of a game's ROMs. However, in most cases, these files are required to run the game, and MAME will complain if they cannot be found.

- Some games such as Neo-Geo, Playchoice-10, Convertible Video System, Deco Cassette, MegaTech, MegaPlay, ST-V Titan, and others need their BIOS ROMs in addition to the game ROMs. The BIOS ROMs often contain ROM code that is used for booting the machine, menu processor code on multi-game systems, and code common to all games on a system. BIOS ROMS must be named correctly and left zipped inside your ROMs folder.
- Older versions of MAME needed decryption tables in order for MAME to emulate Capcom Play System 2 (a.k.a. CPS2) games. These are created by team CPS2Shock.
- Some games in MAME are considered "Clones" of another game. This is often the case when the game in question is simply an alternate version of the same game. Common alternate versions of games include versions with text in other languages, versions with different copyright dates, later versions or updates, bootlegs, etc. "Cloned" games often overlap some of the ROM code as the original or "parent" version of the game. To see if you have any "clones" type "**MAME -listclones**". To run a "cloned game" you simply need to place its parent ROM file in your ROMs folder (leave it zipped).

4.8.7 How can I be sure I have the right ROMs?

MAME checks to be sure you have the right ROMs before emulation begins. If you see any error messages, your ROMs are not those tested to work properly with MAME. You will need to obtain a correct set of ROMs through legal methods.

If you have several games and you wish to verify that they are compatible with the current version of MAME, you can use the *-verifyroms* parameter. For example:

mame -verifyroms robby ...checks your ROMs for the game *Robby Roto* and displays the results on the screen.

mame -verifyroms * >verify.txt ...checks the validity of ALL the ROMs in your ROMS directory, and writes the results to a textfile called *verify.txt*.

4.8.8 Why is it that some games have the US version as the main set, some have Japanese, and some are the World?

Parent and clone sets are a convenience feature to help keep different versions of the same system or software together. The decision on which set to make the parent will always be somewhat arbitrary, but we do have some guidelines:

- Prefer latest release version
- Prefer English language
- Prefer most widespread release
- Prefer most complete version
- Prefer versions that are uncensored, and have story/cutscenes intact
- Prefer versions that keep the original gameplay balance
- Prefer releases from original developers/publishers rather than licensees
- Prefer releases without region-specific notices or warnings

It's not always possible to choose a set that's preferred according to all criteria.

As an example, the World release of Ghouls'n Ghosts (*ghouls*) is the parent of the US release (*ghoulsu*) and the Japanese original Daimakaimura, as it is the most widespread English-language release, and has the story and cutscenes intact.

Another example is Midway Pac-Man (*pacman*), which is a clone of Namco Puck Man (*puckman*), because Pac-Man is a licensed version for the US market, while Puck Man was released by Namco themselves.

4.8.9 How do I legally obtain ROMs or disk images to run on MAME?

You have several options:

- You can obtain a license to them by purchasing one via a distributor or vendor who has proper authority to do so.
- You can download one of the ROM sets that have been released for free to the public for non-commercial use.
- You can purchase an actual arcade PCB, read the ROMs or disks yourself, and let MAME use that data.

Beyond these options, you are on your own.

4.8.10 Isn't copying ROMs a legal gray area?

No, it's not. You are not permitted to make copies of software without the copyright owner's permission. This is a black and white issue.

4.8.11 Can't game ROMs be considered abandonware?

No. Even the companies that went under had their assets purchased by somebody, and that person is the copyright owner.

4.8.12 I had ROMs that worked with an old version of MAME and now they don't. What happened?

As time passes, MAME is perfecting the emulation of older games, even when the results aren't immediately obvious to the user. Often times the better emulation requires more data from the original game to operate. Sometimes the data was overlooked, sometimes it simply wasn't feasible to get at it (for instance, chip "decapping" is a technique that only became affordable very recently for people not working in high-end laboratories). In other cases it's much simpler: more sets of a game were dumped and it was decided to change which sets were which version.

4.8.13 What about those arcade cabinets on eBay that come with all the ROMs?

If the seller does not have a proper license to include the ROMs with their system, they are not legally permitted to include any ROMs with the system. If they have purchased a license to the ROMs in your name from a distributor or vendor with legitimate licenses, then they may include the ROMs with the cabinet. After signing an agreement, cabinet owners that include legitimate licensed ROMs may be permitted to include a version of MAME that runs those ROMs and nothing more.

4.8.14 What about those guys who burn DVDs of ROMs for the price of the media?

What they are doing is just as unlawful as selling the ROMs outright. As long as somebody holds the copyright, making unauthorised copies is unlawful. If someone went on the internet and started a business of selling cheap copies of U2 albums for the price of media, do you think they would get away with it?

Even worse, a lot of these people like to claim that they are helping the project. In reality, they only create more problems for the MAME team. We are not associated with these people in any way, regardless of how "official" they may attempt to appear. By buying from them, you only help criminals profit from selling software they have no right to sell. **Anyone using the MAME name and/or logo to sell such products is also in violation of the MAME trademark.**

4.8.15 But isn't there a special DMCA exemption that makes ROM copying legal?

No, you have misread the exemptions. The exemption allows people to reverse-engineer the copy protection or encryption in computer programs that are obsolete. The exemption simply means that figuring out how these obsolete programs worked is not illegal according to the DMCA. It does not have any effect on the legality of making unauthorised copies of computer programs, which is what you are doing if you make copies of ROMs.

4.8.16 But isn't it OK to download and "try" ROMs for 24 hours?

This is an urban legend that was made up by people who made ROMs available for download from their web sites, in order to justify the fact that they were breaking the law. There is no provision like this in any copyright law.

4.8.17 If I buy a cabinet with legitimate ROMs, can I set it up in a public place to make money?

Probably not. ROMs are typically only licensed for personal, non-commercial purposes.

4.8.18 But I've seen Ultracade and Global VR Classics cabinets out in public places? Why can they do it?

Ultracade had two separate products. The Ultracade product is a commercial machine with commercial licenses to the games. These machines were designed to be put on location and make money, like traditional arcade machines. Their other product is the Arcade Legends series. These are home machines with non- commercial licenses for the games, and can only be legally operated in a private environment. Since their buyout by Global VR they only offer the Global VR Classics cabinet, which is equivalent to the earlier Ultracade product.

4.8.19 HELP! I'm getting a black screen or an error message in regards to DirectX on Windows!

You probably have missing or damaged DirectX runtimes. You can download the latest DirectX setup tool from Microsoft at https://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=35

Additional troubleshooting information can be found on Microsoft's website at https://support.microsoft.com/en-us/kb/179113

4.8.20 I have a controller that doesn't want to work with the standard Microsoft Windows version of MAME, what can I do?

By default, MAME on Microsoft Windows tries to read joystick(s), mouse/mice and keyboard(s) using the RawInput API. This works with most devices, and allows multiple keyboards and mice to be used independently. However, some device drivers are not compatible with RawInput, and it may be necessary to use DirectInput or window events to receive input. This is also the case for most software that simulates mouse or keyboard input, like JoyToKey, VNC or Remote Desktop.

You can try changing the *keyboardprovider*, *mouseprovider*, *joystickprovider* or *lightgunprovider* setting (depending on which kind of device you're having issues with) from `rawinput` to one of the other options such as `dinput` or `win32`. See *OSD-related Options* for details on input provider options

4.8.21 What happened to the MAME support for external OPL2-carrying sound-cards?

MAME 0.23 added support for using a sound card's onboard OPL2 (Yamaha YM3812 chip) instead of emulating the OPL2. This feature was only supported for DOS – it was never supported in official Windows versions of MAME. It dropped entirely as of MAME 0.60, as the OPL2 emulation in MAME had become advanced enough to be a better solution in almost all cases. MAME's OPL2 emulation is now superior to using a sound card's YM3812 in all cases, especially as modern sound cards lack a YM3812.

4.8.22 What happened to the MAME support for autofire?

A Lua plugin providing enhanced autofire support was added in MAME 0.210, and the built-in autofire functionality was removed in MAME 0.216. This plugin has more functionality than the built-in autofire feature it replaced; for example, you can configure alternate buttons for different autofire rates.

You can enable and configure the new autofire system with the following steps:

- Start MAME with no system selected.
- Choose **Configure Options** at the bottom (use **Tab** to move focus, or double-click the menu item).
- Choose **Plugins** near the bottom of the Settings menu.
- Turn **Autofire plugin** on (use **Left**/**Right** or click the arrows to change options).
- Exit MAME completely and start MAME again so the setting takes effect.

The setting will be automatically saved for future use.

See *Autofire Plugin* for more information about using the autofire plugin, or *Plugins* for more information about using plugins with MAME in general.

4.8.23 Does MAME support G-Sync or FreeSync? How do I configure MAME to use them?

MAME supports both G-Sync and FreeSync right out of the box for Windows and Linux, however macOS does not support G-Sync or FreeSync.

- Make sure your monitor is capable of at least 120Hz G-Sync/FreeSync. If your monitor is only capable of 60Hz in G-Sync/FreeSync modes, you will hit problems with drivers such as *Pac-Man* that run at 60.60606Hz and may hit problems with others that are very close to but not quite 60Hz.
- If playing MAME windowed or using the BGFX video system, you'll need to make sure that you have G-Sync/FreeSync turned on for windowed applications as well as full screen in your video driver.
- Be sure to leave triple buffering turned off.
- Turning VSync on is suggested in general with G-Sync and FreeSync.
- Low Latency Mode will not affect MAME performance with G-Sync/FreeSync.

The effects of G-Sync and FreeSync will be most noticeable in drivers that run at refresh rates that are very different from normal PC refresh rates. For instance, the first three *Mortal Kombat* titles run at 54.706841Hz.

CHAPTER

FIVE

MAME COMMAND-LINE USAGE AND OS-SPECIFIC CONFIGURATION

5.1 Universal Command-line Options

This section contains configuration options that are applicable to *all* MAME configurations (including both SDL and Windows native).

- *Commands and Verbs*
- *Patterns*
- *File Names and Directory Paths*
- *Core Verbs*
- *Configuration Verbs*
- *Frontend Verbs*
- *OSD-related Options*
- *OSD Command-Line Verbs*
- *OSD Output Options*
- *Configuration Options*
- *Core Search Path Options*
- *Core Output Directory Options*
- *Core State/Playback Options*
- *Core Performance Options*
- *Core Rotation Options*
- *Core Video Options*
- *Core Full Screen Options*
- *Core Per-Window Options*
- *Core Artwork Options*
- *Core Screen Options*
- *Core Vector Options*

- *Core Video OpenGL Feature Options*
- *Core Video OpenGL GLSL Options*
- *Core Sound Options*
- *Core Input Options*
- *Core Input Automatic Enable Options*
- *Debugging Options*
- *Core Communication Options*
- *Core Misc Options*
- *Scripting Options*
- *HTTP Server Options*

5.1.1 Commands and Verbs

Commands include **mame** itself as well as various tools included with the MAME distribution such as **romcmp** and **srcclean**.

Verbs are actions to take upon something with the command (e.g. **mame -validate pacman** has *mame* as a command and *-validate* as a verb)

5.1.2 Patterns

Many verbs support the use of *patterns*, which are either a system or device short name (e.g. **a2600**, **zorba_kbd**) or a glob pattern that matches either (e.g. **zorba_***).

Depending on the command you're using the pattern with, pattern matching may match systems or systems and devices. It is advised to put quotes around your patterns to avoid having your shell try to expand them against filenames (e.g. **mame -validate "pac*"**).

5.1.3 File Names and Directory Paths

A number of options for specifying directories support multiple paths (for example to search for ROMs in multiple locations). MAME expects multiple paths to be separated with semicolons (`;`).

MAME expands environment variable expressions in paths. The syntax used depends on your operating system. On Windows, `%` (percent) syntax is used. For example `%APPDATA%\mame\cfg` will expand the application data path for the current user's roaming profile. On UNIX-like system (including macOS and Linux), Bourne shell syntax is used, and a leading ~ expands to the current user's home directory. For example, `~/.mame/${HOSTNAME}/cfg` expands to a host-specific path inside the `.mame` directory in the current user's home directory. Note that only simple variable substitutions are supported; more complex expressions supported by Bash, ksh or zsh are not recognized by MAME.

Relative paths are resolved relative to the current working directory. If you start MAME by double-clicking it in Windows Explorer, the working directory is set to the folder containing the MAME executable. If you start MAME by double-clicking it in the macOS Finder, it will open a Terminal window with the working directory is set to your home directory (usually `/Users/<username>`) and start MAME.

If you want behaviour similar to what Windows Explorer provides on macOS, create a script file containing these lines in the directory containing the MAME executable (for example you could call it `mame-here`):

```
#!/bin/sh
cd "`dirname "$0"`"
exec ./mame
```

You should be able to use any text editor. If you have a choice of file format or line ending style, choose UNIX. This assumes you're using a 64-bit release build of MAME, but if you aren't you just need to change `mame` to the name of your MAME executable (e.g. mamed, mamep, mamedp). Once you've created the file, you need to mark it as executable. You can do this by opening a Terminal window, typing **chmod a+x** followed by a space, dragging the file you created onto the window (this causes Terminal to insert the full escaped path to the file), and then ensuring the Terminal window is active and hitting **Return** (or **Enter**) on your keyboard. You can close the Terminal window after doing this. Now if you double-click the script in the Finder, it will open a Terminal window, set the working directory to the location of the script (i.e. the folder containing MAME), and then start MAME.

5.1.4 Core Verbs

Tip

Examples that have the output abbreviated for space reasons will show "**...**" in the output where needed. For instance:
.. code-block:: bash

    A B C ... Z

-help / -h / -?

Displays current MAME version and copyright notice.

Example:

```
mame -help
```

-validate / -valid [<*pattern*>]

Performs internal validation on one or more drivers and devices in the system. Run this before submitting changes to ensure that you haven't violated any of the core system rules.

If a pattern is specified, it will validate systems matching the pattern, otherwise it will validate all systems and devices. Note that if a pattern is specified, it will be matched against systems only (not other devices), and no device type validation will be performed.

Example:

```
mame -validate
Driver ace100 (file apple2.cpp): 1 errors, 0 warnings
Errors:
Software List device 'flop525_orig': apple2_flop_orig.xml: Errors parsing␣
↪software list:
apple2_flop_orig.xml(126.2): Unknown tag: year
apple2_flop_orig.xml(126.8): Unexpected content
apple2_flop_orig.xml(127.2): Unknown tag: publisher
apple2_flop_orig.xml(127.13): Unexpected content
apple2_flop_orig.xml(128.2): Unknown tag: info
apple2_flop_orig.xml(129.2): Unknown tag: sharedfeat
apple2_flop_orig.xml(132.2): Unknown tag: part
apple2_flop_orig.xml(133.3): Tag dataarea found outside of software context
```

(continues on next page)

(continued from previous page)

```
apple2_flop_orig.xml(134.4): Tag rom found outside of part context
apple2_flop_orig.xml(137.3): mismatched tag
```

5.1.5 Configuration Verbs

-createconfig / -cc

Creates the default `mame.ini` file. All the configuration options (not verbs) described below can be permanently changed by editing this configuration file.

Example:

```
mame -createconfig
```

-showconfig / -sc

Displays the current configuration settings. If you route this to a file, you can use it as an INI file.

Example:

```
mame -showconfig > mame.ini
```

This example is equivalent to **-createconfig**.

-showusage / -su

Displays a summary of all the command line options. For options that are not mentioned here, the short summary given by "mame -showusage" is usually a sufficient description.

5.1.6 Frontend Verbs

Note: By default, all the '**-list**' verbs below write info to the standard output (usually the terminal/command window where you typed the command). If you wish to write the info to a text file instead, add this to the end of your command:

> *filename*

where *filename* is the name of the file to save the output in (e.g. `list.txt`). Note that if this file already exists, it will be completely overwritten.

Example:

```
mame -listcrc puckman > list.txt
```

This creates (or overwrites the existing file if already there) `list.txt` and fills the file with the results of **-listcrc puckman**. In other words, the list of each ROM used in Puckman and the CRC for that ROM are written into that file.

-listxml / -lx [<*pattern*>...]

List comprehensive details for all of the supported systems and devices in XML format. The output is quite long, so it is usually better to redirect this into a file. By default all systems are listed; however, you can limit this list by specifying one or more *patterns* after the **-listxml** verb.

This XML output is typically imported into other tools (like graphical front-ends and ROM managers), or processed with scripts query detailed information.

Example:

```
mame galaxian -listxml
<?xml version="1.0"?>
<!DOCTYPE mame [
<!ELEMENT mame (machine+)>
        <!ATTLIST mame build CDATA #IMPLIED>
        <!ATTLIST mame debug (yes|no) "no">
        <!ATTLIST mame mameconfig CDATA #REQUIRED>
        <!ELEMENT machine (description, year?, manufacturer?, biosset*,␣
↪rom*, disk*, device_ref*, sample*, chip*, display*, sound?, input?,␣
↪dipswitch*, configuration*, port*, adjuster*, driver?, feature*, device*,
↪ slot*, softwarelist*, ramoption*)>
                <!ATTLIST machine name CDATA #REQUIRED>
                <!ATTLIST machine sourcefile CDATA #IMPLIED>
...
<mame build="0.216 (mame0216-154-gabddfb0404c-dirty)" debug="no"␣
↪mameconfig="10">
    <machine name="galaxian" sourcefile="galaxian.cpp">
                <description>Galaxian (Namco set 1)</description>
                <year>1979</year>
                <manufacturer>Namco</manufacturer>
        ...
    <machine name="z80" sourcefile="src/devices/cpu/z80/z80.cpp" isdevice=
↪"yes" runnable="no">
                <description>Zilog Z80</description>
    </machine>
</mame>
```

Tip

Output from this command is typically more useful if redirected to an output file. For instance, doing **mame -listxml galaxian > galax.xml** will make `galax.xml` or overwrite any existing data in the file with the results of **-listxml**; this will allow you to view it in a text editor or parse it with external tools.

-listfull / **-ll** [<*pattern*>...]

Example:

```
mame -listfull galaxian*
Name:            Description:
galaxian         "Galaxian (Namco set 1)"
galaxiana        "Galaxian (Namco set 2)"
galaxianbl       "Galaxian (bootleg, set 2)"
galaxianbl2      "Galaxian (bootleg, set 4)"
galaxiani        "Galaxian (Irem)"
galaxianm        "Galaxian (Midway set 1)"
galaxianmo       "Galaxian (Midway set 2)"
galaxiant        "Galaxian (Taito)"
galaxian_sound   "Galaxian Custom Sound"
```

Displays a list of system driver names and descriptions. By default all systems and devices are listed; however, you can limit this list by specifying one or more *patterns* after the **-listfull** verb.

-listsource / **-ls** [<*pattern*>...]

Displays a list of system drivers/devices and the names of the source files where they are defined. Useful for finding which driver a system runs on in order to fix bugs. By default all systems and devices are listed; however, you can limit this list by specifying one or more *pattern* after the **-listsource** verb.

Example:

```
mame galaga -listsource
galaga           galaga.cpp
```

-listclones / -lc [<*pattern*>]

Displays a list of clones. By default all clones are listed; however, you can limit this list by specifying a *pattern* after the **-listsource** verb. If a pattern is specified, MAME will list clones of systems that match the pattern, as well as clones that match the pattern themselves.

Example 1:

```
mame pacman -listclones
Name:            Clone of:
pacman           puckman
```

Example 2:

```
mame puckman -listclones
Name:            Clone of:
abscam           puckman
bucaner          puckman
crockman         puckman
crockmnf         puckman
...
puckmod          puckman
titanpac         puckman
```

-listbrothers / -lb [<*pattern*>]

Displays a list of *brothers*, i.e. other systems that are defined in the same source file as a system that matches the specified *pattern*.

Example:

```
mame galaxian -listbrothers
Source file:         Name:            Parent:
galaxian.cpp         amidar
galaxian.cpp         amidar1          amidar
galaxian.cpp         amidarb          amidar
...
galaxian.cpp         zigzagb
galaxian.cpp         zigzagb2         zigzagb
```

-listcrc [<*pattern*>...]

Displays a full list of CRCs and names of all ROM images referenced by systems and devices matching the specified pattern(s). If no patterns are specified, ROMs referenced by all supported systems and devices will be included.

Example:

```
mame playch10 -listcrc
d52fa07a pch1-c__8t_e-2.8t                    playch10            ␣
↪PlayChoice-10 BIOS
503ee8b1 pck1-c.8t                            playch10            ␣
↪PlayChoice-10 BIOS
123ffa37 pch1-c_8te.8t                        playch10            ␣
↪PlayChoice-10 BIOS
0be8ceb4 pck1-c_fix.8t                        playch10            ␣
↪PlayChoice-10 BIOS
9acffb30 pch1-c__8k.8k                        playch10            ␣
↪PlayChoice-10 BIOS
c1232eee pch1-c__8m_e-1.8m                    playch10            ␣
↪PlayChoice-10 BIOS
30c15e23 pch1-c__8p_e-1.8p                    playch10            ␣
↪PlayChoice-10 BIOS
9acffb30 pch1-c__8k.8k                        playch10            ␣
↪PlayChoice-10 BIOS
c1232eee pch1-c__8m_e-1.8m                    playch10            ␣
↪PlayChoice-10 BIOS
30c15e23 pch1-c__8p_e-1.8p                    playch10            ␣
↪PlayChoice-10 BIOS
9acffb30 pch1-c__8k.8k                        playch10            ␣
↪PlayChoice-10 BIOS
83ebc7a3 pch1-c_8m.8m                         playch10            ␣
↪PlayChoice-10 BIOS
90e1b80c pch1-c_8p-8p                         playch10            ␣
↪PlayChoice-10 BIOS
9acffb30 pch1-c__8k.8k                        playch10            ␣
↪PlayChoice-10 BIOS
c1232eee pch1-c__8m_e-1.8m                    playch10            ␣
↪PlayChoice-10 BIOS
30c15e23 pch1-c__8p_e-1.8p                    playch10            ␣
↪PlayChoice-10 BIOS
e5414ca3 pch1-c-6f.82s129an.6f                playch10            ␣
↪PlayChoice-10 BIOS
a2625c6e pch1-c-6e.82s129an.6e                playch10            ␣
↪PlayChoice-10 BIOS
1213ebd4 pch1-c-6d.82s129an.6d                playch10            ␣
↪PlayChoice-10 BIOS
48de65dc rp2c0x.pal                           playch10            ␣
↪PlayChoice-10 BIOS
```

-listroms / -lr [<*pattern*>...]

Displays a list of ROM images referenced by supported systems/devices that match the specified pattern(s). If no patterns are specified, the results will include *all* supported systems and devices.

Example:

```
mame neogeo -listroms
ROMs required for driver "neogeo".
Name                                  Size Checksum
sp-s2.sp1                           131072 CRC(9036d879)␣
↪SHA1(4f5ed7105b7128794654ce82b51723e16e389543)
```

(continues on next page)

(continued from previous page)

```
sp-s.sp1                                131072 CRC(c7f2fa45)␣
↪SHA1(09576ff20b4d6b365e78e6a5698ea450262697cd)
sp-45.sp1                               524288 CRC(03cc9f6a)␣
↪SHA1(cdf1f49e3ff2bac528c21ed28449cf35b7957dc1)
...
sm1.sm1                                 131072 CRC(94416d67)␣
↪SHA1(42f9d7ddd6c0931fd64226a60dc73602b2819dcf)
000-lo.lo                               131072 CRC(5a86cff2)␣
↪SHA1(5992277debadeb64d1c1c64b0a92d9293eaf7e4a)
sfix.sfix                               131072 CRC(c2ea0cfd)␣
↪SHA1(fd4a618cdcdbf849374f0a50dd8efe9dbab706c3)
```

-listsamples [<*pattern*>]

Displays a list of samples referenced by the specified pattern of system or device names. If no pattern is specified, the results will be *all* systems and devices.

Example:

```
mame armorap -listsamples
Samples required for driver "armorap".
loexp
jeepfire
hiexp
tankfire
tankeng
beep
chopper
```

-verifyroms [<*pattern*>]

Checks for invalid or missing ROM images. By default all drivers that have valid ZIP files or directories in the rompath are verified; however, you can limit this list by specifying a *pattern* after the **-verifyroms** command.

Example:

```
mame gradius -verifyroms
romset gradius [nemesis] is good
1 romsets found, 1 were OK.
```

-verifysamples [<*pattern*>]

Checks for invalid or missing samples. By default all drivers that have valid ZIP files or directories in the samplepath are verified; however, you can limit this list by specifying a *pattern* after the **-verifyroms** command.

Example:

```
mame armorap -verifysamples
sampleset armorap [armora] is good
1 samplesets found, 1 were OK.
```

-romident [*path/to/romstocheck.zip*]

Attempts to identify ROM files, if they are known to MAME, in the specified .zip file or directory. This command can be used to try and identify ROM sets taken from unknown boards. On exit, the errorlevel is

returned as one of the following:

- 0: means all files were identified
- 7: means all files were identified except for 1 or more "non-ROM" files
- 8: means some files were identified
- 9: means no files were identified

Example:

```
mame unknown.rom -romident
Identifying unknown.rom....
unknown.rom         = 456-a07.17l           gradius    Gradius (Japan, ROM␣
↪version)
```

-listdevices / **-ld** [<*pattern*>]

Displays a list of all devices known to be hooked up to a system. The ":" is considered the system itself with the devices list being attached to give the user a better understanding of what the emulation is using.

If slots are populated with devices, any additional slots those devices provide will be visible with **-listdevices** as well. For instance, installing a floppy controller into a PC will expose the disk drive slots.

Example:

```
mame apple2e -listdevices
Driver apple2e (Apple //e):
   <root>                          Apple //e
     a2bus                         Apple II Bus
     a2common                      Apple II Common Components @ 14.31 MHz
     a2video                       Apple II video @ 14.31 MHz
     aux                           Apple IIe AUX Slot
       ext80                       Apple IIe Extended 80-Column Card
     auxbus                        Apple IIe AUX Bus
     ay3600                        AY-5-3600 Keyboard Encoder
     ...
     speaker                       Filtered 1-bit DAC
     tape                          Cassette
```

-listslots / **-lslot** [<*pattern*>]

Show available slots and options for each slot (if available). Primarily used for MAME to allow control over internal plug-in cards, much like PCs needing video, sound and other expansion cards.

If slots are populated with devices, any additional slots those devices provide will be visible with **-listslots** as well. For instance, installing a floppy controller into a PC will expose the disk drive slots.

The slot name (e.g. **ctrl1**) can be used from the command line (**-ctrl1** in this case)

Example:

```
mame apple2e -listslots
SYSTEM           SLOT NAME        SLOT OPTIONS     SLOT DEVICE NAME
---------------- ---------------- ---------------- ------------------------
↪----
apple2e          sl1              4play            4play Joystick Card␣
↪(rev. B)
                                  ...
```

(continues on next page)

(continued from previous page)

```
                                    aevm80          Applied Engineering␣
↪Viewmaster 80
                                    alfam2          ALF MC1 / Apple Music II
                                    ...
                                    zipdrive        Zip Technologies␣
↪ZipDrive

                    ...
                    aux             ext80           Apple IIe Extended 80-
↪Column Card
                                    rw3             Applied Engineering␣
↪RamWorks III
                                    std80           Apple IIe Standard 80-
↪Column Card

                    gameio          compeyes        Digital Vision␣
↪ComputerEyes
                                    joy             Apple II analog␣
↪joysticks
                                    paddles         Apple II paddles
```

-listbios [<*pattern*>]

Show available BIOS options for a system (if available). BIOS options may be available for the system or any devices selected as slot options.

If no pattern is specified, the results will include *all* supported systems.

Example:

```
mamed -listbios apple2 -sl2 grapplus -sl4 videoterm
BIOS options for system Apple ][ (apple2):
    default          Original Monitor
    autostart        Autostart Monitor

  BIOS options for device Orange Micro Grappler+ Printer Interface (-sl2␣
↪grapplus):
      v30              ROM 3.0
      v32              ROM 3.2

  BIOS options for device Videx Videoterm 80 Column Display (-sl4␣
↪videoterm):
      v24_60hz         Firmware v2.4 (60 Hz)
      v24_50hz         Firmware v2.4 (50 Hz)
```

-listmedia / -lm [<*pattern*>]

List available media that the chosen system allows to be used. This includes media types (cartridge, cassette, diskette and more) as well as common file extensions which are supported.

Example:

```
mame coco3 -listmedia
SYSTEM           MEDIA NAME       (brief)    IMAGE FILE EXTENSIONS␣
↪SUPPORTED
```

(continues on next page)

(continued from previous page)

```
---------------- -------------------------- ------------------------------
↪-
coco3            cassette         (cass)     .wav  .cas
                 printout         (prin)     .prn
                 cartridge        (cart)     .ccc  .rom
                 floppydisk1      (flop1)    .dmk  .jvc  .dsk  .vdk  .sdf ␣
↪.os9  .d77  .d88  .1dd  .dfi  .hfe  .imd  .ipf  .mfi  .mfm  .td0  .cqm  .
↪cqi
                 floppydisk2      (flop2)    .dmk  .jvc  .dsk  .vdk  .sdf ␣
↪.os9  .d77  .d88  .1dd  .dfi  .hfe  .imd  .ipf  .mfi  .mfm  .td0  .cqm  .
↪cqi
                 harddisk1        (hard1)    .vhd
                 harddisk2        (hard2)    .vhd
```

-listsoftware / -lsoft [<*pattern*>]

Displays the contents of all software lists that can be used by the system or systems represented by *pattern*.

Example:

```
mame coco3 -listsoftware
<?xml version="1.0"?>
<!DOCTYPE softwarelists [
<!ELEMENT softwarelists (softwarelist*)>
        <!ELEMENT softwarelist (software+)>
                <!ATTLIST softwarelist name CDATA #REQUIRED>
                <!ATTLIST softwarelist description CDATA #IMPLIED>
                <!ELEMENT software (description, year, publisher, info*,␣
↪sharedfeat*, part*)>
        ...
<softwarelists>
        <softwarelist name="coco_cart" description="Tandy Radio Shack␣
↪Color Computer cartridges">
                <software name="7cardstd">
                        <description>7 Card Stud</description>
                        <year>1983</year>
                        <publisher>Tandy</publisher>
                        <info name="developer" value="Intelligent Software
↪"/>
                        <info name="serial" value="26-3074"/>
                        <part name="cart" interface="coco_cart">
                                <dataarea name="rom" size="8192">
                                        <rom name="7 card stud (1983) (26-
↪3074) (intelligent software).rom" size="8192" crc="f38d8c97" sha1=
↪"5cfcb699ce09840dbb52714c8d91b3d86d3a86c3"/>
                                </dataarea>
                        </part>
                </software>
        ...
```

-verifysoftware / -vsoft [<*pattern*>]

Checks for invalid or missing ROM images in your software lists. By default all drivers that have valid ZIP files or directories in the rompath are verified; however, you can limit this list by specifying a specific

driver name or *pattern* after the **-verifysoftware** command.

Example:

```
mame coco3 -verifysoftware
romset coco_cart:7cardstd is good
coco_cart:amazing: a mazing world of malcom mortar (1987)(26-3160)(zct␣
↪systems).rom (16384 bytes) - NEEDS REDUMP
romset coco_cart:amazing is best available
coco_cart:amazing1: a mazing world of malcom mortar (1987)(26-3160)(zct␣
↪systems)[a].rom (16384 bytes) - NEEDS REDUMP
romset coco_cart:amazing1 is best available
romset coco_cart:androne is good
...
```

-getsoftlist / -glist [<*pattern*>]

Displays the contents of a specific softlist with the filename represented by *pattern*.

Example:

```
mame -getsoftlist apple2_flop_orig
<?xml version="1.0"?>
<!DOCTYPE softwarelists [
<!ELEMENT softwarelists (softwarelist*)>
        <!ELEMENT softwarelist (software+)>
                <!ATTLIST softwarelist name CDATA #REQUIRED>
                <!ATTLIST softwarelist description CDATA #IMPLIED>
                <!ELEMENT software (description, year, publisher, info*,␣
↪sharedfeat*, part*)>
                        <!ATTLIST software name CDATA #REQUIRED>
                        <!ATTLIST software cloneof CDATA #IMPLIED>
                        <!ATTLIST software supported (yes|partial|no) "yes
↪">
                        <!ELEMENT description (#PCDATA)>
                        <!ELEMENT year (#PCDATA)>
                        <!ELEMENT publisher (#PCDATA)>
                        <!ELEMENT info EMPTY>
                                <!ATTLIST info name CDATA #REQUIRED>
                                <!ATTLIST info value CDATA #IMPLIED>
                        <!ELEMENT sharedfeat EMPTY>
                                <!ATTLIST sharedfeat name CDATA #REQUIRED>
                                <!ATTLIST sharedfeat value CDATA #IMPLIED>
            ...
```

-verifysoftlist / -vlist [softwarelistname]

Checks a specified software list for missing ROM images if files exist for issued softwarelistname. By default, all drivers that have valid ZIP files or directories in the rompath are verified; however, you can limit this list by specifying a specific softwarelistname (without .XML) after the -verifysoftlist command.

Example:

```
mame -verifysoftlist apple2_flop_orig
romset apple2_flop_orig:agentusa is good
romset apple2_flop_orig:airheart is good
```

(continues on next page)

(continued from previous page)

```
romset apple2_flop_orig:aplpanic is good
romset apple2_flop_orig:alambush is good
romset apple2_flop_orig:ankh is good
romset apple2_flop_orig:aplcdspd is good
romset apple2_flop_orig:agalxian is good
romset apple2_flop_orig:aquatron is good
romset apple2_flop_orig:archon is good
romset apple2_flop_orig:archon2 is good
romset apple2_flop_orig:ardyardv is good
romset apple2_flop_orig:autobahn is good
...
```

5.1.7 OSD-related Options

-uimodekey [*keystring*]

Key used to enable/disable MAME keyboard controls when the emulated system has keyboard inputs. The default setting is **Forward Delete** on macOS or **SCRLOCK** on other operating systems (including Windows and Linux). Use **FN-Delete** on Macintosh computers with notebook/compact keyboards.

Example:

```
mame ibm5150 -uimodekey DEL
```

-controller_map / **-ctrlmap** <*filename*>

Path to a text file containing game controller button and axis mappings in the format used by SDL2 and Steam, or `none` to use only built-in mappings. Must use an ASCII-compatible text encoding with native line endings (e.g. CRLF on Windows). Currently only supported when using the `sdlgame` joystick provider. The default setting is `none`.

A community-sourced list of game controller mappings can be found on GitHub. Besides using a text editor, several tools are available for creating game controller mappings, including SDL2 Gamepad Mapper and SDL2 ControllerMap which is supplied with SDL. You can also configure your controller in Steam's Big Picture mode, then copy the mappings from `SDL_GamepadBind` entries in the **config.vdf** file found in the **config** folder inside your Steam installation folder.

Example:

```
mame -controller_map gamecontrollerdb.txt sf2ce
```

-[no]background_input

Sets whether input is accepted or ignored when MAME does not have UI focus. This setting is ignored when the debugger is enabled. The default is OFF (**-nobackground_input**).

Currently supported for RawInput mouse/keyboard input, DirectInput mouse/keyboard/joystick input and XInput joystick input on Windows, and SDL game controller/joystick input.

Example:

```
mame -background_input ssf2tb
```

-uifontprovider <*module*>

Chooses provider for UI font rendering. The default setting is `auto`.

Table 1: Supported UI font providers per-platform

Microsoft Windows	win	dwrite	auto		sdl[1]	none
macOS			auto	osx	sdl[2]	none
Linux			auto		sdl	none

Example:

```
mame ajax -uifontprovider dwrite
```

-keyboardprovider <*module*>

Chooses how MAME will get keyboard input. The default is `auto`.

Table 2: Supported keyboard input providers per-platform

Microsoft Windows	auto[3]	rawinput	dinput	win32		sdl[4]	none
SDL (macOS and Linux)	auto[5]					sdl	none
macOS native	auto[6]				mac		none

Tip

Note that user-mode keyboard emulation tools such as joy2key will almost certainly require the use of **-keyboardprovider win32** on Windows machines.

Example:

```
mame c64 -keyboardprovider win32
```

-mouseprovider <*module*>

Chooses how MAME will get mouse input. The default is `auto`.

Table 3: Supported mouse input providers per-platform

Microsoft Windows	auto[7]	rawinput	dinput	win32		sdl[8]	none
SDL (macOS and Linux)	auto[9]					sdl	none
macOS native	auto[10]				mac		none

Example:

```
mame indy_4610 -mouseprovider win32
```

-lightgunprovider <*module*>

[1] SDL support on Windows requires that you compile MAME with the support in. By default SDL is not included in Windows builds of MAME.
[2] SDL support on macOS requires that you compile MAME with the support in.
[3] `auto` on Windows will try `rawinput` with fallback to `dinput`.
[4] SDL support on Windows requires that you compile MAME with the support in. By default SDL is not included in Windows builds of MAME.
[5] `auto` on SDL will default to `sdl`.
[6] `auto` on macOS native will default to `mac`.
[7] On Windows, `auto` will try `rawinput` with fallback to `dinput`.
[8] SDL support on Windows requires that you compile MAME with the support in. By default SDL is not included in Windows builds of MAME.
[9] `auto` on SDL will default to `sdl`.
[10] `auto` on macOS native will default to `mac`.

Chooses how MAME will get light gun input. The default is `auto`.

Table 4: Supported light gun input providers per-platform

Microsoft Windows	auto[11]	rawinput	win32	sdl[12]		none
SDL	auto[13]			sdl[14]		none
Linux	auto[13]			sdl	x11	none
macOS native	auto[15]			mac		none

Example:

```
mame lethalen -lightgunprovider x11
```

-joystickprovider <*module*>

Chooses how MAME will get joystick and other game controller input. The default is `auto`.

Table 5: Supported joystick input providers per-platform

Microsoft Windows	auto[16]	winhybrid	dinput	xinput			sdlgame[17]	sdljoy[17]	none
SDL	auto[18]						sdlgame	sdljoy	none
macOS native	auto[19]				macgame	macjoy			none

winhybrid
Uses XInput for compatible game controllers, falling back to DirectInput for other game controllers. Typically provides the best experience on Windows.

dinput
Uses DirectInput for all game controllers. May be useful if you want to use more than four XInput game controllers simultaneously. Note that LT and RT controls are combined with using XInput game controllers via DirectInput.

xinput
Supports up to four XInput game controllers.

macgame
Uses the macOS Game Controller framework for game controllers with button/axis mappings available, including Sony and Microsoft console controllers and MFi-certified controllers. Provides consistent button and axis assignment and meaningful control names.

macjoy
Uses the macOS IOKit framework for all game controllers.

sdlgame
Uses the SDL game controller API for game controllers with button/axis mappings available, falling back to the SDL joystick API for other game controllers. Provides consistent button and axis assignment and meaningful control names for popular game controllers. Use the *controller_map option* to supply mappings for additional game controllers or override built-in mappings.

[11] On Windows, auto will try `rawinput` with fallback to `win32`, or `none` if it doesn't find any.
[12] SDL support on Windows requires that you compile MAME with the support in. By default SDL is not included in Windows builds of MAME.
[13] On SDL, `auto` will default to `sdl`.
[14] SDL support on macOS requires that you compile MAME with the support in.
[15] On macOS native, `auto` will default to `mac`, which reads the system pointer.
[16] On Windows native, auto will default to `winhybrid`.
[17] SDL support on Windows requires that you compile MAME with the support in. By default SDL is not included in Windows builds of MAME.
[18] On SDL, auto will default to `sdlgame`.
[19] On macOS without SDL support, auto will default to `macgame`.

sdljoy
Uses the SDL joystick API for all game controllers.

none
Ignores all game controllers.

Example:

```
mame mk2 -joystickprovider winhybrid
```

-midiprovider <*module*>

Chooses how MAME will communicate with MIDI devices and applications (e.g. music keyboards and synthesisers). Supported options are `pm` to use the PortMidi library, or `none` to disable MIDI input and output (MIDI files can still be played). The default is `auto`, which will use PortMidi if available.

Example:

```
mame -midiprovider none dx100 -midiin canyon.mid
```

-networkprovider <*module*>

Chooses how MAME will provide communication for emulated packet-oriented network interfaces (e.g. Ethernet cards). Supported options are `taptun` to use the TUN/TAP, TAP-Windows or similar, `pcap` to use a pcap library, or `none` to disable communication for emulated network interfaces. Available options depend on your operating system. By default, `taptun` and `none` are available on Windows and Linux, and `pcap` and `none` are available on macOS.

The default is `auto` which will use `taptun` if available, falling back to `pcap`.

Example:

```
mame -networkprovider pcap apple2ee -sl3 uthernet
```

5.1.8 OSD Command-Line Verbs

-listmidi

List available MIDI I/O devices for use with emulation.

Example:

```
mame -listmidi
MIDI input ports:

MIDI output ports:
Microsoft MIDI Mapper (default)
Microsoft GS Wavetable Synth
```

-listnetwork

List available network adapters for use with emulation.

Example 1:

```
mame -listnetwork
No network adapters were found
```

Example 2:

```
mame -listnetwork
Available network adapters:
    Local Area Connection
```

Tip

On Windows, you'll need the TAP driver from OpenVPN for MAME to see any network adapters.

5.1.9 OSD Output Options

-output

Chooses how MAME will handle processing of output notifiers. These are used to connect external outputs such as the LED lights for the Player 1/2 start buttons on certain arcade machines.

You can choose from: `auto`, `none`, `console` or `network`

Note that network port is fixed at 8000.

Example:

```
mame asteroid -output console
led0 = 1
led0 = 0
...
led0 = 1
led0 = 0
```

5.1.10 Configuration Options

-[no]readconfig / -[no]rc

Enables or disables the reading of the config files. When enabled (which is the default), MAME reads the following config files in order:

- `mame.ini`
- `debug.ini` (if the debugger is enabled)
- `vertical.ini` (for systems with vertical monitor orientation)
- `horizont.ini` (for systems with horizontal monitor orientation)
- `raster.ini` (for raster display systems only)
- `vector.ini` (for vector display systems only)
- `lcd.ini` (for matrix display systems only)
- `source/<driver>.ini` (based on the source filename of the driver)
- `<parent>.ini` (for clones only, may be called recursively)
- `<systemname>.ini`

(See *Order of Configurattion File Loading* for further details)

The settings in the later INIs override those in the earlier INIs. So, for example, if you wanted to disable overlay effects in the vector systems, you can create a `vector.ini` with line `effect none` in it, and it will override whatever `effect` value you have in your `mame.ini`.

The default is ON (**-readconfig**).

Example:

```
mame apple2ee -noreadconfig -sl6 diskii -sl7 cffa2 -hard1 TotalReplay.2mg
```

5.1.11 Core Search Path Options

-homepath <*path*>

Specifies a path for Lua plugins to store data.

The default is `.` (that is, in the current working directory).

Example:

```
mame -homepath C:\mame\lua
```

-rompath / **-rp** <*path*>

Specifies one or more paths within which to find ROM or disk images. Multiple paths can be specified by separating them with semicolons.

The default is `roms` (that is, a directory `roms` in the current working directory).

Example:

```
mame -rompath C:\mame\roms;C:\roms\another
```

-hashpath / **-hash_directory** / **-hash** <*path*>

Specifies one or more paths within which to find software definition files. Multiple paths can be specified by separating them with semicolons.

The default is `hash` (that is, a directory `hash` in the current working directory).

Example:

```
mame -hashpath C:\mame\hash;C:\roms\softlists
```

-samplepath / **-sp** <*path*>

Specifies one or more paths within which to find audio sample files. Multiple paths can be specified by separating them with semicolons.

The default is `samples` (that is, a directory `samples` in the current working directory).

Example:

```
mame -samplepath C:\mame\samples;C:\roms\samples
```

-artpath <*path*>

Specifies one or more paths within which to find external layout and artwork files. Multiple paths can be specified by separating them with semicolons.

The default is `artwork` (that is, a directory `artwork` in the current working directory).

Example:

```
mame -artpath C:\mame\artwork;C:\emu\shared-artwork
```

-ctrlrpath <*path*>

Specifies one or more paths within which to find controller configuration files. Multiple paths can be specified by separating them with semicolons. Used in conjunction with the `-ctrlr` option.

The default is `ctrlr` (that is, a directory `ctrlr` in the current working directory).

Example:

```
mame -ctrlrpath C:\mame\ctrlr;C:\emu\controllers
```

-inipath <*path*>

Specifies one or more paths within which to find INI files. Multiple paths can be specified by separating them with semicolons.

On Windows, the default is `.;ini;ini/presets` (that is, search in the current directory first, then in the directory `ini` in the current working directory, and finally the directory `presets` inside that directory).

On macOS, the default is `$HOME/Library/Application Support/mame;$HOME/.mame;.;ini` (that is, search the `mame` folder inside the current user's Application Support folder, followed by the `.mame` folder in the current user's home directory, then the current working directory, and finally the directory `ini` in the current working directory).

On other platforms (including Linux), the default is `$HOME/.mame;.;ini` (that is search the `.mame` directory in the current user's home directory, followed by the current working directory, and finally the directory `ini` in the current working directory).

Example:

```
mame -inipath C:\Users\thisuser\documents\mameini
```

-fontpath <*path*>

Specifies one or more paths within which to find BDF (Adobe Glyph Bitmap Distribution Format) font files. Multiple paths can be specified by separating them with semicolons.

The default is `.` (that is, search in the current working directory).

Example:

```
mame -fontpath C:\mame\;C:\emu\artwork\mamefonts
```

-cheatpath <*path*>

Specifies one or more paths within which to find XML cheat files. Multiple paths can be specified by separating them with semicolons.

The default is `cheat` (that is, a folder called `cheat` located in the current working directory).

Example:

```
mame -cheatpath C:\mame\cheat;C:\emu\cheats
```

-crosshairpath <*path*>

Specifies one or more paths within which to find crosshair image files. Multiple paths can be specified by separating them with semicolons.

The default is `crsshair` (that is, a directory `crsshair` in the current working directory).

Example:

```
mame -crosshairpath C:\mame\crsshair;C:\emu\artwork\crosshairs
```

-pluginspath <*path*>

Specifies one or more paths within which to find Lua plugins for MAME.

The default is `plugins` (that is, a directory `plugins` in the current working directory).

Example:

```
mame -pluginspath C:\mame\plugins;C:\emu\lua
```

-languagepath <*path*>

Specifies one or more paths within which to find language files for localized UI text.

The default is `language` (that is, a directory `language` in the current working directory).

Example:

```
mame -languagepath C:\mame\language;C:\emu\mame-languages
```

-swpath <*path*>

Specifies the default path from which to load loose software image files.

The default is `software` (that is, a directory `software` in the current working directory).

Example:

```
mame -swpath C:\mame\software;C:\emu\mydisks
```

5.1.12 Core Output Directory Options

-cfg_directory <*path*>

Specifies the directory where configuration files are stored. Configuration files are read when starting MAME or when starting an emulated machine, and written on exit. Configuration files preserve settings including input assignment, DIP switch settings, bookkeeping statistics, and debugger window arrangement.

The default is `cfg` (that is, a directory `cfg` in the current working directory). If this directory does not exist, it will be created automatically.

Example:

```
mame -cfg_directory C:\mame\cfg
```

-nvram_directory <*path*>

Specifies the directory where NVRAM files are stored. NVRAM files store the contents of EEPROM, non-volatile RAM (NVRAM), and other programmable devices for systems that used this type of hardware. This data is read when starting an emulated machine and written on exit.

The default is `nvram` (that is, a directory `nvram` in the current working directory)). If this directory does not exist, it will be created automatically.

Example:

```
mame -nvram_directory C:\mame\nvram
```

-input_directory <*path*>

Specifies the directory where input recording files are stored. Input recordings are created using the **-record** option and played back using the **-playback** option.

The default is `inp` (that is, a directory `inp` in the current working directory). If this directory does not exist, it will be created automatically.

Example:

```
mame -input_directory C:\mame\inp
```

-state_directory <*path*>

Specifies the directory where save state files are stored. Save state files are read and written either upon user request, or when using the **-autosave** option.

The default is `sta` (that is, a directory `sta` in the current working directory). If this directory does not exist, it will be created automatically.

Example:

```
mame -state_directory C:\mame\sta
```

-snapshot_directory <*path*>

Specifies the directory where screen snapshots and video recordings are stored when requested by the user.

The default is `snap` (that is, a directory `snap` in the current working directory). If this directory does not exist, it will be created automatically.

Example:

```
mame -snapshot_directory C:\mame\snap
```

-diff_directory <*path*>

Specifies the directory where hard drive difference files are stored. Hard drive difference files store data that is written back to an emulated hard disk, in order to preserve the original image file. The difference files are created when starting an emulated system with a compressed hard disk image.

The default is `diff` (that is, a directory `diff` in the current working directory). If this directory does not exist, it will be created automatically.

Example:

```
mame -diff_directory C:\mame\diff
```

-comment_directory <*path*>

Specifies a directory where debugger comment files are stored. Debugger comment files are written by the debugger when comments are added to the disassembly for a system.

The default is `comments` (that is, a directory `comments` in the current working directory). If this directory does not exist, it will be created automatically.

Example:

```
mame -comment_directory C:\mame\comments
```

5.1.13 Core State/Playback Options

-[no]rewind

When enabled and emulation is paused, automatically creates a save state in memory every time a frame is advanced. Rewind save states can then be loaded consecutively by pressing the rewind single step shortcut key (**Left Shift + Tilde** by default).

The default rewind value is OFF (**-norewind**).

If debugger is in a 'break' state, a save state is instead created every time step in, step over, or step out occurs. In that mode, rewind save states can be loaded by executing the debugger **rewind** (or **rw**) command.

Example:

```
mame -norewind
```

-rewind_capacity <*value*>

Sets the rewind capacity value, in megabytes. It is the total amount of memory rewind savestates can occupy. When capacity is hit, old savestates get erased as new ones are captured. Setting capacity lower than the current savestate size disables rewind. Values below 0 are automatically clamped to 0.

Example:

```
mame -rewind_capacity 30
```

-state <*slot*>

Immediately after starting the specified system, will cause the save state in the specified <slot> to be loaded.

Example:

```
mame -state 1
```

-[no]autosave

When enabled, automatically creates a save state file when exiting MAME and automatically attempts to reload it when later starting MAME with the same system. This only works for systems that have explicitly enabled save state support in their driver.

The default is OFF (**-noautosave**).

Example:

```
mame -autosave
```

-playback / -pb <*filename*>

Specifies a file from which to play back a series of inputs. This feature does not work reliably for all systems, but can be used to watch a previously recorded game session from start to finish.

The default is `NULL` (no playback).

Example:

```
mame pacman -playback worldrecord
```

Tip

You may experience desync in playback if the configuration, NVRAM, and memory card files don't match the original; this is why it is suggested you should only record and playback with all configuration (.cfg), NVRAM (.nv), and memory card files deleted.

-[no]exit_after_playback

When used in conjunction with the **-playback** option, MAME will exit after playing back the input file. By default, MAME continues to run the emulated system after playback completes.

The default is OFF (**-noexit_after_playback**).

Example:

```
mame pacman -playback worldrecord -exit_after_playback
```

-record / -rec <*filename*>

Specifies a file to record all input from a session. This can be used to record a session for later playback. This feature does not work reliably for all systems, but can be used to record a session from start to finish.

The default is `NULL` (no recording).

Example:

```
mame pacman -record worldrecord
```

Tip

You may experience desync in playback if the configuration, NVRAM, and memory card files don't match the original; this is why it is suggested you should only record and playback with all configuration (.cfg), NVRAM (.nv), and memory card files deleted.

-mngwrite <*filename*>

Writes each video frame to the given <filename> in MNG format, producing an animation of the session. Note that **-mngwrite** only writes video frames; it does not save any audio data. Either use **-wavwrite** to record audio and combine the audio and video tracks using video editing software, or use **-aviwrite** to record audio and video to a single file.

The default is `NULL` (no recording).

Example:

```
mame pacman -mngwrite pacman-video
```

-aviwrite <*filename*>

Stream video and sound data to the given <filename> in uncompressed AVI format, producing an animation of the session complete with sound. Note that the AVI format does not changes to resolution or frame rate, uncompressed video consumes a lot of disk space, and recording uncompressed video in realtime requires a fast disk. It may be more practical to record an emulation session using **-record** then make a video of it with **-aviwrite** in combination with **-playback** and **-exit_after_playback** options.

The default is `NULL` (no recording).

Example:

```
mame pacman -playback worldrecord -exit_after_playback -aviwrite␣
↪worldrecord
```

-wavwrite <*filename*>

Writes the final mixer output to the given <filename> in WAV format, producing an audio recording of the session.

The default is `NULL` (no recording).

Example:

```
mame pacman -wavwrite pacsounds
```

-snapname <*name*>

Describes how MAME should name files for snapshots. <name> is a string that provides a template that is used to generate a filename.

Three simple substitutions are provided: the `/` character represents the path separator on any target platform (even Windows); the string `%g` represents the driver name of the current system; and the string `%i` represents an incrementing index. If `%i` is omitted, then each snapshot taken will overwrite the previous one; otherwise, MAME will find the next empty value for `%i` and use that for a filename.

The default is `%g/%i`, which creates a separate folder for each system, and names the snapshots under it starting with 0000 and increasing from there.

In addition to the above, for drivers using different media, like carts or floppy disks, you can also use the `%d_[media]` indicator. Replace [media] with the media switch you want to use.

Example 1:

```
mame robby -snapname foo\%g%i
```

Snapshots will be saved as `snaps\foo\robby0000.png`, `snaps\foo\robby0001.png` and so on.

Example 2:

```
mame nes -cart robby -snapname %g\%d_cart
```

Snapshots will be saved as `snaps\nes\robby.png`.

Example 3:

```
mame c64 -flop1 robby -snapname %g\%d_flop1/%i
```

Snapshots will be saved as `snaps\c64\robby\0000.png`.

-snapsize <*width*>*x*<*height*>

Hard-codes the size for snapshots and movie recording. By default, MAME will create snapshots at the system's current resolution in raw pixels, and will create movies at the system's starting resolution in raw pixels. If you specify this option, then MAME will create both snapshots and movies at the size specified, and will bilinear filter the result.

The default is `auto`.

Example:

```
mame pacman -snapsize 1920x1080
```

Tip

-snapsize does not automatically rotate if the system is vertically oriented, so for vertical systems you'll want to swap the width and height options.

-snapview <*viewname*>

Specifies the view to use when rendering snapshots and videos. The <*viewname*> does not need to be the full name of a view, MAME will choose the first view with a name that has the <*viewname*> as a prefix. For example **-snapview "screen 0 pixel"** will match the "*Screen 0 Pixel Aspect (10:7)*" view.

If the <*viewname*> is `auto` or an empty string, MAME will select a view based on the number of emulated screens in the system, and the available external and internal artwork. MAME tries to select a view that shows all emulated screens by default.

If the <*viewname*> is `native`, MAME uses special internal view to save a separate snapshot for each visible emulated screen, or to record a video for the first visible screen only. The snapshot(s) or video will have the same resolution as the emulated screen(s) with no artwork elements drawn or effects applied.

The default value is `auto`.

Example:

```
mame wrecking -snapview cocktail
```

-[no]snapbilinear

Specify if the snapshot or movie should have bilinear filtering applied. Disabling this off can improve performance while recording video to a file.

The default is ON (**-snapbilinear**).

Example:

```
mame pacman -nosnapbilinear
```

-statename <*name*>

Describes how MAME should store save state files, relative to the state_directory path. <name> is a string that provides a template that is used to generate a relative path.

Two simple substitutions are provided: the `/` character represents the path separator on any target platform (even Windows); the string `%g` represents the driver name of the current system.

The default is `%g`, which creates a separate folder for each system.

In addition to the above, for drivers using different media, like carts or floppy disks, you can also use the `%d_[media]` indicator. Replace `[media]` with the media switch you want to use.

Example 1:

```
mame robby -statename foo\%g
All save states will be stored inside sta\foo\robby\
```

Example 2:

```
mame nes -cart robby -statename %g/%d_cart
All save states will be stored inside sta\nes\robby\
```

Example 3:

```
mame c64 -flop1 robby -statename %g/%d_flop1
All save states will be stored inside sta\c64\robby\
```

Tip

Note that even on Microsoft Windows, you should use / as your path separator for **-statename**

-[no]burnin

Tracks brightness of the screen during play and at the end of emulation generates a PNG that can be used to simulate burn-in effects on other systems. The resulting PNG is created such that the least used-areas of the screen are fully white (since burned-in areas are darker, all other areas of the screen must be lightened a touch).

The intention is that this PNG can be loaded via an artwork file with a low alpha (e.g, 0.1-0.2 seems to work well) and blended over the entire screen.

The PNG files are saved in the snap directory under the <systemname>/burnin-<screen.name>.png.

The default is OFF (**-noburnin**).

Example:

```
mame neogeo -burnin
```

5.1.14 Core Performance Options

-[no]autoframeskip / -[no]afs

Dynamically adjust the frameskip level while you're running the system to maintain full speed. Turning this on overrides the **-frameskip** setting described below.

This is off by default (**-noautoframeskip**).

Example:

```
mame gradius4 -autoframeskip
```

-frameskip / -fs <*level*>

Specifies the frameskip value. This is the number of frames out of every 12 to drop when running. For example, if you specify **-frameskip 2**, MAME will render and display 10 out of every 12 emulated frames. By skipping some frames, you may be able to get full speed emulation for a system that would otherwise be too demanding for your computer.

The default value is **-frameskip 0**, which skips no frames.

Example:

```
mame gradius4 -frameskip 2
```

-seconds_to_run / -str <*seconds*>

This option tells MAME to automatically stop emulation after a fixed number of seconds of emulated time have elapsed. This may be useful for benchmarking and automated testing. By combining this with a fixed set of other command line options, you can set up a consistent environment for benchmarking MAME's

emulation performance. In addition, upon exit, the **-str** option will write a screenshot to the system's snapshot directory with the file name determined by the **-snapname** option.

Example:

```
mame pacman -seconds_to_run 60
```

-[no]throttle

Enable or disable throttling emulation speed. When throttling is enabled, MAME limits emulation speed to so the emulated system will not run faster than the original hardware. When throttling is disabled, MAME runs the emulation as fast as possible. Depending on your settings and the characteristics of the emulated system, performance may be limited by your CPU, graphics card, or even memory performance.

The default is to enable throttling (**-throttle**).

Example:

```
mame pacman -nothrottle
```

-[no]sleep

When enabled along with **-throttle**, MAME will yield the CPU when limiting emulation speed. This allows other programs to use CPU time, assuming the main emulation thread isn't completely utilising a CPU core. This option can potentially cause hiccups in performance if other demanding programs are running.

The default is on (**-sleep**).

Example:

```
mame gradius 4 -nosleep
```

-speed <*factor*>

Changes the way MAME throttles the emulation so that it runs at some multiple of the system's original speed. A <*factor*> of `1.0` means to run the system at its normal speed, a <*factor*> of `0.5` means run at half speed, and a <*factor*> of 2.0 means run at double speed. Note that changing this value affects sound playback as well, which will scale in pitch accordingly. The internal precision of the fraction is two decimal places, so a <*factor*> of `1.002` is rounded to `1.00`.

The default is `1.0` (normal speed).

Example:

```
mame pacman -speed 1.25
```

-[no]refreshspeed / -[no]rs

Allows MAME to adjust the emulation speed so that the refresh rate of the first emulated screen does not exceed the slowest refresh rate for any targeted monitors in your system. Thus, if you have a 60Hz monitor and run a system that is designed to run at 60.6Hz, MAME will reduce the emulation speed to 99% in order to prevent sound hiccups or other undesirable side effects of running at a slower refresh rate.

The default is off (**-norefreshspeed**).

Example:

```
mame pacman -refreshspeed
```

-numprocessors / -np auto|<*value*>

Specify the number of threads to use for work queues. Specifying `auto` will use the value reported by the system or environment variable `OSDPROCESSORS`. This value is internally limited to four times the number of processors reported by the system.

The default is `auto`.

Example:

```
mame gradius4 -numprocessors 2
```

-bench <*n*>

Benchmark for <*n*> emulated seconds. This is equivalent to the following options:

-str <*n*> **-video none -sound none -nothrottle**

Example:

```
mame gradius4 -bench 300
```

-[no]lowlatency

This tells MAME to draw a new frame before throttling to reduce input latency. This is particularly effective with VRR (Variable Refresh Rate) displays.

This may cause frame pacing issues in the form of jitter with some systems (especially newer 3D-based systems or systems that run software akin to an operating system), so the default is off (**-nolowlatency**).

Example:

```
mame bgaregga -lowlatency
```

5.1.15 Core Rotation Options

-[no]rotate

Rotate the system to match its normal state (horizontal/vertical). This ensures that both vertically and horizontally oriented systems show up correctly without the need to rotate your monitor. If you want to keep the system displaying 'raw' on the screen the way it would have in the arcade, turn this option OFF.

The default is ON (**-rotate**).

Example:

```
mame pacman -norotate
```

-[no]ror

-[no]rol

Rotate the system screen to the right (clockwise) or left (counter-clockwise) relative to either its normal state (if **-rotate** is specified) or its native state (if **-norotate** is specified).

The default for both of these options is OFF (**-noror -norol**).

Example 1:

```
mame pacman -ror
```

Example 2:

```
mame pacman -rol
```

-[no]autoror

-[no]autorol

These options are designed for use with pivoting screens that only pivot in a single direction. If your screen only pivots clockwise, use **-autorol** to ensure that the system will fill the screen either horizontally or vertically in one of the directions you can handle. If your screen only pivots counter-clockwise, use **-autoror**.

Example 1:

```
mame pacman -autoror
```

Example 2:

```
mame pacman -autorol
```

Tip

If you have a display that can be rotated, **-autorol** or **-autoror** will allow you to get a larger display for both horizontal and vertical systems.

-[no]flipx

-[no]flipy

Flip (mirror) the system screen either horizontally (**-flipx**) or vertically (**-flipy**). The flips are applied after the **-rotate** and **-ror/-rol** options are applied.

The default for both of these options is OFF (**-noflipx -noflipy**).

Example 1:

```
mame -flipx pacman
```

Example 2:

```
mame -flipy suprmrio
```

5.1.16 Core Video Options

-video <*bgfx|gdi|d3d|opengl|soft|accel|none*>

Specifies which video subsystem to use for drawing. Options here depend on the operating system and whether this is an SDL-compiled version of MAME.

Generally Available:

- Using `bgfx` specifies the new hardware accelerated renderer.
- Using `opengl` tells MAME to render video using OpenGL acceleration.
- Using `none` displays no windows and does no drawing. This is primarily intended for benchmarking emulation without the overhead of the video system.

On Windows:

- Using `gdi` tells MAME to render video using older standard Windows graphics drawing calls. This is the slowest but most compatible option on older versions of Windows or buggy graphics hardware drivers.
- Using `d3d` tells MAME to use Direct3D 9 for rendering. This produces better quality output than `gdi` and enables additional rendering options. It is recommended if you have a 3D-capable video card or onboard Intel video of the HD3000 line or better.

On other platforms (including SDL on Windows or macOS):

- Using `accel` tells MAME to render video using SDL's 2D acceleration if possible.
- Using `soft` uses software rendering for video output. This isn't as fast or as nice as OpenGL, but it will work on any platform.

On macOS native:

- Using `soft` uses software rendering for video output. This isn't as fast or as nice as OpenGL, but it will work on any platform.

Defaults:

- The default on Windows is `d3d`.
- The default for macOS is `opengl` because macOS is guaranteed to have a compliant OpenGL stack.
- The default on all other systems is `soft`.

Example:

```
mame gradius3 -video bgfx
```

-numscreens <*count*>

Tells MAME how many output windows or screens to create. For most systems, a single output window is all you need, but some systems originally used multiple screens (*e.g. Darius and PlayChoice-10 arcade machines*). Some systems with front panel controls and/or status lights also may let you put these in different windows/screens. Each screen (up to 4) has its own independent settings for physical monitor, aspect ratio, resolution, and view, which can be set using the options below.

The default is `1`.

Example 1:

```
mame darius -numscreens 3
```

Example 2:

```
mame pc_cntra -numscreens 2
```

-[no]window / -[no]w

Run MAME in either a window or full screen.

The default is OFF (**-nowindow**).

Example:

```
mame coco3 -window
```

-[no]maximize / -[no]max

Controls initial window size in windowed mode. If it is set on, the window will initially be set to the maximum supported size when you start MAME. If it is turned off, the window will start out at the closest possible size to the original size of the display; it will scale on only one axis where non-square pixels are used. This option only has an effect when the **-window** option is used.

The default is ON (**-maximize**).

Example:

```
mame apple2e -window -nomaximize
```

-[no]keepaspect / -[no]ka

When enabled, MAME preserves the correct aspect ratio for the emulated system's screen(s). This is most often 4:3 or 3:4 for CRT monitors (depending on the orientation), though many other aspect ratios have been used, such as 3:2 (Nintendo Game Boy), 5:4 (some workstations), and various other ratios. If the emulated screen and/or artwork does not fill MAME's screen or Window, the image will be centred and black bars will be added as necessary to fill unused space (either above/below or to the left and right).

When this option is disabled, the emulated screen and/or artwork will be stretched to fill MAME's screen or window. The image will be distorted by non-proportional scaling if the aspect ratio does not match. This is very pronounced when the emulated system uses a vertically-oriented screen and MAME stretches the image to fill a horizontally-oriented screen.

On Windows, when this option is enabled and MAME is running in a window (not full screen), the aspect ratio will be maintained when you resize the window unless you hold the **Control** (or **Ctrl**) key on your keyboard. The window size will not be restricted when this option is disabled.

The default is ON (**-keepaspect**).

The MAME team strongly recommends leaving this option enabled. Stretching systems beyond their original aspect ratio will mangle the appearance of the system in ways that no filtering or shaders can repair.

Example:

```
mame sf2ua -nokeepaspect
```

-[no]waitvsync

Waits for the refresh period on your computer's monitor to finish before starting to draw video to your screen. If this option is off, MAME will just draw to the screen as a frame is ready, even if in the middle of a refresh cycle. This can cause "tearing" artifacts, where the top portion of the screen is out of sync with the bottom portion.

The effect of turning **-waitvsync** on differs a bit between combinations of different operating systems and video drivers.

On Windows, **-waitvsync** will block until video blanking before allowing MAME to draw the next frame, limiting the emulated machine's framerate to that of the host display. Note that this option does not work with **-video gdi** mode in Windows.

On macOS, **-waitvsync** does not block; instead the most recent completely drawn frame will be displayed at vblank. This means that if an emulated system has a higher framerate than your host display, emulated frames will be dropped periodically resulting in motion judder.

On Windows, you should only need to turn this on in windowed mode. In full screen mode, it is only needed if **-triplebuffer** does not remove the tearing, in which case you should use **-notriplebuffer -waitvsync**.

Note that SDL-based MAME support for this option depends entirely on your operating system and video drivers; in general it will not work in windowed mode so **-video opengl** and fullscreen give the greatest

chance of success with SDL builds of MAME.

The default is OFF (**-nowaitvsync**).

Example:

```
mame gradius2 -waitvsync
```

-[no]syncrefresh

Enables speed throttling only to the refresh of your monitor. This means that the system's actual refresh rate is ignored; however, the sound code still attempts to keep up with the system's original refresh rate, so you may encounter sound problems.

This option is intended mainly for those who have tweaked their video card's settings to provide carefully matched refresh rate options. Note that this option does not work with **-video gdi** mode.

The default is OFF (**-nosyncrefresh**).

Example:

```
mame mk -syncrefresh
```

-prescale <*amount*>

Controls the size of the screen images when they are passed off to the graphics system for scaling. At the minimum setting of 1, the screen is rendered at its original resolution before being scaled. At higher settings, the screen is expanded in both axes by a factor of <*amount*> using nearest-neighbor sampling before applying filters or shaders. With **-video d3d**, this produces a less blurry image at the expense of speed.

The default is `1`.

This is supported with all video output types (`bgfx`, `d3d`, etc.) on Windows and is supported with BGFX and OpenGL on other platforms.

Example:

```
mame pacman -video d3d -prescale 3
```

-[no]filter / -[no]d3dfilter / -[no]flt

Enable bilinear filtering on the system screen graphics. When disabled, point filtering is applied, which is crisper but leads to scaling artifacts. If you don't like the filtered look, you are probably better off increasing the **-prescale** value rather than turning off filtering altogether.

The default is ON (**-filter**).

This is supported with OpenGL and D3D video on Windows and is **ONLY** supported with OpenGL on other platforms.

Use `bgfx_screen_chains` in your INI file(s) to adjust filtering with the BGFX video system.

Example:

```
mame pacman -nofilter
```

-[no]unevenstretch

Allow non-integer scaling factors allowing for great window sizing flexibility.

The default is ON. (**-unevenstretch**)

Example:

```
mame dkong -nounevenstretch
```

5.1.17 Core Full Screen Options

-[no]switchres

Enables resolution switching. This option is required for the **-resolution*** options to switch resolutions in full screen mode.

On modern video cards, there is little reason to switch resolutions unless you are trying to achieve the "exact" pixel resolutions of the original systems, which requires significant tweaking. This is also true on LCD displays, since they run with a fixed resolution and switching resolutions on them is just silly. This option does not work with **-video gdi** and **-video bgfx**.

The default is OFF (**-noswitchres**).

Example:

```
mame kof97 -video d3d -switchres -resolution 1280x1024
```

5.1.18 Core Per-Window Options

-screen <*display*>

-screen0 <*display*>

-screen1 <*display*>

-screen2 <*display*>

-screen3 <*display*>

Specifies which physical monitor on your system you wish to have each window use by default. In order to use multiple windows, you must have increased the value of the **-numscreens** option. The name of each display in your system can be determined by running MAME with the -verbose option. The display names are typically in the format of: `\\\\.\\DISPLAYn`, where 'n' is a number from 1 to the number of connected monitors.

The default value for these options is `auto`, which means that the first window is placed on the first display, the second window on the second display, etc.

The **-screen0**, **-screen1**, **-screen2**, **-screen3** parameters apply to the specific window. The **-screen** parameter applies to all windows. The window-specific options override values from the all window option.

Example 1:

```
mame pc_cntra -numscreens 2 -screen0 \\.\DISPLAY1 -screen1 \\.\DISPLAY2
```

Example 2:

```
mame darius -numscreens 3 -screen0 \\.\DISPLAY1 -screen1 \\.\DISPLAY3 -
↪screen2 \\.\DISPLAY2
```

Tip

Using **-verbose** will tell you which displays you have on your system, where they are connected, and what their current resolutions are.

Tip

Multiple Screens may fail to work correctly on some Mac machines as of right now.

-aspect <*width:height*> / **-screen_aspect** <*num:den*>

-aspect0 <*width:height*>

-aspect1 <*width:height*>

-aspect2 <*width:height*>

-aspect3 <*width:height*>

Specifies the physical aspect ratio of the physical monitor for each window. In order to use multiple windows, you must have increased the value of the **-numscreens** option. The physical aspect ratio can be determined by measuring the width and height of the visible screen image and specifying them separated by a colon.

The default value for these options is `auto`, which means that MAME assumes the aspect ratio is proportional to the number of pixels in the desktop video mode for each monitor.

The **-aspect0**, **-aspect1**, **-aspect2**, **-aspect3** parameters apply to the specific window. The **-aspect** parameter applies to all windows. The window-specific options override values from the all window option.

Example 1:

```
mame contra -aspect 16:9
```

Example 2:

```
mame pc_cntra -numscreens 2 -aspect0 16:9 -aspect1 5:4
```

-resolution <*widthxheight[@refresh]*> / **-r** <*widthxheight[@refresh]*>

-resolution0 <*widthxheight[@refresh]*> / **-r0** <*widthxheight[@refresh]*>

-resolution1 <*widthxheight[@refresh]*> / **-r1** <*widthxheight[@refresh]*>

-resolution2 <*widthxheight[@refresh]*> / **-r2** <*widthxheight[@refresh]*>

-resolution3 <*widthxheight[@refresh]*> / **-r3** <*widthxheight[@refresh]*>

Specifies an exact resolution to run in. In full screen mode, MAME will try to use the specific resolution you request. The width and height are required; the refresh rate is optional. If omitted or set to 0, MAME will determine the mode automatically. For example, **-resolution 640x480** will force 640x480 resolution, but MAME is free to choose the refresh rate. Similarly, **-resolution 0x0@60** will force a 60Hz refresh rate, but allows MAME to choose the resolution. The string `auto` is also supported, and is equivalent to `0x0@0`.

In window mode, this resolution is used as a maximum size for the window. This option requires the **-switchres** option as well in order to actually enable resolution switching with **-video d3d**.

The default value for these options is `auto`.

The **-resolution0**, **-resolution1**, **-resolution2**, **-resolution3** parameters apply to the specific window. The **-resolution** parameter applies to all windows. The window-specific options override values from the all window option.

Example:

```
mame pc_cntra -numscreens 2 -resolution0 1920x1080 -resolution1 1280x1024
```

-view <*viewname*>

-view0 <*viewname*>

-view1 <*viewname*>

-view2 <*viewname*>

-view3 <*viewname*>

Specifies the initial view setting for each window/screen. The <*viewname*> does not need to be the full name of a view, MAME will choose the first view with a name that has the <*viewname*> as a prefix. For example **-view "screen 0 pixel"** will match the "*Screen 0 Pixel Aspect (10:7)*" view.

If the <*viewname*> is `auto` or an empty string, MAME will select views based on the number of emulated screens in the system, the number of windows/screens MAME is using, and the available external and internal artwork. MAME tries to select views so that all emulated screens are visible by default.

The default value for these options is `auto`.

The **-view0**, **-view1**, **-view2**, **-view3** parameters apply to the specific window. The **-view** parameter applies to all windows. The window-specific options override values from the all windows option.

Note that view settings saved in the configuration file for the machine take precedence over the initial view settings. If you change the selected views in the Video Options menu, this will be saved in the configuration file for the machine and take precedence over any initial views specified in INI files or on the command line.

Example:

```
mame contra -view native
```

5.1.19 Core Artwork Options

-[no]artwork_crop / -[no]artcrop

Enable cropping of artwork to the system screen area only. This means that vertically oriented systems running full screen can display their artwork to the left and right sides of the screen. This option can also be controlled via the Video Options menu in the user interface.

The default is OFF **-noartwork_crop**.

Example:

```
mame pacman -artwork_crop
```

Tip

-artwork_crop is great for widescreen displays. You will get a full-sized system display and the artwork will fill the empty space on the sides as much as possible.

-fallback_artwork <*artwork name*>

Specifies fallback artwork if no external artwork or internal driver layout is defined. If external artwork for the system is present or a layout is included in the driver for the system, then that will take precedence.

Example:

```
mame coco -fallback_artwork suprmrio
```

Tip

You can use **fallback_artwork <artwork name>** in `horizontal.ini` and `vertical.ini` to specify different fallback artwork choices for horizontal and vertical systems.

-override_artwork <*artwork name*>

Specifies override artwork for external artwork and internal driver layout.

Example:

```
mame galaga -override_artwork puckman
```

-artwork_font / -artfont <*fontname*>

Specifies the font to use for artwork text elements. The same considerations apply as for the UI font (see the *uifont option*).

Note that artwork is typically designed around a sans serif font with tight character spacing (e.g. **Tahoma**, which is the default on Windows). Using a font with wider character spacing or a fixed pitch font (e.g. a Courier family font) may result in text positioning issues.

Example:

```
mame starwbc -artwork_font "Comic Sans MS"
```

5.1.20 Core Screen Options

-brightness <*value*>

Controls the default brightness, or black level, of the system screens. This option does not affect the artwork or other parts of the display. Using the MAME UI, you can individually set the brightness for each system screen; this option controls the initial value for all visible system screens. The standard and default value is `1.0`. Selecting lower values (down to 0.1) will produce a darkened display, while selecting higher values (up to 2.0) will give a brighter display.

Example:

```
mame pacman -brightness 0.5
```

-contrast <*value*>

Controls the contrast, or white level, of the system screens. This option does not affect the artwork or other parts of the display. Using the MAME UI, you can individually set the contrast for each system screen; this option controls the initial value for all visible system screens. The standard and default value is `1.0`. Selecting lower values (down to 0.1) will produce a dimmer display, while selecting higher values (up to 2.0) will give a more saturated display.

Example:

```
mame pacman -contrast 0.5
```

-gamma <*value*>

Controls the gamma, which produces a potentially nonlinear black to white ramp, for the system screens. This option does not affect the artwork or other parts of the display. Using the MAME UI, you can individually set the gamma for each system screen; this option controls the initial value for all visible system screens. The standard and default value is `1.0`, which gives a linear ramp from black to white. Selecting lower values (down to 0.1) will increase the nonlinearity toward black, while selecting higher values (up to 3.0) will push the nonlinearity toward white.

The default is `1.0`.

Example:

```
mame pacman -gamma 0.8
```

-pause_brightness <*value*>

This controls the brightness level when MAME is paused.

The default value is `0.65`.

Example:

```
mame pacman -pause_brightness 0.33
```

-effect <*filename*>

Specifies a single PNG file that is used as an overlay over any system screens in the video display. This PNG file is assumed to live in the root of one of the artpath directories. The pattern in the PNG file is repeated both horizontally and vertically to cover the entire system screen areas (but not any external artwork), and is rendered at the target resolution of the system image.

For **-video gdi** and **-video d3d** modes, this means that one pixel in the PNG will map to one pixel on your output display. The RGB values of each pixel in the PNG are multiplied against the RGB values of the target screen.

The default is `none`, meaning no effect.

Example:

```
mame pacman -effect scanlines
```

5.1.21 Core Vector Options

-beam_width_min <*width*>

Sets the vector beam minimum width. The beam width varies between the minimum and maximum beam widths as the intensity of the vector drawn changes. To disable vector width changes based on intensity, set the maximum equal to the minimum.

Example:

```
mame asteroid -beam_width_min 0.1
```

-beam_width_max <*width*>

Sets the vector beam maximum width. The beam width varies between the minimum and maximum beam widths as the intensity of the vector drawn changes. To disable vector width changes based on intensity, set the maximum equal to the minimum.

Example:

```
mame asteroid -beam_width_max 2
```

-beam_intensity_weight <*weight*>

Sets the vector beam intensity weight. This value determines how the intensity of the vector drawn affects the width. A value of 0 creates a linear mapping from intensity to width. Negative values mean that lower intensities will increase the width toward maximum faster, while positive values will increase the width toward maximum more slowly.

Example:

```
mame asteroid -beam_intensity_weight 0.5
```

-beam_dot_size <*scale*>

Scale factor to apply to the size of single-point dots in vector games. Normally these are rendered according to the computed beam width; however, it is common for this to produce dots that are difficult to see. The beam_dot_size option applies a scale factor on top of the beam width to help them show up better.

The default is `1`.

Example:

```
mame asteroid -beam_dot_size 2
```

-flicker <*value*>

Simulates a vector "flicker" effect, similar to a vector monitor that needs adjustment. This option requires a float argument in the range of 0.00 - 100.00 (0=none, 100=maximum).

The default is `0`.

Example:

```
mame asteroid -flicker 0.15
```

5.1.22 Core Video OpenGL Feature Options

These options are for compatibility in **-video opengl**. If you report rendering artifacts you may be asked to try messing with them by the developers, but normally they should be left at their defaults which results in the best possible video performance.

Tip

Examples are not provided for these options as MAMEdev will provide suitable test options in the case of needing them for debugging.

-[no]gl_forcepow2texture

Always use only power-of-2 sized textures.

The default is OFF. (**-nogl_forcepow2texture**)

-[no]gl_notexturerect

Don't use OpenGL GL_ARB_texture_rectangle.

The default is ON. (**-gl_notexturerect**)

-[no]gl_vbo

Enable OpenGL VBO (Vertex Buffer Objects), if available.

The default is ON. (**-gl_vbo**)

-[no]gl_pbo

Enable OpenGL PBO (Pixel Buffer Objects), if available (default on)

The default is ON. (**-gl_pbo**)

5.1.23 Core Video OpenGL GLSL Options

-[no]gl_glsl

Enable OpenGL GLSL, if available.

The default is OFF (**-nogl_glsl**).

Example:

```
mame galaxian -gl_glsl
```

-gl_glsl_filter

Use OpenGL GLSL shader-based filtering instead of fixed function pipeline-based filtering.

0-plain, 1-bilinear, 2-bicubic

The default is 1. (**-gl_glsl_filter 1**)

Example:

```
mame galaxian -gl_glsl -gl_glsl_filter 0
```

-glsl_shader_mame0

-glsl_shader_mame1

...

-glsl_shader_mame9

Set a custom OpenGL GLSL shader effect to the internal system screen in the given slot. MAME does not include a vast selection of shaders by default; more can be found online.

Example:

```
mame suprmrio -gl_glsl -glsl_shader_mame0 NTSC/NTSC_chain -glsl_shader_
↪mame1 CRT-geom/CRT-geom
```

-glsl_shader_screen0

-glsl_shader_screen1

...

-glsl_shader_screen9

Set a custom OpenGL GLSL shader effect to the whole scaled-up output screen that will be rendered by your graphics card. MAME does not include a vast selection of shaders by default; more can be found online.

Example:

```
mame suprmrio -gl_glsl -glsl_shader_screen0 gaussx -glsl_shader_screen1␣
↪gaussy -glsl_shader_screen2 CRT-geom-halation
```

5.1.24 Core Sound Options

-samplerate <*value*> / **-sr** <*value*>

Sets the audio sample rate. Smaller values (e.g. 11025) cause lower audio quality but faster emulation speed. Higher values (e.g. 48000) cause higher audio quality but slower emulation speed.

The default is `48000`.

Example:

```
mame galaga -samplerate 44100
```

-[no]samples

Use samples if available.

The default is ON (**-samples**).

Example:

```
mame qbert -nosamples
```

-volume / **-vol** <*value*>

Sets the initial sound volume. It can be changed later with the user interface (see Keys section). The volume is in decibels: e.g. "**-volume -12**" will start with -12 dB attenuation. Note that if the volume is changed in the user interface it will be saved to the configuration file for the system. The value from the configuration file for the system has priority over `volume` settings in general INI files.

The default is `0` (no attenuation, or full volume).

Example:

```
mame pacman -volume -30
```

-sound <*wasapi* | *xaudio2* | *coreaudio* | *pipewire* | *pulse* | *sdl* | *portaudio* | *none*>

Specifies which sound module to use. Selecting `none` disables sound output and input altogether (sound hardware is still emulated).

Available features, performance and latency vary between sound modules. You may have to change the value of the *latency option* if you change the sound module.

When using the `sdl` sound subsystem, the audio API to use may be selected by setting the *SDL_AUDIODRIVER* environment variable. Available audio APIs depend on the operating system. On Windows, it may be necessary to set `SDL_AUDIODRIVER=directsound` if no sound output is produced by default.

The default is `wasapi` on Windows. On Mac, `coreaudio` is the default. On all other platforms, `sdl` is the default.

Example:

```
mame pacman -sound portaudio
```

Table 6: Sound module supported platforms and features

Module	Supported OS	Input	Output monitoring	Multi-channel	Device changes
`wasapi`	Windows	Yes	Yes	Yes	Yes
`xaudio2`	Windows	No	No	Yes	Yes
`coreaudio`	macOS	Yes	No	Yes	Yes
`pipewire`	Linux	Yes	?	Yes	Yes
`pulse`	Linux	No	No	Yes	Yes
`sdl`	All[20]	No	No	Yes[21]	No
`portaudio`	All	Yes	Yes[22]	Yes	No

-audio_latency <*value*> / **-alat** <*value*>

Audio latency, conventionally in number of audio frames (1 audio frame is 20ms). It is not required to supply whole numbers, eg. a value of `1.5` is 30ms). Smaller values provide less audio delay while requiring better system performance. Larger values increase audio delay but may help avoid buffer underruns and audio interruptions. A value of `0` will use the default for the selected sound module.

You may need to change the value of this option if you change the sound module using the *sound option*. This option is unsupported on sound modules `pipewire`, `pulse`, `sdl`.

The default is `0`.

Example:

```
mame galaga -audio_latency 2
```

5.1.25 Core Input Options

-[no]coin_lockout / -[no]coinlock

Enables simulation of the "coin lockout" feature that is implemented on a number of arcade game PCBs. It was up to the operator whether or not the coin lockout outputs were actually connected to the coin mechanisms. If this feature is enabled, then attempts to enter a coin while the lockout is active will fail and will display a popup message in the user interface (in debug mode). If this feature is disabled, the coin lockout signal will be ignored.

The default is ON (**-coin_lockout**).

Example:

```
mame suprmrio -coin_lockout
```

-ctrlr <*controller*>

Specifies a controller configuration file, typically used to set more suitable default input assignments for special controllers. Directories specified using the `ctrlrpath` option are searched. Controller configuration files use a similar format to `.cfg` used to save system settings. See *Controller Configuration Files* for more details.

[20] While SDL is not a supported option on official MAME builds for Windows, you can compile MAME with SDL support on Windows.
[21] MAME requires SDL 2.0.16 or later for multi-channel sound support.
[22] PortAudio support for output monitoring depends on the platform and sound API.

The default is NULL (no controller configuration file).

Example:

```
mame dkong -ctrlr xarcade
```

-[no]mouse

Controls whether or not MAME makes use of mouse controllers. When this is enabled, you will likely be unable to use your mouse for other purposes until you exit or pause the system. Supported mouse controllers depend on your *mouseprovider setting*.

Note that if this setting is off (**-nomouse**), mouse input may still be enabled depending on the inputs present on the emulated system and your *automatic input enable settings*. In particular, the default is to enable mouse input when the emulated system has mouse inputs (**-mouse_device mouse**), so MAME will capture your mouse pointer when you run a system with mouse inputs unless you also change the **mouse_device** setting.

The default is OFF (**-nomouse**).

Example:

```
mame centiped -mouse
```

-[no]joystick / -[no]joy

Controls whether or not MAME makes use of game controllers (e.g. joysticks, gamepads and simulation controls). Supported game controllers depend on your *joystickprovider setting*.

When this is enabled, MAME will ask the system about which controllers are connected.

Note that if this setting is off (**-nojoystick**), joystick input may still be enabled depending on the inputs present on the emulated system and your *automatic input enable settings*.

The default is OFF (**-nojoystick**).

Example:

```
mame mappy -joystick
```

-[no]lightgun / -[no]gun

Controls whether or not MAME makes use of lightgun controllers. Note that most lightguns also produce mouse input, so enabling mouse and lightgun controllers simultaneously (using **-lightgun** and **-mouse** together) may produce strange behaviour. Supported lightgun controllers depend on your *lightgunprovider setting*.

Note that if this setting is off (**-nolightgun**), lightgun input may still be enabled depending on the inputs present on the emulated system and your *automatic input enable settings*.

The default is OFF (**-nolightgun**).

Example:

```
mame lethalen -lightgun
```

-[no]multikeyboard / -[no]multikey

Determines whether MAME differentiates between multiple keyboards. Some systems may report more than one keyboard; by default, the data from all of these keyboards is combined so that it looks like a single keyboard.

Turning this option on will enable MAME to report keypresses on different keyboards independently.

The default is OFF (**-nomultikeyboard**).

Example:

```
mame sf2ceua -multikey
```

-[no]multimouse

Determines whether MAME differentiates between multiple mice. Some systems may report more than one mouse device; by default, the data from all of these mice is combined so that it looks like a single mouse. Turning this option on will enable MAME to report mouse movement and button presses on different mice independently.

The default is OFF (**-nomultimouse**).

Example:

```
mame warlords -multimouse
```

-[no]steadykey / -[no]steady

Some systems require two or more buttons to be pressed at exactly the same time to make special moves. Due to limitations in the keyboard hardware, it can be difficult or even impossible to accomplish that using the standard keyboard handling. This option selects a different handling that makes it easier to register simultaneous button presses, but has the disadvantage of making controls less responsive.

The default is OFF (**-nosteadykey**)

Example:

```
mame sf2ua -steadykey
```

-[no]ui_active

Enable user interface on top of emulated keyboard (if present).

The default is OFF (**-noui_active**)

Example:

```
mame apple2e -ui_active
```

-joystick_map <*map*> / **-joymap** <*map*>

Controls how analog joystick values map to digital joystick controls.

Systems such as Pac-Man use a 4-way digital joystick and will exhibit undesired behavior when a diagonal is triggered; in the case of Pac-Man, movement will stop completely at intersections when diagonals are triggered and the game will be considerably harder to play correctly. Many other arcade cabinets used 4-way or 8-way joysticks (as opposed to full analog joysticks), so for true analog joysticks such as flight sticks and analog thumb sticks, this then needs to be mapped down to the expected 4-way or 8-way digital joystick values.

To do this, MAME divides the analog range into a 9x9 grid that looks like this:

insert 9x9 grid picture here

MAME then takes the joystick axis position (for X and Y axes only), maps it to this grid, and then looks up a translation from a joystick map. This parameter allows you to specify the map.

For instance, an 8-way joystick map traditionally looks like this:

insert 8-way map picture here

This mapping gives considerable leeway to the angles accepted for a given direction, so that being approximately in the area of the direction you want will give you the results you want. Without that, if you were slightly off center while holding the stick left, it would not recognize the action correctly.

The default is `auto`, which means that a standard 8-way, 4-way, or 4-way diagonal map is selected automatically based on the input port configuration of the current system.

Generally you will want to set up the **-joystick_map** setting in the per-system `<system>.ini` file as opposed to the main `MAME.INI` file so that the mapping only affects the systems you want it to. See *Multiple Configuration Files* for further details on per-system configuration.

Maps are defined as a string of numbers and characters. Since the grid is 9x9, there are a total of 81 characters necessary to define a complete map. Below is an example map for an 8-way joystick that matches the picture shown above:

777888999 777888999 777888999 444555666 444555666 444555666 111222333 111222333 111222333	Note that the numeric digits correspond to the keys on a numeric keypad. So '7' maps to up+left, '4' maps to left, '5' maps to neutral, etc. In addition to the numeric values, you can specify the character 's', which means "sticky". Sticky map positions will keep the output the same as the last non-sticky input sent to the system.

To specify the map for this parameter, you can specify a string of rows separated by a '.' (which indicates the end of a row), like so:

-joymap 777888999.777888999.777888999.444555666.444555666.444555666.111222333.111222333.111222333

However, this can be reduced using several shorthands supported by the <map> parameter. If information about a row is missing, then it is assumed that any missing data in columns 5-9 are left/right symmetric with data in columns 0-4; and any missing data in columns 0-4 is assumed to be copies of the previous data. The same logic applies to missing rows, except that up/down symmetry is assumed.

By using these shorthands, the 81 character map can be simply specified by this 11 character string: 7778...4445 (which means we then use **-joymap 7778...4445**)

Looking at the first row, 7778 is only 4 characters long. The 5th entry can't use symmetry, so it is assumed to be equal to the previous character '8'. The 6th character is left/right symmetric with the 4th character, giving an '8'. The 7th character is left/right symmetric with the 3rd character, giving a '9' (which is '7' with left/right flipped). Eventually this gives the full 777888999 string of the row.

The second and third rows are missing, so they are assumed to be identical to the first row. The fourth row decodes similarly to the first row, producing 444555666. The fifth row is missing so it is assumed to be the same as the fourth.

The remaining three rows are also missing, so they are assumed to be the up/down mirrors of the first three rows, giving three final rows of 111222333.

With 4-way games, sticky becomes important to avoid problems with diagonals. Typically you would choose a map that looks something like this:

insert 9x9 4-way sticky grid picture here

This means that if you press left, then roll the stick towards up (without re-centering it) you'll pass through the sticky section in the corner. As you do, MAME will read that sticky corner as **left** as that's the last non-sticky input it received. As the roll gets into the upward space of the map, this then switches to an up motion.

This map would look somewhat like:

s8888888s 4s88888s6 44s888s66 444555666 444555666 444555666 44s222s66 4s22222s6 s2222222s	For this mapping, we have a wide range for the cardinal directions on 8, 4, 6, and 2. We have sticky on the meeting points between those cardinal directions where the appropriate direction isn't going to be completely obvious.

To specify the map for this parameter, you can specify a string of rows separated by a '.' (which indicates the end of a row), like so:

-joymap s8888888s.4s88888s6.44s888s66.444555666.444555666.444555666.44s222s66.4s22222s6.s2222222s

Like before, because of the symmetry between top and bottom and left and right, we can shorten this down to:

-joymap s8.4s8.44s8.4445

-joystick_deadzone <*value*> / **-joy_deadzone** <*value*> / **-jdz** <*value*>

If you play with an analog joystick, the center can drift a little. joystick_deadzone tells how far along an axis you must move before the axis starts to change. This option expects a float in the range of 0.0 to 1.0. Where 0 is the center of the joystick and 1 is the outer limit.

The default is `0.15`.

Example:

```
mame sinistar -joystick_deadzone 0.3
```

-joystick_saturation <*value*> / **joy_saturation** <*value*> / **-jsat** <*value*>

If you play with an analog joystick, the ends can drift a little, and may not match in the +/- directions. joystick_saturation tells how far along an axis movement change will be accepted before it reaches the maximum range. This option expects a float in the range of 0.0 to 1.0, where 0 is the center of the joystick and 1 is the outer limit.

The default is `0.85`.

Example:

```
mame sinistar -joystick_saturation 1.0
```

-joystick_threshold <*value*> / **joy_threshold** <*value*> / **-jthresh** <*value*>

When a joystick axis (or other absolute analog axis) is assigned to a digital input, this controls how far it must be moved from the neutral position (or centre) to be considered active or switched on. This option expects a float in the range of 0.0 to 1.0, where 0 means any movement from the neutral position is considered active, and 1 means only the outer limits are considered active. This threshold is **not** adjusted to the range between the dead zone and saturation point.

Note that if a *joystick map* is configured, that will take precedence over this setting when a joystick's main X/Y axes are assigned to digital inputs.

The default is `0.3`.

Example:

```
mame raiden -joystick_threshold 0.2
```

-[no]natural

Allows user to specify whether or not to use a natural keyboard or not. This allows you to start your system in a 'native' mode, depending on your region, allowing compatibility for non-"QWERTY" style keyboards.

The default is OFF (**-nonatural**)

In "emulated keyboard" mode (the default mode), MAME translates pressing/releasing host keys/buttons to emulated keystrokes. When you press/release a key/button mapped to an emulated key, MAME presses/releases the emulated key.

In "natural keyboard" mode, MAME attempts to translate characters to keystrokes. The OS translates keystrokes to characters (similarly to when you type into a text editor), and MAME attempts to translate these characters to emulated keystrokes.

There are a number of unavoidable limitations in "natural keyboard" mode:

- The emulated system driver and/or keyboard device has to support it.
- The selected keyboard layout *must* match the keyboard layout selected in the emulated OS!
- Keystrokes that don't produce characters can't be translated (e.g. pressing a modifier on its own such as **shift**, **ctrl**, or **alt**).
- Holding a key until the character repeats will cause the emulated key to be pressed repeatedly as opposed to being held down.
- Dead key sequences are cumbersome to use at best.
- It won't work at all if IME edit is involved (e.g. for Chinese, Japanese or Korean language input).

Example:

```
mame coco2 -natural
```

-[no]joystick_contradictory

Enable contradictory direction digital joystick input at the same time such as **Left and Right** or **Up and Down** at the same time.

The default is OFF (**-nojoystick_contradictory**)

Example:

```
mame pc_smb -joystick_contradictory
```

-coin_impulse *[n]*

Set coin impulse time based on n (n<0 disable impulse, n==0 obey driver, 0<n set time n).

Default is `0`.

Example:

```
mame contra -coin_impulse 1
```

5.1.26 Core Input Automatic Enable Options

-paddle_device (none | keyboard | mouse | lightgun | joystick)

-adstick_device (none | keyboard | mouse | lightgun | joystick)

-pedal_device (none | keyboard | mouse | `lightgun | joystick)

-dial_device (none | keyboard | mouse | lightgun | joystick)

-trackball_device (none | keyboard | mouse | lightgun | joystick)

-lightgun_device (none | keyboard | mouse | lightgun | joystick)

-positional_device (none | keyboard | mouse | lightgun | joystick)

-mouse_device (none | keyboard | mouse | lightgun | joystick)

Each of these options sets whether mouse, joystick or lightgun controllers should be enabled when running an emulated system that uses a particular class of analog inputs. These options can effectively set *-mouse*, *-joystick* and/or *-lightgun* depending on the type of inputs present on the emulated system. Note that these options *will not* override explicit **-nomouse**, **-nojoystick** and/or **-nolightgun** settings at a higher priority level (e.g. in a more specific INI file or on the command line).

For example, if you specify the option **-paddle_device mouse**, then mouse controls will automatically be enabled when you run a game that has paddle controls (e.g. Super Breakout), even if you specified **-nomouse** .

The default is to automatically enable mouse controls when running emulated systems with mouse inputs (**-mouse_device mouse**).

Example:

```
mame sbrkout -paddle_device mouse
```

Tip

Note that these settings can override **-nomouse**, **-nojoystick** and/or **-nolightgun** depending on the inputs present on the emulated system.

5.1.27 Debugging Options

-[no]verbose / -[no]v

Displays internal diagnostic information. This information is very useful for debugging problems with your configuration.

The default is OFF (**-noverbose**).

Example:

```
mame polepos -verbose
```

Tip

IMPORTANT: When reporting bugs to MAMEdev, please run with **-verbose** and include the resulting information.

-[no]oslog

Output `error.log` messages to the system diagnostic output, if one is present.

By default messages are sent to the standard error output (this is typically displayed in the terminal or command prompt window, or saved to a system log file). On Windows, if a debugger is attached (e.g. the Visual Studio debugger or WinDbg), messages will be sent to the debugger instead.

The default is OFF (**-nooslog**).

Example:

```
mame mappy -oslog
```

-[no]log

Creates a file called error.log which contains all of the internal log messages generated by the MAME core and system drivers. This can be used at the same time as **-oslog** to output the log data to both targets as well.

The default is OFF (**-nolog**).

Example 1:

```
mame qbert -log
```

Example 2:

```
mame qbert -oslog -log
```

-[no]debug

Activates the integrated debugger. By default, pressing the backtick/tilde (~) key during emulation breaks into the debugger. MAME also breaks into the debugger after the initial soft reset on startup if the debugger is active. See *MAME Debugger* for information on using the debugger.

The default is OFF (**-nodebug**).

Example:

```
mame indy_4610 -debug
```

-debugger <*module*>

Chooses the module to use for debugging the target system when the *debug* option is on. Available debugger modules depend on the host platform and build options.

Supported debugger modules:

windows
Win32 GUI debugger (default on Windows). Only supported on Windows.

qt
Qt GUI debugger (default on Linux). Supported on Windows, Linux and macOS, but only included on Linux by default. Set `USE_QTDEBUG=1` when compiling MAME to include the Qt debugger on Windows or macOS.

osx
Cocoa GUI debugger (default on macOS). Only supported on macOS.

imgui
ImgUi GUI debugger displayed in first MAME window. Requires *video* option to be set to **bgfx**. Supported on all platforms with BGFX video output support.

gdbstub
Acts as a remote debugging server for the GNU debugger (GDB). Only a small subset of the CPUs emulated by MAME are supported. Use the *debugger_port* option to set the listening port and the *debugger_host* option to set the address to bind to. Supported on all platforms with TCP socket support.

Example:

```
mame ambush -debug -debugger qt
```

-debugscript <*filename*>

Specifies a file that contains a list of debugger commands to execute immediately upon startup.

The default is `NULL` (*no commands*).

Example:

```
mame galaga -debug -debugscript testscript.txt
```

-[no]update_in_pause

Enables updating of the main screen bitmap while the system is paused. This means that the video update callback will be called repeatedly while the emulation is paused, which can be useful for debugging.

The default is OFF (**-noupdate_in_pause**).

Example:

```
mame indy_4610 -update_in_pause
```

-watchdog <*duration*> / **-wdog** <*duration*>

Enables an internal watchdog timer that will automatically kill the MAME process if more than <*duration*> seconds passes without a frame update. Keep in mind that some systems sit for a while during load time without updating the screen, so <*duration*> should be long enough to cover that.

10-30 seconds on a modern system should be plenty in general.

By default there is no watchdog.

Example:

```
mame ibm_5150 -watchdog 30
```

-debugger_host <*address*>

Set the IP address to listen on to accept GDB connections when using the GDB stub debugger module (see the *debugger* option).

The default is `localhost`.

Example:

```
mame rfjet -debug -debugger gdbstub -debugger_host 0.0.0.0
```

-debugger_port <*port*>

Set the TCP port number to accept GDB connections on when using the GDB stub debugger module (see the *debugger* option).

The default is `23946`.

Example:

```
mame rfjet -debug -debugger gdbstub -debugger_port 2159
```

-debugger_font <*fontname*> / **-dfont** <*fontname*>

Specifies the name of the font to use for debugger windows.

The Windows default font is `Lucida Console`.
The Mac (Cocoa) default font is system fixed-pitch font default (typically `Monaco`).
The Qt default font is `Courier New`.

Example:

```
mame marble -debug -debugger_font "Comic Sans MS"
```

-debugger_font_size <*points*> / **-dfontsize** <*points*>

Specifies the size of the font to use for debugger windows, in points.

The Windows default size is `9` points.
The Qt default size is `11` points.
The Mac (Cocoa) default size is the system default size.

Example:

```
mame marble -debug -debugger_font "Comic Sans MS" -debugger_font_size 16
```

5.1.28 Core Communication Options

-comm_localhost <*string*>

Local address to bind to. This can be a traditional `xxx.xxx.xxx.xxx` address or a string containing a resolvable hostname.

The default is value is `0.0.0.0` (which binds to all local IPv4 addresses).

Example:

```
mame arescue -comm_localhost 192.168.1.2
```

-comm_localport <*string*>

Local port to bind to. This can be any traditional communications port as an unsigned 16-bit integer (0-65535).

The default value is `15112`.

Example:

```
mame arescue -comm_localhost 192.168.1.2 -comm_localport 30100
```

-comm_remotehost <*string*>

Remote address to connect to. This can be a traditional xxx.xxx.xxx.xxx address or a string containing a resolvable hostname.

The default is value is "`0.0.0.0`" (which binds to all local IPv4 addresses).

Example:

```
mame arescue -comm_remotehost 192.168.1.2
```

-comm_remoteport <*string*>

Remote port to connect to. This can be any traditional communications port as an unsigned 16-bit integer (0-65535).

The default value is "`15112`".

Example:

```
mame arescue -comm_remotehost 192.168.1.2 -comm_remoteport 30100
```

-[no]comm_framesync

Synchronize frames between the communications network.

The default is OFF (**-nocomm_framesync**).

Example:

```
mame arescue -comm_remotehost 192.168.1.3 -comm_remoteport 30100 -comm_
↪framesync
```

5.1.29 Core Misc Options

-[no]drc

Enable DRC (dynamic recompiler) CPU cores if available. Turn this option off to use interpreter CPU cores if available. This option does not affect CPUs that only support one core type.

The default is ON (**-drc**).

Example:

```
mame ironfort -nodrc
```

-[no]drc_rwx

Allow DRC CPU cores to use memory that is simultaneously writable and executable if supported. Turning this option off may decrease performance. This option only affects DRC CPU cores, and has no effect in configurations that do not allow memory to be simultaneously writable and executable (e.g. recent versions of macOS and NetBSD).

The default is ON (**-drc_rwx**).

Example:

```
mame fiveside -nodrc_rwx
```

-[no]drc_use_c

Force DRC CPU cores to use the portable C code back-end when a native back-end is available. This option only affects DRC CPU cores, and has no effect if a native DRC back-end is not available.

The default is OFF (**-nodrc_use_c**).

Example:

```
mame vamphalf -drc_use_c
```

-[no]drc_log_uml

Write DRC UML disassembly log.

The default is OFF (**-nodrc_log_uml**).

Example:

```
mame ironfort -drc_log_uml
```

-[no]drc_log_native

Write DRC native disassembly log.

The default is OFF (**-nodrc_log_native**).

Example:

```
mame ironfort -drc_log_native
```

-bios <*biosname*>

Specifies the specific BIOS to use with the current system, for systems that make use of a BIOS. The **-listbios** output will list all of the possible BIOS names for a system, as does the **-listxml** output.

The default is `default`.

Example:

```
mame mslug -bios unibios33
```

-[no]cheat / -[no]c

Activates the cheat menu with autofire options and other tricks from the cheat database, if present. This also activates additional options on the slider menu for overall speed and overclocking/underclocking.

Be advised that savestates created with cheats on may not work correctly with this turned off and vice-versa.

The default is OFF (**-nocheat**).

Example:

```
mame dkong -cheat
```

-[no]skip_gameinfo

Forces MAME to skip displaying the system info screen.

The default is OFF (**-noskip_gameinfo**).

Example:

```
mame samsho5 -skip_gameinfo
```

-uifont <*fontname*>

Specifies the font to use for UI text. If this font cannot be found or cannot be loaded, MAME will fall back to its built-in UI font. Supported fonts depend on the platform and selected UI font provider module. In some configurations, *fontname* can be a system font name or a path to a TrueType font file. In all cases, a path to a BDF (Adobe Glyph Bitmap Distribution Format) font file can be used.

Note that characters available depend on the font, and many fonts do not cover multiple writing systems and languages, or symbols like arrows. Depending on the configuration, MAME may not automatically substitute characters from other fonts. Characters that are not available may be replaced with substitute glyphs (often rectangles).

The default is `default` (use the default font determined by the UI font provider module).

Example:

```
mame -uifont "Comic Sans MS"
```

-ui <*type*>

Specifies the type of UI to use, either `simple` or `cabinet`.

The default is cabinet (**-ui cabinet**).

Example:

```
mame -ui simple
```

-[no]confirm_quit

Display a Confirm Quit dialog to screen on exit, requiring one extra step to exit MAME.

The default is OFF (**-noconfirm_quit**).

Example:

```
mame pacman -confirm_quit
```

-[no]ui_mouse

Displays a mouse cursor when using the built-in MAME user interface.

The default is ON (**-ui_mouse**).

Example:

```
mame -ui_mouse
```

-language <*language*>

Specify a localization language found in the `languagepath` tree.

Example:

```
mame -language Japanese
```

-[no]nvram_save

Save the NVRAM contents when exiting machine emulation. By turning this off, you can retain your previous NVRAM contents as any current changes made will not be saved. Turning this option off will also unconditionally suppress the saving of .nv files associated with some types of software cartridges.

The default is ON (**-nvram_save**).

Example:

```
mame galaga88 -nonvram_save
```

-rtc <value>

Specifies a fixed baseline time to initialize the real-time clock (RTC) chips emulated by MAME. By default, MAME initializes RTC chips using your host computer's current system time.

The option accepts a 14-digit formatted date and time string for <*value*> in the format **YYYYMMDDhhmmss** (e.g., `20260709070000` for July 9, 2026, at 07:00:00).

Tip

If an input macro playback file (such as with **-playback**) is active, it takes absolute precedence to prevent input desynchronization. The **-rtc** command-line option will be ignored, and a warning will be displayed in the console.

5.1.30 Scripting Options

-autoboot_command *"<command>"*

Command string to execute after machine boot (in quotes " "). To issue a quote to the emulation, use """ in the string. Using **\n** will create a new line, issuing what was typed prior as a command.

This works only with systems that support natural keyboard mode.

Example:

```
mame c64 -autoboot_delay 5 -autoboot_command "load """$""",8,1\n"
```

-autoboot_delay *[n]*

Timer delay (in seconds) to trigger command execution on autoboot.

Example:

```
mame c64 -autoboot_delay 5 -autoboot_command "load """$""",8,1\n"
```

-autoboot_script / -script *[filename.lua]*

File containing scripting to execute after machine boot.

Example:

```
mame ibm5150 -autoboot_script myscript.lua
```

-[no]console

Enables emulator Lua Console window.

The default of OFF (**-noconsole**).

Example:

```
mame ibm5150 -console
```

-plugins

Enable the use of Lua Plugins.

The default is ON (**-plugins**).

Example:

```
mame apple2e -plugins
```

-plugin *[plugin shortname]*

A list of Lua Plugins to enable, comma separated.

Example:

```
mame alcon -plugin cheat,discord,autofire
```

-noplugin *[plugin shortname]*

A list of Lua Plugins to disable, comma separated.

Example:

```
mame alcon -noplugin cheat
```

5.1.31 HTTP Server Options

-[no]http

Enable HTTP server.

The default is OFF (**-nohttp**).

Example:

```
mame -http
```

-http_port <*port*>

Choose HTTP server port.

The default is `8080`.

Example:

```
mame apple2 -http -http_port 6502
```

-http_root <*rootfolder*>

Choose HTTP server document root.

The default is `web`.

Example:

```
mame apple2 -http -http_port 6502 -http_root C:\Users\me\appleweb\root
```

5.2 Windows-Specific Command-line Options

This section contains configuration options that are specific to the native (non-SDL) Windows version of MAME.

5.2.1 Performance options

-priority <*priority*>

Sets the thread priority for the MAME threads. By default the priority is left alone to guarantee proper cooperation with other applications. The valid range is -15 to 1, with 1 being the highest priority. The default is *0* (*NORMAL priority*).

-profile *[n]*

Enables profiling, specifying the stack depth of *[n]* to track.

5.2.2 Full screen options

-[no]triplebuffer / -[no]tb

Enables or disables "triple buffering". Normally, MAME just draws directly to the screen, without any fancy buffering. But with this option enabled, MAME creates three buffers to draw to, and cycles between them in order. It attempts to keep things flowing such that one buffer is currently displayed, the second buffer is waiting to be displayed, and the third buffer is being drawn to. **-triplebuffer** will override **-waitvsync**, if the buffer is successfully created. This option does not work with **-video gdi**. The default is OFF (**-notriplebuffer**).

-full_screen_brightness <*value*> / **-fsb** <*value*>

Controls the brightness, or black level, of the entire display. The standard value is 1.0. Lower values (down to 0.1) will produce a darkened display, while higher values (up to 2.0) will give a brighter display. Note that not all video cards have hardware to support this option. This option does not work with **-video gdi**. The default is *1.0*.

-full_screen_contrast <*value*> / **-fsc** <*value*>

Controls the contrast, or white level, of the entire display. The standard value is 1.0. Lower values (down to 0.1) will produce a dimmer display, while higher values (up to 2.0) will give a more saturated display. Note that not all video cards have hardware to support this option. This option does not work with **-video gdi**. The default is *1.0*.

-full_screen_gamma <*value*> / **-fsg** <*value*>

Controls the gamma, which produces a potentially nonlinear black to white ramp, for the entire display. The standard value is 1.0, which gives a linear ramp from black to white. Lower values (down to 0.1) will increase the nonlinearity toward black, while higher values (up to 3.0) will push the nonlinearity toward white. Note that not all video cards have hardware to support this option. This option does not work with **-video gdi**. The default is *1.0*.

5.2.3 Input device options

-[no]dual_lightgun / -[no]dual

Controls whether or not MAME attempts to track two lightguns that appear as a single mouse. This option requires the *lightgun option* to be on and the *lightgunprovider option* to be set to *win32*.

This option supports certain older dual lightgun setups that work by setting the mouse pointer location at the moment a lightgun trigger is activated. The primary and secondary triggers on the first lightgun correspond to the first and second mouse buttons, and the primary and secondary triggers on the second lightgun correspond to the third and fourth mouse buttons.

If you have multiple lightguns connected, you will probably just need to enable the *lightgun option*, use the default *lightgunprovider option* of *rawinput*, and configure each lightgun individually.

The default is OFF (**-nodual_lightgun**).

5.3 SDL-Specific Command-line Options

This section contains configuration options that are specific to any build supported by SDL (including Windows when built with SDL instead of native).

5.3.1 Performance Options

-[no]sdlvideofps

Enable output of benchmark data on the SDL video subsystem, including your system's video driver, X server (if applicable), and OpenGL stack in **-video opengl** mode.

5.3.2 Video Options

-[no]centerh

Center horizontally within the view area. Default is ON (**-centerh**).

-[no]centerv

Center vertically within the view area. Default is ON (**-centerv**).

5.3.3 Video Soft-Specific Options

-scalemode

Scale mode: none, async, yv12, yuy2, yv12x2, yuy2x2 (**-video soft** only). Default is *none*.

5.3.4 SDL Keyboard Mapping

-keymap

Enable keymap. Default is OFF (**-nokeymap**)

-keymap_file <*file*>

Keymap file name. Default is `keymap.dat`.

5.3.5 SDL Input Options

-enable_touch

Enable support for touch input. If this option is switched off, mouse input simulated from touch devices will be used instead. Default is OFF (**-noenable_touch**)

-sixaxis

Use special handling for PlayStation 3 SixAxis controllers. May cause undesirable behaviour with other controllers. Only affects the `sdljoy` joystick provider. Default is OFF (**-nosixaxis**)

-[no]dual_lightgun / -[no]dual

Controls whether or not MAME attempts to track two lightguns that appear as a single mouse. This option requires the *lightgun option* to be on and the *lightgunprovider option* to be set to *sdl*.

This option supports dual lightgun setups that work by setting the mouse pointer location at the moment a lightgun trigger is activated. The primary and secondary triggers on the first lightgun correspond to the first and second mouse buttons, and the primary and secondary triggers on the second lightgun correspond to the third and fourth mouse buttons.

The default is OFF (**-nodual_lightgun**).

5.3.6 SDL Lightgun Mapping

-lightgun_index1 <*name*>
-lightgun_index2 <*name*>
...
-lightgun_index8 <*name*>

Device name or ID mapped to a given lightgun slot.

5.3.7 SDL Low-level Driver Options

-videodriver <*driver*>

SDL video driver to use ('x11', 'directfb', ... or '*auto*' for SDL default)

-audiodriver <*driver*>

SDL audio driver to use ('alsa', 'arts', ... or '*auto*' for SDL default)

-gl_lib <*driver*>

Alternative **libGL.so** to use; '*auto*' for system default

5.4 Command-line Index

This is a complete index of all command-line options and verbs for MAME, suitable for quickly finding a given option.

5.4.1 Universal Command-line Options

This section contains configuration options that are applicable to *all* MAME configurations (including both SDL and Windows native).

Core Verbs

help
validate

Configuration Verbs

createconfig
showconfig
showusage

Frontend Verbs

listxml
listfull
listsource
listclones
listbrothers
listcrc
listroms
listbios
listsamples
verifyroms
verifysamples
romident
listdevices

listslots
listmedia
listsoftware
verifysoftware
getsoftlist
verifysoftlist

OSD-related Options

uimodekey
controller_map
background_input
uifontprovider
keyboardprovider
mouseprovider
lightgunprovider
joystickprovider
midiprovider
networkprovider

OSD CLI Verbs

listmidi
listnetwork

OSD Output Options

output

Configuration Options

noreadconfig

Core Search Path Options

homepath
rompath
hashpath
samplepath
artpath
ctrlrpath
inipath
fontpath
cheatpath
crosshairpath

pluginspath
languagepath
swpath

Core Output Directory Options

cfg_directory
nvram_directory
input_directory
state_directory
snapshot_directory
diff_directory
comment_directory

Core State/Playback Options

[no]rewind / rewind
rewind_capacity
state
[no]autosave
playback
[no]exit_after_playback
record
mngwrite
aviwrite
wavwrite
snapname
snapsize
snapview
[no]snapbilinear
statename
[no]burnin

Core Performance Options

[no]autoframeskip
frameskip
seconds_to_run
[no]throttle
[no]sleep
speed
[no]refreshspeed
numprocessors
bench
[no]lowlatency

Core Rotation Options

[no]rotate
[no]ror
[no]rol
[no]autoror
[no]autorol
[no]flipx
[no]flipy

Core Video Options

video
numscreens
[no]window
[no]maximize
[no]keepaspect
[no]waitvsync
[no]syncrefresh
prescale
[no]filter
[no]unevenstretch

Core Full Screen Options

[no]switchres

Core Per-Window Video Options

screen
aspect
resolution
view

Core Artwork Options

[no]artwork_crop
fallback_artwork
override_artwork
artwork_font

Core Screen Options

brightness
contrast
gamma
pause_brightness
effect

Core Vector Options

beam_width_min
beam_width_max
beam_intensity_weight
beam_dot_size
flicker

Core Video OpenGL Debugging Options

[no]gl_forcepow2texture
[no]gl_notexturerect
[no]gl_vbo
[no]gl_pbo

Core Video OpenGL GLSL Options

[no]gl_glsl
gl_glsl_filter
glsl_shader_mame[0-9]
glsl_shader_screen[0-9]

Core Sound Options

samplerate
[no]samples
volume
sound
audio_latency

Core Input Options

[no]coin_lockout
ctrlr
[no]mouse
[no]joystick
[no]lightgun
[no]multikeyboard
[no]multimouse
[no]steadykey
[no]ui_active
joystick_map
joystick_deadzone
joystick_saturation
joystick_threshold
[no]natural
[no]joystick_contradictory
coin_impulse

Core Input Automatic Enable Options

paddle_device
adstick_device
pedal_device
dial_device
trackball_device
lightgun_device
positional_device
mouse_device

Core Debugging Options

[no]verbose
[no]oslog
[no]log
[no]debug
debugger
debugscript
[no]update_in_pause
watchdog
debugger_host
debugger_port
debugger_font
debugger_font_size

Core Communication Options

comm_localhost
comm_localport
comm_remotehost
comm_remoteport
[no]comm_framesync

Core Misc Options

[no]drc
[no]drc_rwx
[no]drc_use_c
[no]drc_log_uml
[no]drc_log_native
bios
[no]cheat
[no]skip_gameinfo
uifont
ui
[no]confirm_quit
[no]ui_mouse
language
[no]nvram_save
rtc

Scripting Options

autoboot_command
autoboot_delay
autoboot_script
[no]console
[no]plugins
plugin
noplugin

HTTP Server Options

http
http_port
http_root

5.4.2 Windows-Specific Command-line Options

This section contains configuration options that are specific to the native (non-SDL) Windows version of MAME.

Windows Performance Options

priority
profile

Windows Full Screen Options

[no]triplebuffer
full_screen_brightness
full_screen_contrast
full_screen_gamma

Windows Input Device Options

[no]dual_lightgun

5.4.3 SDL-Specific Command-line Options

This section contains configuration options that are specific to any build supported by SDL (including Windows when built with SDL instead of native).

SDL Performance Options

[no]sdlvideofps

SDL Video Options

[no]centerh
[no]centerv

SDL Video Soft-Specific Options

scalemode

SDL Keyboard Mapping

keymap
keymap_file

SDL Input Options

[no]enable_touch
[no]sixaxis
[no]dual_lightgun

SDL Lightgun Mapping

lightgun_index

SDL Low-level Driver Options

videodriver
audiodriver
gl_lib

CHAPTER

SIX

PLUGINS

- *Introduction*
- *Using plugins*
- *Included plugins*

6.1 Introduction

MAME supports plugins that can provide additional functionality. Plugins have been written to communicate with external programs, play games automatically, display internal game structures like hitboxes, provide alternate user interfaces, and automatically test emulation. See *Lua Scripting Interface* for more information about MAME's Lua API.

6.2 Using plugins

To enable plugins, you need to turn on the *plugins option*, and make sure the *pluginspath option* includes the folder where your plugins are stored. You can set the plugins option in an INI file or on the command line. You can set the pluginspath option by selecting **Configure Options** from the system selection menu, then selecting **Configure Directories**, and then selecting **Plugins** (you can also set it in an INI file or on the command line).

Many plugins need to store settings and/or data. The *homepath option* sets the folder where plugins should save data (defaults to the working directory). You can change this by selecting **Configure Options** from the system selection menu, then selecting **Configure Directories**, and then selecting **Plugin Data**.

To turn individual plugins on or off, first make sure plugins are enabled, then select **Configure Options** from the system selection menu, and then select **Plugins**. You will need to completely exit MAME and start it again for changes to the enabled plugins to take effect. You can also use the *plugin option* on the command line, or change the settings in the **plugin.ini** file.

If an enabled plugin needs additional configuration, or if it needs to show information, a **Plugin Options** item will appear in the main menu (accessed by pressing **Tab** during emulation by default).

6.3 Included plugins

MAME includes several plugins providing useful functionality, and serving as sample code that you can use as a starting point when writing your own plugins.

6.3.1 Autofire Plugin

- *Introduction*
- *Autofire buttons settings*
- *Notes and potential pitfalls*

Introduction

The autofire plugin allows you to simulate repeatedly pressing an emulated button by holding down a key or button combination. This can help people with certain disabilities or injuries play shooting games, and may help reduce the risk of repetitive strain injuries (or keyboard damage).

To configure the autofire plugin, activate the main menu (press **Tab** during emulation by default), select **Plugin Options**, and then select **Autofire**. Configured autofire buttons for the current system are listed, along with their repetition rates and activation hotkeys (initially there will be no autofire buttons configured). Select an autofire button to change settings, or choose **Add autofire button** to set up a new autofire button. See *Autofire buttons settings* for details on setting up an autofire button. You can delete an autofire button by highlighting it in the menu and pressing the UI Clear key (Del/Delete/Forward Delete on the keyboard by default).

Autofire settings are saved in the **autofire** folder in the plugin data folder (see the *homepath option*). A file is created for each system with autofire buttons configured, named according to the system's short name (or ROM set name), with the extension `.cfg`. For example, autofire settings for Super-X will be saved in the file **superx.cfg** in the **autofire** folder in your plugin data folder. The autofire settings are stored in JSON format.

Autofire buttons settings

The options for adding a new autofire button or modifying an existing autofire button are the same.

Select **Input** to set the emulated button that you want to simulate pressing repeatedly. Only digital inputs are supported. Typically you'll set this to the primary fire button for shooting games. This is most often *P1 Button 1* or the equivalent for another player, but it might have a different name. On Konami's Gradius games, *P1 Button 2* is the primary fire button.

Select **Hotkey** to set the control (or combination of controls) you'll use to activate the autofire button. This can be any combination that MAME supports for activating a digital input.

On frames and **Off frames** are the number of consecutive emulated video frames that the emulated button will be held and released for, respectively. Adjust the value with the UI Left/Right keys, or click the arrows. Press the UI Clear key to reset the values to one frame. Lower values correspond to pressing the button at a faster rate. Depending on how fast the system reads inputs, you may need higher numbers than 1 for the system to recognise the button being released and pressed again (e.g. 2 on frames and 2 off frames works for Alcon). Experiment with different values to get the best effect.

When adding a new autofire button, there is a **Cancel** option that changes to **Create** after you set the input and hotkey. Select **Create** to finish creating the autofire button and return to the list of autofire buttons. The new autofire button

will be added at the end of the list. Press the UI Back key (Escape/Esc on the keyboard by default), or select **Cancel** before setting the input/hotkey, to return to the previous menu without creating the new autofire button.

When modifying an existing autofire button, select **Done** or press the UI Cancel key to return to the list of autofire buttons. Changes take effect immediately.

When modifying an existing autofire button, select **Delete** to delete the autofire button. The autofire button will be deleted immediately when you select **Delete** without requiring additional confirmation. You can also delete an autofire button by highlighting it in the list of autofire buttons and pressing the UI Clear key (Del/Delete/Forward Delete on the keyboard by default).

Notes and potential pitfalls

Autofire buttons act as if they're wired in parallel with MAME's regular controls. This means that if you set the activation hotkey for an autofire button to a button or key that's also assigned to one of the emulated inputs directly, you may get unexpected results. Using Gradius as an example:

- Suppose you set button 1 on your controller to fire, and set an autofire hotkey to button 1 as well. Holding the button down to shoot will not trigger the autofire effect: the button will never be released as you're holding the non-autofire button 1 down. This will also happen if you set a different button as autofire (say, button 3 in this case), and hold button 1 down while also pressing button 3.
- If you set button 3 on your controller to autofire and assign button 3 to powerup as well, you will trigger the powerup action every time you grab a powerup because the powerup button is also being held down along with the autofire button.

It is recommended that you choose control combinations for autofire hotkeys that are not assigned to any other emulated inputs in the system.

Autofire is not necessarily desirable in all situations. For example using autofire in Super-X with the blue "lightning" weapon equipped at high power levels will only produce a single beam, greatly reducing the weapon's effectiveness. The fire button must be held down to produce all beams. Some shooting games (e.g. Raiden Fighters) require the primary fire button to be held down for a charged special attack. This means it's often necessary to have a non-autofire input for the primary fire button assigned to play effectively.

6.3.2 Console Plugin

The console plugin provides functionality for MAME's interactive Lua console. It is not used directly. Use the *console option* to activate the interactive Lua console. See *Lua Scripting Interface* for more information about MAME's Lua API.

6.3.3 Data Plugin

The data plugin loads information from various external support files so it can be displayed in MAME. If the plugin is enabled, info is show in the **Infos** tab of the right-hand pane on the system and software selection menus. The info viewer can be shown by clicking the toolbar button on the system and software selection menus, or by choosing **External DAT View** from the main menu during emulation (this menu item will not appear if the data plugin is not enabled, or if no information is available for the emulated system).

To set the folders where the data plugin looks for supported files, choose **Configure Options** on the system selection menu, then choose **Configure Directories**, and then choose **DATs**. You can also set the `historypath` option in your **ui.ini** file.

Loading large data files like **history.xml** can take quite a while, so please be patient the first time you start MAME after updating or adding new data files.

The following files are supported:

history.xml
: From Gaming-History (formerly Arcade-History)

mameinfo.dat
: From MASH's MAMEINFO

messinfo.dat
: From progetto-SNAPS MESSINFO.dat

gameinit.dat
: From progetto-SNAPS GameInit.dat

command.dat
: from progetto-SNAPS Command.dat

score3.htm
: Top Scores from the MAME Action Replay Page

Japanese mameinfo.dat and command.dat
: From MAME E2J

sysinfo.dat
: From the defunct Progetto EMMA site

story.dat
: From the defunct MAMESCORE site

If you install hi2txt, the data plugin can also show high scores from non-volatile memory or saved by the *hiscore support plugin* for supported games.

Note that you can only use a single file of each type at a time. You cannot, for example, use the English and Japanese **mameinfo.dat** files simultaneously.

The data plugin creates a **history.db** file in the **data** folder in the plugin data folder (see the *homepath option*). This file stores the information from the support files in a format suitable for rapid loading. It uses the SQLite3 database format.

6.3.4 Discord Presence Plugin

The Discord presence plugin works with the Discord app for Windows, macOS or Linux to set your activity to show what you're doing in MAME. The activity is set to *In menu* if you're using the system or software selection menu, *Playing* if emulation is running, or *Paused* if emulation is paused. The details are set to show the system name and software description if applicable.

6.3.5 Dummy Test Plugin

This is a sample plugin that shows how to set necessary plugin metadata, register callbacks, and display a simple menu. It prints status messages, and it adds a **Dummy** option to the **Plugin Options** menu.

6.3.6 GDB Stub Plugin

The GDB stub plugin acts as a remote debugging server for the GNU debugger (GDB). This allows you to connect to MAME and debug supported systems using GDB. The plugin listens for connections on port 2159 on the IPv4 loopback address (127.0.0.1). Only Intel 80386 (i386) family processors are supported.

See the *debugger option* for another GDB remote debugging implementation with support for more CPUs and configurable listening ports.

6.3.7 Hiscore Support Plugin

The hiscore support plugin saves and restores high scores for games that did not originally save high scores in non-volatile memory. Note that this plugin modifies the contents of memory directly with no coordination with the emulated software, and hence changes behaviour. This may have undesirable effects, including broken gameplay or causing the emulated software to crash.

The plugin includes a **hiscore.dat** file that contains the information on how to save and restore high scores for supported systems. This file must be kept up-to-date when system definitions change in MAME.

High scores can be saved automatically either on exit, or a few seconds after they're updated in memory. To change the setting, activate the main menu (press **Tab** during emulation by default), select **Plugin Options**, and then select **Hiscore Support**. Change the **Save scores** option by highlighting it and using the UI Left/Right keys, or clicking the arrows.

High score data is saved in the **hiscore** folder in the plugin data folder (see the *homepath option*). A file with a name corresponding the system short name (or ROM set name) with the extension `.hi`. For example, high scores for the game Moon Cresta will be saved in the file **mooncrst.hi** in the **hiscore** folder in your plugin data folder. The settings for the hiscore support plugin are stored in the file **plugin.cfg** in the **hiscore** folder in the plugin data folder (this file is in JSON format).

6.3.8 Input Macro Plugin

- *Introduction*
- *Editing input macros*
- *Example macros*
  - *Raiden autofire*
  - *Track & Field sprint cheat*
  - *Street Fighter II Shoryuken*
  - *Virtua Cop Aim Off-Screen*
  - *Invasion: The Abductors Reload*

Introduction

The input macro plugin allows you to trigger a sequence of emulated input actions with a key or button combination. This can help people with disabilities or injuries that make some input sequences difficult. It can also be used as a way to cheat in games that require rapid sequences of inputs, like the running events in Track & Field, or the eating minigame in Daisu-Kiss.

To configure the input macro plugin, activate the main menu (press **Tab** during emulation by default), select **Plugin Options**, and then select **Input Macros**. Configured input macros for the current system are listed, along with their activation combinations (initially there will be no input macros configured). Select a macro to edit it, or choose **Add macro** to set up a new input macro. See *Editing input macros* for details on editing input macros. You can delete an input macro by highlighting it in the menu and pressing the UI Clear key (Del/Delete/Forward Delete on the keyboard by default). You can also delete a macro by selecting the macro to edit it and then selecting **Delete macro** from the menu.

Input macros are saved in the **inputmacro** folder in the plugin data folder (see the *homepath option*). A file is created for each system with input macros configured, named according to the system's short name (or ROM set name), with the extension `.cfg`. For example, input macros for Daisu-Kiss will be saved in the file **daiskiss.cfg** in the **inputmacro** folder in your plugin data folder. The input macros are stored in JSON format.

Editing input macros

The options for editing input macros are the same whether you're creating a new macro or editing an existing macro. Input macros consist of a sequence of *steps*. Each step optionally waits for a configurable delay, then activates one or more emulated inputs for a specified duration. You can choose what should happen if the activation combination is still held when the final step of the macro completes: the emulated inputs can be released, the final step can be prolonged, or the macro can loop back to any step in the sequence.

The settings in first section of the macro editing menu apply to the macro as a whole:

- The **Name** will be used in the list of input macros, so it helps to make it descriptive. Press the UI Select key (Return/Enter on the keyboard or the first button on the first joystick by default) to edit the current name, or press the UI Clear key to type a new name. Press the UI Select key before moving to another menu item to save the new name; press the UI Back key (Escape/Esc on the keyboard by default) to change discard the new name.
- Select **Activation combination** to set the control (or combination of controls) you want to use to activate the macro. Keep in mind that regular input assignments still apply, so you will probably want to use a combination that isn't being used for any other emulated input in the system.
- Set **On release** to specify what should happen if the activation combination is released before the macro completes. When set to *Stop immediately*, any emulated inputs activated by the macro will be released immediately, and no further steps will be processed; when set to *Complete macro*, the macro will continue to be processed until the end of the final step.
- Set **When held** to specify what should happen if the activation combination is held after the final step of the macro completes. When set to *Release*, any inputs activated by the macro will be released, and the macro will not be reactivated until the activation combination is released and pressed again; when set to *Prolong step <n>* where *<n>* is the number of the final step of the macro, the emulated inputs activated by the final step of the macro will remain active until the activation combination is released; when set to *Loop to step <n>* where *<n>* is a step number, macro processing will return to that step, including its delay, if the activation combination is held after the final step completes.

Each step has delay, duration and input settings:

- Set the **Delay** to the number of emulated video frame intervals to wait before activating the inputs for the step. During the delay, no emulated inputs will be activated by the macro. You can reset the setting to zero by pressing the UI Clear key.

- Set the **Duration** to the number of emulated video frame intervals to hold the emulated inputs for the step active before moving to the next step (or completing the macro in the case of the final step). You can reset the setting to one frame by pressing the UI Clear key.
- Set the **Input** settings to the emulated inputs to activate for the step. Only non-toggle digital inputs and non-wrapping analog inputs are supported. Select **Add input** to set multiple inputs for a step (this option will only appear after you set the first input for the initially created step when creating a new macro). If the step has multiple inputs, you can highlight an input on the menu and press the UI Clear key to delete it (all steps must have at least one input, so you can't delete the only input for a step).
- For analog inputs, you can set the desired **Value** that the input should have at each step. You can adjust the value by pressing the UI Left and UI Right keys, or type a numeric value and press the UI Select key to accept the new value. The value will be limited to the valid range for the input. Pressing the UI Clear key will set the default value for the input.
- If the macro has multiple steps, you can select **Delete step** to delete a step (this options does not appear if the macro only has a single step). Remember to check that the **On release** and **When held** settings are correct after deleting steps.

To add a step to the macro, highlight **Add step at position** (below the existing steps), use the UI Left/Right keys or click the arrows to set the position where you'd like to insert the new step, and then press the UI Select key (or double-click the menu item) to add the new step. You will be prompted to set the first input for the new step. Remember to check the **On release** and **When held** settings after adding steps. The **Add step at position** item will only appear after you set the first input for the initially created step when creating a new macro.

When creating a new macro, there is a **Cancel** option that changes to **Create** after you set the activation combination and the first input for the initially created step. Select **Create** to finish creating the macro and return to the list of input macros. The new macro will be added at the end of the list. Press the UI Back key, or select **Cancel** before setting the activation combination/input, to return to the previous menu without creating the new macro.

When editing an existing macro, select **Done** or press the UI Back key to return to the list of input macros. Changes take effect immediately.

When editing an existing macro, select **Delete macro** to delete the macro. The macro will be deleted immediately when you select **Delete macro** without requiring additional confirmation. You can also delete a macro by highlighting it in the list of macros and pressing the UI Clear key (Del/Delete/Forward Delete on the keyboard by default).

Example macros

Raiden autofire

This provides player 1 autofire functionality using the space bar. The same thing could be achieved using the *Autofire Plugin*, but this demonstrates a simple looping macro:

- **Name**: P1 Autofire
- **Activation combination**: Kbd Space
- **On release**: Stop immediately
- **When held**: Loop to step 2
- **Step 1**:
  - **Delay (frames)**: 0
  - **Duration (frames)**: 2
  - **Input 1**: P1 Button 1
- **Step 2**:

  - **Delay (frames)**: 4
  - **Duration (frames)**: 2
  - **Input 1**: P1 Button 1

The first step has no delay so that firing begins as soon as the space bar is pressed. The second step has sufficient delay to ensure the game recognises the button being pressed and released again. The second step is repeated as long as the space bar is held down.

Track & Field sprint cheat

This allows you to run in Konami Track & Field by holding a single button. This takes most of the skill (and fun) out of the game:

- **Name**: P1 Sprint
- **Activation combination**: Kbd Shift
- **On release**: Stop immediately
- **When held**: Loop to step 2
- **Step 1**:
  - **Delay (frames)**: 0
  - **Duration (frames)**: 1
  - **Input 1**: P1 Button 1
- **Step 2**:
  - **Delay (frames)**: 1
  - **Duration (frames)**: 1
  - **Input 1**: P1 Button 3
- **Step 3**:
  - **Delay (frames)**: 1
  - **Duration (frames)**: 1
  - **Input 1**: P1 Button 1

This macro rapidly alternates pressing buttons 1 and 3 – the pattern required to run in the game.

Street Fighter II Shoryuken

This macro allows you to perform a right-facing Shōryūken (Dragon Punch) by pressing a single key:

- **Name**: 1P Shoryuken LP
- **Activation combination**: Kbd M
- **On release**: Complete macro
- **When held**: Prolong step 6
- **Step 1**:
  - **Delay (frames)**: 0

  - **Duration (frames)**: 1
  - **Input 1**: P1 Right
- **Step 2**:
  - **Delay (frames)**: 1
  - **Duration (frames)**: 1
  - **Input 1**: P1 Down
- **Step 3**:
  - **Delay (frames)**: 0
  - **Duration (frames)**: 1
  - **Input 1**: P1 Down
  - **Input 2**: P1 Right
- **Step 4**:
  - **Delay (frames)**: 0
  - **Duration (frames)**: 1
  - **Input 1**: P1 Right
- **Step 5**:
  - **Delay (frames)**: 0
  - **Duration (frames)**: 1
  - **Input 1**: P1 Right
  - **Input 2**: P1 Jab Punch
- **Step 6**:
  - **Delay (frames)**: 0
  - **Duration (frames)**: 1
  - **Input 1**: P1 Jab Punch

This macro involves steps that activate multiple inputs. The macro will complete if the activation combination is released early, allowing you to tap the key momentarily to perform the move. Holding the activation sequence holds down the attack button.

Virtua Cop Aim Off-Screen

This macro allows you to hold a key to simulate pointing the lightgun away from the screen:

- **Name**: 1P Aim Off-Screen
- **Activation combination**: Kbd Alt
- **On release**: Stop immediately
- **When held**: Prolong step 1
- **Step 1**:
  - **Delay (frames)**: 0

  - **Duration (frames)**: 1
  - **Input 1**: Lightgun X
  - **Value (131-630)**: 131
  - **Input 2**: Lightgun Y
  - **Value (36-425)**: 36

This macro involves setting analog input values.

Invasion: The Abductors Reload

In order to reload, this game requires the lightgun to be pointed away from the screen for about a tenth of a second before pulling the trigger, and it requires the trigger to be held for a similar duration. This makes it unsuitable for use with the *Off-Screen Reload Helper plugin*. However, you can use a two-step macro to produce the required actions when pressing a key or button:

- **Name**: 1P Reload
- **Activation combination**: Kbd Shift
- **On release**: Complete macro
- **When held**: Prolong step 2
- **Step 1**:
  - **Delay (frames)**: 0
  - **Duration (frames)**: 6
  - **Input 1**: Lightgun X
  - **Value (0-255)**: 255
- **Step 2**:
  - **Delay (frames)**: 0
  - **Duration (frames)**: 6
  - **Input 1**: Lightgun X
  - **Value (0-255)**: 255
  - **Input 2**: P1 Button 1

This macro involves multiple steps, setting analog input values, activating multiple inputs simultaneously, and completing the macro even if the activation combination is only pressed momentarily.

6.3.9 Layout Plugin

When enabled, the layout plugin allows embedded Lua scripts in layout files to run. Built-in artwork for some machines and some external artwork packages can use Lua scripts to provide enhanced interactive features. See *MAME Layout Scripting* for an introduction to layout file scripting.

6.3.10 Off-Screen Reload Helper Plugin

- *Introduction*
- *Editing off-screen reload helpers*

Introduction

The off-screen reload helper plugin makes it easier to play games that require you to aim a lightgun away from the screen to perform actions (usually reloading). You can use a key or button combination to simulate aiming the lightgun away from the screen and pulling the trigger.

To configure the off-screen reload helper plugin, activate the main menu (press **Tab** during emulation by default), select **Plugin Options**, and then select **Off-Screen Reload Helper**. Configured off-screen reload helpers for the current system are listed, along with their activation sequences (initially there will be no off-screen reload helpers configured). Select a reload helper to edit it, or choose **Add reload helper** to set up a new off-screen reload helper. See *Editing off-screen reload helpers* for details on editing off-screen reload helpers. You can delete an off-screen reload helper by highlighting it in the menu and pressing the UI Clear key (Del/Delete/Forward Delete on the keyboard by default). You can also delete an off-screen reload helper by selecting it to edit it and then selecting **Delete reload helper** from the menu.

This plugin works with games that recognise the simple condition of pulling the lightgun trigger while one of the lightgun axes reports its minimum value. This includes, for example, Lethal Enforcers and Virtua cop. Some games, for example Invasion: The Abductors, have more complex requirements to trigger reloading. Although this plugin is not suitable for use with those games, you can use the *Input Macro plugin* to use a key or button combination to reload easily.

Off-screen reload helpers are saved in the **offscreenreload** folder in the plugin data folder (see the *homepath option*). A file is created for each system with off-screen reload helpers configured, named according to the system's short name (or ROM set name), with the extension `.cfg`. For example, input macros for the parent version of Lethal Enforcers will be saved in the file **lethalen.cfg** in the **offscreenreload** folder in your plugin data folder. The off-screen reload helpers are stored in JSON format.

Editing off-screen reload helpers

The options for editing off-screen reload helpers are the same whether you're creating a new reload helper or editing an existing one. An off-screen reload helper need an *axis input* (usually Y axis for the player's lightgun), and a *trigger input* (usually button 1 for the player, corresponding to the lightgun's trigger).

- Select **Reload combination** to set the control (or combination of controls) you want to use to activate the off-screen reload helper. You will probably want to use a combination that isn't being used for any other emulated input in the system.
- Select **Axis input** to set the axis input (often **Lightgun Y** for the first player, or **Lightgun 2 Y** for the second player). Only non-wrapping analog inputs are supported.
- Select **Trigger input** to set the trigger input (often **P1 Button 1** for the first player, or **P2 Button 1** for the second player). Only non-toggle digital inputs are supported. The plugin attempts to guess the trigger input when you change the axis input.

When creating a new off-screen reload helper, there is a **Cancel** option that changes to **Create** after you set the reload combination, axis input and trigger input. Select **Create** to finish creating the reload helper and return to the list of off-screen reload helpers. The new reload helper will be added at the end of the list. Press the UI Back key, or select **Cancel** before setting the reload sequence/inputs, to return to the previous menu without creating the new reload helper.

When editing an existing off-screen reload helper, select **Done** or press the UI Back key to return to the list of off-screen reload helpers. Changes take effect immediately.

When editing an existing off-screen reload helper, select **Delete reload helper** to delete it. The off-screen reload helper will be deleted immediately when you select **Delete reload helper** without requiring additional confirmation. You can also delete an off-screen reload helper by highlighting it in the list of reload helpers and pressing the UI Clear key (Del/Delete/Forward Delete on the keyboard by default).

6.3.11 Timecode Recorder Plugin

The timecode recorder plugin logs time codes to a text file in conjunction with creating an input recording file to assist people creating gameplay videos. The time code log file is *only* created when making an input recording. The time code log file has the same name as the input recording file with the extension **.timecode** appended. Use the *record* and *input_directory* options to create an input recording and specify the location for the output files.

By default, the plugin records a time code when you press the **F12** key on the keyboard while not pressing either **Shift** or **Alt** key. You can change this setting in the options menu for the plugin (choose **Plugin Options** from the main menu during emulation, and then choose **Timecode Recorder**).

Settings for the plugin are stored in JSON format in the file **plugin.cfg** in the **timecode** folder inside your plugin data folder (see the *homepath option*).

6.3.12 Game Play Timer Plugin

The timer plugin records the total time spent emulating each combination of a system and a software list item, as well as the number of times each combination has been launched. To see the statistics, bring up the main menu (press **Tab** during emulation by default), choose **Plugin Options**, and then choose **Timer**.

This plugin records wall clock time (the real time duration elapsed while emulation is running, according to the host OS) as well as emulated time. The elapsed wall clock time may be shorter than the elapsed emulated time if you turn off throttling or use MAME's "fast forward" feature, or it may be longer than the elapsed emulated time if you pause the emulation of if the emulation is too demanding to run at full speed.

The statistics are stored in the file **timer.db** in the **timer** folder inside your plugin data folder (see the *homepath option*). The file is a SQLite3 database.

CHAPTER

SEVEN

ADVANCED CONFIGURATION

7.1 Multiple Configuration Files

MAME has a very powerful configuration file system that can allow you to tweak settings on a per-game, per-system, or even per-monitor type basis, but requires careful thought about how you arrange your configuration files.

7.1.1 Order of Configurattion File Loading

1. The command line is parsed first, and any settings passed that way *will take precedence over anything in an INI file*.
2. `mame.ini` (or other platform INI; e.g. `mess.ini`) is parsed twice. The first pass may change various path settings, so the second pass is done to see if there is a valid configuration file at that new location (and if so, change settings using that file).
3. `debug.ini` if the debugger is enabled. This is an advanced config file, most people won't need to use it or be concerned by it.
4. Screen orientation INI file (either `horizont.ini` or `vertical.ini`). For example Pac-Man has a vertical screen, so it loads `vertical.ini`, while Street Fighter Alpha uses a horizontal screen, so it loads `horizont.ini`.

   Systems with no monitors, multiple monitors with different orientations, or monitors connected to slot devices will usually load `horizont.ini`.
5. Monitor type INI file (`vector.ini` for vector monitors, `raster.ini` for CRT raster monitors, or `lcd.ini` for LCD/EL/plasma matrix monitors). Pac-Man and Street Fighter Alpha use raster CRTs, so `raster.ini` is loaded here, while Tempest uses a vector monitor, so `vector.ini` is loaded here.

   For systems that have multiple monitor types, such as House Mannequin with its CRT raster monitor and dual LCD matrix monitors, the INI file relevant to the first monitor is used (`raster.ini` in this case). Systems without monitors or with other kinds of monitors will not load an INI file for this step.
6. Driver source file INI file. MAME will attempt to load `source/<`*sourcefile*`>.ini` where <*sourcefile*> is the base name of the source code file where the system driver is defined. A system's source file can be found using **mame -listsource <pattern>** at the command line.

   For instance, Banpresto's Sailor Moon, Atlus's Dodonpachi, and Nihon System's Dangun Feveron all run on similar hardware and are defined in the `cave.cpp` source file, so they will all load `source/cave.ini` at this step.
7. BIOS set INI file (if applicable). For example The Last Soldier uses the Neo-Geo MVS BIOS, so it will load `neogeo.ini`. Systems that don't use a BIOS set won't load an INI file for this step.

8. Parent system INI file. For example The Last Soldier is a clone of The Last Blade / Bakumatsu Roman - Gekka no Kenshi, so it will load `lastblad.ini`. Parent systems will not load an INI file for this step.
9. System INI file. Using the previous example, The Last Soldier will load `lastsold.ini`.

7.1.2 Examples of Config Loading Order

- Brix, which is a clone of Zzyzzyxx. (**mame brix**)
  1. Command line
  2. `mame.ini` (global)
  3. (debugger not enabled, no extra INI file loaded)
  4. `vertical.ini` (screen orientation)
  5. `raster.ini` (monitor type)
  6. `source/jack.ini` (driver source file)
  7. (no BIOS set)
  8. `zzyzzyxx.ini` (parent system)
  9. `brix.ini` (system)
- Super Street Fighter 2 Turbo (**mame ssf2t**)
  1. Command line
  2. `mame.ini` (global)
  3. (debugger not enabled, no extra INI file loaded)
  4. `horizont.ini` (screen orientation)
  5. `raster.ini` (monitor type)
  6. `source/cps2.ini` (driver source file)
  7. (no BIOS set)
  8. (no parent system)
  9. `ssf2t.ini` (system)
- Final Arch (**mame finlarch**)
  1. Command line
  2. `mame.ini` (global)
  3. (debugger not enabled, no extra INI file loaded)
  4. `horizont.ini` (screen orientation)
  5. `raster.ini` (monitor type)
  6. `source/stv.ini` (driver source file)
  7. `stvbios.ini` (BIOS set)
  8. `smleague.ini` (parent system)
  9. `finlarch.ini` (system)

Remember command line argument take precedence over all else!

7.1.3 Tricks to Make Life Easier

Some users may have a wall-mounted or otherwise rotatable monitor, and may wish to actually play vertical games with the rotated display. The easiest way to accomplish this is to put your rotation modifiers into `vertical.ini`, where they will only affect vertical games.

7.1.4 Multiple Search Folders

If you include paths to multiple folders in your *inipath setting*, MAME will search for each INI file in each of the specified folders in the order they are listed. If a file with the same name is found in multiple folders, settings from files found earlier have precedence over settings in files found later. MAME always saves INI files to the first folder specified in your inipath setting.

7.2 MAME Path Handling

MAME has a specific order it uses when checking for user files such as ROM sets and cheat files.

7.2.1 Order of Path Loading

Let's use an example of the cheat file for After Burner 2 for Sega Genesis/MegaDrive (aburner2 in the megadrive softlist), and your cheatpath is set to "cheat" (as per the default) -- this is how MAME will search for that cheat file:

1. `cheat/megadriv/aburner2.xml`
2. `cheat/megadriv.zip` -> `aburner2.xml` Notice that it checks for a .ZIP file first before a .7Z file.
3. `cheat/megadriv.zip` -> `<arbitrary path>/aburner2.xml` It will look for the first (if any) `aburner2.xml` file it can find inside that zip, no matter what the path is.
4. `cheat.zip` -> `megadriv/aburner2.xml` Now it is specifically looking for the file and folder combination, but inside the cheat.zip file.
5. `cheat.zip` -> `<arbitrary path>/megadriv/aburner2.xml` Like before, except looking for the first (if any) `aburner2.xml` inside a `megadriv` folder inside the zip.
6. `cheat/megadriv.7z` -> `aburner2.xml` Now we start checking 7ZIP files.
7. `cheat/megadriv.7z` -> `<arbitrary path>/aburner2.xml`
8. `cheat.7z` -> `megadriv/aburner2.xml`
9. `cheat.7z` -> `<arbitrary path>/megadriv/aburner2.xml` Similar to zip, except now 7ZIP files.

[todo: ROM set loading is slightly more complicated, adding CRC. Get that documented in the next day or two.]

7.3 Shifter Toggle Disable

This is an advanced feature for alternative shifter handling for certain older arcade machines such as Spy Hunter and Outrun that used a two-way toggle switch for the shifter. By default, the shifter is treated as a toggle switch. One press of the mapped control for the shifter will switch it from low to high, and another will switch it back. This may not be ideal if you have access to a physical shifter that works identically to how the original machines did. (The input is on when in one gear, the input is off otherwise)

Note that this feature will *not* help controller users and will not help with games that have more than two shifter states (e.g. five gears in modern racing games)

This feature is not exposed through the graphical user interface in any way, as it is an extremely advanced tweak intended explicitly for people who have this specific need, have the hardware to take advantage of it, and the knowledge to use it correctly.

7.3.1 Disabling and Enabling Shifter Toggle

This example will use the game Spy Hunter (set *spyhunt*) to demonstrate the exact change needed:

You will need to manually edit the game .CFG file in the CFG folder (e.g. `spyhunt.cfg`)

Start by loading MAME with the game in question. In our case, that will be **mame spyhunt**.

Set up the controls as you would please, including mapping the shifter. Exit MAME, open the .cfg file in your text editor of choice.

Inside the `spyhunt.cfg` file, you will find the following for the input. The actual input code in the middle can and will vary depending on the controller number and what input you have mapped.

```
<port tag=":ssio:IP0" type="P1_BUTTON2" mask="16" defvalue="16">
    <newseq type="standard">
        JOYCODE_1_RYAXIS_NEG_SWITCH OR JOYCODE_1_RYAXIS_POS_SWITCH
    </newseq>
</port>
```

The line you need to edit will be the port line defining the actual input. For Spy Hunter, that's going to be *P1_BUTTON2*. Add **toggle="no"** to the end of the tag, like follows:

```
<port tag=":ssio:IP0" type="P1_BUTTON2" mask="16" defvalue="16" toggle="no">
    <newseq type="standard">
        JOYCODE_1_RYAXIS_NEG_SWITCH OR JOYCODE_1_RYAXIS_POS_SWITCH
    </newseq>
</port>
```

Save and exit. To disable this, simply remove the **toggle="no"** from each desired .CFG input.

7.4 BGFX Effects for (nearly) Everyone

- *Introduction*
- *Resolution and Aspect Ratio*
- *Getting Started with BGFX*
- *Configuration Settings*
- *Tweaking BGFX HLSL Settings inside MAME*
- *Using the included pillarbox filters*

7.4.1 Introduction

By default, MAME outputs an idealized version of the video as it would be on the way to the arcade cabinet's monitor, with minimal modification of the output (primarily to stretch the game image back to the aspect ratio the monitor would traditionally have, usually 4:3). This works well, but misses some of the nostalgia factor. Arcade monitors were never ideal, even in perfect condition, and the nature of a CRT display distorts that image in ways that change the appearance significantly.

Modern LCD monitors simply do not look the same, and even computer CRT monitors cannot match the look of an arcade monitor without help.

That's where the new BGFX renderer with HLSL comes into the picture.

HLSL simulates most of the effects that a CRT arcade monitor has on the video, making the result look a lot more authentic. However, HLSL requires some effort on the user's part: the settings you use are going to be tailored to your PC's system specs, and especially the monitor you're using. Additionally, there were hundreds of thousands of monitors out there in arcades. Each was tuned and maintained differently, meaning there is no one correct appearance to judge by either. Basic guidelines will be provided here to help you, but you may also wish to ask for opinions on popular MAME-centric forums.

7.4.2 Resolution and Aspect Ratio

Resolution is a very important subject for HLSL settings. You will want MAME to be using the native resolution of your monitor to avoid additional distortion and lag caused by your monitor upscaling the display image.

While most arcade machines used a 4:3 ratio display (or 3:4 for vertically oriented monitors like Pac-Man), it's difficult to find a consumer display that is 4:3 at this point. The good news is that that extra space on the sides isn't wasted. Many arcade cabinets used bezel artwork around the main display, and should you have the necessary artwork files, MAME will display that artwork. Turn the **Zoom to Screen Area** setting in the video options menu to scale and crop the artwork so the emulated screen fills your display in one direction.

Some older LCD displays used a native resolution of 1280×1024 and were a 5:4 aspect ratio. There's not enough extra space to display artwork, and you'll end up with some very slight pillarboxing, but the results will be still be good and on-par with a 4:3 monitor.

7.4.3 Getting Started with BGFX

You will need to have followed the initial MAME setup instructions elsewhere in this manual before beginning. Official MAME distributions include BGFX as of MAME 0.172, so you don't need to download any additional files.

Open your `mame.ini` file in your text editor of choice (e.g. Notepad), and make sure the following options are set correctly:

- `video bgfx`

Now, you may want to take a moment to look below at the Configuration Settings section to see how to set up these next options.

As explained in *Order of Configurattion File Loading*, MAME has a order in which it processes INI files. The BGFX settings can be edited in `mame.ini`, but to take full advantage of the power of MAME's configuration files, you'll want to copy the BGFX settings from `mame.ini` to one of the other configuration files and make changes there.

In particular, you will want the `bgfx_screen_chains` to be specific to each game.

Save your INI file(s) and you're ready to begin.

7.4.4 Configuration Settings

bgfx_path
: This is where your BGFX shader files are stored. By default, this will be the *bgfx* folder in your MAME installation folder.

bgfx_backend
: Selects a rendering backend for BGFX to use. Possible choices include `auto`, `d3d9`, `d3d11`, `d3d12`, `opengl`, `gles`, `metal`, and `vulkan`. The default is `auto`, which will let MAME choose the best selection for you.

  - `d3d9` -- Direct3D 9.0 Renderer (May require the DirectX End-User Runtime to be installed)
  - `d3d11` -- Direct3D 11.0 Renderer
  - `d3d12` -- Direct3D 12.0 Renderer
  - `opengl` -- OpenGL Renderer (Requires OpenGL drivers, may work better on some video cards, supported on Linux and macOS)
  - `gles` -- OpenGL ES Renderer (Supported with some low-power GPUs)
  - `metal` -- Apple Metal Graphics API (Requires macOS 10.11 El Capitan or newer)
  - `vulkan` -- Vulkan Renderer (Requires Windows or Linux with compatible GPU drivers)
  - `auto` -- MAME will automatically choose the best selection for you.

bgfx_debug
: Enables BGFX debugging features. Most users will not need to use this.

bgfx_vectorcrt
: Enables the persistent BGFX vector CRT renderer for vector games. It uses HDR phosphor accumulation, Gaussian beam rendering, scan-order variation, and bloom. The default is disabled. This option has no effect when using a video backend other than BGFX or when running a raster game.

bgfx_screen_chains
: This dictates how to handle BGFX rendering on a per-display basis. Possible choices include `hlsl`, `unfiltered`, and `default`.

  - `default` -- **default** bilinear filtered output
  - `unfiltered` -- nearest neighbor sampled output

- `hlsl` -- display simulation through shaders
- `crt-geom` -- lightweight CRT simulation
- `crt-geom-deluxe` -- more detailed CRT simulation
- `lcd-grid` -- LCD matrix simulation
- `depixelize` -- vectorizing upscaler for pixel art (Kopf & Lischinski, "Depixelizing Pixel Art"); smooth contours, best on clean low-resolution graphics

We make a distinction between emulated screens (which we'll call a *screen*) and output windows or monitors (which we'll call a *window*, set by the `-numscreens` option) here. Use colons (:) to separate windows, and commas (,) to separate screens in the `-bgfx_screen_chains` setting value.

For the simple single window, single screen case, such as Pac-Man on one physical PC monitor, you can specify one entry like:

```
bgfx_screen_chains hlsl
```

Things get only slightly more complicated when we get to multiple windows and multiple screens.

On a single window, multiple screen game, such as Darius on one physical PC monitor, specify screen chains (one per window) like:

```
bgfx_screen_chains hlsl,hlsl,hlsl
```

This also works with single screen games where you are mirroring the output to more than one physical display. For instance, you could set up Pac-Man to have one unfiltered output for use with video broadcasting while a second display is set up HLSL for playing on.

On a multiple window, multiple screen game, such as Darius on three physical PC monitors, specify multiple entries (one per window) like:

```
bgfx_screen_chains hlsl:hlsl:hlsl
```

Another example game would be Taisen Hot Gimmick, which used two CRTs to show individual player hands to just that player. If using two windows (two physical displays):

```
bgfx_screen_chains hlsl:hlsl
```

One more special case is that Nichibutsu had a special cocktail mahjong cabinet that used a CRT in the middle along with two LCD displays to show each player their hand. We would want the LCDs to be unfiltered and untouched as they were, while the CRT would be improved through HLSL. Since we want to give each player their own full screen display (two physical monitors) along with the LCD, we'll go with:

```
-numscreens 2 -view0 "Player 1" -view1 "Player 2" -video bgfx -bgfx_screen_chains␣
↪hlsl,unfiltered:hlsl,unfiltered
```

This sets up the view for each display respectively, keeping HLSL effect on the CRT for each window (physical display) while going unfiltered for the LCD screens.

If using only one window (one display), keep in mind the game still has three screens, so we would use:

```
bgfx_screen_chains hlsl,unfiltered,unfiltered
```

Note that the commas are on the outside edges, and any colons are in the middle.

bgfx_shadow_mask
This specifies the shadow mask effect PNG file. By default this is **slot-mask.png**.

7.4.5 Tweaking BGFX HLSL Settings inside MAME

Start by loading MAME with the game of your choice (e.g. **mame pacman**).

The tilde key (**~**) brings up the on-screen display options. Use up and down to go through the various settings, while left and right will allow you to change that setting. Results will be shown in real time as you're changing these settings.

Note that settings are individually changeable on a per-screen basis.

BGFX slider settings are saved per-system in CFG files. If the `bgfx_screen_chains` setting has been set (either in an INI file or on the command line), it will set the initial effects. If the `bgfx_screen_chains` setting has not been set, MAME will use the effects you chose the last time you ran the system.

7.4.6 Using the included pillarbox filters

MAME includes example BGFX shaders and layouts for filling unused space on a 16:9 widescreen display with a blurred version of the emulated video. The all the necessary files are included, and just need to be enabled.

For systems using 4:3 horizontal monitors, use these options:

```
-override_artwork bgfx/border_blur -view Horizontal -bgfx_screen_chains crt-geom,
↪pillarbox_left_horizontal,pillarbox_right_horizontal
```

For systems using 3:4 vertical monitors, use these options:

```
-override_artwork bgfx/border_blur -view Vertical -bgfx_screen_chains crt-geom,pillarbox_
↪left_vertical,pillarbox_right_vertical
```

- You can use a different setting in place of `crt-geom` for the effect to apply to the primary screen image in the centre (e.g. `default`, `hlsl` or `lcd-grid`).
- If you've previously changed the view for the system in MAME, the correct pillarboxed view will not be selected by default. Use the video options menu to select the correct view.
- You can add these settings to an INI file to have them apply to certain systems automatically (e.g. **horizont.ini** or **vertical.ini**, or the INI file for a specific system).

7.5 HLSL Effects for Windows

By default, MAME outputs an idealized version of the video as it would be on the way to the arcade cabinet's monitor, with minimal modification of the output (primarily to stretch the game image back to the aspect ratio the monitor would traditionally have, usually 4:3) -- this works well, but misses some of the nostalgia factor. Arcade monitors were never ideal, even in perfect condition, and the nature of a CRT display distorts that image in ways that change the appearance significantly.

Modern LCD monitors simply do not look the same, and even computer CRT monitors cannot match the look of an arcade monitor without help.

That's where HLSL comes into the picture.

HLSL simulates most of the effects that a CRT arcade monitor has on the video, making the result look a lot more authentic. However, HLSL requires some effort on the user's part: the settings you use are going to be tailored to your PC's system specs, and especially the monitor you're using. Additionally, there were hundreds of thousands of monitors out there in arcades. Each was tuned and maintained differently, meaning there is no one correct appearance to judge by either. Basic guidelines will be provided here to help you, but you may also wish to ask for opinions on popular MAME-centric forums.

7.5.1 Resolution and Aspect Ratio

Resolution is a very important subject for HLSL settings. You will want MAME to be using the native resolution of your monitor to avoid additional distortion and lag created by your monitor upscaling the display image.

While most arcade machines used a 4:3 ratio display (or 3:4 for vertically oriented monitors like Pac-Man), it's difficult to find a consumer display that is 4:3 at this point. The good news is that that extra space on the sides isn't wasted. Many arcade cabinets used bezel artwork around the main display, and should you have the necessary artwork files, MAME will display that artwork. Turn the artwork view to Cropped for best results.

Some older LCD displays used a native resolution of 1280x1024 and were a 5:4 aspect ratio. There's not enough extra space to display artwork, and you'll end up with some very slight pillarboxing, but the results will be still be good and on-par with a 4:3 monitor.

7.5.2 Getting Started with HLSL

You will need to have followed the initial MAME setup instructions elsewhere in this manual before beginning. Official MAME distributions include HLSL by default, so you don't need to download any additional files.

Open your `mame.ini` in your text editor of choice (e.g. Notepad), and make sure the following options are set correctly:

- **video d3d**
- **filter 0**

The former is required because HLSL requires Direct3D support. The latter turns off extra filtering that interferes with HLSL output.

Lastly, one more edit will turn HLSL on:

- **hlsl_enable 1**

Save the .INI file and you're ready to begin.

Several presets have been included in the INI folder with MAME, allowing for good quick starting points for Nintendo Game Boy, Nintendo Game Boy Advance, Raster, and Vector monitor settings.

7.5.3 Tweaking HLSL Settings inside MAME

For multiple, complicated to explain reasons, HLSL settings are no longer saved when you exit MAME. This means that while tweaking settings is a little more work on your part, the results will always come out as expected.

Start by loading MAME with the game of your choice (e.g. **mame pacman**)

The tilde key (**~**) brings up the on-screen display options. Use up and down to go through the various settings, while left and right will allow you to change that setting. Results will be shown in real time as you're changing these settings.

Once you've found settings you like, write the numbers down on a notepad and exit MAME.

7.5.4 Configuration Editing

As referenced in *Order of Configurattion File Loading*, MAME has a order in which it processes INI files. The HLSL settings can be edited in `mame.ini`, but to take full advantage of the power of MAME's config files, you'll want to copy the HLSL settings from mame.ini to one of the other config files and make changes there.

For instance, once you've found HLSL settings you think are appropriate for Neo Geo games, you can put those settings into `neogeo.ini` so that all Neo-Geo games will be able to take advantage of those settings without needing to add it to every game INI manually.

7.5.5 Configuration Settings

hlslpath

This is where your HLSL files are stored. By default, this will be the HLSL folder in your MAME installation.

hlsl_snap_width
hlsl_snap_height

Sets the resolution that Alt+F12 HLSL screenshots are output at.

shadow_mask_alpha (*Shadow Mask Amount*)

This defines how strong the effect of the shadowmask is. Acceptable range is from 0 to 1, where 0 will show no shadowmask effect, 1 will be a completely opaque shadowmask, and 0.5 will be 50% transparent.

shadow_mask_tile_mode (*Shadow Mask Tile Mode*)

This defines whether the shadowmask should be tiled based on the screen resolution of your monitor or based on the source resolution of the emulated system. Valid values are 0 for *Screen* mode and 1 for *Source* mode.

shadow_mask_texture
shadow_mask_x_count (*Shadow Mask Pixel X Count*)
shadow_mask_y_count (*Shadow Mask Pixel Y Count*)
shadow_mask_usize (*Shadow Mask U Size*)
shadow_mask_vsize (*Shadow Mask V Size*)
shadow_mask_x_count (*Shadow Mask U Offset*)
shadow_mask_y_count (*Shadow Mask V Offset*)

These settings need to be set in unison with one another. In particular, **shadow_mask_texture** sets rules for how you need to set the other options.

shadow_mask_texture sets the texture of the shadowmask effect. Three shadowmasks are included with MAME: *aperture-grille.png*, *shadow-mask.png*, and *slot-mask.png*

shadow_mask_usize and **shadow_mask_vsize** define the used size of the shadow_mask_texture in percentage, staring at the top-left corner. The means for a texture with the actual size of 24x24 pixel and an u/v size of 0.5,0.5 the top-left 12x12 pixel will be used. Keep in mind to define an u/v size that makes is possible to tile the texture without gaps or glitches. 0.5,0.5 is fine for any shadowmask texture that is included with MAME.

shadow_mask_x_count and **shadow_mask_y_count** define how many screen pixel should be used to display the u/v sized texture. e.g. if you use the example from above and define a x/y count of 12,12 every pixel of the texture will be displayed 1:1 on the screen, if you define a x/y count of 24,24 the texture will be displayed twice as large.

example settings for **shadow_mask.png**:

shadow_mask_texture shadow-mask.png
shadow_mask_x_count 12
shadow_mask_y_count 6 or 12
shadow_mask_usize 0.5
shadow_mask_vsize 0.5

example settings for **slot-mask.png**:

shadow_mask_texture slot-mask.png
shadow_mask_x_count 12
shadow_mask_y_count 8 or 16
shadow_mask_usize 0.5
shadow_mask_vsize 0.5

example settings for **aperture-grille**:

shadow_mask_texture aperture-grille.png
shadow_mask_x_count 12
shadow_mask_y_count 12 or any
shadow_mask_usize 0.5
shadow_mask_vsize 0.5

shadow_mask_uoffset and **shadow_mask_voffset** can be used to tweak the alignment of the final shadowmask in subpixel range. Range is from -1.00 to 1.00, where 0.5 moves the shadowmask by 50 percent of the u/v sized texture.

distortion (*Quadric Distortion Amount*)

This setting determines strength of the quadric distortion of the screen image.

cubic_distortion (*Cubic Distortion Amount*)

This setting determines strength of the cubic distortion of the screen image.

Both distortion factors can be negative to compensate each other. e.g. distortion 0.5 and cubic_distortion -0.5

distort_corner (*Distorted Corner Amount*)

This setting determines strength of distortion of the screen corners, which does not affect the distortion of screen image itself.

round_corner (*Rounded Corner Amount*)

The corners of the display can be rounded off through the use of this setting.

smooth_border (*Smooth Border Amount*)

Sets a smoothened/blurred border around the edges of the screen.

reflection (*Reflection Amount*)

If set above 0, this creates a white reflective blotch on the display. By default, this is put in the upper right corner of the display. By editing the POST.FX file's GetSpotAddend section, you can change the location. Range is from 0.00 to 1.00.

vignetting (*Vignetting Amount*)

When set above 0, will increasingly darken the outer edges of the display in a pseudo-3D effect. Range is from 0.00 to 1.00.

scanline_alpha (*Scanline Amount*)

This defines how strong the effect of the scanlines are. Acceptable range is from 0 to 1, where 0 will show no scanline effect, 1 will be a completely black line, and 0.5 will be 50% transparent. Note that arcade monitors did not have completely black scanlines.

scanline_size (*Overall Scanline Scale*)

The overall spacing of the scanlines is set with this option. Setting it at 1 represents consistent alternating spacing between display lines and scanlines.

scanline_height (*Individual Scanline Scale*)

This determines the overall size of each scanline. Setting lower than 1 makes them thinner, larger than 1 makes them thicker.

scanline_variation (*Scanline Variation*)

This affects the size of each scanline depending on its brightness. Brighter scanlines will be thicker than darker scanline. Acceptable range is from 0 to 2.0, with the default being 1.0. At 0.0 all scanlines will have the same size independent of their brightness.

scanline_bright_scale (*Scanline Brightness Scale*)

Specifies how bright the scanlines are. Larger than 1 will make them brighter, lower will make them dimmer. Setting to 0 will make scanlines disappear entirely.

scanline_bright_offset (*Scanline Brightness Offset*)

This will give scanlines a glow/overdrive effect, softening and smoothing the top and bottom of each scanline.

scanline_jitter (*Scanline Jitter Amount*)

Specifies the wobble or jitter of the scanlines, causing them to jitter on the monitor. Warning: Higher settings may hurt your eyes.

hum_bar_alpha (*Hum Bar Amount*)

Defines the strength of the hum bar effect.

defocus (*Defocus*)

This option will defocus the display, blurring individual pixels like an extremely badly maintained monitor. Specify as X,Y values (e.g. **defocus 1,1**)

converge_x (*Linear Convergence X, RGB*)
converge_y (*Linear Convergence Y, RGB*)
radial_converge_x (*Radial Convergence X, RGB*)
radial_converge_y (*Radial Convergence Y, RGB*)

Adjust the convergence of the red, green, and blue channels in a given direction. Many badly maintained monitors with bad convergence would bleed colored ghosting off-center of a sprite, and this simulates that.

red_ratio (*Red Output from RGB*)
grn_ratio (*Green Output from RGB*)
blu_ratio (*Blue Output from RGB*)

Defines a 3x3 matrix that is multiplied with the RGB signals to simulate color channel interference. For instance, a green channel of (0.100, 1.000, 0.250) is weakened 10% by the red channel and strengthened 25% through the blue channel.

offset (*Signal Offset*)

Strengthen or weakens the current color value of a given channel. For instance, a red signal of 0.5 with an offset of 0.2 will be raised to 0.7

scale (*Signal Scale*)

Applies scaling to the current color value of the channel. For instance, a red signal of 0.5 with a scale of 1.1 will result in a red signal of 0.55

power (*Signal Exponent, RGB*)

Exponentiate the current color value of the channel, also called gamma. For instance, a red signal of 0.5 with red power of 2 will result in a red signal of 0.25

This setting also can be used to adjust line thickness in vector games.

floor (*Signal Floor, RGB*)

Sets the absolute minimum color value of a channel. For instance, a red signal of 0.0 (total absence of red) with a red floor of 0.2 will result in a red signal of 0.2

Typically used in conjunction with artwork turned on to make the screen have a dim raster glow.

phosphor_life (*Phosphor Persistence, RGB*)

How long the color channel stays on the screen, also called phosphor ghosting. 0 gives absolutely no ghost effect, and 1 will leave a contrail behind that is only overwritten by a higher color value.

This also affects vector games quite a bit.

saturation (*Color Saturation*)

Color saturation can be adjusted here.

bloom_blend_mode (*Bloom Blend Mode*)

Determines the mode of the bloom effect. Valid values are 0 for *Brighten* mode and 1 for *Darken* mode, last is only useful for systems with STN LCD.

bloom_scale (*Bloom Scale*)

Determines the intensity of bloom effect. Arcade CRT displays had a tendency towards bloom, where bright colors could bleed out into neighboring pixels. This effect is extremely graphics card intensive, and can be turned completely off to save GPU power by setting it to 0

bloom_overdrive (*Bloom Overdrive, RGB*)

Sets a RGB color, separated by commas, that has reached the brightest possible color and will be overdriven to white. This is only useful on color raster, color LCD, or color vector games.

bloom_lvl0_weight (*Bloom Level 0 Scale*)
bloom_lvl1_weight (*Bloom Level 1 Scale*)
. . . .
bloom_lvl7_weight (*Bloom Level 7 Scale*)
bloom_lvl8_weight (*Bloom Level 8 Scale*)

These define the bloom effect. Range is from 0.00 to 1.00. If used carefully in conjunction with phosphor_life, glowing/ghosting for moving objects can be achieved.

hlsl_write

Enables writing of an uncompressed AVI video with the HLSL effects included with set to *1*. This uses a

massive amount of disk space very quickly, so a large HD with fast write speeds is highly recommended. Default is *0*, which is off.

Suggested defaults for raster-based games:

bloom_lvl0_weight 1.00	Bloom level 0 weight	Full-size target.
bloom_lvl1_weight 0.64	Bloom level 1 weight	1/4 smaller that level 0 target
bloom_lvl2_weight 0.32	Bloom level 2 weight	1/4 smaller that level 1 target
bloom_lvl3_weight 0.16	Bloom level 3 weight	1/4 smaller that level 2 target
bloom_lvl4_weight 0.08	Bloom level 4 weight	1/4 smaller that level 3 target
bloom_lvl5_weight 0.06	Bloom level 5 weight	1/4 smaller that level 4 target
bloom_lvl6_weight 0.04	Bloom level 6 weight	1/4 smaller that level 5 target
bloom_lvl7_weight 0.02	Bloom level 7 weight	1/4 smaller that level 6 target
bloom_lvl8_weight 0.01	Bloom level 8 weight	1/4 smaller that level 7 target

7.5.6 Vector Games

HLSL effects can also be used with vector games. Due to a wide variance of vector settings to optimize for each individual game, it is heavily suggested you add these to per-game INI files (e.g. tempest.ini)

Shadowmasks were only present on color vector games, and should not be used on monochrome vector games. Additionally, vector games did not use scanlines, so that should also be turned off.

Open your INI file in your text editor of choice (e.g. Notepad), and make sure the following options are set correctly:

- **video d3d**
- **filter 0**
- **hlsl_enable 1**

In the Core Vector Options section:

- **beam_width_min 1.0** (*Beam Width Minimum*)
- **beam_width_max 1.0** (*Beam Width Maximum*)
- **beam_intensity_weight 0.0** (*Beam Intensity Weight*)
- **flicker 0.0** (*Vector Flicker*)

In the Vector Post-Processing Options section:

- **vector_beam_smooth 0.0** (*Vector Beam Smooth Amount*)
- **vector_length_scale 0.5** (*Vector Attenuation Maximum*)
- **vector_length_ratio 0.5** (*Vector Attenuation Length Minimum*)

Suggested settings for vector games:

- **bloom_scale** should typically be set higher for vector games than raster games. Try between 0.4 and 1.0 for best effect.
- **bloom_overdrive** should only be used with color vector games.
- **bloom_lvl_weights** should be set as follows:

bloom_lvl0_weight 1.00	Bloom level 0 weight	Full-size target.
bloom_lvl1_weight 0.48	Bloom level 1 weight	1/4 smaller that level 0 target
bloom_lvl2_weight 0.32	Bloom level 2 weight	1/4 smaller that level 1 target
bloom_lvl3_weight 0.24	Bloom level 3 weight	1/4 smaller that level 2 target
bloom_lvl4_weight 0.16	Bloom level 4 weight	1/4 smaller that level 3 target
bloom_lvl5_weight 0.24	Bloom level 5 weight	1/4 smaller that level 4 target
bloom_lvl6_weight 0.32	Bloom level 6 weight	1/4 smaller that level 5 target
bloom_lvl7_weight 0.48	Bloom level 7 weight	1/4 smaller that level 6 target
bloom_lvl8_weight 0.64	Bloom level 8 weight	1/4 smaller that level 7 target

7.6 GLSL Effects for *nix, OS X, and Windows

By default, MAME outputs an idealized version of the video as it would be on the way to the arcade cabinet's monitor, with minimal modification of the output (primarily to stretch the game image back to the aspect ratio the monitor would traditionally have, usually 4:3) -- this works well, but misses some of the nostalgia factor. Arcade monitors were never ideal, even in perfect condition, and the nature of a CRT display distorts that image in ways that change the appearance significantly.

Modern LCD monitors simply do not look the same, and even computer CRT monitors cannot match the look of an arcade monitor without help.

That's where GLSL comes into the picture.

GLSL simulates most of the effects that a CRT arcade monitor has on the video, making the result look a lot more authentic. However, GLSL requires some effort on the user's part: the settings you use are going to be tailored to your PC's system specs, and especially the monitor you're using. Additionally, there were hundreds of thousands of monitors out there in arcades. Each was tuned and maintained differently, meaning there is no one correct appearance to judge by either. Basic guidelines will be provided here to help you, but you may also wish to ask for opinions on popular MAME-centric forums.

7.6.1 Resolution and Aspect Ratio

Resolution is a very important subject for GLSL settings. You will want MAME to be using the native resolution of your monitor to avoid additional distortion and lag created by your monitor upscaling the display image.

While most arcade machines used a 4:3 ratio display (or 3:4 for vertically oriented monitors like Pac-Man), it's difficult to find a consumer display that is 4:3 at this point. The good news is that that extra space on the sides isn't wasted. Many arcade cabinets used bezel artwork around the main display, and should you have the necessary artwork files, MAME will display that artwork. Turn the artwork view to Cropped for best results.

Some older LCD displays used a native resolution of 1280x1024, which is a 5:4 aspect ratio. There's not enough extra space to display artwork, and you'll end up with some very slight pillarboxing, but the results will be on-par with a 4:3 monitor.

7.6.2 Getting Started with GLSL

You will need to have followed the initial MAME setup instructions elsewhere in this manual before beginning. Official MAME distributions include GLSL support by default, but do NOT include the GLSL shader files. You will need to obtain the shader files from third party online sources.

Open your `mame.ini` in your text editor of choice (e.g. Notepad), and make sure the following options are set correctly:

- `video opengl`
- `filter 0`

The former is required because GLSL requires OpenGL support. The latter turns off extra filtering that interferes with GLSL output.

Lastly, one more edit will turn GLSL on:

- `gl_glsl 1`

Save the .INI file and you're ready to begin.

7.6.3 Tweaking GLSL Settings inside MAME

For multiple, complicated to explain reasons, GLSL settings are no longer saved when you exit MAME. This means that while tweaking settings is a little more work on your part, the results will always come out as expected.

Start by loading MAME with the game of your choice (e.g. **mame pacman**)

The tilde key (**~**) brings up the on-screen display options. Use up and down to go through the various settings, while left and right will allow you to change that setting. Results will be shown in real time as you're changing these settings.

Once you've found settings you like, write the numbers down on a notepad and exit MAME.

7.6.4 Configuration Editing

As referenced in *Order of Configurattion File Loading*, MAME has a order in which it processes INI files. The GLSL settings can be edited in `mame.ini`, but to take full advantage of the power of MAME's config files, you'll want to copy the GLSL settings from mame.ini to one of the other config files and make changes there.

For instance, once you've found GLSL settings you think are appropriate for Neo Geo games, you can put those settings into neogeo.ini so that all Neo-Geo games will be able to take advantage of those settings without needing to add it to every game INI manually.

7.6.5 Configuration Settings

gl_glsl

Enables GLSL when set to 1, disabled if set to 0. Defaults to *0*.

gl_glsl_filter

Enables filtering to GLSL output. Reduces jagginess at the cost of blurriness.

glsl_shader_mame0

...

glsl_shader_mame9

Specifies the shaders to run, in the order from 0 to 9. See your shader pack author for details on which to run in which order for best effect.

glsl_shader_screen0

...

glsl_shader_screen9

Specifies screen to apply the shaders on.

7.7 Controller Configuration Files

- *Introduction*
- *Basic structure*
- *Substituting default controls*
- *Overriding defaults by input type*
- *Overriding defaults for specific inputs*
- *Assigning input device numbers*
- *Setting pointer input options*

7.7.1 Introduction

Controller configuration files can be used to modify MAME's default input settings. Controller configuration files may be supplied with an input device to provide more suitable defaults, or used as profiles that can be selected for different situations. MAME includes a few sample controller configuration files in the **ctrlr** folder, designed to provide useful defaults for certain arcade-style controllers.

Controller configuration files are an XML application, using the `.cfg` filename extension. MAME searches for controller configuration files in the directories specified using the *ctrlrpath* option. A controller configuration file is selected by setting the `ctrlr` option to its filename, excluding the `.cfg` extension (e.g. set the `ctrlr` option to `scorpionxg` to use **scorpionxg.cfg**). It is an error if the specified controller configuration file does not exist, or if it contains no sections applicable to the emulated system.

Controller configuration files use implementation-dependent input tokens. The values available and their precise meanings depend on the exact version of MAME used, the input devices connected, the selected input provider modules (*keyboardprovider*, *mouseprovider*, *lightgunprovider* and *joystickprovider* options), and possibly other settings.

7.7.2 Basic structure

Controller configuration files follow a similar format to the system configuration files that MAME uses to save things like input settings and bookkeeping data (created in the folder specified using the *cfg_directory option*). This example shows the overall structure of a controller configuration file:

```
<?xml version="1.0"?>
<mameconfig version="10">
    <system name="default">
        <input>
            <!-- settings affecting all emulated systems go here -->
        </input>
    </system>
    <system name="neogeo">
        <input>
            <!-- settings affecting neogeo and clones go here -->
        </input>
    </system>
    <system name="intellec4.cpp">
        <input>
            <!-- settings affecting all systems defined in intellec4.cpp go here -->
        </input>
    </system>
</mameconfig>
```

The root of a controller configuration file must be a `mameconfig` element, with a `version` attribute specifying the configuration format version (currently `10` – MAME will not load a file using a different version). The `mameconfig` element contains one or more `system` elements, each of which has a `name` attribute specifying the system(s) it applies to. Each `system` element may contain an `input` element which holds the actual `remap` and `port` configuration elements, which will be described later. Each `system` element may also contain a `pointer_input` element to set pointer input options for systems with interactive artwork.

When launching an emulated system, MAME will apply configuration from `system` elements where the value of the `name` attribute meets one of the following criteria:

- If the `name` attribute has the value `default`, it will always be applied (including for the system/software selection menus).
- If the value of the `name` attribute matches the system's short name, the short name of its parent system, or the short name of its BIOS system (if applicable).
- If the value of the `name` attribute matches the name of the source file where the system is defined.

For example, for the game "DaeJeon! SanJeon SuJeon (AJTUE 990412 V1.000)", `system` elements will be applied if their `name` attribute has the value `default` (applies to all systems), `sanjeon` (short name of the system itself), `sasissu` (short name of the parent system), `stvbios` (short name of the BIOS system), or `stv.cpp` (source file where the system is defined).

As another example, a `system` element whose `name` attribute has the value `zac2650.cpp` will be applied for the systems "The Invaders", "Super Invader Attack (bootleg of The Invaders)", and "Dodgem".

Applicable `system` elements are applied in the order they appear in the controller configuration file. Settings from elements that appear later in the file may modify or override settings from elements that appear earlier. Within a `system` element, `remap` elements are applied before `port` elements.

7.7.3 Substituting default controls

You can use a `remap` element to substitute one host input for another in MAME's default input configuration. For example, this substitutes keys on the numeric keypad for the cursor direction keys:

```
<input>
    <remap origcode="KEYCODE_UP" newcode="KEYCODE_8PAD" />
    <remap origcode="KEYCODE_DOWN" newcode="KEYCODE_2PAD" />
    <remap origcode="KEYCODE_LEFT" newcode="KEYCODE_4PAD" />
    <remap origcode="KEYCODE_RIGHT" newcode="KEYCODE_6PAD" />
</input>
```

The `origcode` attribute specifies the token for the host input to be substituted, and the `newcode` attribute specifies the token for the replacement host input. In this case, assignments using the cursor up, down, left and right arrows will be replaced with the numeric 8, 2, 4 and 6 keys on the numeric keypad, respectively.

Note that substitutions specified using `remap` elements only apply to inputs that use MAME's default assignment for the input type. That is, they only apply to default assignments for control types set in the "Input Assignments (general)" menus. They *do not* apply to default control assignments set in driver/device I/O port definitions (using the `PORT_CODE` macro).

MAME applies `remap` elements found inside any applicable `system` element.

7.7.4 Overriding defaults by input type

Use `port` elements with `type` attributes but without `tag` attributes to override the default control assignments for emulated inputs by type:

```
<input>
    <port type="UI_MENU">
        <newseq type="standard">KEYCODE_TAB OR KEYCODE_1 KEYCODE_5</newseq>
    </port>
    <port type="UI_CANCEL">
        <newseq type="standard">KEYCODE_ESC OR KEYCODE_2 KEYCODE_6</newseq>
    </port>

    <port type="P1_BUTTON1">
        <newseq type="standard">KEYCODE_C OR JOYCODE_1_BUTTON1</newseq>
    </port>
    <port type="P1_BUTTON2">
        <newseq type="standard">KEYCODE_LSHIFT OR JOYCODE_1_BUTTON2</newseq>
    </port>
    <port type="P1_BUTTON3">
        <newseq type="standard">KEYCODE_Z OR JOYCODE_1_BUTTON3</newseq>
    </port>
    <port type="P1_BUTTON4">
        <newseq type="standard">KEYCODE_X OR JOYCODE_1_BUTTON4</newseq>
    </port>
</input>
```

This sets the following default input assignments:

Show/Hide Menu (User Interface)
: Tab key, or 1 and 2 keys pressed simultaneously

UI Cancel (User Interface)
Escape key, or 2 and 6 keys pressed simultaneously

P1 Button 1 (Player 1 Controls)
C key, or joystick 1 button 1

P1 Button 2 (Player 1 Controls)
Left Shift key, or joystick 1 button 2

P1 Button 3 (Player 1 Controls)
Z key, or joystick 1 button 3

P1 Button 4 (Player 1 Controls)
X key, or joystick 1 button 4

Note that this will only apply for inputs that use MAME's default assignment for the input type. That is, `port` elements without `tag` attributes only override default assignments for control types set in the "Input Assignments (general)" menus. They *do not* override default control assignments set in driver/device I/O port definitions (using the `PORT_CODE` macro).

MAME applies `port` elements without `tag` attributes found inside any applicable `system` element.

7.7.5 Overriding defaults for specific inputs

Use `port` elements with `tag`, `type`, `mask` and `defvalue` attributes to override defaults for specific inputs. These `port` elements should only occur inside `system` elements that apply to particular systems or source files (i.e. they should not occur inside `system` elements where the `name` attribute has the value `default`). The default control assignments can be overridden, as well as the toggle setting for digital inputs.

The `tag`, `type`, `mask` and `defvalue` are used to identify the affected input. You can find out the values to use for a particular input by changing its control assignment, exiting MAME, and checking the values in the system configuration file (created in the folder specified using the *cfg_directory option*). Note that these values are not guaranteed to be stable, and may change between MAME versions.

Here's an example that overrides defaults for 280-ZZZAP:

```
<system name="280zzzap">
    <input>
        <port tag=":IN0" type="P1_BUTTON2" mask="16" defvalue="0" toggle="no" />
        <port tag=":IN1" type="P1_PADDLE" mask="255" defvalue="127">
            <newseq type="increment">KEYCODE_K</newseq>
            <newseq type="decrement">KEYCODE_J</newseq>
        </port>
    </input>
</system>
```

This sets the controls to steer left and right to the K and J keys, respectively, and disables the toggle setting for the gear shift input.

7.7.6 Assigning input device numbers

Use `mapdevice` elements with `device` and `controller` attributes to assign stable numbers to input devices. Note that all devices explicitly configured in this way must be connected when MAME starts for this to work as expected.

Set the `device` attribute to the device ID of the input device, and set the `controller` attribute to the desired input device token (device type and number).

Here's an example numbering two light guns and two XInput game controllers:

```
<system name="default">
    <input>
        <mapdevice device="VID_D209&amp;PID_1601" controller="GUNCODE_1" />
        <mapdevice device="VID_D209&amp;PID_1602" controller="GUNCODE_2" />
        <mapdevice device="XInput Player 1" controller="JOYCODE_1" />
        <mapdevice device="XInput Player 2" controller="JOYCODE_2" />
    </input>
</system>
```

MAME applies `mapdevice` elements found inside the first applicable `system` element only. To avoid confusion, it's simplest to place the `system` element applying to all systems (`name` attribute set to `default`) first in the file, and use it to assign input device numbers.

7.7.7 Setting pointer input options

A `pointer_input` element may contain `target` elements to set pointer input options for each output screen or window. Each `target` element must have an `index` attribute containing the zero-based index of the screen to which it applies.

Each `target` element may have an `activity_timeout` attribute to set the time after which a mouse pointer that has not moved and has no buttons pressed will be considered inactive. The value is specified in seconds, and must be in the range of 0.1 seconds to 10 seconds, inclusive.

Each `target` element may have a `hide_inactive` element to set whether inactive pointers may be hidden. If the value is `0` (zero), inactive pointers will not be hidden. If the value is `1`, inactive pointers may be hidden, but layout views can still specify that inactive pointers should not be hidden.

Here's an example demonstrating the use of this feature:

```
<system name="default">
    <pointer_input>
        <target index="0" activity_timeout="1.5" />
    </pointer_input>
</system>
<system name="intellec4.cpp">
    <pointer_input>
        <target index="0" hide_inactive="0" />
    </pointer_input>
</system>
```

For all systems, pointers over the first output screen or window will be considered inactive after not moving for 1.5 seconds with no buttons pressed. For systems defined in `intellec4.cpp`, inactive pointers over the first window will not be hidden.

7.8 Stable Controller IDs

By default, MAME does not assign stable numbers to input devices. For instance, a game pad controller may be assigned to "Joy 1" initially, but after restarting, the same game pad may be reassigned to "Joy 3".

The reason is that MAME assigns numbers to input devices in the based on enumeration order. Factors that can cause this to change include disconnecting and reconnecting USB or Bluetooth devices, changing ports/hubs, and even just restarting the computer. Input device numbers can be quite unpredictable.

This is where the `mapdevice` configuration setting comes into the picture. By adding this setting to a *controller configuration file*, you can ensure that a given input device is always assigned the same number in MAME.

7.8.1 Using mapdevice

The `mapdevice` XML element is added to the `input` XML element in the controller configuration file. It requires two attributes, `device` and `controller`. Note that `mapdevice` elements only take effect in the controller configuration file (set using the *-ctrlr option*) – they are ignored in system configuration files and the default configuration file.

The `device` attribute specifies the device ID of the input device to match. It may also be a substring of the device ID. To obtain the device ID for an input device, select it in the *Input Devices menu*, and then select **Copy Device ID**. The device ID will be copied to the clipboard. You can also see input device IDs by turning on verbose logging (more on this later). The format of device IDs depends the type of device, selected input provider module and operating system. Your input device IDs may look very different to the examples here.

The `controller` attribute specifies the input token for the input device type (i.e. `JOYCODE`, `GUNCODE`, `MOUSECODE`) and number to assign to the device, separated by an underscore. Numbering starts from 1. For example the token for the first joystick device will be `JOYCODE_1`, the second will be `JOYCODE_2`, and so on.

7.8.2 Example

Here's an example:

```
<mameconfig version="10">
    <system name="default">
        <input>
            <mapdevice device="VID_D209&amp;amp;PID_1601" controller="GUNCODE_1" />
            <mapdevice device="VID_D209&amp;amp;PID_1602" controller="GUNCODE_2" />
            <mapdevice device="XInput Player 1" controller="JOYCODE_1" />
            <mapdevice device="XInput Player 2" controller="JOYCODE_2" />

            <port type="P1_JOYSTICK_UP">
                <newseq type="standard">
                    JOYCODE_1_YAXIS_UP_SWITCH OR KEYCODE_8PAD
                </newseq>
            </port>
            ...
```

In the above example, we have four device mappings specified:

- The first two `mapdevice` elements map player 1 and 2 light guns to Gun 1 and Gun 2, respectively. We use a substring of the full device IDs to match each devices. Note that, since this is XML, we needed to escape the ampersands (&) as `&`.
- The last two `mapdevices` elements map player 1 and player 2 gamepad controllers to Joy 1 and Joy 2, respectively. In this case, these are XInput game controllers.

7.8.3 Listing Available Devices

There are two ways to obtain device IDs: by copying them from the *Input Devices menu*, or by *turning on verbose logging* and finding the messages logged when input devices are added.

To reach the Input Devices menu from the system selection menu, select **General Settings**, and the select **Input Devices**. To reach the input devices menu from the *main menu*, select **Input Settings**, then select **Input Devices**. From the Input Devices menu, select a device, then select **Copy Device ID** to copy its device ID to the clipboard.

To use verbose logging, run MAME with the `-v` or `-verbose` option on the command line. Search the output for messages starting with "Input: Adding…" that show recognised input devices and their respective IDs.

Here an example:

Input: Adding lightgun #1:
Input: Adding lightgun #2:
Input: Adding lightgun #3: HID-compliant mouse (**device id: \\?\HID#VID_045E&PID_0053#7&18297dcb&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd}**)
Input: Adding lightgun #4: HID-compliant mouse (**device id: \\?\HID#IrDeviceV2&Col08#2&2818a073&0&0007#{378de44c-56ef-11d1-bc8c-00a0c91405dd}**)
Input: Adding lightgun #5: HID-compliant mouse (**device id: \\?\HID#VID_D209&PID_1602&MI_02#8&389ab7f3&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd}**)
Input: Adding lightgun #6: HID-compliant mouse (**device id: \\?\HID#VID_D209&PID_1601&MI_02#9&375eebb1&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd}**)
Input: Adding lightgun #7: HID-compliant mouse (**device id: \\?\HID#VID_1241&PID_1111#8&198f3adc&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd}**)
Skipping DirectInput for XInput compatible joystick Controller (XBOX 360 For Windows).
Input: Adding joystick #1: ATRAK Device #1 (**device id: ATRAK Device #1**)
Skipping DirectInput for XInput compatible joystick Controller (XBOX 360 For Windows).
Input: Adding joystick #2: ATRAK Device #2 (**device id: ATRAK Device #2**)
Input: Adding joystick #3: XInput Player 1 (**device id: XInput Player 1**)
Input: Adding joystick #4: XInput Player 2 (**device id: XInput Player 2**)

Furthermore, when devices are reassigned using `mapdevice` elements in the controller configuration file, you'll see that in the verbose log output, too, such as:

Input: Remapped lightgun #1: HID-compliant mouse (device id: \\?\HID#VID_D209&PID_1601&MI_02#9&375eebb1&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd})
Input: Remapped lightgun #2: HID-compliant mouse (device id: \\?\HID#VID_D209&PID_1602&MI_02#8&389ab7f3&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd})
Input: Remapped joystick #1: XInput Player 1 (device id: XInput Player 1)
Input: Remapped joystick #2: XInput Player 2 (device id: XInput Player 2)

7.8.4 Limitations

You can only assign stable numbers to devices if MAME receives stable, unique device IDs from the input device provider and operating system. This is not always the case. For example the SDL joystick provider is not capable of providing unique IDs for many USB game controllers.

If not all configured devices are connected when MAME starts, the devices that are connected may not be numbered as expected.

7.9 Linux Lightguns

Many lightguns (especially the Ultimarc AimTrak) may work better in MAME under Linux when using a slightly more complicated configuration. The instructions here are for getting an AimTrak working on Ubuntu using udev and Xorg, but other Linux distributions and lightguns may work with some changes to the steps.

7.9.1 Configure udev rules

For the AimTrak, each lightgun exposes several USB devices once connected: 2 mouse emulation devices, and 1 joystick emulation device. We need to instruct libinput via udev to ignore all but the correct emulated mouse device. This prevents each lightgun from producing multiple mouse devices, which would result in non-deterministic selection between the "good" and "bad" emulated mouse devices by Xorg.

Create a new file named `/etc/udev/rules.d/65-aimtrak.rules` and place the following contents into it:

```
# Set mode (0666) & disable libinput handling to avoid X11 picking up the wrong
# interfaces/devices.
SUBSYSTEMS=="usb", ATTRS{idVendor}=="d209", ATTRS{idProduct}=="160*",
   MODE="0666", ENV{ID_INPUT}="", ENV{LIBINPUT_IGNORE_DEVICE}="1"

# For ID_USB_INTERFACE_NUM==2, re-enable libinput handling.
SUBSYSTEMS=="usb", ATTRS{idVendor}=="d209", ATTRS{idProduct}=="160*",
    ENV{ID_USB_INTERFACE_NUM}=="02", ENV{ID_INPUT}="1",
    ENV{LIBINPUT_IGNORE_DEVICE}="0"
```

This configuration will be correct for the AimTrak lightguns, however each brand of lightgun will require their own settings.

7.9.2 Configure Xorg inputs

Next, we'll configure Xorg to treat the lightguns as a "Floating" device. This is important for multiple lightguns to work correctly and ensures each gun's emulated mouse pointer is NOT merged with the main system mouse pointer.

In `/etc/X11/xorg.conf.d/60-aimtrak.conf` we will need:

```
Section "InputClass"
    Identifier "AimTrak Guns"
    MatchDevicePath "/dev/input/event*"
    MatchUSBID "d209:160*"
```

(continues on next page)

(continued from previous page)

```
    Driver "libinput"
    Option "Floating" "yes"
    Option "AutoServerLayout" "no"
EndSection
```

7.9.3 Configure MAME

Next, we'll need to configure MAME via `mame.ini` to use the new lightgun device(s):

```
lightgun 1
lightgun_device lightgun
lightgunprovider x11
```

These first three lines tell MAME to enable lightgun support, to tell MAME that we're using a lightgun instead of a mouse, and to use the x11 provider:

```
lightgun_index1 "Ultimarc ATRAK Device #1"
lightgun_index2 "Ultimarc ATRAK Device #2"
lightgun_index3 "Ultimarc ATRAK Device #3"
lightgun_index4 "Ultimarc ATRAK Device #4"
```

Lastly, as many lightgun games require aiming away from the screen and firing to reload, you may reloading and we're using a device that represents that as a separate button, you will need to configure the *off-screen reload helper plugin* or *input macro plugin* to translate this to the axis and button inputs that these games expect.

CHAPTER

EIGHT

MAME DEBUGGER

- *Introduction*
- *Debugger commands*
- *Specifying devices and address spaces*
- *Debugger expression syntax*
  - *Numbers*
  - *Boolean values*
  - *Memory accesses*
  - *Functions*

8.1 Introduction

MAME includes an interactive low-level debugger that targets the emulated system. This can be a useful tool for diagnosing emulation issues, developing software to run on vintage systems, creating cheats, ROM hacking, or just investigating how software works.

Use the `-debug` command line option to start MAME with the debugger activated. By default, pressing the back-tick/tilde (~) during emulation breaks into the debugger (this can be changed by reassigning the **Break in Debugger** input).

The exact appearance of the debugger depends on your operating system and the options MAME was built with. All variants of the debugger provide a multi-window interface for viewing the contents of memory and disassembled code.

The debugger console window is a special window that shows the contents of CPU registers and disassembled code around the current program counter address, and provides a command-line interface to most of the debugging functionality.

8.2 Debugger commands

Debugger commands are described in the sections below. You can also type **help <topic>** in the debugger console, where **<topic>** is the name of a command, to see documentation directly in MAME.

8.2.1 General Debugger Commands

help
: displays built-in help in the console

do
: evaluates the given expression

symlist
: lists registered symbols

softreset
: executes a soft reset

hardreset
: executes a hard reset

print
: prints one or more <item>s to the console

printf
: prints one or more <item>s to the console using <format>

logerror
: outputs one or more <item>s to the error.log

tracelog
: outputs one or more <item>s to the trace file using <format>

tracesym
: outputs one or more <item>s to the trace file

history
: displays recently visited PC addresses and opcodes

trackpc
: visually track visited opcodes

trackmem
: record which PC writes to each memory address

pcatmem
: query which PC wrote to a given memory address

rewind
: go back in time by loading the most recent rewind state

statesave
: save a state file for the emulated system

stateload
: load a state file for the emulated system

snap
: save a screen snapshot

source
: read commands from file and executes them one by one

time
: prints the current machine time to the console

quit
: exit the debugger and end the emulation session

help

help [<topic>]

Displays built-in debugger help in the debugger console. If no **<topic>** is specified, top-level topics are listed. Most debugger commands have correspondingly named help topics.

Examples:

`help`
: Lists top-level help topics.

`help expressions`
: Displays built-in help for debugger expression syntax.

`help wpiset`
: Displays built-in help for the *wpiset* command.

Back to *General Debugger Commands*

do

do <expression>

The **do** command simply evaluates the supplied expression. This is often used to set or modify device state variable (e.g. CPU registers), or to write to memory. See *Debugger expression syntax* for details about expression syntax.

Examples:

`do pc = 0`
: Sets the register **pc** to 0.

Back to *General Debugger Commands*

symlist

symlist [<cpu>]

Lists registered symbols and their values. If **<cpu>** is not specified, symbols in the global symbol table are displayed; otherwise, symbols specific to the device **<cpu>** are displayed. Symbols are listed alphabetically. Read-only symbols are noted. See *Specifying devices and address spaces* for details on how to specify a CPU.

Examples:

`symlist`
: Displays the global symbol table.

`symlist 2`
: Displays the symbols for the third CPU in the system (zero-based index).

`symlist audiocpu`
: Displays symbols for the CPU with the absolute tag `:audiocpu`.

Back to *General Debugger Commands*

softreset

softreset

Executes a soft reset. This calls the reset member functions of all the devices in the system (by default, pressing **F3** during emulation has the same effect).

Examples:

`softreset`
: Executes a soft reset.

Back to *General Debugger Commands*

hardreset

hardreset

Executes a hard reset. This tears down the emulation session and starts another session with the same system and options (by default, pressing **Shift+F3** during emulation has the same effect). Note that this will lose history in the debugger console and error log.

Examples:

`hardreset`
: Executes a hard reset.

Back to *General Debugger Commands*

print

print <item>[,...]

The **print** command prints the results of one or more expressions to the debugger console as hexadecimal numbers.

Examples:

`print pc`
: Prints the value of the **pc** register the console as a hex number.

`print a,b,a+b`
: Prints **a**, **b**, and the value of **a+b** to the console as hex numbers.

Back to *General Debugger Commands*

printf

printf <format>[,<argument>[,...]]

Prints a C-style formatted message to the debugger console. Only a very limited subset of format specifiers and escape sequences are available:

%c
: Prints the corresponding argument as an 8-bit character.

%[-][0][<n>]d
: Prints the corresponding argument as a decimal number with optional left justification, zero fill and minimum field width.

%[-][0][<n>]o
: Prints the corresponding argument as an octal number with optional left justification, zero fill and minimum field width.

%[-][0][<n>]x
: Prints the corresponding argument as a lowercase hexadecimal number with optional left justification, zero fill and minimum field width.

%[-][0][<n>]X
: Prints the corresponding argument as an uppercase hexadecimal number with optional left justification, zero fill and minimum field width.

%[-][<n>][.[<n>]]s
: Prints a null-terminated string of 8-bit characters from the address and address space given by the corresponding argument, with optional left justification, minimum and maximum field widths.

%%
: Prints a literal percent symbol.

\n
: Prints a line break.

****
: Prints a literal backslash.

All other format specifiers are ignored.

Examples:

printf "PC=%04X",pc
: Prints PC=<pcval> where **<pcval>** is the hexadecimal value of the **pc** register with a minimum of four digits and zero fill.

printf "A=%d, B=%d\\nC=%d",a,b,a+b
: Prints A=<aval>, B=<bval> on one line, and C=<a+bval> on a second line.

Back to *General Debugger Commands*

logerror

logerror <format>[,<argument>[,...]]

Prints a C-style formatted message to the error log. See *printf* for details about the limited set of supported format specifiers and escape sequences.

Examples:

`logerror "PC=%04X",pc`
: Logs PC=<pcval> where **<pcval>** is the hexadecimal value of the **pc** register with a minimum of four digits and zero fill.

`logerror "A=%d, B=%d\\nC=%d",a,b,a+b`
: Logs A=<aval>, B=<bval> on one line, and C=<a+bval> on a second line.

Back to *General Debugger Commands*

tracelog

tracelog <format>[,<argument>[,...]]

Prints a C-style formatted message to the currently open trace file (see *trace* for more information). If no trace file is open, this command has no effect. See *printf* for details about the limited set of supported format specifiers and escape sequences.

Examples:

`tracelog "PC=%04X",pc`
: Outputs PC=<pcval> where **<pcval>** is the hexadecimal value of the **pc** register with a minimum of four digits and zero fill if a trace log file is open.

`tracelog "A=%d, B=%d\\nC=%d",a,b,a+b`
: Outputs A=<aval>, B=<bval> on one line, and C=<a+bval> on a second line if a trace log file is open.

Back to *General Debugger Commands*

tracesym

tracesym <item>[,...]

Prints the specified symbols to the currently open trace file (see *trace* for more information). If no trace file is open, this command has no effect.

Examples:

`tracesym pc`
: Outputs PC=<pcval> where **<pcval>** is the value of the **pc** register in its default format if a trace log file is open.

Back to *General Debugger Commands*

history

history [<CPU>[,<length>]]

Displays recently visited PC addresses, and disassembly of the instructions at those addresses. If present, the first argument selects the CPU (see *Specifying devices and address spaces* for details); if no CPU is specified, the visible CPU is assumed. The second argument, if present, limits the maximum number of addresses shown. Addresses are shown in order from least to most recently visited.

Examples:

`history ,5`
: Displays up to five most recently visited PC addresses and instructions for the visible CPU.

`history 3`
: Displays recently visited PC addresses and instructions for the fourth CPU in the system (zero-based index).

`history audiocpu,1`
: Displays the most recently visited PC address and instruction for the CPU with the absolute tag `:audiocpu`.

trackpc

trackpc [<enable>[,<CPU>[,<clear>]]]

Turns visited PC address tracking for disassembly views on or off. Instructions at addresses visited while tracking is on are highlighted in debugger disassembly views. The first argument is a Boolean specifying whether tracking should be turned on or off (defaults to on). The second argument specifies the CPU to enable or disable tracking for (see *Specifying devices and address spaces* for details); if no CPU is specified, the visible CPU is assumed. The third argument is a Boolean specifying whether existing data should be cleared (defaults to false).

Examples:

`trackpc 1`
: Begin or tracking the current CPU's PC.

`trackpc 1,0,1`
: Begin or continue tracking PC on the first CPU in the system (zero-based index), but clear the history tracked so far.

Back to *General Debugger Commands*

trackmem

trackmem [<enable>,[<CPU>,[<clear>]]]

Enables or disables logging the PC address each time a memory address is written to. The first argument is a Boolean specifying whether tracking should be enabled or disabled (defaults to enabled). The second argument specifies the CPU to enable or disable tracking for (see *Specifying devices and address spaces* for details); if no CPU is specified, the visible CPU is assumed. The third argument is a Boolean specifying whether existing data should be cleared (defaults to false).

Use *pcatmem* to retrieve this data. Right-clicking a debugger memory view will also display the logged PC value for the given address in some configurations.

Examples:

`trackmem`
: Begin or continue tracking memory writes for the visible CPU.

`trackmem 1,0,1`
: Begin or continue tracking memory writes for the first CPU in the system (zero-based index), but clear existing tracking data.

Back to *General Debugger Commands*

pcatmem

pcatmem[{d|i|o}] <address>[:<space>]

Returns the PC value at the time the specified address was most recently written to. The argument is the requested address, optionally followed by a colon and a CPU and/or address space (see *Specifying devices and address spaces* for details). The optional **d**, **i** or **o** suffix controls the default address space for the command.

Tracking must be enabled for the data this command uses to be recorded (see *trackmem*). Right-clicking a debugger memory view will also display the logged PC value for the given address in some configurations.

Examples:

`pcatmem 400000`
: Print the PC value when location 400000 in the visible CPU's program space was most recently written.

`pcatmem 3bc:io`
: Print the PC value when location 3bc in the visible CPU's `io` space was most recently written.

`pcatmem 1400:audiocpu`
: Print the PC value when location 1400 in the CPU `:audiocpu`'s program space was most recently written.

Back to *General Debugger Commands*

rewind

rewind

Loads the most recent RAM-based saved state. When enabled, rewind states are saved when *step*, *over* and *out* commands are used, storing the machine state as of the moment before stepping. May be abbreviated to `rw`.

Consecutively loading rewind states can work like reverse execution. Depending on which steps forward were taken previously, the behavior can be similar to GDB's **reverse-stepi** and **reverse-next** commands. All output for this command is currently echoed into the running machine window.

Previous memory and PC tracking statistics are cleared. Actual reverse execution does not occur.

Examples:

`rewind`
: Load the previous RAM-based save state.

`rw`
: Abbreviated form of the command.

Back to *General Debugger Commands*

statesave

statesave <filename>

Creates a save state at the current moment in emulated time. The state file is written to the configured save state directory (see the *state_directory* option), and the **.sta** extension is automatically appended to the specified file name. May be abbreviates to `ss`.

All output from this command is currently echoed into the running machine window.

Examples:

`statesave foo`
: Saves the emulated machine state to the file **foo.sta** in the configured save state directory.

`ss bar`
: Abbreviated form of the command – saves the emulated machine state to **bar.sta**.

Back to *General Debugger Commands*

stateload

stateload <filename>

Restores a saved state file from disk. The specified state file is read from the configured save state directory (see the *state_directory* option), and the **.sta** extension is automatically appended to the specified file name. May be abbreviated to `sl`.

All output for this command is currently echoed into the running machine window. Previous memory and PC tracking statistics are cleared.

Examples:

`stateload foo`
: Loads state from file **foo.sta** to the configured save state directory.

`sl bar`
: Abbreviated form of the command – loads the file **bar.sta**.

Back to *General Debugger Commands*

snap

snap [<filename>[,<scrnum>]]

Takes a snapshot of the emulated video display and saves it to the configured snapshot directory (see the *snapshot_directory* option). If a file name is specified, a single screenshot for the specified screen is saved using the specified filename (or the first emulated screen in the system if a screen is not specified). If a file name is not specified, the configured snapshot view and file name pattern are used (see the *snapview* and *snapname* options).

If a file name is specified, the **.png** extension is automatically appended. The screen number is specified as a zero-based index, as seen in the names of automatically-generated single-screen views in MAME's video options menus.

Examples:

`snap`
: Takes a snapshot using the configured snapshot view and file name options.

`snap shinobi`
: Takes a snapshot of the first emulated video screen and saves it as **shinobi.png** in the configured snapshot directory.

Back to *General Debugger Commands*

source

source <filename>

Reads the specified file in text mode and executes each line as a debugger command. This is similar to running a shell script or batch file.

Examples:

`source break_and_trace.cmd`
: Reads and executes debugger commands from **break_and_trace.cmd**.

Back to *General Debugger Commands*

time

Prints the total elapsed emulated time to the debugger console.

Examples:

`time`
: Prints the elapsed emulated time.

Back to *General Debugger Commands*

quit

quit

Closes the debugger and ends the emulation session immediately. Either exits MAME or returns to the system selection menu, depending on whether the system was specified on the command line when starting MAME.

Examples:

`quit`
: Exits the emulation session immediately.

Back to *General Debugger Commands*

8.2.2 Memory Debugger Commands

dasm
: disassemble code to a file

find
: search emulated memory for data

fill
: fill emulated memory with specified pattern

dump
: dump emulated memory to a file as text

strdump
: dump delimited strings from emulated memory to a file

save
save binary data from emulated memory to a file

saver
save binary data from a memory region to a file

load
load binary data from a file to emulated memory

loadr
load binary data from a file to a memory region

map
map a logical address to the corresponding physical address and handler

memdump
dump current memory maps to a file

dasm

dasm <filename>,<address>,<length>[,<opcodes>[,<CPU>]]

Disassembles program memory to the file specified by the **<filename>** parameter. The **<address>** parameter specifies the address to start disassembling from, and the **<length>** parameter specifies how much memory to disassemble. The range **<address>** through **<address>+<length>-1**, inclusive, will be disassembled to the file. By default, raw opcode data is output with each line. The optional **<opcodes>** parameter is a Boolean that enables disables this feature. By default, program memory for the visible CPU is disassembled. To disassemble program memory for a different CPU, specify it using the optional fifth parameter (see *Specifying devices and address spaces* for details).

Examples:

`dasm venture.asm,0,10000`
Disassembles addresses 0-ffff for the visible CPU, including raw opcode data, to the file **venture.asm**.

`dasm harddriv.asm,3000,1000,0,2`
Disassembles addresses 3000-3fff for the third CPU in the system (zero-based index), without raw opcode data, to the file **harddriv.asm**.

Back to *Memory Debugger Commands*

find

f[ind][{d|i|o}] <address>[:<space>],<length>[,<data>[,...]] f[ind] <address>:<memory>.{m|s},<length>[,<data>[,...]]

Search through memory for the specified sequence of data. The **<address>** is the address to begin searching from, optionally followed by a device and/or address space (see *Specifying devices and address spaces* for details); the **<length>** specifies how much memory to search. If an address space is not specified, the command suffix sets the address space: `find` defaults to the first address space exposed by the device, `findd` defaults to the space with index 1 (data), `findi` default to the space with index 2 (I/O), and `findo` defaults to the space with index 3 (opcodes).

The **<data>** can either be a quoted string, a numeric value or expression, or the wildcard character `?`. By default, strings imply a byte-sized search; by default non-string data is searched using the native word size of the address space. To override the search size for non-string data, you can prefix values with `b.` to force byte-sized search, `w.` for word-sized search, `d.` for double word-sized search, and `q.` for quadruple word-sized search. Overrides propagate to subsequent values, so if you want to search for a sequence of words, you need only prefix the first value with `w.`. Also note that you can intermix sizes to perform more complex searches.

The entire range **<address>** through **<address>+<length>-1**, inclusive, will be searched for the sequence, and all occurrences will be displayed.

Examples:

find 0,10000,"HIGH SCORE",0
Searches the address range 0-ffff in the program space of the visible CPU for the string "HIGH SCORE" followed by a 0 byte.

find 300:tms9918a,100,w.abcd,4567
Searches the address range 300-3ff in the first address space exposed by the device with the absolute tag `:tms9918a` for the word-sized value abcd followed by the word-sized value 4567.

find 0,8000,"AAR",d.0,"BEN",w.0
Searches the address range 0000-7fff for the string "AAR" followed by a dword-sized 0 followed by the string "BEN", followed by a word-sized 0.

Back to *Memory Debugger Commands*

fill

fill[{d|i|o}] <address>[:<space>],<length>[,<data>[,...]] fill <address>:<memory>.{m|s},<length>[,<data>[,...]]

Overwrite a block of memory with copies of the supplied data sequence. The **<address>** specifies the address to begin writing at, optionally followed by a device and/or address space (see *Specifying devices and address spaces* for details); the **<length>** specifies how much memory to fill. If an address space is not specified, the command suffix sets the address space: `fill` defaults to the first address space exposed by the device, `filld` defaults to the space with index 1 (data), `filli` default to the space with index 2 (I/O), and `fillo` defaults to the space with index 3 (opcodes).

The **<data>** can either be a quoted string, or a numeric value or expression. By default, non-string data is written using the native word size of the address space. To override the data size for non-string data, you can prefix values with `b.` to force byte-sized fill, `w.` for word-sized fill, `d.` for double word-sized fill, and `q.` for quadruple word-sized fill. Overrides propagate to subsequent values, so if you want to fill with a series of words, you need only prefix the first value with `w.`. Also note that you can intermix sizes to perform more complex fills. The fill operation may be truncated if a page fault occurs or if part of the sequence or string would fall beyond **<address>+<length>-1**.

Back to *Memory Debugger Commands*

dump

dump[{d|i|o}] <filename>,<address>[:<space>],<length>[,<group>[,<ascii>[,<rowsize>]]] dump <filename>,<address>:<memory>.{m|s},<length>[,<group>[,<ascii>[,<rowsize>]]]

Dump memory to the text file specified by the **<filename>** parameter. The **<address>** specifies the address to start dumping from, optionally followed by a device and/or address space (see *Specifying devices and address spaces* for details); the **<length>** specifies how much memory to dump. If an address space is not specified, the command suffix sets the address space: `dump` defaults to the first address space exposed by the device, `dumpd` defaults to the space with index 1 (data), `dumpi` default to the space with index 2 (I/O), and `dumpo` defaults to the space with index 3 (opcodes).

The range **<address>** through **<address>+<length>-1**, inclusive, will be output to the file. By default, the data will be output using the native word size of the address space. You can override this by specifying the **<group>** parameter, which can be used to group the data in 1-, 2-, 4- or 8-byte chunks. The optional **<ascii>** parameter is a Boolean value used to enable or disable output of ASCII characters on the right of each line (enabled by default). The optional **<rowsize>** parameter specifies the amount of data on each line in address units (defaults to 16 bytes).

Examples:

dump venture.dmp,0,10000
Dumps addresses 0-ffff from the program space of the visible CPU in 1-byte chunks, including ASCII data, to the file **venture.dmp**.

`dumpd harddriv.dmp,3000:3,1000,4,0`
: Dumps data memory addresses 3000-3fff from the fourth CPU in the system (zero-based index) in 4-byte chunks, without ASCII data, to the file **harddriv.dmp**.

`dump vram.dmp,0:sms_vdp:videoram,4000,1,false,8`
: Dumps `videoram` space addresses 0000-3fff from the device with the absolute tag path `:sms_vdp` in 1-byte chunks, without ASCII data, with 8 bytes per line, to the file **vram.dmp**.

Back to *Memory Debugger Commands*

strdump

strdump[{d|i|o}] <filename>,<address>[:<space>],<length>[,<term>] strdump <filename>,<address>:<memory>.{m|s},<length>[,<term>]

Dump memory to the text file specified by the **<filename>** parameter. The **<address>** specifies the address to start dumping from, optionally followed by a device and/or address space (see *Specifying devices and address spaces* for details); the **<length>** specifies how much memory to dump. If an address space is not specified, the command suffix sets the address space: `strdump` defaults to the first address space exposed by the device, `strdumpd` defaults to the space with index 1 (data), `strdumpi` default to the space with index 2 (I/O), and `strdumpo` defaults to the space with index 3 (opcodes).

By default, the data will be interpreted as a series of NUL-terminated (ASCIIZ) strings, the dump will have one string per line, and C-style escapes sequences will be used for bytes that do not represent printable ASCII characters. The optional **<term>** parameter can be used to specify a different string terminator character. If **<term>** is equal to -0x80, strings will be treated as terminating after any byte with the most significant bit set, and the most significant bit will be ignored when printing characters.

Back to *Memory Debugger Commands*

save

save[{d|i|o}] <filename>,<address>[:<space>],<length> save <filename>,<address>:<memory>.{m|s},<length>

Save raw memory to the binary file specified by the **<filename>** parameter. The **<address>** specifies the address to start saving from, optionally followed by a device and/or address space (see *Specifying devices and address spaces* for details); the **<length>** specifies how much memory to save. If an address space is not specified, the command suffix sets the address space: `save` defaults to the first address space exposed by the device, `saved` defaults to the space with index 1 (data), `savei` default to the space with index 2 (I/O), and `saveo` defaults to the space with index 3 (opcodes).

The range **<address>** through **<address>+<length>-1**, inclusive, will be output to the file.

Examples:

`save venture.bin,0,10000`
: Saves addresses 0-ffff from the program space of the current CPU to the binary file **venture.bin**

`saved harddriv.bin,3000:3,1000`
: Saves data memory addresses 3000-3fff from the fourth CPU in the system (zero-based index) to the binary file **harddriv.bin**.

`save vram.bin,0:sms_vdp:videoram,4000`
: Saves `videoram` space addresses 0000-3fff from the device with the absolute tag path `:sms_vdp` to the binary file **vram.bin**.

Back to *Memory Debugger Commands*

saver

saver <filename>,<address>,<length>,<region>

Save raw content of memory region specified by the **<region>** parameter to the binary file specified by the **<filename>** parameter. Regions tags follow the same rules as device tags (see *Specifying devices and address spaces*). The **<address>** specifies the address to start saving from, and the **<length>** specifies how much memory to save. The range **<address>** through **<address>+<length>-1**, inclusive, will be output to the file.

Alternetevely use *save* syntax: `save <filename>,<address>:<region>.m,<length>`

Examples:

`saver data.bin,200,100,:monitor`
: Saves `:monitor` region addresses 200-2ff to the binary file **data.bin**.

`saver cpurom.bin,1000,400,.`
: Saves addresses 1000-13ff from the memory region with the same tag as the visible CPU to the binary file **cpurom.bin**.

Back to *Memory Debugger Commands*

load

load[{d|i|o}] <filename>,<address>[:<space>][,<length>] load <filename>,<address>:<memory>.{m|s}[,<length>]

Load raw memory from the binary file specified by the **<filename>** parameter. The **<address>** specifies the address to start loading to, optionally followed by a device and/or address space (see *Specifying devices and address spaces* for details); the **<length>** specifies how much memory to load. If an address space is not specified, the command suffix sets the address space: `load` defaults to the first address space exposed by the device, `loadd` defaults to the space with index 1 (data), `loadi` default to the space with index 2 (I/O), and `loado` defaults to the space with index 3 (opcodes).

The range **<address>** through **<address>+<length>-1**, inclusive, will be read in from the file. If the **<length>** is omitted, if it is zero, or if it is greater than the total length of the file, the entire contents of the file will be loaded but no more.

Note that this has the same effect as writing to the address space from a debugger memory view, or using the `b@`, `w@`, `d@` or `q@` memory accessors in debugger expressions. Read-only memory will not be overwritten, and writing to I/O addresses may have effects beyond setting register values.

Examples:

`load venture.bin,0,10000`
: Loads addresses 0-ffff in the program space for the visible CPU from the binary file **venture.bin**.

`loadd harddriv.bin,3000,1000,3`
: Loads data memory addresses 3000-3fff in the program space for the fourth CPU in the system (zero-based index) from the binary file **harddriv.bin**.

`load vram.bin,0:sms_vdp:videoram`
: Loads the `videoram` space for the device with the absolute tag path `:sms_vdp` starting at address 0000 with the entire content of the binary file **vram.bin**.

Back to *Memory Debugger Commands*

loadr

loadr <filename>,<address>,<length>,<region>

Load memory in the memory region specified by the **<region>** with raw data from the binary file specified by the **<filename>** parameter. Regions tags follow the same rules as device tags (see *Specifying devices and address spaces*). The **<address>** specifies the address to start loading to, and the **<length>** specifies how much memory to load. The range **<address>** through **<address>+<length>-1**, inclusive, will be read in from the file. If the **<length>** is zero, or is greater than the total length of the file, the entire contents of the file will be loaded but no more.

Alternetevely use *load* syntax: `load <filename>,<address>:<region>.m[,<length>]`

Examples:

`loadr data.bin,200,100,:monitor`
Loads `:monitor` region addresses 200-2ff from the binary file **data.bin**.

`loadr cpurom.bin,1000,400,.`
Loads addresses 1000-13ff in the memory region with the same tag as the visible CPU from the binary file **cpurom.bin**.

Back to *Memory Debugger Commands*

map

map[{d|i|o}] <address>[:<space>]

Map a logical memory address to the corresponding physical address, as well as reporting the handler name. The address may optionally be followed by a colon and device and/or address space (see *Specifying devices and address spaces* for details). If an address space is not specified, the command suffix sets the address space: `map` defaults to the first address space exposed by the device, `mapd` defaults to the space with index 1 (data), `mapi` default to the space with index 2 (I/O), and `mapo` defaults to the space with index 3 (opcodes).

Examples:

`map 152d0`
Gives the physical address and handler name for logical address 152d0 in program memory for the visible CPU.

`map 107:sms_vdp`
Gives the physical address and handler name for logical address 107 in the first address space for the device with the absolute tag path `:sms_vdp`.

Back to *Memory Debugger Commands*

memdump

memdump [<filename>,[<device>]]

Dumps the current memory maps to the file specified by **<filename>**, or **memdump.log** if omitted. If **<device>** is specified (see *Specifying devices and address spaces*), only memory maps for the part of the device tree rooted at this device will be dumped.

Examples:

`memdump mylog.log`
Dumps memory maps for all devices in the system to the file **mylog.log**.

`memdump`
Dumps memory maps for all devices in the system to the file **memdump.log**.

`memdump audiomaps.log,audiopcb`
: Dumps memory maps for the device with the absolute tag path `:audiopcb` and all its child devices to the file **audiomaps.log**.

`memdump mylog.log,1`
: Dumps memory maps for the second CPU device in the system (zero-based index) and all its child devices to the file **mylog.log**.

Back to *Memory Debugger Commands*

8.2.3 Execution Debugger Commands

step
: single step for <count> instructions (F11)

over
: single step over <count> instructions (F10)

out
: single step until the current subroutine/exception handler returns (Shift-F11)

go
: resume execution (F5)

gbt
: resume execution until next true branch is executed

gbf
: resume execution until next false branch is executed

gex
: resume execution until exception is raised

gint
: resume execution until interrupt is taken (F7)

gni
: resume execution until next further instruction

gtime
: resume execution until the given delay has elapsed

gvblank
: resume execution until next vertical blanking interval (F8)

next
: resume execution until the next CPU switch (F6)

focus
: focus debugger only on <CPU>

ignore
: stop debugging on <CPU>

observe
: resume debugging on <CPU>

trace
: trace the specified CPU to a file

traceover
: trace the specified CPU to a file skipping subroutines

traceflush
: flush all open trace files.

step

s[tep] [<count>]

Single steps one or more instructions on the currently executing CPU. Executes one instruction if **<count>** is omitted, or steps **<count>** instructions if it is supplied.

Examples:

`s`
: Steps forward one instruction on the current CPU.

`step 4`
: Steps forward four instructions on the current CPU.

Back to *Execution Debugger Commands*

over

o[ver] [<count>]

The over command single steps "over" one or more instructions on the currently executing CPU, stepping over subroutine calls and exception handler traps and counting them as a single instruction. Note that when stepping over a subroutine call, code may execute on other CPUs before the subroutine returns.

Steps over one instruction if **<count>** is omitted, or steps over **<count>** instructions if it is supplied.

Note that the step over functionality may not be implemented for all CPU types. If it is not implemented, then `over` will behave exactly like *step*.

Examples:

`o`
: Steps forward over one instruction on the current CPU.

`over 4`
: Steps forward over four instructions on the current CPU.

Back to *Execution Debugger Commands*

out

out

Single steps until a return from subroutine or return from exception instruction is encountered. Note that because it detects return from exception conditions, if you attempt to step out of a subroutine and an interrupt/exception occurs before the subroutine completes, execution may stop prematurely at the end of the exception handler.

Note that the step out functionality may not be implemented for all CPU types. If it is not implemented, then `out` will behave exactly like *step*.

Example:

`out`
: Steps until a subroutine or exception handler returns.

Back to *Execution Debugger Commands*

go

g[o] [<address>]

Resumes execution. Control will not be returned to the debugger until a breakpoint or watchpoint is triggered, or a debugger break is manually requested. If the optional **<address>** is supplied, a temporary unconditional breakpoint will be set for the visible CPU at the specified address. It will be cleared automatically when triggered.

Examples:

`g`
: Resume execution until a breakpoint/watchpoint is triggered, or a break is manually requested.

`g 1234`
: Resume execution, stopping at address 1234, unless another condition causes execution to stop before then.

Back to *Execution Debugger Commands*

gbf

gbf [<condition>]

Resumes execution. Control will not be returned to the debugger until a breakpoint or watchpoint is triggered, or until a conditional branch or skip instruction is not taken, following any delay slots.

The optional **<condition>** parameter lets you specify an expression that will be evaluated each time a conditional branch is encountered. If the result of the expression is true (non-zero), execution will be halted after the branch if it is not taken; otherwise, execution will continue with no notification.

Examples:

`gbf`
: Resume execution until a breakpoint/watchpoint is triggered, or until the next false branch.

`gbf {pc != 1234}`
: Resume execution until the next false branch, disregarding the instruction at address 1234.

Back to *Execution Debugger Commands*

gbt

gbt [<condition>]

Resumes execution. Control will not be returned to the debugger until a breakpoint or watchpoint is triggered, or until a conditional branch or skip instruction is taken, following any delay slots.

The optional **<condition>** parameter lets you specify an expression that will be evaluated each time a conditional branch is encountered. If the result of the expression is true (non-zero), execution will be halted after the branch if it is taken; otherwise, execution will continue with no notification.

Examples:

`gbt`
: Resume execution until a breakpoint/watchpoint is triggered, or until the next true branch.

`gbt {pc != 1234}`
: Resume execution until the next true branch, disregarding the instruction at address 1234.

Back to *Execution Debugger Commands*

gex

ge[x] [<exception>,[<condition>]]

Resumes execution. Control will not be returned to the debugger until a breakpoint or watchpoint is triggered, or until an exception condition is raised on the current CPU. Use the optional **<exception>** parameter to stop execution only for a specific exception condition. If **<exception>** is omitted, execution will stop for any exception condition.

The optional **<condition>** parameter lets you specify an expression that will be evaluated each time the specified exception condition is raised. If the result of the expression is true (non-zero), the exception will halt execution; otherwise, execution will continue with no notification.

Examples:

`gex`
: Resume execution until a breakpoint/watchpoint is triggered, or until any exception condition is raised on the current CPU.

`ge 2`
: Resume execution until a breakpoint/watchpoint is triggered, or until exception condition 2 is raised on the current CPU.

Back to *Execution Debugger Commands*

gint

gi[nt] [<irqline>]

Resumes execution. Control will not be returned to the debugger until a breakpoint or watchpoint is triggered, or until an interrupt is asserted and acknowledged on the current CPU. Use the optional **<irqline>** parameter to stop execution only for a specific interrupt line being asserted and acknowledged. If **<irqline>** is omitted, execution will stop when any interrupt is acknowledged.

Examples:

`gi`
: Resume execution until a breakpoint/watchpoint is triggered, or any interrupt is asserted and acknowledged on the current CPU.

`gint 4`
: Resume execution until a breakpoint/watchpoint is triggered, or interrupt request line 4 is asserted and acknowledged on the current CPU.

Back to *Execution Debugger Commands*

gni

gni [<count>]

Resumes execution. Control will not be returned to the debugger until a breakpoint or watchpoint is triggered. A temporary unconditional breakpoint is set at the program address **<count>** instructions sequentially past the current one. When this breakpoint is hit, it is automatically removed.

The **<count>** parameter is optional and defaults to 1 if omitted. If **<count>** is specified as zero, the command does nothing. **<count>** is not permitted to exceed 512 decimal.

Examples:

`gni`
: Resume execution until a breakpoint/watchpoint is triggered, including the temporary breakpoint set at the address of the following instruction.

`gni 2`
: Resume execution until a breakpoint/watchpoint is triggered. A temporary breakpoint is set two instructions past the current one.

Back to *Execution Debugger Commands*

gtime

gt[ime] <milliseconds>

Resumes execution. Control will not be returned to the debugger until a specified interval of emulated time has elapsed. The interval is specified in milliseconds.

Example:

`gtime #10000`
: Resume execution for ten seconds of emulated time.

Back to *Execution Debugger Commands*

gvblank

gv[blank]

Resumes execution. Control will not be returned to the debugger until a breakpoint or watchpoint is triggered, or until the beginning of the vertical blanking interval for an emulated screen.

Example:

`gv`
: Resume execution until a breakpoint/watchpoint is triggered, or a vertical blanking interval starts.

Back to *Execution Debugger Commands*

next

n[ext]

Resumes execution until a different CPU is scheduled. Execution will not stop when a CPU is scheduled if it is ignored due to the use of *ignore* or *focus*.

Example:

`n`
: Resume execution, stopping when a different CPU that is not ignored is scheduled.

Back to *Execution Debugger Commands*

focus

focus <CPU>

Focus exclusively on to the specified **<CPU>**, ignoring all other CPUs. The **<CPU>** argument can be a device tag or debugger CPU number (see *Specifying devices and address spaces* for details). This is equivalent to using the *ignore* command to ignore all CPUs besides the specified CPU.

Examples:

`focus 1`
: Focus exclusively on the second CPU in the system (zero-based index), ignoring all other CPUs.

`focus audiopcb:melodycpu`
: Focus exclusively on the CPU with the absolute tag path `:audiopcb:melodycpu`.

Back to *Execution Debugger Commands*

ignore

ignore [<CPU>[,<CPU>[,...]]]

Ignores the specified CPUs in the debugger. CPUs can be specified by tag or debugger CPU number (see *Specifying devices and address spaces* for details). The debugger never shows execution for ignored CPUs, and breakpoints or watchpoints on ignored CPUs have no effect. If no CPUs are specified, currently ignored CPUs will be listed. Use the *observe* command to stop ignoring a CPU.

Note that you cannot ignore all CPUs; at least CPU must be observed at all times.

Examples:

`ignore audiocpu`
: Ignore the CPU with the absolute tag path `:audiocpu` when using the debugger.

`ignore 2,3,4`
: Ignore the third, fourth and fifth CPUs in the system (zero-based indices) when using the debugger.

`ignore`
: List the CPUs that are currently being ignored by the debugger.

Back to *Execution Debugger Commands*

observe

observe [<CPU>[,<CPU>[,...]]]

Allow interaction with the specified CPUs in the debugger. CPUs can be specified by tag or debugger CPU number (see *Specifying devices and address spaces* for details). This command reverses the effects of the *ignore* command. If no CPUs are specified, currently observed CPUs will be listed.

Examples:

`observe audiocpu`
: Stop ignoring the CPU with the absolute tag path `:audiocpu` when using the debugger.

`observe 2,3,4`
: Stop ignoring the third, fourth and fifth CPUs in the system (zero-based indices) when using the debugger.

`observe`
: List the CPUs that are currently being observed by the debugger.

Back to *Execution Debugger Commands*

trace

trace {<filename>|off}[,<CPU>[,[noloop|logerror][,<action>]]]

Starts or stops tracing for execution of the specified **<CPU>**, or the currently visible CPU if no CPU is specified. To enable tracing, specify the trace log file name in the **<filename>** parameter. To disable tracing, use the keyword `off` for the **<filename>** parameter. If the **<filename>** argument begins with two right angle brackets (**>>**), it is treated as a directive to open the file for appending rather than overwriting.

The optional third parameter is a flags field. The supported flags are `noloop` and `logerror`. Multiple flags must be separated by `|` (pipe) characters. By default, loops are detected and condensed to a single line. If the `noloop` flag is specified, loops will not be detected and every instruction will be logged as executed. If the `logerror` flag is specified, error log output will be included in the trace log.

The optional **<action>** parameter is a debugger command to execute before each trace message is logged. Generally, this will include a *tracelog* or *tracesym* command to include additional information in the trace log. Note that you may need to surround the action within braces `{ }` to ensure commas and semicolons within the command are not interpreted in the context of the `trace` command itself.

Examples:

`trace joust.tr`
: Begin tracing the execution of the currently visible CPU, logging output to the file **joust.tr**.

`trace dribling.tr,maincpu`
: Begin tracing the execution of the CPU with the absolute tag path `:maincpu:`, logging output to the file **dribling.tr**.

`trace starswep.tr,,noloop`
: Begin tracing the execution of the currently visible CPU, logging output to the file **starswep.tr**, with loop detection disabled.

`trace starswep.tr,1,logerror`
: Begin tracing the execution of the second CPU in the system (zero-based index), logging output along with error log output to the file **starswep.tr**.

`trace starswep.tr,0,logerror|noloop`
: Begin tracing the execution of the first CPU in the system (zero-based index), logging output along with error log output to the file **starswep.tr**, with loop detection disabled.

`trace >>pigskin.tr`
: Begin tracing execution of the currently visible CPU, appending log output to the file **pigskin.tr**.

`trace off,0`
: Turn off tracing for the first CPU in the system (zero-based index).

`trace asteroid.tr,,,{tracelog "A=%02X ",a}`
: Begin tracing the execution of the currently visible CPU, logging output to the file **asteroid.tr**. Before each line, output **A=<aval>** to the trace log.

Back to *Execution Debugger Commands*

traceover

traceover {<filename>|off}[,<CPU>[,[noloop|logerror][,<action>]]]

Starts or stops tracing for execution of the specified **<CPU>**, or the currently visible CPU if no CPU is specified. When a subroutine call is encountered, tracing will skip over the subroutine. The same algorithm is used as is used in the *step over* command. It will not work properly with recursive functions, or if the return address does not immediately follow the call instruction.

This command accepts the same parameters as the *trace* command. Please refer to the corresponding section for a detailed description of options and more examples.

Examples:

`traceover joust.tr`
: Begin tracing the execution of the currently visible CPU, logging output to the file **joust.tr**.

`traceover dribling.tr,maincpu`
: Begin tracing the execution of the CPU with the absolute tag path `:maincpu:`, logging output to the file **dribling.tr**.

`traceover starswep.tr,,noloop`
: Begin tracing the execution of the currently visible CPU, logging output to the file **starswep.tr**, with loop detection disabled.

`traceover off,0`
: Turn off tracing for the first CPU in the system (zero-based index).

`traceover asteroid.tr,,,{tracelog "A=%02X ",a}`
: Begin tracing the execution of the currently visible CPU, logging output to the file **asteroid.tr**. Before each line, output **A=<aval>** to the trace log.

Back to *Execution Debugger Commands*

traceflush

traceflush

Flushes all open trace log files to disk.

Example:

`traceflush`
: Flush trace log files.

Back to *Execution Debugger Commands*

8.2.4 Breakpoint Debugger Commands

bpset
: sets a breakpoint at <address>

bpclear
: clears a specific breakpoint or all breakpoints

bpdisable
: disables a specific breakpoint or all breakpoints

bpenable
: enables a specific breakpoint or all breakpoints

bplist
: lists breakpoints

Breakpoints halt execution and activate the debugger before a CPU executes an instruction at a particular address.

bpset

bp[set] <address>[:<CPU>][,<condition>[,<action>]]

Sets a new execution breakpoint at the specified **<address>**. The **<address>** may optionally be followed by a colon and a tag or debugger CPU number to set a breakpoint for a specific CPU. If no CPU is specified, the breakpoint will be set for the CPU currently visible in the debugger.

The optional **<condition>** parameter lets you specify an expression that will be evaluated each time the breakpoint address is hit. If the result of the expression is true (non-zero), the breakpoint will halt execution; otherwise, execution will continue with no notification. The optional **<action>** parameter provides a command to be executed whenever the breakpoint is hit and the **<condition>** is true. Note that you may need to surround the action with braces `{ }` to ensure commas and semicolons within the command are not interpreted in the context of the `bpset` command itself.

Each breakpoint that is set is assigned a numeric index which can be used to refer to it in other breakpoint commands. Breakpoint indices are unique throughout a session.

Examples:

`bp 1234`
: Set a breakpoint for the visible CPU that will halt execution whenever the PC is equal to 1234.

`bp 23456,a0 == 0 && a1 == 0`
: Set a breakpoint for the visible CPU that will halt execution whenever the PC is equal to 23456 *and* the expression `a0 == 0 && a1 == 0` is true.

`bp 3456:audiocpu,1,{ printf "A0=%08X\n",a0 ; g }`
: Set a breakpoint for the CPU with the absolute tag path `:audiocpu` that will halt execution whenever the PC is equal to 3456. When this happens, print **A0=<a0val>** to the debugger console and resume execution.

`bp 45678:2,a0==100,{ a0 = ff ; g }`
: Set a breakpoint on the third CPU in the system (zero-based index) that will halt execution whenever the PC is equal to 45678 and the expression `a0 == 100` is true. When that happens, set **a0** to ff and resume execution.

`temp0 = 0 ; bp 567890,++temp0 >= 10`
: Set a breakpoint for the visible CPU that will halt execution whenever the PC is equal to 567890 and the expression `++temp0 >= 10` is true. This effectively breaks only after the breakpoint has been hit sixteen times.

Back to *Breakpoint Debugger Commands*

bpclear

bpclear [<bpnum>[,...]]

Clear breakpoints. If **<bpnum>** is specified, the breakpoints referred to will be cleared. If **<bpnum>** is not specified, all breakpoints will be cleared.

Examples:

`bpclear 3`
: Clear the breakpoint with index 3.

`bpclear`
: Clear all breakpoints.

Back to *Breakpoint Debugger Commands*

bpdisable

bpdisable [<bpnum>[,...]]

Disable breakpoints. If **<bpnum>** is specified, the breakpoints referred to will be disabled. If **<bpnum>** is not specified, all breakpoints will be disabled.

Note that disabling a breakpoint does not delete it, it just temporarily marks the breakpoint as inactive. Disabled breakpoints will not cause execution to halt, their associated condition expressions will not be evaluated, and their associated commands will not be executed.

Examples:

`bpdisable 3`
: Disable the breakpoint with index 3.

`bpdisable`
: Disable all breakpoints.

Back to *Breakpoint Debugger Commands*

bpenable

bpenable [<bpnum>[,...]]

Enable breakpoints. If **<bpnum>** is specified, the breakpoint referred to will be enabled. If **<bpnum>** is not specified, all breakpoints will be enabled.

Examples:

`bpenable 3`
: Enable the breakpoint with index 3.

`bpenable`
: Enable all breakpoints.

Back to *Breakpoint Debugger Commands*

bplist

bplist [<CPU>]

List current breakpoints, along with their indices and any associated conditions or actions. If no **<CPU>** is specified, breakpoints for all CPUs in the system will be listed; if a **<CPU>** is specified, only breakpoints for that CPU will be listed. The **<CPU>** can be specified by tag or by debugger CPU number (see *Specifying devices and address spaces* for details).

Examples:

`bplist`
: List all breakpoints.

`bplist .`
: List all breakpoints for the visible CPU.

`bplist maincpu`
: List all breakpoints for the CPU with the absolute tag path `:maincpu`.

Back to *Breakpoint Debugger Commands*

8.2.5 Watchpoint Debugger Commands

wpset
: sets memory access watchpoints

wpclear
: clears watchpoints

wpdisable
: disables watchpoints

wpenable
: enables enables watchpoints

wplist
: lists watchpoints

Watchpoints halt execution and activate the debugger when a CPU accesses a location in a particular memory range.

wpset

wp[{d|i|o}][set] <address>[:<space>],<length>,<type>[,<condition>[,<action>]]

Sets a new watchpoint starting at the specified **<address>** and extending for **<length>**. The range of the watchpoint is **<address>** through **<address>+<length>-1**, inclusive. The **<address>** may optionally be followed by a CPU and/or address space (see *Specifying devices and address spaces* for details). If an address space is not specified, the command suffix sets the address space: `wpset` defaults to the first address space exposed by the CPU, `wpdset` defaults to the space with index 1 (data), `wpiset` defaults to the space with index 2 (I/O), and `wposet` defaults to the space with index 3 (opcodes). The **<type>** parameter specifies the access types to trap on – it can be one of three values: `r` for read accesses, `w` for write accesses, or `rw` for both read and write accesses.

The optional **<condition>** parameter lets you specify an expression that will be evaluated each time the watchpoint is triggered. If the result of the expression is true (non-zero), the watchpoint will halt execution; otherwise, execution will continue with no notification. The optional **<action>** parameter provides a command to be executed whenever the watchpoint is triggered and the **<condition>** is true. Note that you may need to surround the action with braces `{ }` to ensure commas and semicolons within the command are not interpreted in the context of the `wpset` command itself.

Each watchpoint that is set is assigned a numeric index which can be used to refer to it in other watchpoint commands. Watchpoint indices are unique throughout a session.

To make **<condition>** expressions more useful, two variables are available: for all watchpoints, the variable **wpaddr** is set to the access address that triggered the watchpoint; for write watchpoints, the variable **wpdata** is set to the data being written.

Examples:

`wp 1234,6,rw`
: Set a watchpoint for the visible CPU that will halt execution whenever a read or write to the first address space occurs in the address range 1234-1239, inclusive.

`wp 23456:data,a,w,wpdata == 1`
: Set a watchpoint for the visible CPU that will halt execution whenever a write to the `data` space occurs in the address range 23456-2345f and the data written is equal to 1.

`wp 3456:maincpu,20,r,1,{ printf "Read @ %08X\n",wpaddr ; g }`
: Set a watchpoint for the CPU with the absolute tag path `:maincpu` that will halt execution whenever a read from the first address space occurs in the address range 3456-3475. When this happens, print **Read @ <wpaddr>** to the debugger console and resume execution.

`temp0 = 0 ; wp 45678,1,w,wpdata==f0,{ temp0++ ; g }`
: Set a watchpoint for the visible CPU that will halt execution whenever a write to the first address space occurs at address 45678 where the value being written is equal to f0. When this happens, increment the variable **temp0** and resume execution.

Back to *Watchpoint Debugger Commands*

wpclear

wpclear [<wpnum>[,...]]

Clear watchpoints. If **<wpnum>** is specified, the watchpoints referred to will be cleared. If **<wpnum>** is not specified, all watchpoints will be cleared.

Examples:

`wpclear 3`
: Clear the watchpoint with index 3.

`wpclear`
: Clear all watchpoints.

Back to *Watchpoint Debugger Commands*

wpdisable

wpdisable [<wpnum>[,...]]

Disable watchpoints. If **<wpnum>** is specified, the watchpoints referred to will be disabled. If **<wpnum>** is not specified, all watchpoints will be disabled.

Note that disabling a watchpoint does not delete it, it just temporarily marks the watchpoint as inactive. Disabled watchpoints will not cause execution to halt, their associated condition expressions will not be evaluated, and their associated commands will not be executed.

Examples:

`wpdisable 3`
: Disable the watchpoint with index 3.

`wpdisable`
: Disable all watchpoints.

Back to *Watchpoint Debugger Commands*

wpenable

wpenable [<wpnum>[,...]]

Enable watchpoints. If **<wpnum>** is specified, the watchpoints referred to will be enabled. If **<wpnum>** is not specified, all watchpoints will be enabled.

Examples:

`wpenable 3`
: Enable the watchpoint with index 3.

`wpenable`
: Enable all watchpoints.

Back to *Watchpoint Debugger Commands*

wplist

wplist [<CPU>]

List current watchpoints, along with their indices and any associated conditions or actions. If no **<CPU>** is specified, watchpoints for all CPUs in the system will be listed; if a **<CPU>** is specified, only watchpoints for that CPU will be listed. The **<CPU>** can be specified by tag or by debugger CPU number (see *Specifying devices and address spaces* for details).

Examples:

`wplist`
: List all watchpoints.

`wplist .`
: List all watchpoints for the visible CPU.

`wplist maincpu`
: List all watchpoints for the CPU with the absolute tag path `:maincpu`.

Back to *Watchpoint Debugger Commands*

8.2.6 Registerpoint Debugger Commands

rpset
: sets a registerpoint to trigger on a condition

rpclear
: clears registerpoints

rpdisable
: disables a registerpoint

rpenable
: enables registerpoints

rplist
: lists registerpoints

Registerpoints evaluate an expression each time a CPU executes an instruction and halt execution and activate the debugger if the result is true (non-zero).

rpset

rp[set] <condition>[,<action>]

Sets a new registerpoint which will be triggered when the expression supplied as the **<condition>** evaluates to true (non-zero). Note that the condition may need to be surrounded with braces `{ }` to prevent it from being interpreted as an assignment. The optional **<action>** parameter provides a command to be executed whenever the registerpoint is triggered. Note that you may need to surround the action with braces `{ }` to ensure commas and semicolons within the command are not interpreted in the context of the `rpset` command itself.

Each registerpoint that is set is assigned a numeric index which can be used to refer to it in other registerpoint commands. Registerpoint indices are unique throughout a session.

Examples:

`rp {PC==150}`
: Set a registerpoint that will halt execution whenever the **PC** register equals 150.

`temp0=0; rp {PC==150},{temp0++; g}`
: Set a registerpoint that will increment the variable **temp0** whenever the **PC** register equals 150.

`rp {temp0==5}`
: Set a registerpoint that will halt execution whenever the **temp0** variable equals 5.

Back to *Registerpoint Debugger Commands*

rpclear

rpclear [<rpnum>,[,...]]

Clears registerpoints. If **<rpnum>** is specified, the registerpoints referred to will be cleared. If **<rpnum>** is not specified, all registerpoints will be cleared.

Examples:

`rpclear 3`
: Clear the registerpoint with index 3.

`rpclear`
: Clear all registerpoints.

Back to *Registerpoint Debugger Commands*

rpdisable

rpdisable [<rpnum>[,...]]

Disables registerpoints. If **<rpnum>** is specified, the registerpoints referred to will be disabled. If **<rpnum>** is not specified, all registerpoints will be disabled.

Note that disabling a registerpoint does not delete it, it just temporarily marks the registerpoint as inactive. Disabled registerpoints will not cause execution to halt, their condition expressions will not be evaluated, and their associated commands will not be executed.

Examples:

`rpdisable 3`
: Disable the registerpoint with index 3.

`rpdisable`
: Disable all registerpoints.

Back to *Registerpoint Debugger Commands*

rpenable

rpenable [<rpnum>[,...]]

Enables registerpoints. If **<rpnum>** is specified, the registerpoints referred to will be enabled. If **<rpnum>** is not specified, all registerpoints will be enabled.

Examples:

`rpenable 3`
: Enable the registerpoint with index 3.

`rpenable`
: Enable all registerpoints.

Back to *Registerpoint Debugger Commands*

rplist

rplist [<CPU>]

List current registerpoints, along with their indices and conditions, and any associated actions. If no **<CPU>** is specified, registerpoints for all CPUs in the system will be listed; if a **<CPU>** is specified, only registerpoints for that CPU will be listed. The **<CPU>** can be specified by tag or by debugger CPU number (see *Specifying devices and address spaces* for details).

Examples:

`rplist`
: List all registerpoints.

`rplist .`
: List all registerpoints for the visible CPU.

`rplist maincpu`
: List all registerpoints for the CPU with the absolute tag path `:maincpu`.

Back to *Registerpoint Debugger Commands*

8.2.7 Exceptionpoint Debugger Commands

epset
: sets a new exceptionpoint

epclear
: clears a specific exceptionpoint or all exceptionpoints

epdisable
: disables a specific exceptionpoint or all exceptionpoints

epenable
: enables a specific exceptionpoint or all exceptionpoints

eplist
: lists exceptionpoints

Exceptionpoints halt execution and activate the debugger when a CPU raises a particular exception number.

epset

ep[set] <type>[,<condition>[,<action>]]

Sets a new exceptionpoint for exceptions of type **<type>**. The optional **<condition>** parameter lets you specify an expression that will be evaluated each time the exceptionpoint is hit. If the result of the expression is true (non-zero), the exceptionpoint will actually halt execution at the start of the exception handler; otherwise, execution will continue with no notification. The optional **<action>** parameter provides a command that is executed whenever the exceptionpoint is hit and the **<condition>** is true. Note that you may need to embed the action within braces `{ }` in order to prevent commas and semicolons from being interpreted as applying to the `epset` command itself.

The numbering of exceptions depends upon the CPU type. Causes of exceptions may include internally or externally vectored interrupts, errors occurring within instructions and system calls.

Each exceptionpoint that is set is assigned an index which can be used in other exceptionpoint commands to reference this exceptionpoint.

Examples:

`ep 2`
: Set an exception that will halt execution whenever the visible CPU raises exception number 2.

`ep a, ir == a9c9`
: Set an exception that will halt execution whenever the visible CPU raises exception number 10 (the "A-line" opcode trap on a MC68000) when IR is equal to 0xa9c9.

Back to *Exceptionpoint Debugger Commands*

epclear

epclear [<epnum>[,...]]

The epclear command clears exceptionpoints. If **<epnum>** is specified, only the requested exceptionpoints are cleared, otherwise all exceptionpoints are cleared.

Examples:

`epclear 3`
: Clear exceptionpoint index 3.

`epclear`
: Clear all exceptionpoints.

Back to *Exceptionpoint Debugger Commands*

epdisable

epdisable [<epnum>[,...]]

The epdisable command disables exceptionpoints. If **<epnum>** is specified, only the requested exceptionpoints are disabled, otherwise all exceptionpoints are disabled. Note that disabling an exceptionpoint does not delete it, it just temporarily marks the exceptionpoint as inactive.

Examples:

`epdisable 3`
: Disable exceptionpoint index 3.

`epdisable`
: Disable all exceptionpoints.

Back to *Exceptionpoint Debugger Commands*

epenable

epenable [<epnum>[,...]]

The epenable command enables exceptionpoints. If **<epnum>** is specified, only the requested exceptionpoints are enabled, otherwise all exceptionpoints are enabled.

Examples:

`epenable 3`
: Enable exceptionpoint index 3.

`epenable`
: Enable all exceptionpoints.

Back to *Exceptionpoint Debugger Commands*

eplist

eplist

The eplist command lists all the current exceptionpoints, along with their index and any conditions or actions attached to them.

Back to *Exceptionpoint Debugger Commands*

8.2.8 Code Annotation Debugger Commands

comadd
: adds a comment to the disassembled code at given address

comdelete
: removes a comment from the given address

comsave
: save the current comments to file

comlist
: list comments stored in the comment file for the system

commit
: combined `comadd` and `comsave` command

comadd

comadd <address>,<comment>

Sets the specified comment for the specified address in the disassembled code for the visible CPU. This command may be abbreviated to `//`.

Examples:

`comadd 0,hello world.`
: Adds the comment "hello world." to the code at address 0.

`// 10,undocumented opcode!`
Adds the comment "undocumented opcode!" to the code at address 10.

comdelete

comdelete

Deletes the comment at the specified address for the visible CPU.

Examples:

`comdelete 10`
Deletes the comment at code address 10 for the visible CPU.

comsave

comsave

Saves the current comments to the XML comment file for the emulated system. This file will be loaded by the debugger the next time the system is run with debugging enabled. The directory for saving these files is set using the *comment_directory* option.

Examples:

`comsave`
Saves the current comments to the comment file for the system.

comlist

comlist

Reads the comments stored in the XML comment file for the emulated system and prints them to the debugger console. This command does not affect the comments for the current session, it reads the file directly. The directory for these files is set using the *comment_directory* option.

Examples:

comlist
Shows comments stored in the comment file for the system.

commit

commit <address>,<comment>

Sets the specified comment for the specified address in the disassembled code for the visible CPU, and saves comments to the file for the current emulated system (equivalent to *comadd* followed by *comsave*). This command may be abbreviated to `/*`.

Examples:

`commit 0,hello world.`
Adds the comment "hello world." to the code at address 0 for the visible CPU and saves comments.

`/* 10,undocumented opcode!`
Adds the comment "undocumented opcode!" to the code at address 10 for the visible CPU and saves comments.

8.2.9 Cheat Debugger Commands

cheatinit
initialize the cheat search to the selected memory area

cheatrange
add selected memory area to cheat search

cheatnext
filter cheat candidates by comparing to previous values

cheatnextf
filter cheat candidates by comparing to initial value

cheatlist
show the list of cheat search matches or save them to a file

cheatundo
undo the last cheat search (state only)

The debugger includes basic cheat search functionality, which works by saving the contents of memory, and then filtering locations according to how the values change.

We'll demonstrate use of the cheat search functionality to make an infinite lives cheat for Raiden (`raiden`):

- Start the game with the debugger active. Allow the game to run, insert a coin, and start a game, then break into the debugger.
- Ensure the main CPU is visible, and start a search for 8-bit unsigned values using the *cheatinit command*:

```
>cheatinit ub
36928 cheat locations initialized for NEC V30 ':maincpu' program space
```

- Allow the game to run, lose a life and break into the debugger.
- Use the *cheatnext command* to filter locations that have decreased by 1:

```
>cheatnext -,1
12 cheats found
```

- Allow the game to run, lose another life, break into the debugger, and filter locations that have decreased by 1 again:

```
>cheatnext -,1
Address=0B85 Start=03 Current=01
1 cheats found
```

- Use the *cheatlist command* to save the cheat candidate to a file:

```
>cheatlist raiden-p1-lives.xml
```

- The file now contains an XML fragment with cheat to set the candidate location to the initial value:

```
<cheat desc="Possibility 1: 00B85 (01)">
  <script state="run">
    <action>:maincpu.pb@0x00B85=0x03</action>
  </script>
</cheat>
```

cheatinit

cheatinit [[<sign>[<width>[<swap>]]],[<address>,<length>[,<space>]]]

Initialize the cheat search to writable RAM areas in the specified address space. May be abbreviated to `ci`.

The first argument specifies the data format to search for. The **<sign>** may be **u** for unsigned or **s** for signed, the **<width>** may be **b** for 8-bit (byte), **w** for 16-bit (word), **d** for 32-bit (double word), or **q** for 64-bit (quadruple word); **<swap>** may be **s** for reversed byte order. If the first argument is omitted or empty, the data format from the previous cheat search is used, or unsigned 8-bit format if this is the first cheat search.

The **<address>** specifies the address to start searching from, and the **<length>** specifies how much memory to search. If specified, writable RAM in the range **<address>** through **<address>+<length>-1**, inclusive, will be searched; otherwise, all writable RAM in the address space will be searched.

See *Specifying devices and address spaces* for details on specifying address spaces. If the address space is not specified, it defaults to the first address space exposed by the visible CPU.

Examples:

`cheatinit ub,0x1000,0x10`
: Initialize the cheat search for unsigned 8-bit values in addresses 0x1000-0x100f in the program space of the visible CPU.

`cheatinit sw,0x2000,0x1000,1`
: Initialize the cheat search for signed 16-bit values in addresses 0x2000-0x2fff in the program space of the second CPU in the system (zero-based index).

`cheatinit uds,0x0000,0x1000`
: Initialize the cheat search for unsigned 64-bit values with reverse byte order in addresses 0x0000-0x0fff in the program space of the visible CPU.

Back to *Cheat Debugger Commands*

cheatrange

cheatrange <address>,<length>

Add writable RAM areas to the cheat search. May be abbreviated to `cr`. Before using this command, the *cheatinit command* must be used to initialize the cheat search and set the address space and data format.

The **<address>** specifies the address to start searching from, and the **<length>** specifies how much memory to search. Writable RAM in the range **<address>** through **<address>+<length>-1**, inclusive, will be added to the areas to search.

Examples:

`cheatrange 0x1000,0x10`
: Add addresses 0x1000-0x100f to the areas to search for cheats.

Back to *Cheat Debugger Commands*

cheatnext

cheatnext <condition>[,<comparisonvalue>]

Filter candidates by comparing to the previous search values. If five or fewer candidates remain, they will be shown in the debugger console. May be abbreviated to cn.

Possible **<condition>** arguments:

`all`
: Use to update the last value without changing the current matches (the **<comparisonvalue>** is not used).

`equal` (`eq`)
: Without **<comparisonvalue>**, search for values that are equal to the previous search; with **<comparisonvalue>**, search for values that are equal to the **<comparisonvalue>**.

`notequal` (`ne`)
: Without **<comparisonvalue>**, search for values that are not equal to the previous search; with **<comparisonvalue>**, search for values that are not equal to the **<comparisonvalue>**.

`decrease` (`de`, `-`)
: Without **<comparisonvalue>**, search for values that have decreased since the previous search; with **<comparisonvalue>**, search for values that have decreased by the **<comparisonvalue>** since the previous search.

`increase` (`in`, `+`)
: Without **<comparisonvalue>**, search for values that have increased since the previous search; with **<comparisonvalue>**, search for values that have increased by the **<comparisonvalue>** since the previous search.

`decreaseorequal` (`deeq`)
: Search for values that have decreased or are unchanged since the previous search (the **<comparisonvalue>** is not used).

`increaseorequal` (`ineq`)
: Search for values that have increased or are unchanged since the previous search (the **<comparisonvalue>** is not used).

`smallerof` (`lt`, `<`)
: Search for values that are less than the **<comparisonvalue>** (the **<comparisonvalue>** is required).

`greaterof` (`gt`, `>`)
: Search for values that are greater than the **<comparisonvalue>** (the **<comparisonvalue>** is required).

`changedby` (`ch`, `~`)
: Search for values that have changed by the **<comparisonvalue>** since the previous search (the **<comparisonvalue>** is required).

Examples:

`cheatnext increase`
: Search for all values that have increased since the previous search.

`cheatnext decrease,1`
: Search for all values that have decreased by 1 since the previous search.

Back to *Cheat Debugger Commands*

cheatnextf

cheatnextf <condition>[,<comparisonvalue>]

Filter candidates by comparing to the initial search values. If five or fewer candidates remain, they will be shown in the debugger console. May be abbreviated to `cn`.

Possible **<condition>** arguments:

`all`
: Use to update the last value without changing the current matches (the **<comparisonvalue>** is not used).

`equal` (`eq`)
: Without **<comparisonvalue>**, search for values that are equal to the initial search; with **<comparisonvalue>**, search for values that are equal to the **<comparisonvalue>**.

`notequal` (`ne`)
: Without **<comparisonvalue>**, search for values that are not equal to the initial search; with **<comparisonvalue>**, search for values that are not equal to the **<comparisonvalue>**.

`decrease` (`de`, `-`)
: Without **<comparisonvalue>**, search for values that have decreased since the initial search; with **<comparisonvalue>**, search for values that have decreased by the **<comparisonvalue>** since the initial search.

`increase` (`in`, `+`)
: Without **<comparisonvalue>**, search for values that have increased since the initial search; with **<comparisonvalue>**, search for values that have increased by the **<comparisonvalue>** since the initial search.

`decreaseorequal` (`deeq`)
: Search for values that have decreased or are unchanged since the initial search (the **<comparisonvalue>** is not used).

`increaseorequal` (`ineq`)
: Search for values that have increased or are unchanged since the initial search (the **<comparisonvalue>** is not used).

`smallerof` (`lt`, `<`)
: Search for values that are less than the **<comparisonvalue>** (the **<comparisonvalue>** is required).

`greaterof` (`gt`, `>`)
: Search for values that are greater than the **<comparisonvalue>** (the **<comparisonvalue>** is required).

`changedby` (`ch`, `~`)
: Search for values that have changed by the **<comparisonvalue>** since the initial search (the **<comparisonvalue>** is required).

Examples:

`cheatnextf increase`
: Search for all values that have increased since the initial search.

`cheatnextf decrease,1`
: Search for all values that have decreased by 1 since the initial search.

Back to *Cheat Debugger Commands*

cheatlist

cheatlist [<filename>]

Without **<filename>**, show the current cheat matches in the debugger console; with **<filename>**, save the current cheat matches in basic XML format to the specified file. May be abbreviated to `cl`.

Examples:

`cheatlist`
Show the current matches in the console.

`cheatlist cheat.xml`
Save the current matches to the file **cheat.xml** in XML format.

Back to *Cheat Debugger Commands*

cheatundo

cheatundo

Undo filtering of cheat candidates by the most recent *cheatnext* or *cheatnextf* command. Note that the previous values *are not* rolled back. May be abbreviated to `cu`.

Examples:

`cheatundo`
Restore cheat candidates filtered out by the most recent *cheatnext* or *cheatnextf* command.

Back to *Cheat Debugger Commands*

8.2.10 Media Image Debugger Commands

images
lists all image devices and mounted media images

mount
mounts a media image file to an image device

unmount
unmounts the media image from a device

images

images

Lists the instance names for media images devices in the system and the currently mounted media images, if any. Brief instance names, as allowed for command line media options, are listed. Mounted software list items are displayed as the list name, software item short name, and part name, separated by colons; other mounted images are displayed as file names.

Example:

`images`
Lists image device instance names and mounted media.

Back to *Media Image Debugger Commands*

mount

mount <instance>,<filename>

Mounts a file on a media device. The device may be specified by its instance name or brief instance name, as allowed for command line media options.

Some media devices allow software list items to be mounted using this command by supplying the short name of the software list item in place of a filename for the **<filename>** parameter.

Examples:

`mount flop1,os1xutls.td0`
: Mount the file **os1xutls.td0** on the media device with instance name **flop1**.

`mount cart,10yard`
: Mount the software list item with short name **10yard** on the media device with instance name **cart**.

Back to *Media Image Debugger Commands*

unmount

unmount <instance>

Unmounts the mounted media image (if any) from a device. The device may be specified by its instance name or brief instance name, as allowed for command line media options.

Examples:

unmount cart
: Unmounts any media image mounted on the device with instance name **cart**.

Back to *Media Image Debugger Commands*

8.3 Specifying devices and address spaces

Many debugger commands accept parameters specifying which device to operate on. If a device is not specified explicitly, the CPU currently visible in the debugger is used. Devices can be specified by tag, or by CPU number:

- Tags are the colon-separated paths that MAME uses to identify devices within a system. You see them in options for configuring slot devices, in debugger disassembly and memory viewer source lists, and various other places within MAME's UI.
- CPU numbers are monotonically incrementing numbers that the debugger assigns to CPU-like devices within a system, starting at zero. The **cpunum** symbol holds the CPU number for the currently visible CPU in the debugger (you can see it by entering the command **print cpunum** in the debugger console).

If a tag starts with a caret (**^**) or dot (**.**), it is interpreted relative to the CPU currently visible in the debugger, otherwise it is interpreted relative to the root machine device. If a device argument is ambiguously valid as both a tag and a CPU number, it will be interpreted as a tag.

Examples:

`maincpu`
: The device with the absolute tag `:maincpu`.

`^melodypsg`
: The sibling device of the visible CPU with the tag `melodypsg`.

`.:adc`
: The child device of the visible CPU with the tag `adc`.

`2`
: The third CPU-like device in the system (zero-based index).

Commands that operate on memory extend this by allowing the device tag or CPU number to be optionally followed by an address space identifier. Address space identifiers are tag-like strings. You can see them in debugger memory viewer source lists. If the address space identifier is omitted, a default address space will be used. Usually, this is the address space that appears first for the device. Many commands have variants with **d**, **i** and **o** (data, I/O and opcodes) suffixes that default to the address spaces at indices 1, 2 and 3, respectively, as these have special significance for CPU-like devices.

In ambiguous cases, the default address space of a child device will be used rather than a specific address space.

Examples:

`ram`
: The default address space of the device with the absolute tag `:ram`, or the `ram` space of the visible CPU.

`.:io`
: The default address space of the child device of the visible CPU with the tag `io`, or the `io` space of the visible CPU.

`:program`
: The default address space of the device with the absolute tag `:program`, or the `program` space of the root machine device.

`^vdp`
: The default address space of the sibling device of the visible CPU with the tag `vdp`.

`^:data`
: The default address space of the sibling device of the visible CPU with the tag `data`, or the `data` space of the parent device of the visible CPU.

`1:rom`
: The default address space of the child device of the second CPU in the system (zero-based index) with the tag `rom`, or the `rom` space of the second CPU in the system.

`2`
: The default address space of the third CPU-like device in the system (zero-based index).

If a command takes an emulated memory address as a parameter, the address may optionally be followed by an address space specification, as described above.

Examples:

`0220`
: Address 0220 in the default address space for the visible CPU.

`0378:io`
: Address 0378 in the default address space of the device with the absolute tag `:io`, or the `io` space of the visible CPU.

`1234:.:rom`
: Address 1234 in the default address space of the child device of the visible CPU with the tag `:rom`, or the `rom` space of the visible CPU.

`1260:^vdp`
: Address 1260 in the default address space of the sibling device of the visible CPU with the tag `vdp`.

8008:^:data
Address 8008 in the default address space of the sibling device of the visible CPU with the tag `data`, or the `data` space of the parent device of the visible CPU.

9660::ram
Address 9660 in the default address space of the device with the absolute tag `:ram`, or the `ram` space of the root machine device.

1883:vram.m
Address 1883 in the memory region with the absolute tag `:vram`.

1923:sprites.s
Address 1923 in the memory share with the absolute tag `:sprites`.

The examples here include a lot of corner cases, but in general the debugger should take the most likely meaning for a device or address space specification.

8.4 Debugger expression syntax

Expressions can be used anywhere a numeric or Boolean parameter is expected. The syntax for expressions is similar to a subset of C-style expression syntax, with full operator precedence and parentheses. There are a few operators missing (notably the ternary conditional operator), and a few new ones (memory accessors).

The table below lists all the operators, ordered from highest to lowest precedence:

( )
Standard parentheses

++ --
Postfix increment/decrement

++ -- ~ ! - + b@ w@ d@ q@ b! w! d! q!
Prefix increment/decrement, binary complement, logical complement, unary identity/negation, memory access

*** / %**
Multiplication, division, modulo

+ -
Addition, subtraction

<< >>
Bitwise left/right shift

< <= > >=
Less than, less than or equal, greater than, greater than or equal

== !=
Equal, not equal

&
Bitwise intersection (and)

^
Bitwise exclusive or

|
Bitwise union (or)

&&
Logical conjunction (and)

`||`
: Logical disjunction (or)

`= *= /= %= += -= <<= >>= &= |= ^=`
: Assignment and modifying assignment

`,`
: Separate terms, function parameters

Major differences from C expression semantics:

- All numbers are unsigned 64-bit values. In particular, this means negative numbers are not possible.
- The logical conjunction and disjunction operators && and || do not have short-circuit properties – both sides of the expression are always evaluated.

8.4.1 Numbers

Literal numbers are prefixed according to their bases:

- Hexadecimal (base-16) with `$` or `0x`
- Decimal (base-10) with `#`
- Octal (base-8) with `0o`
- Binary (base-2) with `0b`
- Unprefixed numbers are hexadecimal (base-16).

Examples:

- `123` is 123 hexadecimal (291 decimal)
- `$123` is 123 hexadecimal (291 decimal)
- `0x123` is 123 hexadecimal (291 decimal)
- `#123` is 123 decimal
- `0o123` is 123 octal (83 decimal)
- `0b1001` is 1001 binary (9 decimal)
- `0b123` is invalid

8.4.2 Boolean values

Any expression that evaluates to a number can be used where a Boolean value is required. Zero is treated as false, and all non-zero values are treated as true. Additionally, the string `true` is treated as true, and the string `false` is treated as false.

An empty string may be supplied as an argument for Boolean parameters to debugger commands to use the default value, even when subsequent parameters are specified.

8.4.3 Memory accesses

The memory access prefix operators allow reading from and writing to emulated address spaces. The memory prefix operators specify the access size and whether side effects are disabled, and may optionally be preceded by an address space specification. The supported access sizes and side effect modes are as follows:

- `b` specifies an 8-bit access (byte)
- `w` specifies a 16-bit access (word)
- `d` specifies a 32-bit access (double word)
- `q` specifies a 64-bit access (quadruple word)
- `@` suppress side effects
- `!` do not suppress side effects

Suppressing side effects of a read access yields the value reading from address would, with no further effects. For example reading a mailbox with side effects disabled will not clear the pending flag, and reading a FIFO with side effects disabled will not remove an item.

For write accesses, suppressing side effects doesn't change behaviour in most cases – you want to see the effects of writing to a location. However, there are some exceptions where it is useful to separate multiple effects of a write access. For example:

- Some registers need to be written in sequence to avoid race conditions. The debugger can issue multiple writes at the same point in emulated time, so these race conditions can be avoided trivially. For example writing to the MC68HC05 output compare register high byte (OCRH) inhibits compare until the output compare register low byte (OCRL) is written to prevent race conditions. Since the debugger can write to both locations at the same instant from the emulated machine's point of view, the race condition is not usually relevant. It's more error-prone if you can accidentally set hidden state when all you really want to do is change the value, so writing to OCRH with side effects suppressed does not inhibit compare, it just changes the value in the output compare register.
- Writing to some registers has multiple effects that may be useful to separate for debugging purposes. Using the MC68HC05 as an example again, writing to OCRL changes the value in the output compare register, and also clears the output compare flag (OCF) and enables compare if it was previously inhibited by writing to OCRH. Writing to OCRL with side effects disabled just changes the value in the register without clearing OCF or enabling compare, since it's useful for debugging purposes. Writing to OCRL with side effects enabled has the additional effects.

The size may optionally be preceded by an access type specification:

- `p` or `lp` specifies a logical address defaulting to space 0 (program)
- `d` or `ld` specifies a logical address defaulting to space 1 (data)
- `i` or `li` specifies a logical address defaulting to space 2 (I/O)
- `3` or `l3` specifies a logical address defaulting to space 3 (opcodes)
- `pp` specifies a physical address defaulting to space 0 (program)
- `pd` specifies a physical address defaulting to space 1 (data)
- `pi` specifies a physical address defaulting to space 2 (I/O)
- `p3` specifies a physical address defaulting to space 3 (opcodes)
- `r` specifies direct read/write pointer access defaulting to space 0 (program)
- `o` specifies direct read/write pointer access defaulting to space 3 (opcodes)
- `m` specifies a memory region

- `s` specifies a memory share

Finally, this may be preceded by a tag and/or address space name followed by a dot (`.`).

That may seem like a lot to digest, so let's look at the simplest examples:

b@<addr>
: Refers to the byte at **<addr>** in the program space of the current CPU while suppressing side effects

b!<addr>
: Refers to the byte at **<addr>** in the program space of the current CPU, *not* suppressing side effects such as reading a mailbox clearing the pending flag, or reading a FIFO removing an item

w@<addr> and w!<addr>
: Refer to the word at **<addr>** in the program space of the current CPU, suppressing or not suppressing side effects, respectively.

d@<addr> and d!<addr>
: Refer to the double word at **<addr>** in the program space of the current CPU, suppressing or not suppressing side effects, respectively.

q@<addr> and q!<addr>
: Refer to the quadruple word at **<addr>** in the program space of the current CPU, suppressing or not suppressing side effects, respectively.

Adding access types gives additional possibilities:

dw@300
: Refers to the word at 300 in the data space of the current CPU while suppressing side effects

id@400
: Refers to the double word at 400 in the I/O space of the current CPU while suppressing side effects

ppd!<addr>
: Refers to the double word at physical address **<addr>** in the program space of the current CPU while not suppressing side effects

rw@<addr>
: Refers to the word at address **<addr>** in the program space of the current CPU using direct read/write pointer access

If we want to access an address space of a device other than the current CPU, an address space beyond the first four indices, or a memory region, we need to include a tag or name:

ramport.b@<addr>
: Refers to the byte at address **<addr>** in the `ramport` space of the current CPU

audiocpu.dw@<addr>
: Refers to the word at address **<addr>** in the data space of the CPU with absolute tag `:audiocpu`

maincpu:status.b@<addr>
: Refers to the byte at address **<addr>** in the `status` space of the CPU with the absolute tag `:maincpu`

monitor.mb@78
: Refers to the byte at 78 in the memory region with the absolute tag `:monitor`

..md@202
: Refers to the double word at address 202 in the memory region with the same tag path as the current CPU.

Some combinations are not useful. For example physical and logical addresses are equivalent for some CPUs, and direct read/write pointer accesses never have side effects. Accessing a memory region (`m` access type) requires a tag to be specified.

Memory accesses can be used as both lvalues and rvalues, so you can write `b@100 = ff` to store a byte in memory.

8.4.4 Functions

The debugger supports a number of useful utility functions in expressions.

min(<a>, <b>)
Returns the lesser of the two arguments.

max(<a>, <b>)
Returns the greater of the two arguments.

if(<cond>, <trueval>, <falseval>)
Returns **<trueval>** if **<cond>** is true (non-zero), or **<falseval>** otherwise. Note that the expressions for **<trueval>** and **<falseval>** are both evaluated irrespective of whether **<cond>** is true or false.

abs(<x>)
Reinterprets the argument as a 64-bit signed integer and returns the absolute value.

bit(<x>, <n>[, <w>])
Extracts and right-aligns a bit field **<w>** bits wide from **<x>** with least significant bit position **<n>**, counting from the least significant bit. If **<w>** is omitted, a single bit is extracted.

s8(<x>)
Sign-extends the argument from 8 bits to 64 bits (overwrites bits 8 through 63, inclusive, with the value of bit 7, counting from the least significant bit).

s16(<x>)
Sign-extends the argument from 16 bits to 64 bits (overwrites bits 16 through 63, inclusive, with the value of bit 15, counting from the least significant bit).

s32(<x>)
Sign-extends the argument from 32 bits to 64 bits (overwrites bits 32 through 63, inclusive, with the value of bit 31, counting from the least significant bit).

CHAPTER

NINE

LUA SCRIPTING INTERFACE

- *Introduction*
- *Features*
- *API reference*
- *Interactive Lua console tutorial*

9.1 Introduction

MAME provides Lua script bindings for a useful set of core functionality. This feature first appeared in version 0.148, when a minimal Lua interface was implemented. Today, the Lua interface is rich enough to let you inspect and manipulate device state, access CPU registers, read and write memory, and draw custom graphical overlays.

There are three ways to use MAME's Lua scripting capabilities:

- Using the *interactive Lua console*, enabled by the *console option*.
- By providing a script file to run using the *-autoboot_script option*. The *-autoboot_delay option* controls how long MAME waits after starting the emulated system before running the script.
- By writing *Lua plugins*. Several plugins are included with MAME.

Internally, MAME makes extensive use of Sol3 to implement Lua bindings.

The Lua API is not yet declared stable and may suddenly change without prior notice. However, we expose methods to let you know at runtime which API version you are running against, and most objects support some level of runtime introspection.

9.2 Features

The API is not yet complete, but this is a partial list of capabilities exposed to Lua scripts:

- Session information (application version, current emulated system)
- Session control (starting, pausing, resetting, stopping)
- Event hooks (on frame painting and on user events)
- Device introspection (device tree listing, memory and register enumeration)
- Screen introspection (screens listing, screen details, frame counting)

- Screen overlay drawing (text, lines, boxes on multiple screens)
- Memory read/write (8/16/32/64 bits, signed and unsigned)
- Register and state control (state enumeration, get and set)

9.3 API reference

9.3.1 Lua Common Types and Globals

- *Containers*
- *Emulator interface*

Containers

Many properties yield container wrappers. Container wrappers are cheap to create, and provide an interface that is similar to a read-only table. The complexity of operations may vary. Container wrappers usually provide most of these operations:

#c
: Get the number of items in the container.

c[k]
: Returns the item corresponding to the key `k`, or `nil` if the key is not present.

pairs(c)
: Iterate over container by key and value. The key is what you would pass to the index operator or the `get` method to get the value.

ipairs(c)
: Iterate over container by index and value. The index is what you would pass to the `at` method to get the value (this may be the same as the key for some containers).

c:empty()
: Returns a Boolean indicating whether there are no items in the container.

c:get(k)
: Returns the item corresponding to the key `k`, or `nil` if the key is not present. Usually equivalent to the index operator.

c:at(i)
: Returns the value at the 1-based index `i`, or `nil` if it is out of range.

c:find(v)
: Returns the key for item `v`, or `nil` if it is not in the container. The key is what you would pass to the index operator to get the value.

c:index_of(v)
: Returns the 1-based index for item `v`, or `nil` if it is not in the container. The index is what you would pass to the `at` method to get the value.

Emulator interface

The emulator interface `emu` provides access to core functionality. Many classes are also available as properties of the emulator interface.

Methods

emu.wait(duration, ...)
Yields for the specified duration in terms of emulated time. The duration may be specified as an *attotime* or a number in seconds. Any additional arguments are returned from the coroutine. Returns a Boolean indicating whether the duration expired normally.

All outstanding calls to `emu.wait` will return `false` immediately if a saved state is loaded or the emulation session ends. Calling this function from callbacks that are not run as coroutines will raise an error.

emu.wait_next_update(...)
Yields until the next video/UI update. Any arguments are returned from the coroutine. Calling this function from callbacks that are not run as coroutines will raise an error.

emu.wait_next_frame(...)
Yields until the next emulated frame completes. Any arguments are returned from the coroutine. Calling this function from callbacks that are not run as coroutines will raise an error.

emu.add_machine_reset_notifier(callback)
Add a callback to receive notifications when the emulated system is reset. Returns a *notifier subscription*.

emu.add_machine_stop_notifier(callback)
Add a callback to receive notifications when the emulated system is stopped. Returns a *notifier subscription*.

emu.add_machine_pause_notifier(callback)
Add a callback to receive notifications when the emulated system is paused. Returns a *notifier subscription*.

emu.add_machine_resume_notifier(callback)
Add a callback to receive notifications when the emulated system is resumed after being paused. Returns a *notifier subscription*.

emu.add_machine_frame_notifier(callback)
Add a callback to receive notifications when an emulated frame completes. Returns a *notifier subscription*.

emu.add_machine_pre_save_notifier(callback)
Add a callback to receive notification before the emulated system state is saved. Returns a *notifier subscription*.

emu.add_machine_post_load_notifier(callback)
Add a callback to receive notification after the emulated system is restored to a previously saved state. Returns a *notifier subscription*.

emu.register_sound_update(callback)
Add a callback to receive new samples that have been created. The samples are coming from the sound devices for which the hook property has been set to true. The callback gets one parameter which is a hash with device tag as key and a (channel-sized) vector of (buffer-sized) vector of samples in the -1..1 range.

emu.print_error(message)
Print an error message.

emu.print_warning(message)
Print a warning message.

emu.print_info(message)
Print an informational message.

emu.print_verbose(message)
Print a verbose diagnostic message (disabled by default).

emu.print_debug(message)
Print a debug message (only enabled for debug builds by default).

emu.lang_translate([context], message)
Look up a message with optional context in the current localised message catalog. Returns the message unchanged if no corresponding localised message is found.

emu.subst_env(string)
Substitute environment variables in a string. The syntax is dependent on the host operating system.

9.3.2 Lua Core Classes

Many of MAME's core classes used to implement an emulation session are available to Lua scripts.

- *Notifier subscription*
- *Attotime*
- *Output proxy*
- *MAME machine manager*
- *Running machine*
- *Video manager*
- *Sound manager*
- *Output manager*
- *Parameters manager*
- *UI manager*
- *System driver metadata*
- *Lua plugin*

Notifier subscription

Wraps MAME's `util::notifier_subscription` class, which manages a subscription to a broadcast notification.

Methods

subscription:unsubscribe()
Unsubscribes from notifications. The subscription will become inactive and no future notifications will be received.

Properties

subscription.is_active (read-only)
A Boolean indicating whether the subscription is active. A subscription becomes inactive after explicitly unsubscribing or if the underlying notifier is destroyed.

Attotime

Wraps MAME's `attotime` class, which represents a high-precision time interval. Attotime values support addition and subtraction with other attotime values, and multiplication and division by integers.

Instantiation

emu.attotime()
Creates an attotime value representing zero (i.e. no elapsed time).

emu.attotime(seconds, attoseconds)
Creates an attotime with the specified whole and fractional parts.

emu.attotime(attotime)
Creates a copy of an existing attotime value.

emu.attotime.from_double(seconds)
Creates an attotime value representing the specified number of seconds.

emu.attotime.from_ticks(periods, frequency)
Creates an attotime representing the specified number of periods of the specified frequency in Hertz.

emu.attotime.from_seconds(seconds)
Creates an attotime value representing the specified whole number of seconds.

emu.attotime.from_msec(milliseconds)
Creates an attotime value representing the specified whole number of milliseconds.

emu.attotime.from_usec(microseconds)
Creates an attotime value representing the specified whole number of microseconds.

emu.attotime.from_nsec(nanoseconds)
Creates an attotime value representing the specified whole number of nanoseconds.

Methods

t:as_double()
Returns the time interval in seconds as a floating-point value.

t:as_hz()
Interprets the interval as a period and returns the corresponding frequency in Hertz as a floating-point value. Returns zero if `t.is_never` is true. The interval must not be zero.

t:as_khz()
Interprets the interval as a period and returns the corresponding frequency kilohertz as a floating-point value. Returns zero if `t.is_never` is true. The interval must not be zero.

t:as_mhz()
Interprets the interval as a period and returns the corresponding frequency megahertz as a floating-point value. Returns zero if `t.is_never` is true. The interval must not be zero.

t:as_ticks(frequency)
Returns the interval as a whole number of periods at the specified frequency. The frequency is specified in Hertz.

Properties

t.is_zero (read-only)
A Boolean indicating whether the value represents no elapsed time.

t.is_never (read-only)
A Boolean indicating whether the value is greater than the maximum number of whole seconds that can be represented (treated as an unreachable time in the future or overflow).

t.attoseconds (read-only)
The fraction seconds portion of the interval in attoseconds.

t.seconds (read-only)
The number of whole seconds in the interval.

t.msec (read-only)
The number of whole milliseconds in the fractional seconds portion of the interval.

t.usec (read-only)
The number of whole microseconds in the fractional seconds portion of the interval.

t.nsec (read-only)
The number of whole nanoseconds in the fractional seconds portion of the interval.

Output proxy

Wraps MAME's `output_proxy` class, which can be used to get or set the value of an output.

Instantiation

manager.machine.devices[tag]:output(name)
Gets a proxy to an output by name relative to a device. The output will not be created if it does not already exist.

Methods

output:exists()
Returns a Boolean indicating whether the output exists.

output:name()
Returns the output's name if it exists, or `nil` otherwise.

output:get()
Returns the current value of the output if it exists, or a value stored by the output proxy if the output does no exist.

output:set(val)
Sets the value of the output if it exists, or stores the value in the output proxy if the output does not exist. Note that if the output does not exist, the new value will not be reflected by other output proxies created with the same device and name

MAME machine manager

Wraps MAME's `mame_machine_manager` class, which holds the running machine, UI manager, and other global components.

Instantiation

manager
The MAME machine manager is available as a global variable in the Lua environment.

Properties

manager.machine (read-only)
The *running machine* for the current emulation session.

manager.ui (read-only)
The *UI manager* for the current session.

manager.options (read-only)
The emulation options for the current session.

manager.plugins[] (read-only)
Gets information about the *Lua plugins* that are present, indexed by name. The index get, `at` and `index_of` methods have O(n) complexity.

Running machine

Wraps MAME's `running_machine` class, which represents an emulation session. It provides access to the other core objects that implement an emulation session as well as the emulated device tree.

Instantiation

manager.machine
Gets the running machine instance for the current emulation session.

Methods

machine:exit()
Schedules an exit from the current emulation session. This will either return to the system selection menu or exit the application, depending on how it was started. This method returns immediately, before the scheduled exit takes place.

machine:hard_reset()
Schedules a hard reset. This is implemented by tearing down the emulation session and starting another emulation session for the same system. This method returns immediately, before the scheduled reset takes place.

machine:soft_reset()
Schedules a soft reset. This is implemented by calling the reset method of the root device, which is propagated down the device tree. This method returns immediately, before the scheduled reset takes place.

machine:save(filename)
Schedules saving machine state to the specified file. If the file name is a relative path, it is considered to be relative to the first configured save state directory. This method returns immediately, before the machine state is saved. If this method is called when a save or load operation is already pending, the previously pending operation will be cancelled.

machine:load(filename)
Schedules loading machine state from the specified file. If the file name is a relative path, the configured save state directories will be searched. This method returns immediately, before the machine state is saved. If this method is called when a save or load operation is already pending, the previously pending operation will be cancelled.

machine:popmessage([msg])
Displays a pop-up message to the user. If the message is not provided, the currently displayed pop-up message (if any) will be hidden.

machine:logerror(msg)
Writes the message to the machine error log. This may be displayed in a debugger window, written to a file, or written to the standard error output.

machine:side_effects_disabled()
Returns a Boolean indicating whether side effects are disabled.

Properties

machine.time (read-only)
The elapsed emulated time for the current session as an *attotime*.

machine.system (read-only)
The *driver metadata* for the current system.

machine.parameters (read-only)
The *parameters manager* for the current emulation session.

machine.video (read-only)
The *video manager* for the current emulation session.

machine.sound (read-only)
The *sound manager* for the current emulation session.

machine.output (read-only)
The *output manager* for the current emulation session.

machine.memory (read-only)
The *emulated memory manager* for the current session.

machine.ioport (read-only)
The *I/O port manager* for the current emulation session.

machine.input (read-only)
The *input manager* for the current emulation session.

machine.natkeyboard (read-only)
Gets the *natural keyboard manager*, used for controlling keyboard and keypad input to the emulated system.

machine.uiinput (read-only)
The *UI input manager* for the current emulation session.

machine.render (read-only)
The *render manager* for the current emulation session.

machine.debugger (read-only)
The *debugger manager* for the current emulation session, or `nil` if the debugger is not enabled.

machine.options (read-only)
The user-specified options for the current emulation session.

machine.samplerate (read-only)
The output audio sample rate in Hertz.

machine.paused (read-only)
A Boolean indicating whether emulation is not currently running, usually because the session has been paused or the emulated system has not completed starting.

machine.exit_pending (read-only)
A Boolean indicating whether the emulation session is scheduled to exit.

machine.hard_reset_pending (read-only)
A Boolean indicating whether a hard reset of the emulated system is pending.

machine.devices (read-only)
A *device enumerator* that yields all *devices* in the emulated system.

machine.palettes (read-only)
A *device enumerator* that yields all *palette devices* in the emulated system.

machine.screens (read-only)
A *device enumerator* that yields all *screen devices* in the emulated system.

machine.cassettes (read-only)
A *device enumerator* that yields all *cassette image devices* in the emulated system.

machine.cdplayers (read-only)
A *device enumerator* that yields all *audio CD player devices* in the emulated system.

machine.images (read-only)
A *device enumerator* that yields all *media image devices* in the emulated system.

machine.slots (read-only)
A *device enumerator* that yields all *slot devices* in the emulated system.

Video manager

Wraps MAME's `video_manager` class, which is responsible for coordinating emulated video drawing, speed throttling, and reading host inputs.

Instantiation

manager.machine.video
Gets the video manager for the current emulation session.

Methods

video:frame_update()
Updates emulated screens, reads host inputs, and updates video output.

video:snapshot()
Saves snapshot files according to the current configuration. If MAME is configured to take native emulated screen snapshots, one snapshot will be saved for each emulated screen that is visible in a host window/screen with the current view configuration. If MAME is not configured to use take native emulated screen snapshots or if the system has no emulated screens, a single snapshot will be saved using the currently selected snapshot view.

video:begin_recording([filename], [format])
Stops any video recordings currently in progress and starts recording either the visible emulated screens or the current snapshot view, depending on whether MAME is configured to take native emulated screen snapshots.

If the file name is not supplied, the configured snapshot file name is used. If the file name is a relative path, it is interpreted relative to the first configured snapshot directory. If the format is supplied, it must be `"avi"` or `"mng"`. If the format is not supplied, it defaults to AVI.

video:end_recording()
Stops any video recordings that are in progress.

video:snapshot_size()
Returns the width and height in pixels of snapshots created with the current snapshot target configuration and emulated screen state. This may be configured explicitly by the user, or calculated based on the selected snapshot view and the resolution of any visible emulated screens.

video:snapshot_pixels()
Returns the pixels of a snapshot created using the current snapshot target configuration as 32-bit integers packed into a binary string in host Endian order. Pixels are organised in row-major order, from left to right then top to bottom. Pixel values are colours in RGB format packed into 32-bit integers.

Properties

video.speed_factor (read-only)
Configured emulation speed adjustment in per mille (i.e. the ratio to normal speed multiplied by 1,000).

video.throttled (read/write)
A Boolean indicating whether MAME should wait before video updates to avoid running faster than the target speed.

video.throttle_rate (read/write)
The target emulation speed as a ratio of full speed adjusted by the speed factor (i.e. 1 is normal speed adjusted by the speed factor, larger numbers are faster, and smaller numbers are slower).

video.frameskip (read/write)
The number of emulated video frames to skip drawing out of every twelve, or -1 to automatically adjust the number of frames to skip to maintain the target emulation speed.

video.speed_percent (read-only)
The current emulated speed as a percentage of the full speed adjusted by the speed factor.

video.effective_frameskip (read-only)
The number of emulated frames that are skipped out of every twelve.

video.skip_this_frame (read-only)
A Boolean indicating whether the video manager will skip drawing emulated screens for the current frame.

video.snap_native (read-only)
A Boolean indicating whether the video manager will take native emulated screen snapshots. In addition to the relevant configuration setting, the emulated system must have at least one emulated screen.

video.is_recording (read-only)
A Boolean indicating whether any video recordings are currently in progress.

video.snapshot_target (read-only)
The *render target* used to produce snapshots and video recordings.

Sound manager

Wraps MAME's `sound_manager` class, which manages the emulated sound stream graph and coordinates sound output.

Instantiation

manager.machine.sound
Gets the sound manager for the current emulation session.

Methods

sound:start_recording([filename])
Starts recording to a WAV file. Has no effect if currently recording. If the file name is not supplied, uses the configured WAV file name (from command line or INI file), or has no effect if no WAV file name is configured. Returns `true` if recording started, or `false` if recording is already in progress, opening the output file failed, or no file name was supplied or configured.

sound:stop_recording()
Stops recording and closes the file if currently recording to a WAV file.

sound:get_samples()
Returns the current contents of the output sample buffer as a binary string. Samples are 16-bit integers in host byte order. Samples for left and right stereo channels are interleaved.

Properties

sound.muted (read-only)
A Boolean indicating whether sound output is muted for any reason.

sound.ui_mute (read/write)
A Boolean indicating whether sound output is muted at the request of the user.

sound.debugger_mute (read/write)
A Boolean indicating whether sound output is muted at the request of the debugger.

sound.system_mute (read/write)
A Boolean indicating whether sound output is muted at the request of the emulated system.

sound.volume (read/write)
The output volume in decibels. Should generally be a negative or zero.

sound.recording (read-only)
A Boolean indicating whether sound output is currently being recorded to a WAV file.

Output manager

Wraps MAME's `output_manager` class, providing access to system outputs that can be used for interactive artwork or consumed by external programs.

Instantiation

manager.machine.output
Gets the output manager for the current emulation session.

Parameters manager

Wraps MAME's `parameters_manager` class, which provides a simple key-value store for metadata from system ROM definitions.

Instantiation

manager.machine.parameters
Gets the parameters manager for the current emulation session.

Methods

parameters:lookup(tag)
Gets the value for the specified parameter if it is set, or an empty string if it is not set.

parameters:add(tag, value)
Sets the specified parameter if it is not set. Has no effect if the specified parameter is already set.

UI manager

Wraps MAME's `mame_ui_manager` class, which handles menus and other user interface functionality.

Instantiation

manager.ui
Gets the UI manager for the current session.

Methods

ui:get_char_width(ch)
Gets the width of a Unicode character as a proportion of the width of the UI container in the current font at the configured UI line height.

ui:get_string_width(str)
Gets the width of a string as a proportion of the width of the UI container in the current font at the configured UI line height.

ui:set_aggressive_input_focus(enable)
On some platforms, this controls whether MAME should accept input focus in more situations than when its windows have UI focus.

ui:get_general_input_setting(type, [player])
Gets a description of the configured *input sequence* for the specified input type and player suitable for using in prompts. The input type is an enumerated value. The player number is a zero-based index. If the player number is not supplied, it is assumed to be zero.

Properties

ui.options (read-only)
The UI options for the current session.

ui.line_height (read-only)
The configured UI text line height as a proportion of the height of the UI container.

ui.menu_active (read-only)
A Boolean indicating whether an interactive UI element is currently active. Examples include menus and slider controls.

ui.ui_active (read/write)
A Boolean indicating whether UI control inputs are currently enabled.

ui.single_step (read/write)
A Boolean controlling whether the emulated system should be automatically paused when the next frame is drawn. This property is automatically reset when the automatic pause happens.

ui.show_fps (read/write)
A Boolean controlling whether the current emulation speed and frame skipping settings should be displayed.

ui.show_profiler (read/write)
A Boolean controlling whether profiling statistics should be displayed.

System driver metadata

Provides some metadata for an emulated system.

Instantiation

emu.driver_find(name)
Gets the driver metadata for the system with the specified short name, or `nil` if no such system exists.

manager.machine.system
Gets the driver metadata for the current system.

Properties

driver.name (read-only)
The short name of the system, as used on the command line, in configuration files, and when searching for resources.

driver.description (read-only)
The full display name for the system.

driver.year (read-only)
The release year for the system. May contain question marks if not known definitively.

driver.manufacturer (read-only)
The manufacturer, developer or distributor of the system.

driver.parent (read-only)
The short name of parent system for organisation purposes, or `"0"` if the system has no parent.

driver.compatible_with (read-only)
The short name of a system that this system is compatible with software for, or `nil` if the system is not listed as compatible with another system.

driver.source_file (read-only)
The source file where this system driver is defined. The path format depends on the toolchain the emulator was built with.

driver.rotation (read-only)
A string indicating the rotation applied to all screens in the system after the screen orientation specified in the machine configuration is applied. Will be one of `"rot0"`, `"rot90"`, `"rot180"` or `"rot270"`.

driver.not_working (read-only)
A Boolean indicating whether the system is marked as not working.

driver.supports_save (read-only)
A Boolean indicating whether the system supports save states.

driver.no_cocktail (read-only)
A Boolean indicating whether screen flipping in cocktail mode is unsupported.

driver.is_bios_root (read-only)
A Boolean indicating whether this system represents a system that runs software from removable media without media present.

driver.requires_artwork (read-only)
A Boolean indicating whether the system requires external artwork to be usable.

driver.unofficial (read-only)
A Boolean indicating whether this is an unofficial but common user modification to a system.

driver.no_sound_hw (read-only)
A Boolean indicating whether the system has no sound output hardware.

driver.mechanical (read-only)
A Boolean indicating whether the system depends on mechanical features that cannot be properly simulated.

driver.is_incomplete (read-only)
A Boolean indicating whether the system is a prototype with incomplete functionality.

Lua plugin

Provides a description of an available Lua plugin.

Instantiation

manager.plugins[name]
Gets the description of the Lua plugin with the specified name, or `nil` if no such plugin is available

Properties

plugin.name (read-only)
The short name of the plugin, used in configuration and when accessing the plugin programmatically.

plugin.description (read-only)
The display name for the plugin.

plugin.type (read-only)
The plugin type. May be `"plugin"` for user-loadable plugins, or `"library"` for libraries providing common functionality to multiple plugins.

plugin.directory (read-only)
The path to the directory containing the plugin's files.

plugin.start (read-only)
A Boolean indicating whether the plugin enabled.

9.3.3 Lua Device Classes

Several device classes and device mix-ins classes are exposed to Lua. Devices can be looked up by tag or enumerated.

- *Device enumerators*
- *Device*
- *Palette device*
- *Screen device*
- *Vector device*
- *Cassette image device*
- *Audio CD player interface*
- *Image device interface*
- *Sound device interface*
- *Slot device interface*
- *Device state entry*
- *Media image format*
- *Slot option*

Device enumerators

Device enumerators are special containers that allow iterating over devices and looking up devices by tag. A device enumerator can be created to find any kind of device, to find devices of a particular type, or to find devices that implement a particular interface. When iterating using `pairs` or `ipairs`, devices are returned by walking the device tree depth-first in creation order.

The index get operator looks up a device by tag. It returns `nil` if no device with the specified tag is found, or if the device with the specified tag does not meet the type/interface requirements of the device enumerator. The complexity is O(1) if the result is cached, but an uncached device lookup is expensive. The `at` method has O(n) complexity.

If you create a device enumerator with a starting point other than the root machine device, passing an absolute tag or a tag containing parent references to the index operator may return a device that would not be discovered by iteration. If you create a device enumerator with restricted depth, devices that would not be found due to being too deep in the hierarchy can still be looked up by tag.

Creating a device enumerator with depth restricted to zero can be used to downcast a device or test whether a device implements a certain interface. For example this will test whether a device implements the media image interface:

```
image_intf = emu.image_enumerator(device, 0):at(1)
if image_intf then
    print(string.format("Device %s mounts images", device.tag))
end
```

Instantiation

manager.machine.devices
: Returns a device enumerator that will iterate over *devices* in the system.

manager.machine.palettes
: Returns a device enumerator that will iterate over *palette devices* in the system.

manager.machine.screens
: Returns a device enumerator that will iterate over *screen devices* in the system.

manager.machine.cassettes
: Returns a device enumerator that will iterate over *cassette image devices* in the system.

manager.machine.cdplayers
: Returns a device enumerator that will iterate over *audio CD player devices* in the system.

manager.machine.images
: Returns a device enumerator that will iterate over *media image devices* in the system.

manager.machine.slots
: Returns a device enumerator that will iterate over *slot devices* in the system.

manager.machine.sounds
: Returns a device enumerator that will iterate over *sound devices* in the system.

emu.device_enumerator(device, [depth])
: Returns a device enumerator that will iterate over *devices* in the sub-tree starting at the specified device. The specified device will be included. If the depth is provided, it must be an integer specifying the maximum number of levels to iterate below the specified device (i.e. 1 will limit iteration to the device and its immediate children).

emu.palette_enumerator(device, [depth])
: Returns a device enumerator that will iterate over *palette devices* in the sub-tree starting at the specified device. The specified device will be included if it is a palette device. If the depth is provided, it must be an integer

specifying the maximum number of levels to iterate below the specified device (i.e. 1 will limit iteration to the device and its immediate children).

emu.screen_enumerator(device, [depth])
Returns a device enumerator that will iterate over *screen devices* in the sub-tree starting at the specified device. The specified device will be included if it is a screen device. If the depth is provided, it must be an integer specifying the maximum number of levels to iterate below the specified device (i.e. 1 will limit iteration to the device and its immediate children).

emu.cassette_enumerator(device, [depth])
Returns a device enumerator that will iterate over *cassette image devices* in the sub-tree starting at the specified device. The specified device will be included if it is a cassette image device. If the depth is provided, it must be an integer specifying the maximum number of levels to iterate below the specified device (i.e. 1 will limit iteration to the device and its immediate children).

emu.cdplayer_enumerator(device, [depth])
Returns a device enumerator that will iterate over *audio CD player devices* in the sub-tree starting at the specified device. The specified device will be included if it is an audio CD player device. If the depth is provided, it must be an integer specifying the maximum number of levels to iterate below the specified device (i.e. 1 will limit iteration to the device and its immediate children).

emu.image_enumerator(device, [depth])
Returns a device enumerator that will iterate over *media image devices* in the sub-tree starting at the specified device. The specified device will be included if it is a media image device. If the depth is provided, it must be an integer specifying the maximum number of levels to iterate below the specified device (i.e. 1 will limit iteration to the device and its immediate children).

emu.slot_enumerator(device, [depth])
Returns a device enumerator that will iterate over *slot devices* in the sub-tree starting at the specified device. The specified device will be included if it is a slot device. If the depth is provided, it must be an integer specifying the maximum number of levels to iterate below the specified device (i.e. 1 will limit iteration to the device and its immediate children).

Device

Wraps MAME's `device_t` class, which is a base of all device classes.

Instantiation

manager.machine.devices[tag]
Gets a device by tag relative to the root machine device, or `nil` if no such device exists.

manager.machine.devices[tag]:subdevice(tag)
Gets a device by tag relative to another arbitrary device, or `nil` if no such device exists.

Methods

device:subtag(tag)
Converts a tag relative to the device to an absolute tag.

device:siblingtag(tag)
Converts a tag relative to the device's parent device to an absolute tag.

device:memshare(tag)
Gets a *memory share* by tag relative to the device, or `nil` if no such memory share exists.

device:membank(tag)
Gets a *memory bank* by tag relative to the device, or `nil` if no such memory bank exists.

device:memregion(tag)
Gets a *memory region* by tag relative to the device, or `nil` if no such memory region exists.

device:ioport(tag)
Gets an *I/O port* by tag relative to the device, or `nil` if no such I/O port exists.

device:output(name)
Gets an *output proxy* for the output with the specified name relative to the device. Note that the output will not be created if it does not exist.

device:subdevice(tag)
Gets a device by tag relative to the device.

device:siblingdevice(tag)
Gets a device by tag relative to the device's parent.

device:parameter(tag)
Gets a parameter value by tag relative to the device, or an empty string if the parameter is not set.

Properties

device.tag (read-only)
The device's absolute tag in canonical form.

device.basetag (read-only)
The last component of the device's tag (i.e. its tag relative to its immediate parent), or `"root"` for the root machine device.

device.name (read-only)
The full display name for the device's type.

device.shortname (read-only)
The short name of the devices type (this is used, e.g. on the command line, when looking for resource like ROMs or artwork, and in various data files).

device.owner (read-only)
The device's immediate parent in the device tree, or `nil` for the root machine device.

device.configured (read-only)
A Boolean indicating whether the device has completed configuration.

device.started (read-only)
A Boolean indicating whether the device has completed starting.

device.outputs[] (read-only)
A collection of *output proxies* for the device's outputs, indexed by name. The index operator and `index_of` methods have O(log2(n)) complexity; all other supported operations have O(1) complexity.

device.debug (read-only)
The *debugger interface* to the device if it is a CPU device, or `nil` if it is not a CPU device or the debugger is not enabled.

device.state[] (read-only)
The *state entries* for devices that expose the register state interface, indexed by symbol, or `nil` for other devices. The index operator and `index_of` methods have O(n) complexity; all other supported operations have O(1) complexity.

device.spaces[] (read-only)
A table of the device's *address spaces*, indexed by name. Only valid for devices that implement the memory interface. Note that the names are specific to the device type and have no special significance.

Palette device

Wraps MAME's `device_palette_interface` class, which represents a device that translates pen values to colours.

Colours are represented in alpha/red/green/blue (ARGB) format. Channel values range from 0 (transparent or off) to 255 (opaque or full intensity), inclusive. Colour channel values are not pre-multiplied by the alpha value. Channel values are packed into the bytes of 32-bit unsigned integers, in the order alpha, red, green, blue from most-significant to least-significant byte.

Instantiation

manager.machine.palettes[tag]
Gets a palette device by tag relative to the root machine device, or `nil` if no such device exists or it is not a palette device.

Methods

palette:pen(index)
Gets the remapped pen number for the specified palette index.

palette:pen_color(pen)
Gets the colour for the specified pen number.

palette:pen_contrast(pen)
Gets the contrast value for the specified pen number. The contrast is a floating-point number.

palette:pen_indirect(index)
Gets the indirect pen index for the specified palette index.

palette:indirect_color(index)
Gets the indirect pen colour for the specified palette index.

palette:set_pen_color(pen, color)
Sets the colour for the specified pen number. The colour may be specified as a single packed 32-bit value; or as individual red, green and blue channel values, in that order.

palette:set_pen_red_level(pen, level)
Sets the red channel value of the colour for the specified pen number. Other channel values are not affected.

palette:set_pen_green_level(pen, level)
Sets the green channel value of the colour for the specified pen number. Other channel values are not affected.

palette:set_pen_blue_level(pen, level)
Sets the blue channel value of the colour for the specified pen number. Other channel values are not affected.

palette:set_pen_contrast(pen, factor)
Sets the contrast value for the specified pen number. The value must be a floating-point number.

palette:set_pen_indirect(pen, index)
Sets the indirect pen index for the specified pen number.

palette:set_indirect_color(index, color)
Sets the indirect pen colour for the specified palette index. The colour may be specified as a single packed 32-bit value; or as individual red, green and blue channel values, in that order.

palette:set_shadow_factor(factor)
Sets the contrast value for the current shadow group. The value must be a floating-point number.

palette:set_highlight_factor(factor)
Sets the contrast value for the current highlight group. The value must be a floating-point number.

palette:set_shadow_mode(mode)
Sets the shadow mode. The value is the index of the desired shadow table.

Properties

palette.palette (read-only)
The underlying *palette* managed by the device.

palette.entries (read-only)
The number of colour entries in the palette.

palette.indirect_entries (read-only)
The number of indirect pen entries in the palette.

palette.black_pen (read-only)
The index of the fixed black pen entry.

palette.white_pen (read-only)
The index of the fixed white pen.

palette.shadows_enabled (read-only)
A Boolean indicating whether shadow colours are enabled.

palette.highlights_enabled (read-only)
A Boolean indicating whether highlight colours are enabled.

palette.device (read-only)
The underlying *device*.

Screen device

Wraps MAME's `screen_device` class, which represents an emulated video output.

Instantiation

manager.machine.screens[tag]
: Gets a screen device by tag relative to the root machine device, or `nil` if no such device exists or it is not a screen device.

Base classes

- *Device*

Methods

screen:orientation()
: Returns the rotation angle in degrees (will be one of 0, 90, 180 or 270), whether the screen is flipped left-to-right, and whether the screen is flipped top-to-bottom. This is the final screen orientation after the screen orientation specified in the machine configuration and the rotation for the system driver are applied.

screen:time_until_pos(v, [h])
: Gets the time remaining until the raster reaches the specified position. If the horizontal component of the position is not specified, it defaults to zero (0, i.e. the beginning of the line). The result is a floating-point number in units of seconds.

screen:time_until_vblank_start()
: Gets the time remaining until the start of the vertical blanking interval. The result is a floating-point number in units of seconds.

screen:time_until_vblank_end()
: Gets the time remaining until the end of the vertical blanking interval. The result is a floating-point number in units of seconds.

screen:snapshot([filename])
: Saves a screen snapshot in PNG format. If no filename is supplied, the configured snapshot path and name format will be used. If the supplied filename is not an absolute path, it is interpreted relative to the first configured snapshot path. The filename may contain conversion specifiers that will be replaced by the system name or an incrementing number.

  Returns a file error if opening the snapshot file failed, or `nil` otherwise.

screen:pixel(x, y)
: Gets the pixel at the specified location. Coordinates are in pixels, with the origin at the top left corner of the visible area, increasing to the right and down. Returns either a palette index or a colour in RGB format packed into a 32-bit integer. Returns zero (0) if the specified point is outside the visible area.

screen:pixels()
: Returns all visible pixels, the visible area width and visible area height.

  Pixels are returned as 32-bit integers packed into a binary string in host Endian order. Pixels are organised in row-major order, from left to right then top to bottom. Pixels values are either palette indices or colours in RGB format packed into 32-bit integers.

screen:draw_box(left, top, right, bottom, [line], [fill])
Draws an outlined rectangle with edges at the specified positions.

Coordinates are floating-point numbers in units of emulated screen pixels, with the origin at (0, 0). Note that emulated screen pixels often aren't square. The coordinate system is rotated if the screen is rotated, which is usually the case for vertical-format screens. Before rotation, the origin is at the top left, and coordinates increase to the right and downwards. Coordinates are limited to the screen area.

The fill and line colours are in alpha/red/green/blue (ARGB) format. Channel values are in the range 0 (transparent or off) to 255 (opaque or full intensity), inclusive. Colour channel values are not pre-multiplied by the alpha value. The channel values must be packed into the bytes of a 32-bit unsigned integer, in the order alpha, red, green, blue from most-significant to least-significant byte. If the line colour is not provided, the UI text colour is used; if the fill colour is not provided, the UI background colour is used.

screen:draw_line(x0, y0, x1, y1, [color])
Draws a line from (x0, y0) to (x1, y1).

Coordinates are floating-point numbers in units of emulated screen pixels, with the origin at (0, 0). Note that emulated screen pixels often aren't square. The coordinate system is rotated if the screen is rotated, which is usually the case for vertical-format screens. Before rotation, the origin is at the top left, and coordinates increase to the right and downwards. Coordinates are limited to the screen area.

The line colour is in alpha/red/green/blue (ARGB) format. Channel values are in the range 0 (transparent or off) to 255 (opaque or full intensity), inclusive. Colour channel values are not pre-multiplied by the alpha value. The channel values must be packed into the bytes of a 32-bit unsigned integer, in the order alpha, red, green, blue from most-significant to least-significant byte. If the line colour is not provided, the UI text colour is used.

screen:draw_text(x|justify, y, text, [foreground], [background])
Draws text at the specified position. If the screen is rotated the text will be rotated.

If the first argument is a number, the text will be left-aligned at this X coordinate. If the first argument is a string, it must be `"left"`, `"center"` or `"right"` to draw the text left-aligned at the left edge of the screen, horizontally centred on the screen, or right-aligned at the right edge of the screen, respectively. The second argument specifies the Y coordinate of the maximum ascent of the text.

Coordinates are floating-point numbers in units of emulated screen pixels, with the origin at (0, 0). Note that emulated screen pixels often aren't square. The coordinate system is rotated if the screen is rotated, which is usually the case for vertical-format screens. Before rotation, the origin is at the top left, and coordinates increase to the right and downwards. Coordinates are limited to the screen area.

The foreground and background colours are in alpha/red/green/blue (ARGB) format. Channel values are in the range 0 (transparent or off) to 255 (opaque or full intensity), inclusive. Colour channel values are not pre-multiplied by the alpha value. The channel values must be packed into the bytes of a 32-bit unsigned integer, in the order alpha, red, green, blue from most-significant to least-significant byte. If the foreground colour is not provided, the UI text colour is used; if the background colour is not provided, it is fully transparent.

Properties

screen.width (read-only)
The width of the bitmap produced by the emulated screen in pixels.

screen.height (read-only)
The height of the bitmap produced by the emulated screen in pixels.

screen.refresh (read-only)
The screen's configured refresh rate in Hertz (this may not reflect the current value).

screen.refresh_interval (read-only)
The screen's configured refresh interval as an *attotime* (this may not reflect the current value).

screen.xoffset (read-only)
The screen's default X position offset. This is a floating-point number where one (1) corresponds to the X size of the screen's container. This may be useful for restoring the default after adjusting the X offset via the screen's container.

screen.yoffset (read-only)
The screen's default Y position offset. This is a floating-point number where one (1) corresponds to the Y size of the screen's container. This may be useful for restoring the default after adjusting the Y offset via the screen's container.

screen.xscale (read-only)
The screen's default X scale factor, as a floating-point number. This may be useful for restoring the default after adjusting the X scale via the screen's container.

screen.yscale (read-only)
The screen's default Y scale factor, as a floating-point number. This may be useful for restoring the default after adjusting the Y scale via the screen's container.

screen.pixel_period (read-only)
The interval taken to draw a horizontal pixel, as a floating-point number in units of seconds.

screen.scan_period (read-only)
The interval taken to draw a scan line (including the horizontal blanking interval), as a floating-point number in units of seconds.

screen.frame_period (read-only)
The interval taken to draw a complete frame (including blanking intervals), as a floating-point number in units of seconds.

screen.frame_number (read-only)
The current frame number for the screen. This increments monotonically each frame interval.

screen.container (read-only)
The *render container* used to draw the screen.

screen.palette (read-only)
The *palette device* used to translate pixel values to colours, or `nil` if the screen uses a direct colour pixel format.

Vector device

Wraps MAME's `vector_device` class, which represents an emulated X/Y vector display.

Instantiation

manager.machine.vector_devices[tag]
Gets a vector device by tag relative to the root machine device, or `nil` if no such device exists or it is not a vector device.

Base classes

- *Device*

Methods

vector:add_frame_begin_notifier(callback)
Add a callback to receive notifications when the processing of vectors begins. The callback is not passed any arguments. Returns a *notifier subscription*.

vector:add_frame_end_notifier(callback)
Add a callback to receive notifications when the processing of vectors begins. The callback is not passed any arguments. Returns a *notifier subscription*.

vector:add_line_notifier(callback)
Add a callback to receive notifications when the vector beam moves between two points with a non-zero beam intensity. The callback is passed the following arguments, in order:

- **lastx:** The old horizontal beam position.
- **lasty:** The old vertical beam position.
- **x:** The current horizontal beam position.
- **y:** The current vertical beam position.
- **color:** The beam color, as a 24-bit RGB value.
- **intensity:** The beam intensity, as an 8-bit value.
- **width:** The configured pixel width of the virtual vector display.
- **height:** The configured pixel width of the virtual vector display.

Returns a *notifier subscription*.

vector:add_move_notifier(callback)
Add a callback to receive notifications when the vector beam moves with an intensity of zero. Provided as a convenience feature as the previous position and intensity of the beam is not meaningful when "jumping" the vector beam. The callback is passed the following arguments, in order:

- **x:** The current horizontal beam position.
- **y:** The current vertical beam position.
- **color:** The beam color, as a 24-bit RGB value.
- **width:** The configured pixel width of the virtual vector display.
- **height:** The configured pixel width of the virtual vector display.

Returns a *notifier subscription*.

Cassette image device

Wraps MAME's `cassette_image_device` class, representing a compact cassette mechanism typically used by a home computer for program storage.

Instantiation

manager.machine.cassettes[tag]
Gets a cassette image device by tag relative to the root machine device, or `nil` if no such device exists or it is not a cassette image device.

Base classes

- *Device*
- *Image device interface*

Methods

cassette:stop()
Disables playback.

cassette:play()
Enables playback. The cassette will play if the motor is enabled.

cassette:forward()
Sets forward play direction.

cassette:reverse()
Sets reverse play direction.

cassette:seek(time, whence)
Jump to the specified position on the tape. The time is a floating-point number in units of seconds, relative to the point specified by the whence argument. The whence argument must be one of `"set"`, `"cur"` or `"end"` to seek relative to the start of the tape, the current position, or the end of the tape, respectively.

Properties

cassette.is_stopped (read-only)
A Boolean indicating whether the cassette is stopped (i.e. not recording and not playing).

cassette.is_playing (read-only)
A Boolean indicating whether playback is enabled (i.e. the cassette will play if the motor is enabled).

cassette.is_recording (read-only)
A Boolean indicating whether recording is enabled (i.e. the cassette will record if the motor is enabled).

cassette.motor_state (read/write)
A Boolean indicating whether the cassette motor is enabled.

cassette.speaker_state (read/write)
A Boolean indicating whether the cassette speaker is enabled.

cassette.position (read-only)
The current position as a floating-point number in units of seconds relative to the start of the tape.

cassette.length (read-only)
The length of the tape as a floating-point number in units of seconds, or zero (0) if no tape image is mounted.

Audio CD player interface

Wraps MAME's `device_cd_player_interface` class which is a mix-in implemented by devices containing a hand-operated audio CD player whose output is fed to the emulated system, for example through a cassette or joystick port adapter.

Instantiation

manager.machine.cdplayers[tag]
Gets an audio CD player device by tag relative to the root machine device, or `nil` if no such device exists or it does not contain an audio CD player.

Methods

cdplayer:play()
Starts or resumes playback from the current position.

cdplayer:pause()
Pauses playback, keeping the current position.

cdplayer:stop()
Stops playback and returns to the start of the current track.

cdplayer:previous_track()
Returns to the start of the current track if playback is more than two seconds into it, otherwise moves to the start of the previous track.

cdplayer:next_track()
Moves to the start of the next track.

Properties

cdplayer.is_stopped (read-only)
A Boolean indicating whether the player is stopped.

cdplayer.is_playing (read-only)
A Boolean indicating whether the player is playing.

cdplayer.is_paused (read-only)
A Boolean indicating whether the player is paused.

cdplayer.track (read/write)
The current track number, starting at one. Setting it moves to the start of that track without changing whether the player is playing.

cdplayer.track_count (read-only)
The number of tracks on the disc, or zero (0) if no disc image is mounted.

cdplayer.position (read-only)
The current position as a floating-point number in units of seconds relative to the start of the current track.

cdplayer.length (read-only)
The length of the current track as a floating-point number in units of seconds, or zero (0) if no disc image is mounted.

cdplayer.device (read-only)
The underlying *device*.

cdplayer.image (read-only)
The *CD image device* holding the disc.

Image device interface

Wraps MAME's `device_image_interface` class which is a mix-in implemented by devices that can load media image files.

Instantiation

manager.machine.images[tag]
Gets an image device by tag relative to the root machine device, or `nil` if no such device exists or it is not a media image device.

Methods

image:load(filename)
Loads the specified file as a media image. Returns `nil` if no error or a string describing an error if an error occurred.

image:load_software(name)
Loads a media image described in a software list. Returns `nil` if no error or a string describing an error if an error occurred.

image:unload()
Unloads the mounted image.

image:create(filename)
Creates and mounts a media image file with the specified name. Returns `nil` if no error or a string describing an error if an error occurred.

image:display()
Returns a "front panel display" string for the device, if supported. This can be used to show status information, like the current head position or motor state.

image:add_media_change_notifier(callback)
Add a callback to receive notifications when a media image is loaded or unloaded for the device. The callback is passed a single string argument which will be `"loaded"` if a media image has been loaded or `"unloaded"` if the previously loaded media image has been unloaded. Returns a *notifier subscription*.

Properties

image.is_readable (read-only)
A Boolean indicating whether the device supports reading.

image.is_writeable (read-only)
A Boolean indicating whether the device supports writing.

image.must_be_loaded (read-only)
A Boolean indicating whether the device requires a media image to be loaded in order to start.

image.is_reset_on_load (read-only)
A Boolean indicating whether the device requires a hard reset to change media images (usually for cartridge slots that contain hardware in addition to memory chips).

image.image_type_name (read-only)
A string for categorising the media device.

image.instance_name (read-only)
The instance name of the device in the current configuration. This is used for setting the media image to load on the command line or in INI files. This is not stable, it may have a number appended that may change depending on slot configuration.

image.brief_instance_name (read-only)
The brief instance name of the device in the current configuration. This is used for setting the media image to load on the command line or in INI files. This is not stable, it may have a number appended that may change depending on slot configuration.

image.formatlist[] (read-only)
The *media image formats* supported by the device, indexed by name. The index operator and `index_of` methods have O(n) complexity; all other supported operations have O(1) complexity.

image.exists (read-only)
A Boolean indicating whether a media image file is mounted.

image.readonly (read-only)
A Boolean indicating whether a media image file is mounted in read-only mode.

image.filename (read-only)
The full path to the mounted media image file, or `nil` if no media image is mounted.

image.crc (read-only)
The 32-bit cyclic redundancy check of the content of the mounted image file if the mounted media image was not loaded from a software list, is mounted read-only and is not a CD-ROM, or zero (0) otherwise.

image.loaded_through_softlist (read-only)
A Boolean indicating whether the mounted media image was loaded from a software list, or `false` if no media image is mounted.

image.software_list_name (read-only)
The short name of the software list if the mounted media image was loaded from a software list.

image.software_longname (read-only)
The full name of the software item if the mounted media image was loaded from a software list, or `nil` otherwise.

image.software_publisher (read-only)
The publisher of the software item if the mounted media image was loaded from a software list, or `nil` otherwise.

image.software_year (read-only)
The release year of the software item if the mounted media image was loaded from a software list, or `nil` otherwise.

image.software_parent (read-only)
The short name of the parent software item if the mounted media image was loaded from a software list, or `nil` otherwise.

image.device (read-only)
The underlying *device*.

Sound device interface

Wraps MAME's `device_sound_interface` class which is a mix-in implemented by devices that input and/or output sound.

Instantiation

manager.machine.sounds[tag]
Gets an sound device by tag relative to the root machine device, or `nil` if no such device exists or it is not a slot device.

Properties

sound.inputs (read-only)
Number of sound inputs of the device.

sound.outputs (read-only)
Number of sound outputs of the device.

sound.microphone (read-only)
True if the device is a microphone, false otherwise

sound.speaker (read-only)
True if the device is a speaker, false otherwise

sound.io_positions[] (read-only)
Non-empty only for microphones and speakers, indicates the positions of the inputs or outputs as (x, y, z) coordinates (e.g. [-0.2, 0.0, 1.0])

sound.io_names[] (read-only)
Non-empty only for microphones and speakers, indicates the positions of the inputs or outputs as strings (e.g. Front Left)

sound.hook
A boolean indicating whether to tap the output samples of this device in the global sound hook.

sound.device (read-only)
The underlying *device*.

Slot device interface

Wraps MAME's `device_slot_interface` class which is a mix-in implemented by devices that instantiate a user-specified child device.

Instantiation

manager.machine.slots[tag]
Gets an slot device by tag relative to the root machine device, or `nil` if no such device exists or it is not a slot device.

Properties

slot.fixed (read-only)
A Boolean indicating whether this is a slot with a card specified in machine configuration that cannot be changed by the user.

slot.has_selectable_options (read-only)
A Boolean indicating whether the slot has any user-selectable options (as opposed to options that can only be selected programmatically, typically for fixed slots or to load media images).

slot.options[] (read-only)
The *slot options* describing the child devices that can be instantiated by the slot, indexed by option value. The `at` and `index_of` methods have O(n) complexity; all other supported operations have O(1) complexity.

slot.device (read-only)
The underlying *device*.

Device state entry

Wraps MAME's `device_state_entry` class, which allows access to named registers exposed by a *device*. Supports conversion to string for display.

Instantiation

manager.machine.devices[tag].state[symbol]
Gets a state entry for a given device by symbol.

Properties

entry.value (read/write)
The numeric value of the state entry, as either an integer or floating-point number. Attempting to set the value of a read-only state entry raises an error.

entry.symbol (read-only)
The state entry's symbolic name.

entry.visible (read-only)
A Boolean indicating whether the state entry should be displayed in the debugger register view.

entry.writeable (read-only)
A Boolean indicating whether it is possible to modify the state entry's value.

entry.is_float (read-only)
A Boolean indicating whether the state entry's value is a floating-point number.

entry.datamask (read-only)
A bit mask of the valid bits of the value for integer state entries.

entry.datasize (read-only)
The size of the underlying value in bytes for integer state entries.

entry.max_length (read-only)
The maximum display string length for the state entry.

Media image format

Wraps MAME's `image_device_format` class, which describes a media file format supported by a *media image device*.

Instantiation

manager.machine.images[tag].formatlist[name]
Gets a media image format supported by a given device by name.

Properties

format.name (read-only)
An abbreviated name used to identify the format. This often matches the primary filename extension used for the format.

format.description (read-only)
The full display name of the format.

format.extensions[] (read-only)
Yields a table of filename extensions used for the format.

format.option_spec (read-only)
A string describing options available when creating a media image using this format. The string is not intended to be human-readable.

Slot option

Wraps MAME's `device_slot_interface::slot_option` class, which represents a child device that a *slot device* can be configured to instantiate.

Instantiation

manager.machine.slots[tag].options[name]
Gets a slot option for a given *slot device* by name (i.e. the value used to select the option).

Properties

option.name (read-only)
The name of the slot option. This is the value used to select this option on the command line or in an INI file.

option.device_fullname (read-only)
The full display name of the device type instantiated by this option.

option.device_shortname (read-only)
The short name of the device type instantiated by this option.

option.selectable (read-only)
A Boolean indicating whether the option may be selected by the user (options that are not user-selectable are typically used for fixed slots or to load media images).

option.default_bios (read-only)
The default BIOS setting for the device instantiated using this option, or `nil` if the default BIOS specified in the device's ROM definitions will be used.

option.clock (read-only)
The configured clock frequency for the device instantiated using this option. This is an unsigned 32-bit integer. If the eight most-significant bits are all set, it is a ratio of the parent device's clock frequency, with the numerator in bits 12-23 and the denominator in bits 0-11. If the eight most-significant bits are not all set, it is a frequency in Hertz.

9.3.4 Lua Memory System Classes

MAME's Lua interface exposes various memory system objects, including address spaces, memory shares, memory banks, and memory regions. Scripts can read from and write to the emulated memory system.

- *Memory manager*
- *Address space*
- *Pass-through handler*
- *Address map*
- *Address map entry*
- *Address map handler data*
- *Memory share*
- *Memory bank*
- *Memory region*

Memory manager

Wraps MAME's `memory_manager` class, which allows the memory shares, banks and regions in a system to be enumerated.

Instantiation

manager.machine.memory
Gets the global memory manager instance for the emulated system.

Properties

memory.shares[]
The *memory shares* in the system, indexed by absolute tag. The `at` and `index_of` methods have O(n) complexity; all other supported operations have O(1) complexity.

memory.banks[]
The *memory banks* in the system, indexed by absolute tag. The `at` and `index_of` methods have O(n) complexity; all other supported operations have O(1) complexity.

memory.regions[]
The *memory regions* in the system, indexed by absolute tag. The `at` and `index_of` methods have O(n) complexity; all other supported operations have O(1) complexity.

Address space

Wraps MAME's `address_space` class, which represents an address space belonging to a device.

Instantiation

manager.machine.devices[tag].spaces[name]
Gets the address space with the specified name for a given device. Note that names are specific to the device type.

Methods

space:read_i{8,16,32,64}(addr)
Reads a signed integer value of the size in bits from the specified address.

space:read_u{8,16,32,64}(addr)
Reads an unsigned integer value of the size in bits from the specified address.

space:write_i{8,16,32,64}(addr, val)
Writes a signed integer value of the size in bits to the specified address.

space:write_u{8,16,32,64}(addr, val)
Writes an unsigned integer value of the size in bits to the specified address.

space:readv_i{8,16,32,64}(addr)
Reads a signed integer value of the size in bits from the specified virtual address. The address is translated with the debug read intent. Returns zero if address translation fails.

space:readv_u{8,16,32,64}(addr)
Reads an unsigned integer value of the size in bits from the specified virtual address. The address is translated with the debug read intent. Returns zero if address translation fails.

space:writev_i{8,16,32,64}(addr, val)
Writes a signed integer value of the size in bits to the specified virtual address. The address is translated with the debug write intent. Does not write if address translation fails.

space:writev_u{8,16,32,64}(addr, val)
Writes an unsigned integer value of the size in bits to the specified virtual address. The address is translated with the debug write intent. Does not write if address translation fails.

space:read_direct_i{8,16,32,64}(addr)
Reads a signed integer value of the size in bits from the specified address one byte at a time by obtaining a read pointer for each byte address. If a read pointer cannot be obtained for a byte address, the corresponding result byte will be zero.

space:read_direct_u{8,16,32,64}(addr)
Reads an unsigned integer value of the size in bits from the specified address one byte at a time by obtaining a read pointer for each byte address. If a read pointer cannot be obtained for a byte address, the corresponding result byte will be zero.

space:write_direct_i{8,16,32,64}(addr, val)
Writes a signed integer value of the size in bits to the specified address one byte at a time by obtaining a write pointer for each byte address. If a write pointer cannot be obtained for a byte address, the corresponding byte will not be written.

space:write_direct_u{8,16,32,64}(addr, val)
Writes an unsigned integer value of the size in bits to the specified address one byte at a time by obtaining a write pointer for each byte address. If a write pointer cannot be obtained for a byte address, the corresponding byte will not be written.

space:read_range(start, end, width, [step])
Reads a range of addresses as a binary string. The end address must be greater than or equal to the start address. The width must be 8, 16, 30 or 64. If the step is provided, it must be a positive number of elements.

space:add_change_notifier(callback)
Add a callback to receive notifications for handler changes in address space. The callback function is passed a single string as an argument, either `r` if read handlers have potentially changed, `w` if write handlers have potentially changed, or `rw` if both read and write handlers have potentially changed.

Returns a *notifier subscription*.

space:install_read_tap(start, end, name, callback)
Installs a *pass-through handler* that will receive notifications on reads from the specified range of addresses in the address space. The start and end addresses are inclusive. The name must be a string, and the callback must be a function. Returns the new pass-through handler.

The callback is passed three arguments for the access offset, the data read, and the memory access mask. The offset is the absolute offset into the address space. To modify the data being read, return the modified value from the callback function as an integer. If the callback does not return an integer, the data will not be modified.

space:install_write_tap(start, end, name, callback)
Installs a *pass-through handler* that will receive notifications on write to the specified range of addresses in the address space. The start and end addresses are inclusive. The name must be a string, and the callback must be a function. Returns the new pass-through handler.

The callback is passed three arguments for the access offset, the data written, and the memory access mask. The offset is the absolute offset into the address space. To modify the data being written, return the modified value from the callback function as an integer. If the callback does not return an integer, the data will not be modified.

Properties

space.name (read-only)
The display name for the address space.

space.shift (read-only)
The address granularity for the address space specified as the shift required to translate a byte address to a native address. Positive values shift towards the most significant bit (left) and negative values shift towards the least significant bit (right).

space.index (read-only)
The zero-based space index. Some space indices have special meanings for the debugger.

space.address_mask (read-only)
The address mask for the space.

space.data_width (read-only)
The data width for the space in bits.

space.endianness (read-only)
The Endianness of the space (`"big"` or `"little"`).

space.map (read-only)
The configured *address map* for the space or `nil`.

Pass-through handler

Tracks a pass-through handler installed in an *address space*. A memory pass-through handler receives notifications on accesses to a specified range of addresses, and can modify the data that is read or written if desired. Note that pass-through handler callbacks are not run as coroutines.

Instantiation

manager.machine.devices[tag].spaces[name]:install_read_tap(start, end, name, callback)
Installs a pass-through handler that will receive notifications on reads from the specified range of addresses in an *address space*.

manager.machine.devices[tag].spaces[name]:install_write_tap(start, end, name, callback)
Installs a pass-through handler that will receive notifications on writes to the specified range of addresses in an *address space*.

Methods

passthrough:reinstall()
Reinstalls the pass-through handler in the address space. May be necessary if the handler is removed due to other changes to handlers in the address space.

passthrough:remove()
Removes the pass-through handler from the address space. The associated callback will not be called in response to future memory accesses.

Properties

passthrough.addrstart (read-only)
The inclusive start address of the address range monitored by the pass-through handler (i.e. the lowest address that the handler will be notified for).

passthrough.addrend (read-only)
The inclusive end address of the address range monitored by the pass-through handler (i.e. the highest address that the handler will be notified for).

passthrough.name (read-only)
The display name for the pass-through handler.

Address map

Wraps MAME's `address_map` class, used to configure handlers for an address space.

Instantiation

manager.machine.devices[tag].spaces[name].map
Gets the configured address map for an address space, or `nil` if no map is configured.

Properties

map.spacenum (read-only)
The address space number of the address space the map is associated with.

map.device (read-only)
The device that owns the address space the map is associated with.

map.unmap_value (read-only)
The constant value to return from unmapped reads.

map.global_mask (read-only)
Global mask to be applied to all addresses when accessing the space.

map.entries[] (read-only)
The configured *entries* in the address map. Uses 1-based integer indices. The index operator and the `at` method have O(n) complexity.

Address map entry

Wraps MAME's `address_map_entry` class, representing an entry in a configured address map.

Instantiation

manager.machine.devices[tag].spaces[name].map.entries[index]
Gets an entry from the configured map for an address space.

Properties

entry.address_start (read-only)
Start address of the entry's range.

entry.address_end (read-only)
End address of the entry's range (inclusive).

entry.address_mirror (read-only)
Address mirror bits.

entry.address_mask (read-only)
Address mask bits. Only valid for handlers.

entry.mask (read-only)
Lane mask, indicating which data lines on the bus are connected to the handler.

entry.cswidth (read-only)
The trigger width for a handler that isn't connected to all the data lines.

entry.read (read-only)
Additional data for the read handler.

entry.write (read-only)
Additional data for the write handler.

entry.share (read-only)
Memory share tag for making RAM entries accessible or `nil`.

entry.region (read-only)
Explicit memory region tag for ROM entries, or `nil`. For ROM entries, `nil` infers the region from the device tag.

entry.region_offset (read-only)
Starting offset in memory region for ROM entries.

Address map handler data

Wraps MAME's `map_handler_data` class, which provides configuration data to handlers in address maps.

Instantiation

manager.machine.devices[tag].spaces[name].map.entries[index].read
Gets the read handler data for an address map entry.

manager.machine.devices[tag].spaces[name].map.entries[index].write
Gets the write handler data for an address map entry.

Properties

data.handlertype (read-only)
Handler type. Will be one of `"none"`, `"ram"`, `"rom"`, `"nop"`, `"unmap"`, `"delegate"`, `"port"`, `"bank"`, `"submap"`, or `"unknown"`. Note that multiple handler type values can yield `"delegate"` or `"unknown"`.

data.bits (read-only)
Data width for the handler in bits.

data.name (read-only)
Display name for the handler, or `nil`.

data.tag (read-only)
Tag for I/O ports and memory banks, or `nil`.

Memory share

Wraps MAME's `memory_share` class, representing a named allocated memory zone.

Instantiation

manager.machine.memory.shares[tag]
Gets a memory share by absolute tag, or `nil` if no such memory share exists.

manager.machine.devices[tag]:memshare(tag)
Gets a memory share by tag relative to a device, or `nil` if no such memory share exists.

Methods

share:read_i{8,16,32,64}(offs)
Reads a signed integer value of the size in bits from the specified offset in the memory share.

share:read_u{8,16,32,64}(offs)
Reads an unsigned integer value of the size in bits from the specified offset in the memory share.

share:write_i{8,16,32,64}(offs, val)
Writes a signed integer value of the size in bits to the specified offset in the memory share.

share:write_u{8,16,32,64}(offs, val)
Writes an unsigned integer value of the size in bits to the specified offset in the memory share.

Properties

share.tag (read-only)
The absolute tag of the memory share.

share.size (read-only)
The size of the memory share in bytes.

share.length (read-only)
The length of the memory share in native width elements.

share.endianness (read-only)
The Endianness of the memory share (`"big"` or `"little"`).

share.bitwidth (read-only)
The native element width of the memory share in bits.

share.bytewidth (read-only)
The native element width of the memory share in bytes.

Memory bank

Wraps MAME's `memory_bank` class, representing a named memory zone indirection.

Instantiation

manager.machine.memory.banks[tag]
Gets a memory region by absolute tag, or `nil` if no such memory bank exists.

manager.machine.devices[tag]:membank(tag)
Gets a memory region by tag relative to a device, or `nil` if no such memory bank exists.

Properties

bank.tag (read-only)
The absolute tag of the memory bank.

bank.entry (read/write)
The currently selected zero-based entry number.

Memory region

Wraps MAME's `memory_region` class, representing a memory region used to store read-only data like ROMs or the result of fixed decryptions.

Instantiation

manager.machine.memory.regions[tag]
Gets a memory region by absolute tag, or `nil` if no such memory region exists.

manager.machine.devices[tag]:memregion(tag)
Gets a memory region by tag relative to a device, or `nil` if no such memory region exists.

Methods

region:read(offs, len)
Reads up to the specified length in bytes from the specified offset in the memory region. The bytes read will be returned as a string. If the specified length extends beyond the end of the memory region, the returned string will be shorter than requested. Note that the data will be in host byte order.

region:read_i{8,16,32,64}(offs)
Reads a signed integer value of the size in bits from the specified offset in the memory region. The offset is specified in bytes. Reading beyond the end of the memory region returns zero.

region:read_u{8,16,32,64}(offs)
Reads an unsigned integer value of the size in bits from the specified offset in the memory region. The offset is specified in bytes. Reading beyond the end of the memory region returns zero.

region:write_i{8,16,32,64}(offs, val)
Writes a signed integer value of the size in bits to the specified offset in the memory region. The offset is specified in bytes. Attempting to write beyond the end of the memory region has no effect.

region:write_u{8,16,32,64}(offs, val)
Writes an unsigned integer value of the size in bits to the specified offset in the memory region. The offset is specified in bytes. Attempting to write beyond the end of the memory region has no effect.

Properties

region.tag (read-only)
The absolute tag of the memory region.

region.size (read-only)
The size of the memory region in bytes.

region.length (read-only)
The length of the memory region in native width elements.

region.endianness (read-only)
The Endianness of the memory region (`"big"` or `"little"`).

region.bitwidth (read-only)
The native element width of the memory region in bits.

region.bytewidth (read-only)
The native element width of the memory region in bytes.

9.3.5 Lua Input System Classes

Allows scripts to get input from the user, and access I/O ports in the emulated system.

- *I/O port manager*
- *Natural keyboard manager*
- *Keyboard input device*
- *I/O port*
- *I/O port field*
- *Live I/O port field state*
- *Input type*
- *Input manager*
- *Input code poller*
- *Input sequence poller*
- *Input sequence*
- *Host input device class*

- *Host input device*
- *Host input device item*
- *UI input manager*

I/O port manager

Wraps MAME's `ioport_manager` class, which provides access to emulated I/O ports and handles input configuration.

Instantiation

manager.machine.ioport
Gets the global I/O port manager instance for the emulated machine.

Methods

ioport:count_players()
Returns the number of player controllers in the system.

ioport:type_pressed(type, [player])
Returns a Boolean indicating whether the specified input is currently pressed. The input type may be an enumerated value or an *input type* entry. If the input type is an enumerated value, the player number may be supplied as a zero-based index; if the player number is not supplied, it is assumed to be zero. If the input type is an input type entry, the player number may not be supplied separately.

ioport:type_name(type, [player])
Returns the display name for the specified input type and player number. The input type is an enumerated value. The player number is a zero-based index. If the player number is not supplied, it is assumed to be zero.

ioport:type_group(type, player)
Returns the input group for the specified input type and player number. The input type is an enumerated value. The player number is a zero-based index. Returns an integer giving the grouping for the input. If the player number is not supplied, it is assumed to be zero.

This should be called with values obtained from I/O port fields to provide canonical grouping in an input configuration UI.

ioport:type_seq(type, [player], [seqtype])
Get the configured *input sequence* for the specified input type, player number and sequence type. The input type may be an enumerated value or an *input type* entry. If the input type is an enumerated value, the player number may be supplied as a zero-based index; if the player number is not supplied, it is assumed to be zero. If the input type is an input type entry, the player number may not be supplied separately. If the sequence type is supplied, it must be `"standard"`, `"increment"` or `"decrement"`; if it is not supplied, it is assumed to be `"standard"`.

This provides access to general input assignments.

ioport:set_type_seq(type, [player], seqtype, seq)
Set the configured *input sequence* for the specified input type, player number and sequence type. The input type may be an enumerated value or an *input type* entry. If the input type is an enumerated value, the player number must be supplied as a zero-based index. If the input type is an input type entry, the player number may not be supplied separately. The sequence type must be `"standard"`, `"increment"` or `"decrement"`.

This allows general input assignments to be set.

ioport:token_to_input_type(string)
Returns the input type and player number for the specified input type token string.

ioport:input_type_to_token(type, [player])
Returns the token string for the specified input type and player number. If the player number is not supplied, it assumed to be zero.

Properties

ioport.types[] (read-only)
Gets the supported *input types*. Keys are arbitrary indices. All supported operations have O(1) complexity.

ioport.ports[]
Gets the emulated *I/O ports* in the system. Keys are absolute tags. The `at` and `index_of` methods have O(n) complexity; all other supported operations have O(1) complexity.

Natural keyboard manager

Wraps MAME's `natural_keyboard` class, which manages emulated keyboard and keypad inputs.

Instantiation

manager.machine.natkeyboard
Gets the global natural keyboard manager instance for the emulated machine.

Methods

natkeyboard:post(text)
Post literal text to the emulated machine. The machine must have keyboard inputs with character bindings, and the correct keyboard input device must be enabled.

natkeyboard:post_coded(text)
Post text to the emulated machine. Brace-enclosed codes are interpreted in the text. The machine must have keyboard inputs with character bindings, and the correct keyboard input device must be enabled.

The recognised codes are `{BACKSPACE}`, `{BS}`, `{BKSP}`, `{DEL}`, `{DELETE}`, `{END}`, `{ENTER}`, `{ESC}`, `{HOME}`, `{INS}`, `{INSERT}`, `{PGDN}`, `{PGUP}`, `{SPACE}`, `{TAB}`, `{F1}`, `{F2}`, `{F3}`, `{F4}`, `{F5}`, `{F6}`, `{F7}`, `{F8}`, `{F9}`, `{F10}`, `{F11}`, `{F12}`, and `{QUOTE}`.

natkeyboard:paste()
Post the contents of the host clipboard to the emulated machine. The machine must have keyboard inputs with character bindings, and the correct keyboard input device must be enabled.

natkeyboard:dump()
Returns a string with a human-readable description of the keyboard and keypad input devices in the system, whether they are enabled, and their character bindings.

Properties

natkeyboard.empty (read-only)
A Boolean indicating whether the natural keyboard manager's input buffer is empty.

natkeyboard.full (read-only)
A Boolean indicating whether the natural keyboard manager's input buffer is full.

natkeyboard.can_post (read-only)
A Boolean indicating whether the emulated system supports posting character data via the natural keyboard manager.

natkeyboard.is_posting (read-only)
A Boolean indicating whether posted character data is currently being delivered to the emulated system.

natkeyboard.in_use (read/write)
A Boolean indicating whether "natural keyboard" mode is enabled. When "natural keyboard" mode is enabled, the natural keyboard manager translates host character input to emulated system keystrokes.

natkeyboard.keyboards[]
Gets the *keyboard/keypad input devices* in the emulated system, indexed by absolute device tag. Index get has O(n) complexity; all other supported operations have O(1) complexity.

Keyboard input device

Represents a keyboard or keypad input device managed by the *natural keyboard manager*. Note that this is not a *device* class.

Instantiation

manager.machine.natkeyboard.keyboards[tag]
Gets the keyboard input device with the specified tag, or `nil` if the tag does not correspond to a keyboard input device.

Properties

keyboard.device (read-only)
The underlying *device*.

keyboard.tag (read-only)
The absolute tag of the underlying device.

keyboard.basetag (read-only)
The last component of the tag of the underlying device, or `"root"` for the root machine device.

keyboard.name (read-only)
The human-readable description of the underlying device type.

keyboard.shortname (read-only)
The identifier for the underlying device type.

keyboard.is_keypad (read-only)
A Boolean indicating whether the underlying device has keypad inputs but no keyboard inputs. This is used when determining which keyboard input devices should be enabled by default.

keyboard.enabled (read/write)
A Boolean indicating whether the device's keyboard and/or keypad inputs are enabled.

I/O port

Wraps MAME's `ioport_port` class, representing an emulated I/O port.

Instantiation

manager.machine.ioport.ports[tag]
Gets an emulated I/O port by absolute tag, or `nil` if the tag does not correspond to an I/O port.

manager.machine.devices[devtag]:ioport(porttag)
Gets an emulated I/O port by tag relative to a device, or `nil` if no such I/O port exists.

Methods

port:read()
Read the current input value. Returns a 32-bit integer.

port:write(value, mask)
Write to the I/O port output fields that are set in the specified mask. The value and mask must be 32-bit integers. Note that this does not set values for input fields.

port:field(mask)
Get the first *I/O port field* corresponding to the bits that are set in the specified mask, or `nil` if there is no corresponding field.

Properties

port.device (read-only)
The device that owns the I/O port.

port.tag (read-only)
The absolute tag of the I/O port

port.active (read-only)
A mask indicating which bits of the I/O port correspond to active fields (i.e. not unused or unassigned bits).

port.live (read-only)
The live state of the I/O port.

port.fields[] (read-only)
Gets a table of *fields* indexed by name.

I/O port field

Wraps MAME's `ioport_field` class, representing a field within an I/O port.

Instantiation

manager.machine.ioport.ports[tag]:field(mask)
Gets a field for the given port by bit mask.

manager.machine.ioport.ports[tag].fields[name]
Gets a field for the given port by display name.

Methods

field:set_value(value)
Set the value of the I/O port field. For digital fields, the value is compared to zero to determine whether the field should be active; for analog fields, the value must be right-aligned and in the correct range.

field:clear_value()
Clear programmatically overridden value and restore the field's regular behaviour.

field:set_sensitivity(sensitivity)
Set the sensitivity or gain for analog fields. Raises an error for digital fields.

field:set_keydelta(sensitivity)
Set the sensitivity to corresponding digital increment/decrement inputs for analog fields. Raises an error for digital fields.

field:set_centerdelta(sensitivity)
Set the speed at which non-wrapping analog fields return to their default values after associated digital increment/decrement inputs are released. Raises an error for digital fields and wrapping analog fields.

field:set_analog_reverse(reverse)
Set whether an analog field's value increases in the opposite direction to the convention (e.g. larger values when a pedal is released or a joystick is moved to the left). Raises an error for digital fields.

field:set_input_seq(seqtype, seq)
Set the *input sequence* for the specified sequence type. This is used to configure per-machine input settings. The sequence type must be `"standard"`, `"increment"` or `"decrement"`.

field:input_seq(seq_type)
Get the configured *input sequence* for the specified sequence type. This gets per-machine input assignments. The sequence type must be `"standard"`, `"increment"` or `"decrement"`.

field:set_default_input_seq(seq_type, seq)
Set the default *input sequence* for the specified sequence type. This overrides the default input assignment for a specific input. The sequence type must be `"standard"`, `"increment"` or `"decrement"`.

field:default_input_seq(seq_type)
Gets the default *input sequence* for the specified sequence type. If the default assignment is not overridden, this returns the general input assignment for the field's input type. The sequence type must be `"standard"`, `"increment"` or `"decrement"`.

field:keyboard_codes(shift)
Gets a table of characters corresponding to the field for the specified shift state. The shift state is a bit mask of active shift keys.

Properties

field.device (read-only)
The device that owns the port that the field belongs to.

field.port (read-only)
The *I/O port* that the field belongs to.

field.live (read-only)
The *live state* of the field.

field.type (read-only)
The input type of the field. This is an enumerated value.

field.name (read-only)
The display name for the field.

field.default_name (read-only)
The name for the field from the emulated system's configuration (cannot be overridden by scripts or plugins).

field.player (read-only)
Zero-based player number for the field.

field.mask (read-only)
Bits in the I/O port corresponding to this field.

field.defvalue (read-only)
The field's default value.

field.minvalue (read-only)
The minimum allowed value for analog fields, or `nil` for digital fields.

field.maxvalue (read-only)
The maximum allowed value for analog fields, or `nil` for digital fields.

field.sensitivity (read-only)
The sensitivity or gain for analog fields, or `nil` for digital fields.

field.default_sensitivity (read-only)
The default sensitivity or gain for analog fields, or `nil` for digital fields.

field.keydelta (read-only)
The sensitivity to corresponding digital increment/decrement inputs for analog fields, or `nil` for digital fields.

field.default_keydelta (read-only)
The default sensitivity to corresponding digital increment/decrement inputs for analog fields, or `nil` for digital fields.

field.centerdelta (read-only)
The speed at which non-wrapping analog fields return to their default values after associated digital increment/decrement inputs are released, or `nil` for digital fields and wrapping analog fields.

field.default_centerdelta (read-only)
The default speed at which non-wrapping analog fields return to their default values after associated digital increment/decrement inputs are released, or `nil` for digital fields and wrapping analog fields.

field.analog_reverse (read-only)
A Boolean indicating whether the value increases in the opposite direction to the convention for analog fields (e.g. larger values when a pedal is released or a joystick is moved to the left), or `nil` for digital fields.

field.default_analog_reverse (read-only)
A Boolean indicating whether the value increases in the opposite direction to the convention by default for analog fields (e.g. larger values when a pedal is released or a joystick is moved to the left), or `nil` for digital fields.

field.way (read-only)
The number of directions allowed by the restrictor plate/gate for a digital joystick, or zero (0) for other inputs.

field.type_class (read-only)
The type class for the input field – one of `"keyboard"`, `"controller"`, `"config"`, `"dipswitch"` or `"misc"`.

field.is_analog (read-only)
A Boolean indicating whether the field is an analog axis or positional control.

field.is_digital_joystick (read-only)
A Boolean indicating whether the field corresponds to a digital joystick switch.

field.enabled (read-only)
A Boolean indicating whether the field is enabled.

field.cocktail (read-only)
A Boolean indicating whether the field is only used when the system is configured for a cocktail table cabinet.

field.toggle (read-only)
A Boolean indicating whether the field corresponds to a hardware toggle switch or push-on, push-off button.

field.analog_reset (read-only)
A Boolean indicating whether the field corresponds to an incremental position input (e.g. a dial or trackball axis) that should be reset to zero for every video frame.

field.analog_wraps (read-only)
A Boolean indicating whether the field corresponds to an analog input that wraps from one end of its range to the other (e.g. an incremental position input like a dial or trackball axis).

field.analog_invert (read-only)
A Boolean indicating whether the field corresponds to an analog input that has its value ones-complemented.

field.impulse (read-only)
A Boolean indicating whether the field corresponds to a digital input that activates for a fixed amount of time.

field.crosshair_scale (read-only)
The scale factor for translating the field's range to crosshair position. A value of one (1) translates the field's full range to the full width or height the screen.

field.crosshair_offset (read-only)
The offset for translating the field's range to crosshair position.

field.user_value (read/write)
The value for DIP switch or configuration settings.

field.settings[] (read-only)
Gets a table of the currently enabled settings for a DIP switch or configuration field, indexed by value.

Live I/O port field state

Wraps MAME's `ioport_field_live` class, representing the live state of an I/O port field.

Instantiation

manager.machine.ioport.ports[tag]:field(mask).live
Gets the live state for an I/O port field.

Properties

live.name
Display name for the field.

Input type

Wraps MAME's `input_type_entry` class, representing an emulated input type or emulator UI input type. Input types are uniquely identified by the combination of their enumerated type value and player index.

Instantiation

manager.machine.ioport.types[index]
Gets a supported input type.

Properties

type.type (read-only)
An enumerated value representing the type of input.

type.group (read-only)
An integer giving the grouping for the input type. Should be used to provide canonical grouping in an input configuration UI.

type.player (read-only)
The zero-based player number, or zero for non-player controls.

type.token (read-only)
The token string for the input type, used in configuration files.

type.name (read-only)
The display name for the input type.

type.is_analog (read-only)
A Boolean indicating whether the input type is analog or digital. Inputs that only have on and off states are considered digital, while all other inputs are considered analog, even if they can only represent discrete values or positions.

Input manager

Wraps MAME's `input_manager` class, which reads host input devices and checks whether configured inputs are active.

Instantiation

manager.machine.input
Gets the global input manager instance for the emulated system.

Methods

input:code_value(code)
Gets the current value for the host input corresponding to the specified code. Returns a signed integer value, where zero is the neutral position.

input:code_pressed(code)
Returns a Boolean indicating whether the host input corresponding to the specified code has a non-zero value (i.e. it is not in the neutral position).

input:code_pressed_once(code)
Returns a Boolean indicating whether the host input corresponding to the specified code has moved away from the neutral position since the last time it was checked using this function. The input manager can track a limited number of inputs this way.

input:code_name(code)
Get display name for an input code.

input:code_to_token(code)
Get token string for an input code. This should be used when saving configuration.

input:code_from_token(token)
Convert a token string to an input code. Returns the invalid input code if the token is not valid or belongs to an input device that is not present.

input:seq_pressed(seq)
Returns a Boolean indicating whether the supplied *input sequence* is currently pressed.

input:seq_clean(seq)
Remove invalid elements from the supplied *input sequence*. Returns the new, cleaned input sequence.

input:seq_name(seq)
Get display text for an *input sequence*.

input:seq_to_tokens(seq)
Convert an *input sequence* to a token string. This should be used when saving configuration.

input:seq_from_tokens(tokens)
Convert a token string to an *input sequence*. This should be used when loading configuration.

input:axis_code_poller()
Returns an *input code poller* for obtaining an analog host input code.

input:switch_code_poller()
Returns an *input code poller* for obtaining a host switch input code.

input:keyboard_code_poller()
Returns an *input code poller* for obtaining a host switch input code that only considers keyboard input devices.

input:axis_sequence_poller()
Returns an *input sequence poller* for obtaining an *input sequence* for configuring an analog input assignment.

input:axis_sequence_poller()
Returns an *input sequence poller* for obtaining an *input sequence* for configuring a digital input assignment.

Properties

input.device_classes[] (read-only)
Gets a table of host *input device classes* indexed by name.

Input code poller

Wraps MAME's `input_code_poller` class, used to poll for host inputs being activated.

Instantiation

manager.machine.input:axis_code_poller()
Returns an input code poller that polls for analog inputs being activated.

manager.machine.input:switch_code_poller()
Returns an input code poller that polls for host switch inputs being activated.

manager.machine.input:keyboard_code_poller()
Returns an input code poller that polls for host switch inputs being activated, only considering keyboard input devices.

Methods

poller:reset()
Resets the polling logic. Active switch inputs are cleared and initial analog input positions are set.

poller:poll()
Returns an input code corresponding to the first relevant host input that has been activated since the last time the method was called. Returns the invalid input code if no relevant input has been activated.

Input sequence poller

Wraps MAME's `input_sequence_poller` poller class, which allows users to assign host input combinations to emulated inputs and other actions.

Instantiation

manager.machine.input:axis_sequence_poller()
Returns an input sequence poller for assigning host inputs to an analog input.

manager.machine.input:switch_sequence_poller()
Returns an input sequence poller for assigning host inputs to a switch input.

Methods

poller:start([seq])
Start polling. If a sequence is supplied, it is used as a starting sequence: for analog inputs, the user can cycle between the full range, and the positive and negative portions of an axis; for switch inputs, an "or" code is appended and the user can add an alternate host input combination.

poller:poll()
Polls for user input and updates the sequence if appropriate. Returns a Boolean indicating whether sequence input is complete. If this method returns false, you should continue polling.

Properties

poller.sequence (read-only)
The current *input sequence*. This is updated while polling. It is possible for the sequence to become invalid.

poller.valid (read-only)
A Boolean indicating whether the current input sequence is valid.

poller.modified (read-only)
A Boolean indicating whether the sequence was changed by any user input since starting polling.

Input sequence

Wraps MAME's `input_seq` class, representing a combination of host inputs that can be read or assigned to an emulated input. Input sequences can be manipulated using *input manager* methods. Use an *input sequence poller* to obtain an input sequence from the user.

Instantiation

emu.input_seq()
Creates an empty input sequence.

emu.input_seq(seq)
Creates a copy of an existing input sequence.

Methods

seq:reset()
Clears the input sequence, removing all items.

seq:set_default()
Sets the input sequence to a single item containing the metavalue specifying that the default setting should be used.

Properties

seq.empty (read-only)
A Boolean indicating whether the input sequence is empty (contains no items, indicating an unassigned input).

seq.length (read-only)
The number of items in the input sequence.

seq.is_valid (read-only)
A Boolean indicating whether the input sequence is a valid. To be valid, it must contain at least one item, all items must be valid codes, all product groups must contain at least one item that is not negated, and items referring to absolute and relative axes must not be mixed within a product group.

seq.is_default (read-only)
A Boolean indicating whether the input sequence specifies that the default setting should be used.

Host input device class

Wraps MAME's `input_class` class, representing a category of host input devices (e.g. keyboards or joysticks).

Instantiation

manager.machine.input.device_classes[name]
Gets an input device class by name.

Properties

devclass.name (read-only)
The device class name.

devclass.enabled (read-only)
A Boolean indicating whether the device class is enabled.

devclass.multi (read-only)
A Boolean indicating whether the device class supports multiple devices, or inputs from all devices in the class are combined and treated as a single device.

devclass.devices[] (read-only)
Gets a table of *host input devices* in the class. Keys are one-based indices.

Host input device

Wraps MAME's `input_device` class, representing a host input device.

Instantiation

manager.machine.input.device_classes[name].devices[index]
Gets a specific host input device.

Properties

inputdev.name (read-only)
Display name for the device. This is not guaranteed to be unique.

inputdev.id (read-only)
Unique identifier string for the device. This may not be human-readable.

inputdev.devindex (read-only)
Device index within the device class. This is not necessarily the same as the index in the `devices` property of the device class – the `devindex` indices may not be contiguous.

inputdev.items (read-only)
Gets a table of *input items*, indexed by item ID. The item ID is an enumerated value.

Host input device item

Wraps MAME's `input_device_item` class, representing a single host input (e.g. a key, button, or axis).

Instantiation

manager.machine.input.device_classes[name].devices[index].items[id]
Gets an individual host input item. The item ID is an enumerated value.

Properties

item.name (read-only)
The display name of the input item. Note that this is just the name of the item itself, and does not include the device name. The full display name for the item can be obtained by calling the `code_name` method on the *input manager* with the item's code.

item.code (read-only)
The input item's identification code. This is used by several *input manager* methods.

item.token (read-only)
The input item's token string. Note that this is a token fragment for the item itself, and does not include the device portion. The full token for the item can be obtained by calling the `code_to_token` method on the *input manager* with the item's code.

item.current (read-only)
The item's current value. This is a signed integer where zero is the neutral position.

UI input manager

Wraps MAME's `ui_input_manager` class, which is used for high-level input.

Instantiation

manager.machine.uiinput
: Gets the global UI input manager instance for the machine.

Methods

uiinput:reset()
: Clears pending events and UI input states. Should be called when leaving a modal state where input is handled directly (e.g. configuring an input combination).

uiinput:pressed(type)
: Returns a Boolean indicating whether the specified UI input has been pressed. The input type is an enumerated value.

uiinput:pressed_repeat(type, speed)
: Returns a Boolean indicating whether the specified UI input has been pressed or auto-repeat has been triggered at the specified speed. The input type is an enumerated value; the speed is an interval in sixtieths of a second.

Properties

uiinput.presses_enabled (read/write)
: Whether the UI input manager will check for UI inputs frame updates.

9.3.6 Lua Render System Classes

The render system is responsible for drawing what you see in MAME's windows, including emulated screens, artwork, and UI elements.

- *Render bounds*
- *Render colour*
- *Palette*
- *Bitmap*
- *Render texture*
- *Render manager*
- *Render target*
- *Render container*
- *Container user settings*
- *Layout file*
- *Layout view*

- *Layout view item*
- *Layout element*

Render bounds

Wraps MAME's `render_bounds` class, which represents a rectangle using floating-point coordinates.

Instantiation

emu.render_bounds()
Creates a render bounds object representing a unit square, with top left corner at (0, 0) and bottom right corner at (1, 1). Note that render target coordinates don't necessarily have equal X and Y scales, so this may not represent a square in the final output.

emu.render_bounds(left, top, right, bottom)
Creates a render bounds object representing a rectangle with top left corner at (x0, y0) and bottom right corner at (x1, y1).

The arguments must all be floating-point numbers.

Methods

bounds:includes(x, y)
Returns a Boolean indicating whether the specified point falls within the rectangle. The rectangle must be normalised for this to work (right greater than left and bottom greater than top).

The arguments must both be floating-point numbers.

bounds:set_xy(left, top, right, bottom)
Set the rectangle's position and size in terms of the positions of the edges.

The arguments must all be floating-point numbers.

bounds:set_wh(left, top, width, height)
Set the rectangle's position and size in terms of the top left corner position, and the width and height.

The arguments must all be floating-point numbers.

Properties

bounds.x0 (read/write)
The leftmost coordinate in the rectangle (i.e. the X coordinate of the left edge or the top left corner).

bounds.x1 (read/write)
The rightmost coordinate in the rectangle (i.e. the X coordinate of the right edge or the bottom right corner).

bounds.y0 (read/write)
The topmost coordinate in the rectangle (i.e. the Y coordinate of the top edge or the top left corner).

bounds.y1 (read/write)
The bottommost coordinate in the rectangle (i.e. the Y coordinate of the bottom edge or the bottom right corner).

bounds.width (read/write)
The width of the rectangle. Setting this property changes the position of the rightmost edge.

bounds.height (read/write)
The height of the rectangle. Setting this property changes the position of the bottommost edge.

bounds.aspect (read-only)
The width-to-height aspect ratio of the rectangle. Note that this is often in render target coordinates which don't necessarily have equal X and Y scales. A rectangle representing a square in the final output doesn't necessarily have an aspect ratio of 1.

Render colour

Wraps MAME's `render_color` class, which represents an ARGB (alpha, red, green, blue) format colour. Channels are floating-point values ranging from zero (0, transparent alpha or colour off) to one (1, opaque or full colour intensity). Colour channel values are not pre-multiplied by the alpha channel value.

Instantiation

emu.render_color()
Creates a render colour object representing opaque white (all channels set to 1). This is the identity value – ARGB multiplication by this value will not change a colour.

emu.render_color(a, r, g, b)
Creates a render colour object with the specified alpha, red, green and blue channel values.

The arguments must all be floating-point numbers in the range from zero (0) to one (1), inclusive.

Methods

color:set(a, r, g, b)
Sets the colour object's alpha, red, green and blue channel values.

The arguments must all be floating-point numbers in the range from zero (0) to one (1), inclusive.

Properties

color.a (read/write)
Alpha value, in the range of zero (0, transparent) to one (1, opaque).

color.r (read/write)
Red channel value, in the range of zero (0, off) to one (1, full intensity).

color.g (read/write)
Green channel value, in the range of zero (0, off) to one (1, full intensity).

color.b (read/write)
Blue channel value, in the range of zero (0, off) to one (1, full intensity).

Palette

Wraps MAME's `palette_t` class, which represents a table of colours that can be looked up by zero-based index. Palettes always contain additional special entries for black and white.

Each colour has an associated contrast adjustment value. Each adjustment group has associated brightness and contrast adjustment values. The palette also has overall brightness, contrast and gamma adjustment values.

Colours are represented in alpha/red/green/blue (ARGB) format. Channel values range from 0 (transparent or off) to 255 (opaque or full intensity), inclusive. Colour channel values are not pre-multiplied by the alpha value. Channel values are packed into the bytes of 32-bit unsigned integers, in the order alpha, red, green, blue from most-significant to least-significant byte.

Instantiation

emu.palette(colors, [groups])
Creates a palette with the specified number of colours and brightness/contrast adjustment groups. The number of colour groups defaults to one if not specified. Colours are initialised to black, brightness adjustment is initialised to 0.0, contrast adjustment initialised to 1.0, and gamma adjustment is initialised to 1.0.

Methods

palette:entry_color(index)
Gets the colour at the specified zero-based index.

Index values range from zero to the number of colours in the palette minus one. Returns black if the index is greater than or equal to the number of colours in the palette.

palette:entry_contrast(index)
Gets the contrast adjustment for the colour at the specified zero-based index. This is a floating-point number.

Index values range from zero to the number of colours in the palette minus one. Returns 1.0 if the index is greater than or equal to the number of colours in the palette.

palette:entry_adjusted_color(index, [group])
Gets a colour with brightness, contrast and gamma adjustments applied.

If the group is specified, colour index values range from zero to the number of colours in the palette minus one, and group values range from zero to the number of adjustment groups in the palette minus one.

If the group is not specified, index values range from zero to the number of colours multiplied by the number of adjustment groups plus one. Index values may be calculated by multiplying the zero-based group index by the number of colours in the palette, and adding the zero-based colour index. The last two index values correspond to the special entries for black and white, respectively.

Returns black if the specified combination of index and adjustment group is invalid.

palette:entry_set_color(index, color)
Sets the colour at the specified zero-based index. The colour may be specified as a single packed 32-bit value; or as individual red, green and blue channel values, in that order.

Index values range from zero to the number of colours in the palette minus one. Raises an error if the index value is invalid.

palette:entry_set_red_level(index, level)
Sets the red channel value of the colour at the specified zero-based index. Other channel values are not affected.

Index values range from zero to the number of colours in the palette minus one. Raises an error if the index value is invalid.

palette:entry_set_green_level(index, level)
Sets the green channel value of the colour at the specified zero-based index. Other channel values are not affected.

Index values range from zero to the number of colours in the palette minus one. Raises an error if the index value is invalid.

palette:entry_set_blue_level(index, level)
Sets the blue channel value of the colour at the specified zero-based index. Other channel values are not affected.

Index values range from zero to the number of colours in the palette minus one. Raises an error if the index value is invalid.

palette:entry_set_contrast(index, level)
Sets the contrast adjustment value for the colour at the specified zero-based index. This must be a floating-point number.

Index values range from zero to the number of colours in the palette minus one. Raises an error if the index value is invalid.

palette:group_set_brightness(group, brightness)
Sets the brightness adjustment value for the adjustment group at the specified zero-based index. This must be a floating-point number.

Group values range from zero to the number of adjustment groups in the palette minus one. Raises an error if the index value is invalid.

palette:group_set_contrast(group, contrast)
Sets the contrast adjustment value for the adjustment group at the specified zero-based index. This must be a floating-point number.

Group values range from zero to the number of adjustment groups in the palette minus one. Raises an error if the index value is invalid.

Properties

palette.colors (read-only)
The number of colour entries in each group of colours in the palette.

palette.groups (read-only)
The number of groups of colours in the palette.

palette.max_index (read-only)
The number of valid colour indices in the palette.

palette.black_entry (read-only)
The index of the special entry for the colour black.

palette.white_entry (read-only)
The index of the special entry for the colour white.

palette.brightness (write-only)
The overall brightness adjustment for the palette. This is a floating-point number.

palette.contrast (write-only)
The overall contrast adjustment for the palette. This is a floating-point number.

palette.gamma (write-only)
The overall gamma adjustment for the palette. This is a floating-point number.

Bitmap

Wraps implementations of MAME's `bitmap_t` and `bitmap_specific` classes, which represent two-dimensional bitmaps stored in row-major order. Pixel coordinates are zero-based, increasing to the right and down. Several pixel formats are supported.

Instantiation

emu.bitmap_ind8(palette, [width, height], [xslop, yslop])
Creates an 8-bit indexed bitmap. Each pixel is a zero-based, unsigned 8-bit index into a *palette*.

If no width and height are specified, they are assumed to be zero. If the width is specified, the height must also be specified. The X and Y slop values set the amount of extra storage in pixels to reserve at the left/right of each row and top/bottom of each column, respectively. If an X slop value is specified, a Y slop value must be specified as well. If no X and Y slop values are specified, they are assumed to be zero (the storage will be sized to fit the bitmap content). If the width and/or height is less than or equal to zero, no storage will be allocated, irrespective of the X and Y slop values, and the width and height of the bitmap will both be set to zero.

The initial clipping rectangle is set to the entirety of the bitmap.

emu.bitmap_ind16(palette, [width, height], [xslop, yslop])
Creates a 16-bit indexed bitmap. Each pixel is a zero-based, unsigned 16-bit index into a *palette*.

If no width and height are specified, they are assumed to be zero. If the width is specified, the height must also be specified. The X and Y slop values set the amount of extra storage in pixels to reserve at the left/right of each row and top/bottom of each column, respectively. If an X slop value is specified, a Y slop value must be specified as well. If no X and Y slop values are specified, they are assumed to be zero (the storage will be sized to fit the bitmap content). If the width and/or height is less than or equal to zero, no storage will be allocated, irrespective of the X and Y slop values, and the width and height of the bitmap will both be set to zero.

The initial clipping rectangle is set to the entirety of the bitmap.

emu.bitmap_ind32(palette, [width, height], [xslop, yslop])
Creates a 32-bit indexed bitmap. Each pixel is a zero-based, unsigned 32-bit index into a *palette*.

If no width and height are specified, they are assumed to be zero. If the width is specified, the height must also be specified. The X and Y slop values set the amount of extra storage in pixels to reserve at the left/right of each row and top/bottom of each column, respectively. If an X slop value is specified, a Y slop value must be specified as well. If no X and Y slop values are specified, they are assumed to be zero (the storage will be sized to fit the bitmap content). If the width and/or height is less than or equal to zero, no storage will be allocated, irrespective of the X and Y slop values, and the width and height of the bitmap will both be set to zero.

The initial clipping rectangle is set to the entirety of the bitmap.

emu.bitmap_ind64(palette, [width, height], [xslop, yslop])
Creates a 64-bit indexed bitmap. Each pixel is a zero-based, unsigned 64-bit index into a *palette*.

If no width and height are specified, they are assumed to be zero. If the width is specified, the height must also be specified. The X and Y slop values set the amount of extra storage in pixels to reserve at the left/right of each row and top/bottom of each column, respectively. If an X slop value is specified, a Y slop value must be specified as well. If no X and Y slop values are specified, they are assumed to be zero (the storage will be sized to fit the bitmap content). If the width and/or height is less than or equal to zero, no storage will be allocated, irrespective of the X and Y slop values, and the width and height of the bitmap will both be set to zero.

The initial clipping rectangle is set to the entirety of the bitmap.

emu.bitmap_yuy16([width, height], [xslop], yslop])
Creates a Y'CbCr format bitmap with 4:2:2 chroma subsampling (horizontal pairs of pixels have individual luma values but share chroma values). Each pixel is a 16-bit integer value. The most significant byte of the pixel

value is the unsigned 8-bit Y' (luma) component of the pixel colour. For each horizontal pair of pixels, the least significant byte of the first pixel (even zero-based X coordinate) value is the signed 8-bit Cb value for the pair of pixels, and the least significant byte of the second pixel (odd zero-based X coordinate) value is the signed 8-bit Cr value for the pair of pixels.

If no width and height are specified, they are assumed to be zero. If the width is specified, the height must also be specified. The X and Y slop values set the amount of extra storage in pixels to reserve at the left/right of each row and top/bottom of each column, respectively. If an X slop value is specified, a Y slop value must be specified as well. If no X and Y slop values are specified, they are assumed to be zero (the storage will be sized to fit the bitmap content). If the width and/or height is less than or equal to zero, no storage will be allocated, irrespective of the X and Y slop values, and the width and height of the bitmap will both be set to zero.

The initial clipping rectangle is set to the entirety of the bitmap.

emu.bitmap_rgb32([width, height], [xslop, yslop])
Creates an RGB format bitmap with no alpha (transparency) channel. Each pixel is represented by a 32-bit integer value. The most significant byte of the pixel value is ignored. The remaining three bytes, from most significant to least significant, are the unsigned 8-bit unsigned red, green and blue channel values (larger values correspond to higher intensities).

If no width and height are specified, they are assumed to be zero. If the width is specified, the height must also be specified. The X and Y slop values set the amount of extra storage in pixels to reserve at the left/right of each row and top/bottom of each column, respectively. If an X slop value is specified, a Y slop value must be specified as well. If no X and Y slop values are specified, they are assumed to be zero (the storage will be sized to fit the bitmap content). If the width and/or height is less than or equal to zero, no storage will be allocated, irrespective of the X and Y slop values, and the width and height of the bitmap will both be set to zero.

The initial clipping rectangle is set to the entirety of the bitmap.

emu.bitmap_argb32([width, height], [xslop, yslop])
Creates an ARGB format bitmap. Each pixel is represented by a 32-bit integer value. The most significant byte of the pixel is the 8-bit unsigned alpha (transparency) channel value (smaller values are more transparent). The remaining three bytes, from most significant to least significant, are the unsigned 8-bit unsigned red, green and blue channel values (larger values correspond to higher intensities). Colour channel values are not pre-multiplied by the alpha channel value.

If no width and height are specified, they are assumed to be zero. If the width is specified, the height must also be specified. The X and Y slop values set the amount of extra storage in pixels to reserve at the left/right of each row and top/bottom of each column, respectively. If an X slop value is specified, a Y slop value must be specified as well. If no X and Y slop values are specified, they are assumed to be zero (the storage will be sized to fit the bitmap content). If the width and/or height is less than or equal to zero, no storage will be allocated, irrespective of the X and Y slop values, and the width and height of the bitmap will both be set to zero.

The initial clipping rectangle is set to the entirety of the bitmap.

emu.bitmap_ind8(source, [x0, y0, x1, y1])
Creates an 8-bit indexed bitmap representing a view of a portion of an existing bitmap. The initial clipping rectangle is set to the bounds of the view. The source bitmap will be locked, preventing resizing and reallocation.

If no coordinates are specified, the new bitmap will represent a view of the source bitmap's current clipping rectangle. If coordinates are specified, the new bitmap will represent a view of the rectangle with top left corner at (x0, y0) and bottom right corner at (x1, y1) in the source bitmap. Coordinates are in units of pixels. The bottom right coordinates are inclusive.

The source bitmap must be owned by the Lua script and must use the 8-bit indexed format. Raises an error if coordinates are specified representing a rectangle not fully contained within the source bitmap's clipping rectangle.

emu.bitmap_ind16(source, [x0, y0, x1, y1])
Creates a 16-bit indexed bitmap representing a view of a portion of an existing bitmap. The initial clipping

rectangle is set to the bounds of the view. The source bitmap will be locked, preventing resizing and reallocation.

If no coordinates are specified, the new bitmap will represent a view of the source bitmap's current clipping rectangle. If coordinates are specified, the new bitmap will represent a view of the rectangle with top left corner at (x0, y0) and bottom right corner at (x1, y1) in the source bitmap. Coordinates are in units of pixels. The bottom right coordinates are inclusive.

The source bitmap must be owned by the Lua script and must use the 16-bit indexed format. Raises an error if coordinates are specified representing a rectangle not fully contained within the source bitmap's clipping rectangle.

emu.bitmap_ind32(source, [x0, y0, x1, y1])

Creates a 32-bit indexed bitmap representing a view of a portion of an existing bitmap. The initial clipping rectangle is set to the bounds of the view. The source bitmap will be locked, preventing resizing and reallocation.

If no coordinates are specified, the new bitmap will represent a view of the source bitmap's current clipping rectangle. If coordinates are specified, the new bitmap will represent a view of the rectangle with top left corner at (x0, y0) and bottom right corner at (x1, y1) in the source bitmap. Coordinates are in units of pixels. The bottom right coordinates are inclusive.

The source bitmap must be owned by the Lua script and must use the 32-bit indexed format. Raises an error if coordinates are specified representing a rectangle not fully contained within the source bitmap's clipping rectangle.

emu.bitmap_ind64(source, [x0, y0, x1, y1])

Creates a 64-bit indexed bitmap representing a view of a portion of an existing bitmap. The initial clipping rectangle is set to the bounds of the view. The source bitmap will be locked, preventing resizing and reallocation.

If no coordinates are specified, the new bitmap will represent a view of the source bitmap's current clipping rectangle. If coordinates are specified, the new bitmap will represent a view of the rectangle with top left corner at (x0, y0) and bottom right corner at (x1, y1) in the source bitmap. Coordinates are in units of pixels. The bottom right coordinates are inclusive.

The source bitmap must be owned by the Lua script and must use the 64-bit indexed format. Raises an error if coordinates are specified representing a rectangle not fully contained within the source bitmap's clipping rectangle.

emu.bitmap_yuy16(source, [x0, y0, x1, y1])

Creates a Y'CbCr format bitmap with 4:2:2 chroma subsampling representing a view of a portion of an existing bitmap. The initial clipping rectangle is set to the bounds of the view. The source bitmap will be locked, preventing resizing and reallocation.

If no coordinates are specified, the new bitmap will represent a view of the source bitmap's current clipping rectangle. If coordinates are specified, the new bitmap will represent a view of the rectangle with top left corner at (x0, y0) and bottom right corner at (x1, y1) in the source bitmap. Coordinates are in units of pixels. The bottom right coordinates are inclusive.

The source bitmap must be owned by the Lua script and must use the Y'CbCr format. Raises an error if coordinates are specified representing a rectangle not fully contained within the source bitmap's clipping rectangle.

emu.bitmap_rgb32(source, [x0, y0, x1, y1])

Creates an RGB format bitmap representing a view of a portion of an existing bitmap. The initial clipping rectangle is set to the bounds of the view. The source bitmap will be locked, preventing resizing and reallocation.

If no coordinates are specified, the new bitmap will represent a view of the source bitmap's current clipping rectangle. If coordinates are specified, the new bitmap will represent a view of the rectangle with top left corner at (x0, y0) and bottom right corner at (x1, y1) in the source bitmap. Coordinates are in units of pixels. The bottom right coordinates are inclusive.

The source bitmap must be owned by the Lua script and must use the RGB format. Raises an error if coordinates are specified representing a rectangle not fully contained within the source bitmap's clipping rectangle.

emu.bitmap_argb32(source, [x0, y0, x1, y1])
Creates an ARGB format bitmap representing a view of a portion of an existing bitmap. The initial clipping rectangle is set to the bounds of the view. The source bitmap will be locked, preventing resizing and reallocation.

If no coordinates are specified, the new bitmap will represent a view of the source bitmap's current clipping rectangle. If coordinates are specified, the new bitmap will represent a view of the rectangle with top left corner at (x0, y0) and bottom right corner at (x1, y1) in the source bitmap. Coordinates are in units of pixels. The bottom right coordinates are inclusive.

The source bitmap must be owned by the Lua script and must use the ARGB format. Raises an error if coordinates are specified representing a rectangle not fully contained within the source bitmap's clipping rectangle.

emu.bitmap_argb32.load(data)
Creates an ARGB format bitmap from data in PNG, JPEG (JFIF/EXIF) or Microsoft DIB (BMP) format. Raises an error if the data invalid or not a supported format.

Methods

bitmap:cliprect()
Returns the left, top, right and bottom coordinates of the bitmap's clipping rectangle. Coordinates are in units of pixels; the bottom and right coordinates are inclusive.

bitmap:reset()
Sets the width and height to zero, and frees the pixel storage if the bitmap owns its own storage, or releases the source bitmap if the it represents a view of another bitmap.

The bitmap must be owned by the Lua script. Raises an error if the bitmap's storage is referenced by another bitmap or a *texture*.

bitmap:allocate(width, height, [xslop, yslop])
Reallocates storage for the bitmap, sets its width and height, and sets the clipping rectangle to the entirety of the bitmap. If the bitmap already owns allocated storage, it will always be freed and reallocated; if the bitmap represents a view of another bitmap, the source bitmap will be released. The storage will be filled with pixel value zero.

The X and Y slop values set the amount of extra storage in pixels to reserve at the left/right of each row and top/bottom of each column, respectively. If an X slop value is specified, a Y slop value must be specified as well. If no X and Y slop values are specified, they are assumed to be zero (the storage will be sized to fit the bitmap content). If the width and/or height is less than or equal to zero, no storage will be allocated, irrespective of the X and Y slop values, and the width and height of the bitmap will both be set to zero.

The bitmap must be owned by the Lua script. Raises an error if the bitmap's storage is referenced by another bitmap or a *texture*.

bitmap:resize(width, height, [xslop, yslop])
Changes the width and height, and sets the clipping rectangle to the entirety of the bitmap.

The X and Y slop values set the amount of extra storage in pixels to reserve at the left/right of each row and top/bottom of each column, respectively. If an X slop value is specified, a Y slop value must be specified as well. If no X and Y slop values are specified, they are assumed to be zero (rows will be stored contiguously, and the top row will be placed at the beginning of the bitmap's storage).

If the bitmap already owns allocated storage and it is large enough for the updated size, it will be used without being freed; if it is too small for the updated size, it will always be freed and reallocated. If the bitmap represents a view of another bitmap, the source bitmap will be released. If storage is allocated, it will be filled with pixel value zero (if existing storage is used, its contents will not be changed).

Raises an error if the bitmap's storage is referenced by another bitmap or a *texture*.

bitmap:wrap(source, [x0, y0, x1, y1])
Makes the bitmap represent a view of a portion of another bitmap and sets the clipping rectangle to the bounds of the view.

If no coordinates are specified, the target bitmap will represent a view of the source bitmap's current clipping rectangle. If coordinates are specified, the target bitmap will represent a view of the rectangle with top left corner at (x0, y0) and bottom right corner at (x1, y1) in the source bitmap. Coordinates are in units of pixels. The bottom right coordinates are inclusive.

The source bitmap will be locked, preventing resizing and reallocation. If the target bitmap owns allocated storage, it will be freed; if it represents a view of another bitmap, the current source bitmap will be released.

The source and target bitmaps must both be owned by the Lua script and must use the same pixel format. Raises an error if coordinates are specified representing a rectangle not fully contained within the source bitmap's clipping rectangle; if the bitmap's storage is referenced by another bitmap or a *texture*; or if the source and target are the same bitmap.

bitmap:pix(x, y)
Returns the colour value of the pixel at the specified location. Coordinates are zero-based in units of pixels.

bitmap:pixels([x0, y0, x1, y1])
Returns the pixels, width and height of the portion of the bitmap with top left corner at (x0, y0) and bottom right corner at (x1, y1). Coordinates are in units of pixels. The bottom right coordinates are inclusive. If coordinates are not specified, the bitmap's clipping rectangle is used.

Pixels are returned packed into a binary string in host Endian order. Pixels are organised in row-major order, from left to right then top to bottom. The size and format of the pixel values depends on the format of the bitmap. Raises an error if coordinates are specified representing a rectangle not fully contained within the bitmap's clipping rectangle.

bitmap:fill(color, [x0, y0, x1, y1])
Fills a portion of the bitmap with the specified colour value. If coordinates are not specified, the clipping rectangle is filled; if coordinates are specified, the intersection of the clipping rectangle and the rectangle with top left corner at (x0, y0) and bottom right corner at (x1, y1) is filled. Coordinates are in units of pixels. The bottom right coordinates are inclusive.

bitmap:plot(x, y, color)
Sets the colour value of the pixel at the specified location if it is within the clipping rectangle. Coordinates are zero-based in units of pixels.

bitmap:plot_box(x, y, width, height, color)
Fills the intersection of the clipping rectangle and the rectangle with top left (x, y) and the specified height and width with the specified colour value. Coordinates and dimensions are in units of pixels.

bitmap:resample(dest, [color])
Copies the bitmap into the destination bitmap, scaling to fill the destination bitmap and using a re-sampling filter. Only ARGB format source and destination bitmaps are supported. The source pixel values will be multiplied by the colour if it is supplied. It must be a *render colour*.

Properties

bitmap.palette (read/write)
The *palette* used to translate pixel values to colours. Only applicable for bitmaps that use indexed pixel formats.

bitmap.width (read-only)
Width of the bitmap in pixels.

bitmap.height (read-only)
Height of the bitmap in pixels.

bitmap.rowpixels (read-only)
Row stride of the bitmap's storage in pixels. That is, the difference in pixel offsets of the pixels at the same horizontal location in consecutive rows. May be greater than the width.

bitmap.rowbytes (read-only)
Row stride of the bitmap's storage in bytes. That is, the difference in byte addresses of the pixels at the same horizontal location in consecutive rows.

bitmap.bpp (read-only)
Size of the type used to represent pixels in the bitmap in bits (may be larger than the number of significant bits).

bitmap.valid (read-only)
A Boolean indicating whether the bitmap has storage available (may be false for empty bitmaps).

bitmap.locked (read-only)
A Boolean indicating whether the bitmap's storage is referenced by another bitmap or a *texture*.

Render texture

Wraps MAME's `render_texture` class, representing a texture that cam be drawn in a *render container*. Render textures must be freed before the emulation session ends.

Instantiation

manager.machine.render:texture_alloc(bitmap)
Creates a render texture based on a *bitmap*. The bitmap must be owned by the Lua script, and must use the Y'CbCr, RGB or ARGB format. The bitmap's storage will be locked, preventing resizing and reallocation.

Methods

texture:free()
Frees the texture. The storage of the underlying bitmap will be released.

Properties

texture.valid (read-only)
A Boolean indicating whether the texture is valid (false if the texture has been freed).

Render manager

Wraps MAME's `render_manager` class, responsible for managing render targets and textures.

Instantiation

manager.machine.render
Gets the global render manager instance for the emulation session.

Methods

render:texture_alloc(bitmap)
Creates a *render texture* based on a *bitmap*. The bitmap must be owned by the Lua script, and must use the Y'CbCr, RGB or ARGB pixel format. The bitmap's storage will be locked, preventing resizing and reallocation. Render textures must be freed before the emulation session ends.

Properties

render.max_update_rate (read-only)
The maximum update rate in Hertz. This is a floating-point number.

render.ui_target (read-only)
The *render target* used to draw the user interface (including menus, sliders and pop-up messages). This is usually the first host window or screen.

render.ui_container (read-only)
The *render container* used for drawing the user interface.

render.targets[] (read-only)
The list of render targets, including output windows and screens, as well as hidden render targets used for things like rendering screenshots. Uses 1-based integer indices. The index operator and the `at` method have O(n) complexity.

Render target

Wrap's MAME's `render_target` class, which represents a video output channel. This could be a host window or screen, or a hidden target used for rendering screenshots.

Instantiation

manager.machine.render.targets[index]
Gets a render target by index.

manager.machine.render.ui_target
Gets the render target used to display the user interface (including menus, sliders and pop-up messages). This is usually the first host window or screen.

manager.machine.video.snapshot_target
Gets the render target used to produce snapshots and video recordings.

Properties

target.ui_container (read-only)
The *render container* for drawing user interface elements over this render target, or `nil` for hidden render targets (targets that are not shown to the user directly).

target.index (read-only)
The 1-based index of the render target. This has O(n) complexity.

target.width (read-only)
The width of the render target in output pixels. This is an integer.

target.height (read-only)
The height of the render target in output pixels. This is an integer.

target.pixel_aspect (read-only)
The width-to-height aspect ratio of the render target's pixels. This is a floating-point number.

target.hidden (read-only)
A Boolean indicating whether this is an internal render target that is not displayed to the user directly (e.g. the render target used to draw screenshots).

target.is_ui_target (read-only)
A Boolean indicating whether this is the render target used to display the user interface.

target.max_update_rate (read/write)
The maximum update rate for the render target in Hertz.

target.orientation (read/write)
The target orientation flags. This is an integer bit mask, where bit 0 (0x01) is set to mirror horizontally, bit 1 (0x02) is set to mirror vertically, and bit 2 (0x04) is set to mirror along the top left-bottom right diagonal.

target.view_names[]
The names of the available views for this render target. Uses 1-based integer indices. The `find` and `index_of` methods have O(n) complexity; all other supported operations have O(1) complexity.

target.current_view (read-only)
The currently selected view for the render target. This is a *layout view* object.

target.view_index (read/write)
The 1-based index of the selected view for this render target.

target.visibility_mask (read-only)
An integer bit mask indicating which item collections are currently visible for the current view.

target.screen_overlay (read/write)
A Boolean indicating whether screen overlays are enabled.

target.zoom_to_screen (read/write)
A Boolean indicating whether the render target is configured to scale so that the emulated screen(s) fill as much of the output window/screen as possible.

Render container

Wraps MAME's `render_container` class.

Instantiation

manager.machine.render.ui_container
Gets the render container used to draw the user interface, including menus, sliders and pop-up messages.

manager.machine.render.targets[index].ui_container
Gets the render container used to draw user interface elements over a particular render target.

manager.machine.screens[tag].container
Gets the render container used to draw a given screen.

Methods

container:draw_box(left, top, right, bottom, [line], [fill])
Draws an outlined rectangle with edges at the specified positions.

Coordinates are floating-point numbers in the range of 0 (zero) to 1 (one), with (0, 0) at the top left and (1, 1) at the bottom right of the window or the screen that shows the user interface. Note that the aspect ratio is usually not square. Coordinates are limited to the window or screen area.

The fill and line colours are in alpha/red/green/blue (ARGB) format. Channel values are in the range 0 (transparent or off) to 255 (opaque or full intensity), inclusive. Colour channel values are not pre-multiplied by the alpha value. The channel values must be packed into the bytes of a 32-bit unsigned integer, in the order alpha, red, green, blue from most-significant to least-significant byte. If the line colour is not provided, the UI text colour is used; if the fill colour is not provided, the UI background colour is used.

container:draw_line(x0, y0, x1, y1, [color])
Draws a line from (x0, y0) to (x1, y1).

Coordinates are floating-point numbers in the range of 0 (zero) to 1 (one), with (0, 0) at the top left and (1, 1) at the bottom right of the window or the screen that shows the user interface. Note that the aspect ratio is usually not square. Coordinates are limited to the window or screen area.

The line colour is in alpha/red/green/blue (ARGB) format. Channel values are in the range 0 (transparent or off) to 255 (opaque or full intensity), inclusive. Colour channel values are not pre-multiplied by the alpha value. The channel values must be packed into the bytes of a 32-bit unsigned integer, in the order alpha, red, green, blue from most-significant to least-significant byte. If the line colour is not provided, the UI text colour is used.

container:draw_quad(texture, x0, y0, x1, y1, [color])
Draws a textured rectangle with top left corner at (x0, y0) and bottom right corner at (x1, y1). If a colour is specified, the ARGB channel values of the texture's pixels are multiplied by the corresponding values of the specified colour.

Coordinates are floating-point numbers in the range of 0 (zero) to 1 (one), with (0, 0) at the top left and (1, 1) at the bottom right of the window or the screen that shows the user interface. Note that the aspect ratio is usually not square. If the rectangle extends beyond the container's bounds, it will be cropped.

The colour is in alpha/red/green/blue (ARGB) format. Channel values are in the range 0 (transparent or off) to 255 (opaque or full intensity), inclusive. Colour channel values are not pre-multiplied by the alpha value. The channel values must be packed into the bytes of a 32-bit unsigned integer, in the order alpha, red, green, blue from most-significant to least-significant byte.

container:draw_text(x|justify, y, text, [foreground], [background])
Draws text at the specified position. If the screen is rotated the text will be rotated.

If the first argument is a number, the text will be left-aligned at this X coordinate. If the first argument is a string, it must be `"left"`, `"center"` or `"right"` to draw the text left-aligned at the left edge of the window or screen, horizontally centred in the window or screen, or right-aligned at the right edge of the window or screen, respectively. The second argument specifies the Y coordinate of the maximum ascent of the text.

Coordinates are floating-point numbers in the range of 0 (zero) to 1 (one), with (0, 0) at the top left and (1, 1) at the bottom right of the window or the screen that shows the user interface. Note that the aspect ratio is usually not square. Coordinates are limited to the window or screen area.

The foreground and background colours are in alpha/red/green/blue (ARGB) format. Channel values are in the range 0 (transparent or off) to 255 (opaque or full intensity), inclusive. Colour channel values are not pre-multiplied by the alpha value. The channel values must be packed into the bytes of a 32-bit unsigned integer, in the order alpha, red, green, blue from most-significant to least-significant byte. If the foreground colour is not provided, the UI text colour is used; if the background colour is not provided, it is fully transparent.

Properties

container.user_settings (read/write)
The container's *user settings*. This can be used to control a number of image adjustments.

container.orientation (read/write)
The container orientation flags. This is an integer bit mask, where bit 0 (0x01) is set to mirror horizontally, bit 1 (0x02) is set to mirror vertically, and bit 2 (0x04) is set to mirror along the top left-bottom right diagonal.

container.xscale (read/write)
The container's X scale factor. This is a floating-point number.

container.yscale (read/write)
The container's Y scale factor. This is a floating-point number.

container.xoffset (read/write)
The container's X offset. This is a floating-point number where one (1) corresponds to the X size of the container.

container.yoffset (read/write)
The container's Y offset. This is a floating-point number where one (1) corresponds to the Y size of the container.

container.is_empty (read-only)
A Boolean indicating whether the container has no items.

Container user settings

Wraps MAME's `render_container::user_settings` class, representing image adjustments applied to a *render container*.

Instantiation

manager.machine.screens[tag].container
Gets the current render container user settings for a given emulated screen.

Properties

settings.orientation (read/write)
The container orientation flags. This is an integer bit mask, where bit 0 (0x01) is set to mirror horizontally, bit 1 (0x02) is set to mirror vertically, and bit 2 (0x04) is set to mirror along the top left-bottom right diagonal.

settings.brightness (read/write)
The brightness adjustment applied to the container. This is a floating-point number.

settings.contrast (read/write)
The contrast adjustment applied to the container. This is a floating-point number.

settings.gamma (read/write)
The gamma adjustment applied to the container. This is a floating-point number.

settings.xscale (read/write)
The container's X scale factor. This is a floating-point number.

settings.yscale (read/write)
The container's Y scale factor. This is a floating-point number.

settings.xoffset (read/write)
The container's X offset. This is a floating-point number where one (1) represents the X size of the container.

settings.yoffset (read/write)
The container's Y offset. This is a floating-point number where one (1) represents the Y size of the container.

Layout file

Wraps MAME's `layout_file` class, representing the views loaded from a layout file for use by a render target. Note that layout file callbacks are not run as coroutines.

Instantiation

A layout file object is supplied to its layout script in the `file` variable. Layout file objects are not instantiated directly from Lua scripts.

Methods

layout:set_resolve_tags_callback(cb)
Set a function to perform additional tasks after the emulated machine has finished starting, tags in the layout views have been resolved, and the default view item handlers have been set up. The function must accept no arguments.

Call with `nil` to remove the callback.

Properties

layout.device (read-only)
The device that caused the layout file to be loaded. Usually the root machine device for external layouts.

layout.elements[] (read-only)
The *elements* created from the layout file. Elements are indexed by name (i.e. the value of the `name` attribute). The index get method has O(1) complexity, and the `at` and `index_of` methods have O(n) complexity.

layout.views[] (read-only)
The *views* created from the layout file. Views are indexed by unqualified name (i.e. the value of the `name` attribute). Views are ordered how they appear in the layout file when iterating or using the `at` method. The index get, `at` and `index_of` methods have O(n) complexity.

Note that some views in the XML file may not be created. For example views that reference screens provided by slot card devices will not be created if said slot card devices are not present in the emulated system.

Layout view

Wraps MAME's `layout_view` class, representing a view that can be displayed in a render target. Views are created from XML layout files, which may be loaded from external artwork, internal to MAME, or automatically generated based on the screens in the emulated system. Note that layout view callbacks are not run as coroutines.

Instantiation

manager.machine.render.targets[index].current_view
Gets the currently selected view for a given render target.

file.views[name]
Gets the view with the specified name from a *layout file*. This is how layout scripts generally obtain views.

Methods

view:has_screen(screen)
Returns a Boolean indicating whether the screen is present in the view. This is true for screens that are present but not visible because the user has hidden the item collection they belong to.

view:set_prepare_items_callback(cb)
Set a function to perform additional tasks before the view items are added to the render target in preparation for drawing a video frame. The function must accept no arguments. Call with `nil` to remove the callback.

view:set_preload_callback(cb)
Set a function to perform additional tasks after preloading visible view items. The function must accept no arguments. Call with `nil` to remove the callback.

This function may be called when the user selects a view or makes an item collection visible. It may be called multiple times for a view, so avoid repeating expensive tasks.

view:set_recomputed_callback(cb)
Set a function to perform additional tasks after the view's dimensions are recomputed. The function must accept no arguments. Call with `nil` to remove the callback.

View coordinates are recomputed in various events, including the window being resized, entering or leaving full-screen mode, and changing the zoom to screen area setting.

view:set_pointer_updated_callback(cb)
Set a function to receive notifications when a pointer enters, moves or changes button states over the view. The function must accept nine arguments:

- The pointer type (`mouse`, `pen`, `touch` or `unknown`).
- The pointer ID (a non-negative integer that will not change for the lifetime of a pointer).
- The device ID for grouping pointers to recognise multi-touch gestures (non-negative integer).
- Horizontal position in layout coordinates.
- Vertical position in layout coordinates.
- A bit mask representing the currently pressed buttons.
- A bit mask representing the buttons that were pressed in this update.
- A bit mask representing the buttons that were released in this update.
- The click count (positive for multi-click actions, or negative if a click is turned into a hold or drag).

Call with `nil` to remove the callback.

view:set_pointer_left_callback(cb)
Set a function to receive notifications when a pointer leaves the view normally. The function must accept seven arguments:

- The pointer type (`mouse`, `pen`, `touch` or `unknown`).
- The pointer ID (a non-negative integer that will not change for the lifetime of a pointer). The ID may be reused for a new pointer after receiving this notification.
- The device ID for grouping pointers to recognise multi-touch gestures (non-negative integer).
- Horizontal position in layout coordinates.
- Vertical position in layout coordinates.
- A bit mask representing the buttons that were released in this update.
- The click count (positive for multi-click actions, or negative if a click is turned into a hold or drag).

Call with `nil` to remove the callback.

view:set_pointer_aborted_callback(cb)
Set a function to receive notifications when a pointer leaves the view abnormally. The function must accept seven arguments:

- The pointer type (`mouse`, `pen`, `touch` or `unknown`).
- The pointer ID (a non-negative integer that will not change for the lifetime of a pointer). The ID may be reused for a new pointer after receiving this notification.
- The device ID for grouping pointers to recognise multi-touch gestures (non-negative integer).
- Horizontal position in layout coordinates.
- Vertical position in layout coordinates.
- A bit mask representing the buttons that were released in this update.
- The click count (positive for multi-click actions, or negative if a click is turned into a hold or drag).

Call with `nil` to remove the callback.

view:set_forget_pointers_callback(cb)
Set a function to receive notifications when the view should stop processing pointer input. The function must accept no arguments. Call with `nil` to remove the callback.

This can happen in a number of situations, including the view configuration changing or a menu taking over input handling.

Properties

view.items[] (read-only)
The screen and layout element *items* in the view. This container does not support iteration by key using `pairs`; only iteration by index using `ipairs` is supported. The key is the value of the `id` attribute if present. Only items with `id` attributes can be looked up by key. The index get method has O(1) complexity, and the `at` and `index_of` methods have O(n) complexity.

view.name (read-only)
The display name for the view. This may be qualified to indicate the device that caused the layout file to be loaded when it isn't the root machine device.

view.unqualified_name (read-only)
The unqualified name of the view, exactly as it appears in the `name` attribute in the XML layout file.

view.visible_screen_count (read-only)
The number of screens items currently enabled in the view.

view.effective_aspect (read-only)
The effective width-to-height aspect ratio of the view in its current configuration.

view.bounds (read-only)
A *render bounds* object representing the effective bounds of the view in its current configuration. The coordinates are in view units, which are arbitrary but assumed to have square aspect ratio.

view.has_art (read-only)
A Boolean indicating whether the view has any non-screen items, including items that are not visible because the user has hidden the item collection that they belong to.

view.show_pointers (read/write)
A Boolean that sets whether mouse and pen pointers should be displayed for the view.

view.hide_inactive_pointers (read/write)
A Boolean that sets whether mouse pointers for the view should be hidden after a period of inactivity.

Layout view item

Wraps MAME's `layout_view_item` class, representing an item in a *layout view*. An item is drawn as a rectangular textured surface. The texture is supplied by an emulated screen or a layout element. Note that layout view item callbacks are not run as coroutines.

Instantiation

layout.views[name].items[id]
Get a view item by ID. The item must have an `id` attribute in the XML layout file to be looked up by ID.

Methods

item:set_state(state)
Set the value used as the element state and animation state in the absence of bindings. The argument must be an integer.

item:set_element_state_callback(cb)
Set a function to call to obtain the element state for the item. The function must accept no arguments and return an integer. Call with `nil` to restore the default element state callback (based on bindings in the XML layout file).

Note that the function must not access the item's `element_state` property, as this will result in infinite recursion.

This callback will not be used to obtain the animation state for the item, even if the item lacks explicit animation state bindings in the XML layout file.

item:set_animation_state_callback(cb)
Set a function to call to obtain the animation state for the item. The function must accept no arguments and return an integer. Call with `nil` to restore the default animation state callback (based on bindings in the XML layout file).

Note that the function must not access the item's `animation_state` property, as this will result in infinite recursion.

item:set_bounds_callback(cb)
Set a function to call to obtain the bounds for the item. The function must accept no arguments and return a *render bounds* object in render target coordinates. Call with `nil` to restore the default bounds callback (based on the item's animation state and `bounds` child elements in the XML layout file).

Note that the function must not access the item's `bounds` property, as this will result in infinite recursion.

item:set_color_callback(cb)
Set a function to call to obtain the multiplier colour for the item. The function must accept no arguments and return a *render colour* object. Call with `nil` to restore the default colour callback (based on the item's animation state and `color` child elements in the XML layout file).

Note that the function must not access the item's `color` property, as this will result in infinite recursion.

item:set_scroll_size_x_callback(cb)
Set a function to call to obtain the size of the horizontal scroll window as a proportion of the associated element's width. The function must accept no arguments and return a floating-point value. Call with `nil` to restore the default horizontal scroll window size callback (based on the `xscroll` child element in the XML layout file).

Note that the function must not access the item's `scroll_size_x` property, as this will result in infinite recursion.

item:set_scroll_size_y_callback(cb)
Set a function to call to obtain the size of the vertical scroll window as a proportion of the associated element's height. The function must accept no arguments and return a floating-point value. Call with `nil` to restore the default vertical scroll window size callback (based on the `yscroll` child element in the XML layout file).

Note that the function must not access the item's `scroll_size_y` property, as this will result in infinite recursion.

item:set_scroll_pos_x_callback(cb)
Set a function to call to obtain the horizontal scroll position. A value of zero places the horizontal scroll window at the left edge of the associated element. If the item does not wrap horizontally, a value of 1.0 places the horizontal scroll window at the right edge of the associated element; if the item wraps horizontally, a value of 1.0

corresponds to wrapping back to the left edge of the associated element. The function must accept no arguments and return a floating-point value. Call with `nil` to restore the default horizontal scroll position callback (based on bindings in the `xscroll` child element in the XML layout file).

Note that the function must not access the item's `scroll_pos_x` property, as this will result in infinite recursion.

item:set_scroll_pos_y_callback(cb)
Set a function to call to obtain the vertical scroll position. A value of zero places the vertical scroll window at the top edge of the associated element. If the item does not wrap vertically, a value of 1.0 places the vertical scroll window at the bottom edge of the associated element; if the item wraps vertically, a value of 1.0 corresponds to wrapping back to the left edge of the associated element. The function must accept no arguments and return a floating-point value. Call with `nil` to restore the default vertical scroll position callback (based on bindings in the `yscroll` child element in the XML layout file).

Note that the function must not access the item's `scroll_pos_y` property, as this will result in infinite recursion.

Properties

item.id (read-only)
Get the optional item identifier. This is the value of the `id` attribute in the XML layout file if present, or `nil`.

item.element (read-only)
The *element* used to draw the item, or `nil` for screen items.

item.bounds_animated (read-only)
A Boolean indicating whether the item's bounds depend on its animation state.

item.color_animated (read-only)
A Boolean indicating whether the item's colour depends on its animation state.

item.bounds (read-only)
The item's bounds for the current state. This is a *render bounds* object in render target coordinates.

item.color (read-only)
The item's colour for the current state. The colour of the screen or element texture is multiplied by this colour. This is a *render colour* object.

item.scroll_wrap_x (read-only)
A Boolean indicating whether the item wraps horizontally.

item.scroll_wrap_y (read-only)
A Boolean indicating whether the item wraps vertically.

item.scroll_size_x (read/write)
Get the item's horizontal scroll window size for the current state, or set the horizontal scroll window size to use in the absence of bindings. This is a floating-point value representing a proportion of the associated element's width.

item.scroll_size_y (read/write)
Get the item's vertical scroll window size for the current state, or set the vertical scroll window size to use in the absence of bindings. This is a floating-point value representing a proportion of the associated element's height.

item.scroll_pos_x (read/write)
Get the item's horizontal scroll position for the current state, or set the horizontal scroll position size to use in the absence of bindings. This is a floating-point value.

item.scroll_pos_y (read/write)
Get the item's vertical scroll position for the current state, or set the vertical position size to use in the absence of bindings. This is a floating-point value.

item.blend_mode (read-only)
Get the item's blend mode. This is an integer value, where 0 means no blending, 1 means alpha blending, 2 means RGB multiplication, 3 means additive blending, and -1 allows the items within a container to specify their own blending modes.

item.orientation (read-only)
Get the item orientation flags. This is an integer bit mask, where bit 0 (0x01) is set to mirror horizontally, bit 1 (0x02) is set to mirror vertically, and bit 2 (0x04) is set to mirror along the top left-bottom right diagonal.

item.element_state (read-only)
Get the current element state. This will call the element state callback function to handle bindings.

item.animation_state (read-only)
Get the current animation state. This will call the animation state callback function to handle bindings.

Layout element

Wraps MAME's `layout_element` class, representing a visual element that can be drawn in a *layout view*. Elements are created from XML layout files, which may be loaded from external artwork or internal to MAME. Note that layout element callbacks are not run as coroutines.

Instantiation

layout.elements[name]
Gets a layout element by name.

layout.views[name].items[id].element
Gets the layout element used to draw a *view item*.

Methods

element:invalidate()
Invalidate all cached textures for the element, ensuring it will be redrawn when the next video frame is drawn.

element.set_draw_callback(cb)
Set a function to call the perform additional drawing after the element's components have been drawn. The function is passed two arguments: the element state (an integer) and the 32-bit ARGB *bitmap* at the required size. The function must not attempt to resize the bitmap. Call with `nil` to remove the callback.

Properties

element.default_state (read-only)
The integer default state for the element if set or `nil`.

9.3.7 Lua Debugger Classes

Some of MAME's core debugging features can be controlled from Lua script. The debugger must be enabled to use the debugger features (usually by passing `-debug` on the command line).

- *Symbol table*
- *Parsed expression*
- *Symbol entry*
- *Debugger manager*
- *Device debugger interface*
- *Breakpoint*
- *Watchpoint*
- *Expression error*

Symbol table

Wrap's MAME's `symbol_table` class, providing named symbols that can be used in expressions. Note that symbol tables can be created and used even when the debugger is not enabled.

Instantiation

emu.symbol_table(machine)
Creates a new symbol table in the context of the specified machine,

emu.symbol_table(parent, [device])
Creates a new symbol table with the specified parent symbol table. If a device is specified and it implements `device_memory_interface`, it will be used as the base for looking up address spaces and memory regions. Note that if a device that does not implement `device_memory_interface` is supplied, it will not be used (address spaces and memory regions will be looked up relative to the root device).

emu.symbol_table(device)
Creates a new symbol table in the context of the specified device. If the device implements `device_memory_interface`, it will be used as the base for looking up address spaces and memory regions. Note that if a device that does not implement `device_memory_interface` is supplied, it will only be used to determine the machine context (address spaces and memory regions will be looked up relative to the root device).

Methods

symbols:set_memory_modified_func(cb)
Set a function to call when memory is modified via the symbol table. No arguments are passed to the function and any return values are ignored. Call with `nil` to remove the callback.

symbols:add(name, [value])
Adds a named integer symbol. The name must be a string. If a value is supplied, it must be an integer. If a value is supplied, a read-only symbol is added with the supplied value. If no value is supplied, a read/write symbol is created with and initial value of zero. If a symbol entry with the specified name already exists in the symbol table, it will be replaced.

Returns the new *symbol entry*.

symbols:add(name, getter, [setter], [format])
Adds a named integer symbol using getter and optional setter callbacks. The name must be a string. The getter must be a function returning an integer for the symbol value. If supplied, the setter must be a function that accepts a single integer argument for the new value of the symbol. A format string for displaying the symbol value may optionally be supplied. If a symbol entry with the specified name already exists in the symbol table, it will be replaced.

Returns the new *symbol entry*.

symbols:add(name, minparams, maxparams, execute)
Adds a named function symbol. The name must be a string. The minimum and maximum numbers of parameters must be integers. If a symbol entry with the specified name already exists in the symbol table, it will be replaced.

Returns the new *symbol entry*.

symbols:find(name)
Returns the *symbol entry* with the specified name, or `nil` if there is no symbol with the specified name in the symbol table.

symbols:find_deep(name)
Returns the *symbol entry* with the specified name, or `nil` if there is no symbol with the specified name in the symbol table or any of its parent symbol tables.

symbols:value(name)
Returns the integer value of the symbol with the specified name, or zero if there is no symbol with the specified name in the symbol table or any of its parent symbol tables. Raises an error if the symbol with specified name is a function symbol.

symbols:set_value(name, value)
Sets the value of the symbol with the specified name. Raises an error if the symbol with the specified name is a read-only integer symbol or if it is a function symbol. Has no effect if there is no symbol with the specified name in the symbol table or any of its parent symbol tables.

symbols:memory_value(name, space, offset, size, disable_se)
Read a value from memory. Supply the name or tag of the address space or memory region to read from, or `nil` to use the address space or memory region implied by the `space` argument. See *memory accesses in debugger expressions* for access type specifications that can be used for the `space` argument. The access size is specified in bytes, and must be 1, 2, 4 or 8. The `disable_se` argument specifies whether memory access side effects should be disabled.

symbols:set_memory_value(name, space, offset, value, size, disable_se)
Write a value to memory. Supply the name or tag of the address space or memory region to write to, or `nil` to use the address space or memory region implied by the `space` argument. See *memory accesses in debugger expressions* for access type specifications that can be used for the `space` argument. The access size is specified in bytes, and must be 1, 2, 4 or 8. The `disable_se` argument specifies whether memory access side effects should be disabled.

symbols:read_memory(space, address, size, apply_translation)
Read a value from an address space. The access size is specified in bytes, and must be 1, 2, 4, or 8. If the `apply_translation` argument is true, the address will be translated with debug read intention. Returns a value of the requested size with all bits set if address translation fails.

symbols:write_memory(space, address, data, size, apply_translation)
Write a value to an address space. The access size is specified in bytes, and must be 1, 2, 4, or 8. If the `apply_translation` argument is true, the address will be translated with debug write intention. The symbol table's memory modified function will be called after the value is written. The value will not be written and the symbol table's memory modified function will not be called if address translation fails.

Properties

symbols.entries[]
The *symbol entries* in the symbol table, indexed by name. The `at` and `index_of` methods have O(n) complexity; all other supported operations have O(1) complexity.

symbols.parent (read-only)
The parent symbol table, or `nil` if the symbol table has no parent.

Parsed expression

Wraps MAME's `parsed_expression` class, which represents a tokenised debugger expression. Note that parsed expressions can be created and used even when the debugger is not enabled.

Instantiation

emu.parsed_expression(symbols)
Creates an empty expression that will use the supplied *symbol table* to look up symbols.

emu.parsed_expression(symbols, string, [default_base])
Creates an expression by parsing the supplied string, looking up symbols in the supplied *symbol table*. If the default base for interpreting integer literals is not supplied, 16 is used (hexadecimal). Raises an *expression error* if the string contains syntax errors or uses undefined symbols.

Methods

expression:set_default_base(base)
Set the default base for interpreting numeric literals. The base must be a positive integer.

expression:parse(string)
Parse a debugger expression string. Replaces the current contents of the expression if it is not empty. Raises an *expression error* if the string contains syntax errors or uses undefined symbols. The previous content of the expression is not preserved when attempting to parse an invalid expression string.

expression:execute()
Evaluates the expression, returning an unsigned integer result. Raises an *expression error* if the expression cannot be evaluated (e.g. attempting to call a function with an invalid number of arguments).

Properties

expression.is_empty (read-only)
A Boolean indicating whether the expression contains no tokens.

expression.original_string (read-only)
The original string that was parsed to create the expression.

expression.symbols (read/write)
The *symbol table* used for to look up symbols in the expression.

Symbol entry

Wraps MAME's `symbol_entry` class, which represents an entry in a *symbol table*. Note that symbol entries must not be used after the symbol table they belong to is destroyed.

Instantiation

symbols:add(name, [value])
Adds an integer symbol to a *symbol table*, returning the new symbol entry.

symbols:add(name, getter, [setter], [format])
Adds an integer symbol to a *symbol table*, returning the new symbol entry.

symbols:add(name, minparams, maxparams, execute)
Adds function symbol to a *symbol table*, returning the new symbol entry.

Properties

entry.name (read-only)
The name of the symbol entry.

entry.format (read-only)
The format string used to convert the symbol entry to text for display.

entry.is_function (read-only)
A Boolean indicating whether the symbol entry is a callable function.

entry.is_lval (read-only)
A Boolean indicating whether the symbol entry is an integer symbol that can be set (i.e. whether it can be used on the left-hand side of assignment expressions).

entry.value (read/write)
The integer value of the symbol entry. Attempting to set the value raises an error if the symbol entry is read-only. Attempting to get or set the value of a function symbol raises an error.

Debugger manager

Wraps MAME's `debugger_manager` class, providing the main interface to control the debugger.

Instantiation

manager.machine.debugger
Returns the global debugger manager instance, or `nil` if the debugger is not enabled.

Methods

debugger:command(str)
Execute a debugger console command. The argument is the command string. The output is sent to both the debugger console and the Lua console.

Properties

debugger.consolelog[] (read-only)
The lines in the console log (output from debugger commands). This container only supports index and length operations.

debugger.errorlog[] (read-only)
The lines in the error log (`logerror` output). This container only supports index and length operations.

debugger.visible_cpu (read/write)
The CPU device with debugger focus. Changes become visible in the debugger console after the next step. Setting to a device that is not a CPU has no effect.

debugger.execution_state (read/write)
Either `"run"` if the emulated system is running, or `"stop"` if it is stopped in the debugger.

Device debugger interface

Wraps MAME's `device_debug` class, providing the debugger interface to an emulated CPU device.

Instantiation

manager.machine.devices[tag].debug
Returns the debugger interface for an emulated CPU device, or `nil` if the device is not a CPU.

Methods

debug:step([cnt])
Step by the specified number of instructions. If the instruction count is not provided, it defaults to a single instruction.

debug:go()
Run the emulated CPU.

debug:bpset(addr, [cond], [act])
Set a breakpoint at the specified address, with an optional condition and action. If the action is not specified, it defaults to just breaking into the debugger. Returns the breakpoint number for the new breakpoint.

If specified, the condition must be a debugger expression that will be evaluated each time the breakpoint is hit. Execution will only be stopped if the expression evaluates to a non-zero value. If the condition is not specified, it defaults to always active.

debug:bpenable([bp])
Enable the specified breakpoint, or all breakpoints for the device if no breakpoint number is specified. Returns whether the specified number matched a breakpoint if a breakpoint number is specified, or `nil` if no breakpoint number is specified.

debug:bpdisable([bp])
Disable the specified breakpoint, or all breakpoints for the device if no breakpoint number is specified. Returns whether the specified number matched a breakpoint if a breakpoint number is specified, or `nil` if no breakpoint number is specified.

debug:bpclear([bp])
Clear the specified breakpoint, or all breakpoints for the device if no breakpoint number is specified. Returns whether the specified number matched a breakpoint if a breakpoint number is specified, or `nil` if no breakpoint number is specified.

debug:bplist()
Returns a table of breakpoints for the device. The keys are the breakpoint numbers, and the values are *breakpoint objects*.

debug:wpset(space, type, addr, len, [cond], [act])
Set a watchpoint over the specified address range, with an optional condition and action. The type must be `"r"`, `"w"` or `"rw"` for a read, write or read/write breakpoint. If the action is not specified, it defaults to just breaking into the debugger. Returns the watchpoint number for the new watchpoint.

If specified, the condition must be a debugger expression that will be evaluated each time the breakpoint is hit. Execution will only be stopped if the expression evaluates to a non-zero value. The variable 'wpaddr' is set to the address that actually triggered the watchpoint, the variable 'wpdata' is set to the data that is being read or written, and the variable 'wpsize' is set to the size of the data in bytes. If the condition is not specified, it defaults to always active.

debug:wpenable([wp])
Enable the specified watchpoint, or all watchpoints for the device if no watchpoint number is specified. Returns whether the specified number matched a watchpoint if a watchpoint number is specified, or `nil` if no watchpoint number is specified.

debug:wpdisable([wp])
Disable the specified watchpoint, or all watchpoints for the device if no watchpoint number is specified. Returns whether the specified number matched a watchpoint if a watchpoint number is specified, or `nil` if no watchpoint number is specified.

debug:wpclear([wp])
Clear the specified watchpoint, or all watchpoints for the device if no watchpoint number is specified. Returns whether the specified number matched a watchpoint if a watchpoint number is specified, or `nil` if no watchpoint number is specified.

debug:wplist(space)
Returns a table of watchpoints for the specified address space of the device. The keys are the watchpoint numbers, and the values are *watchpoint objects*.

Breakpoint

Wraps MAME's `debug_breakpoint` class, representing a breakpoint for an emulated CPU device.

Instantiation

manager.machine.devices[tag].debug:bplist()[bp]
Gets the specified breakpoint for an emulated CPU device, or `nil` if no breakpoint corresponds to the specified index.

Properties

breakpoint.index (read-only)
The breakpoint's index. The can be used to enable, disable or clear the breakpoint via the *CPU debugger interface*.

breakpoint.enabled (read/write)
A Boolean indicating whether the breakpoint is currently enabled.

breakpoint.address (read-only)
The breakpoint's address.

breakpoint.condition (read-only)
A debugger expression evaluated each time the breakpoint is hit. The action will only be triggered if this expression evaluates to a non-zero value. An empty string if no condition was specified.

breakpoint.action (read-only)
An action the debugger will run when the breakpoint is hit and the condition evaluates to a non-zero value. An empty string if no action was specified.

Watchpoint

Wraps MAME's `debug_watchpoint` class, representing a watchpoint for an emulated CPU device.

Instantiation

manager.machine.devices[tag].debug:wplist(space)[wp]
Gets the specified watchpoint for an address space of an emulated CPU device, or `nil` if no watchpoint in the address space corresponds to the specified index.

Properties

watchpoint.index (read-only)
The watchpoint's index. The can be used to enable, disable or clear the watchpoint via the *CPU debugger interface*.

watchpoint.enabled (read/write)
A Boolean indicating whether the watchpoint is currently enabled.

watchpoint.type (read-only)
Either `"r"`, `"w"` or `"rw"` for a read, write or read/write watchpoint.

watchpoint.address (read-only)
The starting address of the watchpoint's address range.

watchpoint.length (read-only)
The length of the watchpoint's address range.

watchpoint.condition (read-only)
A debugger expression evaluated each time the watchpoint is hit. The action will only be triggered if this expression evaluates to a non-zero value. An empty string if no condition was specified.

watchpoint.action (read-only)
An action the debugger will run when the watchpoint is hit and the condition evaluates to a non-zero value. An empty string if no action was specified.

Expression error

Wraps MAME's `expression_error` class, describing an error occurring while parsing or executing a debugger expression. Raised on errors when using *parsed expressions*. Can be converted to a string to provide a description of the error.

Properties

err.code (read-only)
An implementation-dependent number representing the category of error. Should not be displayed to the user.

err.offset (read-only)
The offset within the expression string where the error was encountered.

9.4 Interactive Lua console tutorial

First run an arcade game in MAME at the command prompt with the `-console` and `-window` options to enable the Lua console:

```
$ mame -console -window YOUR_SYSTEM
       /|  /|    /|     /|  /|    _______
      / | / |   / |    / | / |   /      /
     /  |/  |  /  |   /  |/  |  /  ____/
    /       | /   |  /       | /  /_
   /        |/    | /        |/  __/
  /  /|  /|    /| |/  /|  /|    /____
 /  / | / |   / |    / | / |        /
/ _/  |/  /  /  |___/  |/  /_______/
        /  /
       / _/

mame 0.255
Copyright (C) Nicola Salmoria and the MAME team

Lua 5.4
Copyright (C) Lua.org, PUC-Rio

[MAME]>
```

At this point, your game is probably running in attract mode. Let's pause it:

```
[MAME]> emu.pause()
[MAME]>
```

Even without textual feedback on the console, you'll notice the game is now paused. In general, commands are quiet and only print error messages.

You can check the version of MAME you are running with:

```
[MAME]> print(emu.app_name() .. " " .. emu.app_version())
mame 0.255
```

Let's examine the emulated screens. First, enumerate the *screen devices* in the system:

```
[MAME]> for tag, screen in pairs(manager.machine.screens) do print(tag) end
:screen
```

`manager.machine` is the *running machine* object for the current emulation session. We will be using this frequently. `screens` is a *device enumerator* that yields all emulated screens in the system. Most arcade games only have one main screen. In our case, the main and only screen has the absolute tag `:screen`. We can examine it further:

```
[MAME]> -- keep a reference to the main screen in a variable
[MAME]> s = manager.machine.screens[':screen']
[MAME]> print(s.width .. 'x' .. s.height)
320x224
```

Several methods are available for drawing an overlay on the screen using lines, rectangles and text:

```
[MAME]> -- define a function for drawing an overlay and call it
[MAME]> function draw_overlay()
[MAME]>> s:draw_text(40, 40, 'foo') -- (x0, y0, msg)
[MAME]>> s:draw_box(20, 20, 80, 80, 0xff00ffff, 0) -- (x0, y0, x1, y1, line-color, fill-
↪color)
[MAME]>> s:draw_line(20, 20, 80, 80, 0xff00ffff) -- (x0, y0, x1, y1, line-color)
[MAME]>> end
[MAME]> draw_overlay()
```

This will draw some useless lines and text over the screen. However, when the emulated system is resumed, your overlay needs to be refreshed or it will just disappear. In order to do this, you have to register your function to be called on every video update:

```
[MAME]> emu.register_frame_done(draw_overlay, 'frame')
```

All colors are specified in ARGB format (eight bits per channel). The coordinate origin (0,0) normally corresponds to the top-left corner of the screen.

As with screens, you can examine all the emulated devices in the running system:

```
[MAME]> for tag, device in pairs(manager.machine.devices) do print(tag) end
:audiocpu
:maincpu
:saveram
:screen
:palette
[...]
```

For some of them, you can also inspect and manipulate memory and state:

```
[MAME]> cpu = manager.machine.devices[':maincpu']
[MAME]> -- enumerate, read and write register state
[MAME]> for k, v in pairs(cpu.state) do print(k) end
CURPC
rPC
IR
CURFLAGS
SSR
D0
[...]
[MAME]> print(cpu.state["D0"].value)
303
[MAME]> cpu.state['D0'].value = 255
[MAME]> print(cpu.state['D0'].value)
255
```

```
[MAME]> -- inspect memory
[MAME]> for name, space in pairs(cpu.spaces) do print(name) end
program
cpu_space
[MAME]> mem = cpu.spaces['program']
[MAME]> print(mem:read_i8(0xc000))
41
```

Note that many objects support symbol completion if you type part of a method or property name and press the Tab key:

```
[MAME]>print(mem:read_<TAB>
read_direct_i8
read_u16
read_range
read_direct_u16
read_direct_i64
read_i64
read_i32
read_direct_u64
read_i8
read_u32
read_u8
read_u64
read_direct_u32
read_direct_i16
read_direct_i32
read_direct_u8
read_i16
[MAME]>print(mem:read_direct_i8
```

CHAPTER

TEN

MAME EXTERNAL TOOLS

This section describes additional tools that are built alongside and typically distributed with MAME.

10.1 chdman – CHD (Compressed Hunks of Data) File Manager

chdman can be used to create, convert, check the integrity of and extract data from media images in CHD (Compressed Hunks of Data) format.

The basic usage is `chdman <command> <option>...`

- *Common options*
- *Commands*
  - *info*
  - *verify*
  - *createraw*
  - *createhd*
  - *createcd*
  - *createdvd*
  - *createld*
  - *extractraw*
  - *extracthd*
  - *extractcd*
  - *extractdvd*
  - *extractld*
  - *addmeta*
  - *delmeta*
  - *dumpmeta*
  - *listtemplates*
- *Compression algorithms*

10.1.1 Common options

The options available depend on the command, but the following options are used by multiple commands:

--input <file> / -i <file>
Specify the input file. This option is required for most commands. The input file formats supported depend on the command

--inputparent <chdfile> / -ip <chdfile>
Specify the parent CHD file for the input file. This option is supported for most commands that operate on CHD format input files. This option must be used if the input file is a *delta CHD*, storing only the hunks that differ from its parent CHD,

--inputstartbyte <offset> / -isb <offset>
Specify the offset to the data in the input file in bytes. This is useful for creating CHD format files from input files that contain a header before the start of the data, or for extracting partial content from a CHD format file. May not be specified in combination with the `--inputstarthunk`/`-ish` option.

--inputstarthunk <offset> / -ish <offset>
Specify the offset to the data in the input file in hunks. May not be specified in combination with the `--inputstartbyte`/`-isb` option.

--inputbytes <length> / -ib <length>
Specify the amount of input data to use in bytes, starting from the offset to the data in the input file. This is useful for creating CHD format files from input files that contain additional content after the data, or for extracting partial content from a CHD format file. May not be specified in combination with the `--inputhunks`/`-ih` option.

--inputhunks <length> / -ih <length>
Specify the amount of input data to use in hunks, starting from the offset to the data in the input file. May not be specified in combination with the `--inputbytes`/`-ib` option.

--output <file> / -o <file>
Specify the output file name. This option is required for commands that produce output files. The output file formats supported depend on the command.

--outputparent <chdfile> / -op <chdfile>
Specify the parent CHD file for the output file. This option is supported for commands that produce CHD format output files. Using this option produces a *delta CHD*, storing only the hunks that differ from its parent CHD. The parent CHD should be the same media type and size, with the same hunk size.

--compression none|<type>[,<type>]... / -c none|<type>[,<type>]...
Specify compression algorithms to use. This option is supported for commands that produce CHD format output files. Specify `none` to disable compression, or specify up to four comma-separated compression algorithms. See *compression algorithms* for supported compression algorithms. Compression must be disable to create writable media image files.

--hunksize <bytes> / -hs <bytes>
Specifies the hunk size in bytes. This option is supported for commands that produce CHD format output files. The hunk size must be no smaller than 16 bytes and no larger than 1048576 bytes (1 MiB). The hunk size must be a multiple of the sector size or unit size of the media. Larger hunk sizes may give better compression ratios, but reduce performance for small random reads as an entire hunk needs to be read and decompressed at a time.

--force / -f
Overwrite output files if they already exist. This option is supported for commands that produce output files.

--verbose / -v
Enable verbose output. This prints more detailed information for some commands.

--numprocessors <count> / -np <count>
Limit the maximum number of concurrent threads used for data compression. This option is supported for

commands that produce CHD format output files.

10.1.2 Commands

info

Display information about a CHD format file. Information includes:

- CHD format version and compression algorithms used.
- Compressed and uncompressed sizes and overall compression ratio.
- Hunk size, unit size and number of hunks in the file.
- SHA1 digests of the data and metadata.

Common options supported:

- `--input <file>` / `-i <file>` (required)
- `--verbose` / `-v`

verify

Verify the integrity of a CHD format file. The input file must be a read-only CHD format file (the integrity of writable CHD files cannot be verified). Note that this command modifies its input file if the `--fix`/`-f` option is specified.

Common options supported:

- `--input <file>` / `-i <file>` (required)
- `--inputparent <chdfile>` / `-ip <chdfile>`

Additional options:

- `--fix` / `-f`

createraw

Create a CHD format file from a raw media image.

Common options supported:

- `--input <file>` / `-i <file>` (required)
- `--inputstartbyte <offset>` / `-isb <offset>`
- `--inputstarthunk <offset>` / `-ish <offset>`
- `--inputbytes <length>` / `-ib <length>`
- `--inputhunks <length>` / `-ih <length>`
- `--output <file>` / `-o <file>` (required)
- `--outputparent <chdfile>` / `-op <chdfile>`
- `--compression none|<type>[,<type>]...` / `-c none|<type>[,<type>]...`
- `--hunksize <bytes>` / `-hs <bytes>`
- `--force` / `-f`

- `--numprocessors <count>` / `-np <count>`

Additional options:

--unitsize <bytes> / -us <bytes> (required)
The unit size for the output CHD file in bytes. This is the smallest unit of data that can be addressed within the CHD file. It should match the sector size or page size of the source media. The hunk size must be a whole multiple of the unit size. The unit size must be specified if no parent CHD file for the output is supplied. If a parent CHD file for the output is supplied, the unit size must match the unit size of the parent CHD file.

If the `--hunksize` or `-hs` option is not supplied, the default will be:

- The hunk size of the parent CHD file if a parent CHD file for the output is supplied.
- The smallest whole multiple of the unit size not larger than 4 KiB if the unit size is not larger than 4 KiB (4096 bytes).
- The unit size if it is larger than 4 KiB (4096 bytes).

If the `--compression` or `-c` option is not supplied, it defaults to `lzma,zlib,huff,flac`.

createhd

Create a CHD format hard disk image file.

Common options supported:

- `--input <file>` / `-i <file>`
- `--inputstartbyte <offset>` / `-isb <offset>`
- `--inputstarthunk <offset>` / `-ish <offset>`
- `--inputbytes <length>` / `-ib <length>`
- `--inputhunks <length>` / `-ih <length>`
- `--output <file>` / `-o <file>` (required)
- `--outputparent <chdfile>` / `-op <chdfile>`
- `--compression none|<type>[,<type>]...` / `-c none|<type>[,<type>]...`
- `--hunksize <bytes>` / `-hs <bytes>` (required)
- `--force` / `-f`
- `--numprocessors <count>` / `-np <count>`

Additional options:

- `--sectorsize <bytes>` / `-ss <bytes>`
- `--size <bytes>` / `-s <bytes>`
- `--chs <cylinders>,<heads>,<sectors>` / `-chs <cylinders>,<heads>,<sectors>`
- `--template <template>` / `-tp <template>`

Creates a blank (zero-filled) hard disk image if no input file is supplied. The input start/length (`--inputstartbyte`/`-isb`, `--inputstarthunk`/`-ish`, `--inputbytes`/`-ib` and `--inputhunks`/`-ih` options) cannot be used if no input file is supplied.

If the `--hunksize` or `-hs` option is not supplied, the default will be:

- The hunk size of the parent CHD file if a parent CHD file for the output is supplied.

- The smallest whole multiple of the sector size not larger than 4 KiB if the sector size is not larger than 4 KiB (4096 bytes).
- The sector size if it is larger than 4 KiB (4096 bytes).

If the `--compression` or `-c` option is not supplied, it defaults to `lzma,zlib,huff,flac` if an input file is supplied, or `none` if no input file is supplied.

createcd

Create a CHD format CD-ROM image file.

Common options supported:

- `--input <file>` / `-i <file>` (required)
- `--output <file>` / `-o <file>` (required)
- `--outputparent <chdfile>` / `-op <chdfile>`
- `--compression none|<type>[,<type>]...` / `-c none|<type>[,<type>]...`
- `--hunksize <bytes>` / `-hs <bytes>` (required)
- `--force` / `-f`
- `--numprocessors <count>` / `-np <count>`

If the `--hunksize` or `-hs` option is not supplied, the default will be the hunk size of the parent CHD if a parent CHD file for the output is supplied, or eight sectors per hunk (19,584 bytes) otherwise.

If the `--compression` or `-c` option is not supplied, it defaults to `cdlz,cdzl,cdfl`.

createdvd

Create a CHD format DVD-ROM image file.

Common options supported:

- `--input <file>` / `-i <file>` (required)
- `--inputstartbyte <offset>` / `-isb <offset>`
- `--inputstarthunk <offset>` / `-ish <offset>`
- `--inputbytes <length>` / `-ib <length>`
- `--inputhunks <length>` / `-ih <length>`
- `--output <file>` / `-o <file>` (required)
- `--outputparent <chdfile>` / `-op <chdfile>`
- `--compression none|<type>[,<type>]...` / `-c none|<type>[,<type>]...`
- `--hunksize <bytes>` / `-hs <bytes>` (required)
- `--force` / `-f`
- `--numprocessors <count>` / `-np <count>`

If the `--hunksize` or `-hs` option is not supplied, the default will be the hunk size of the parent CHD if a parent CHD file for the output is supplied, or two sectors per hunk (4096 bytes) otherwise.

If the `--compression` or `-c` option is not supplied, it defaults to `lzma,zlib,huff,flac`.

createld

Create a CHD format LaserDisc image file.

Common options supported:

- `--input <file>` / `-i <file>` (required)
- `--output <file>` / `-o <file>` (required)
- `--outputparent <chdfile>` / `-op <chdfile>`
- `--compression none|<type>[,<type>]...` / `-c none|<type>[,<type>]...`
- `--hunksize <bytes>` / `-hs <bytes>` (required)
- `--force` / `-f`
- `--numprocessors <count>` / `-np <count>`

Additional options:

- `--inputstartframe <offset>` / `-isf <offset>`
- `--inputframes <length>` / `-if <length>`

If the `--compression` or `-c` option is not supplied, it defaults to `avhu`.

extractraw

Extract data from a CHD format raw media image.

Common options supported:

- `--input <file>` / `-i <file>` (required)
- `--inputparent <chdfile>` / `-ip <chdfile>`
- `--inputstartbyte <offset>` / `-isb <offset>`
- `--inputstarthunk <offset>` / `-ish <offset>`
- `--inputbytes <length>` / `-ib <length>`
- `--inputhunks <length>` / `-ih <length>`
- `--output <file>` / `-o <file>` (required)
- `--force` / `-f`

extracthd

Extract data from a CHD format hard disk image.

Common options supported:

- `--input <file>` / `-i <file>` (required)
- `--inputparent <chdfile>` / `-ip <chdfile>`
- `--inputstartbyte <offset>` / `-isb <offset>`
- `--inputstarthunk <offset>` / `-ish <offset>`
- `--inputbytes <length>` / `-ib <length>`

- `--inputhunks <length>` / `-ih <length>`
- `--output <file>` / `-o <file>` (required)
- `--force` / `-f`

extractcd

Extract data from a CHD format CD-ROM image.

Common options supported:

- `--input <file>` / `-i <file>` (required)
- `--inputparent <chdfile>` / `-ip <chdfile>`
- `--output <file>` / `-o <file>` (required)
- `--force` / `-f`

Additional options:

- `--outputbin <file>` / `-ob <file>`
- `--splitbin` / `-sb`

extractdvd

Extract data from a CHD format DVD-ROM image.

Common options supported:

- `--input <file>` / `-i <file>` (required)
- `--inputparent <chdfile>` / `-ip <chdfile>`
- `--inputstartbyte <offset>` / `-isb <offset>`
- `--inputstarthunk <offset>` / `-ish <offset>`
- `--inputbytes <length>` / `-ib <length>`
- `--inputhunks <length>` / `-ih <length>`
- `--output <file>` / `-o <file>` (required)
- `--force` / `-f`

extractld

Extract data from a CHD format DVD-ROM image.

Common options supported:

- `--input <file>` / `-i <file>` (required)
- `--inputparent <chdfile>` / `-ip <chdfile>`
- `--output <file>` / `-o <file>` (required)
- `--force` / `-f`

Additional options:

- `--inputstartframe <offset>` / `-isf <offset>`

- --inputframes <length> / -if <length>

addmeta

Add a metadata item to a CHD format file. Note that this command modifies its input file.

Common options supported:

- --input <file> / -i <file> (required)

Additional options:

- --tag <tag> / -t <tag> (required)
- --index <index> / -ix <index>
- --valuetext <text> / -vt <text>
- --valuefile <file> / -vf <file>
- --nochecksum / -nocs

delmeta

Delete a metadata item from a CHD format file. Note that this command modifies its input file.

Common options supported:

- --input <file> / -i <file> (required)

Additional options:

- --tag <tag> / -t <tag> (required)
- --index <index> / -ix <index>

dumpmeta

Extract metadata items from a CHD format file to the standard output or to a file.

Common options supported:

- --input <file> / -i <file> (required)
- --output <file> / -o <file>
- --force / -f

Additional options:

- --tag <tag> / -t <tag> (required)
- --index <index> / -ix <index>

listtemplates

List available hard disk templates. This command does not accept any options.

10.1.3 Compression algorithms

The following compression algorithms are supported:

zlib – zlib deflate
Compresses data using the zlib deflate algorithm.

zstd – Zstandard
Compresses data using the Zstandard algorithm. This gives very good compression and decompression performance with better compression ratios than zlib deflate, but older software may not support CHD files that use Zstandard compression.

lzma – Lempel-Ziv-Markov chain algorithm
Compresses data using the Lempel-Ziv-Markov-chain algorithm (LZMA). This gives high compression ratios at the cost of poor compression and decompression performance.

huff – Huffman coding
Compresses data using 8-bit Huffman entropy coding.

flac – Free Lossless Audio Codec
Compresses data as two-channel (stereo) 16-bit 44.1 kHz PCM audio using the Free Lossless Audio Codec (FLAC). This gives good compression ratios if the media contains 16-bit PCM audio data.

cdzl – zlib deflate for CD-ROM data
Compresses audio data and subchannel data from CD-ROM sectors separately using the zlib deflate algorithm.

cdzs – Zstandard for CD-ROM data
Compresses audio data and subchannel data from CD-ROM sectors separately using the Zstandard algorithm. This gives very good compression and decompression performance with better compression ratios than zlib deflate, but older software may not support CHD files that use Zstandard compression.

cdlz - Lempel-Ziv-Markov chain algorithm/zlib deflate for CD-ROM data
Compresses audio data and subchannel data from CD-ROM sectors separately using the Lempel-Ziv-Markov chain algorithm (LZMA) for audio data and the zlib deflate algorithm for subchannel data. This gives high compression ratios at the cost of poor compression and decompression performance.

cdfl – Free Lossless Audio Codec/zlib deflate for CD-ROM data
Compresses audio data and subchannel data from CD-ROM sectors separately using the Free Lossless Audio Codec (FLAC) for audio data and the zlib deflate algorithm for subchannel data. This gives good compression ratios for audio CD tracks.

avhu – Huffman coding for audio-visual data
This is a specialised compression algorithm for audio-visual (A/V) data. It should only be used for LaserDisc CHD files.

10.2 Imgtool - *A generic image manipulation tool for MAME*

Imgtool is a tool for the maintenance and manipulation of disk and other types of images that MAME users need to deal with. Functions include retrieving and storing files and CRC checking/validation.

Imgtool is part of the MAME project. It shares large portions of code with MAME, and its existence would not be if it were not for MAME. As such, the distribution terms are the same as MAME. Please read the MAME license thoroughly.

Some portions of Imgtool are Copyright (c) 1989, 1993 The Regents of the University of California. All rights reserved.

10.2.1 Using Imgtool

Imgtool is a command line program that contains several "subcommands" that actually do all of the work. Most commands are invoked in a manner along the lines of this:

imgtool <subcommand> <format> <image> ...

- **<subcommand>** is the name of the subcommand
- **<format>** is the format of the image
- **<image>** is the filename of the image

Example usage:

imgtool dir coco_jvc_rsdos myimageinazip.zip

imgtool get coco_jvc_rsdos myimage.dsk myfile.bin mynewfile.txt

imgtool getall coco_jvc_rsdos myimage.dsk

Further details vary with each subcommand. Also note that not all subcommands are applicable or supported for different image formats.

10.2.2 Imgtool Subcommands

create

imgtool create <format> <imagename> [--(createoption)=value]

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Creates an image

dir

imgtool dir <format> <imagename> [path]

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Lists the contents of an image

get

imgtool get <format> <imagename> <filename> [newname] [--filter=filter] [--fork=fork]

- <format> is the image format, e.g. coco_jvc_rsdos

- <imagename> is the image filename; can specify a ZIP file for image name

Gets a single file from an image

put

imgtool put <format> <imagename> <filename>... <destname> [--(fileoption)=value] [--filter=filter] [--fork=fork]

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Puts a single file on an image (wildcards supported)

getall

imgtool getall <format> <imagename> [path] [--filter=filter]

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Gets all files off an image

del

imgtool del <format> <imagename> <filename>...

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Deletes a file on an image

mkdir

imgtool mkdir <format> <imagename> <dirname>

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Creates a subdirectory on an image

rmdir

imgtool rmdir <format> <imagename> <dirname>...

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Deletes a subdirectory on an image

readsector

imgtool readsector <format> <imagename> <track> <head> <sector> <filename>

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

Read a sector on an image and output it to specified <filename>

writesector

imgtool writesector <format> <imagename> <track> <head> <sector> <filename>

- <format> is the image format, e.g. coco_jvc_rsdos

- <imagename> is the image filename; can specify a ZIP file for image name

Write a sector to an image from specified <filename>

identify

- <format> is the image format, e.g. coco_jvc_rsdos
- <imagename> is the image filename; can specify a ZIP file for image name

imgtool identify <imagename>

listformats

Lists all image file formats supported by imgtool

listfilters

Lists all filters supported by imgtool

listdriveroptions

imgtool listdriveroptions <format>

- <format> is the image format, e.g. coco_jvc_rsdos

Lists all format-specific options for the 'put' and 'create' commands

10.2.3 Imgtool Filters

Filters are a means to process data being written into or read out of an image in a certain way. Filters can be specified on the get, put, and getall commands by specifying --filter=xxxx on the command line. Currently, the following filters are supported:

ascii

Translates end-of-lines to the appropriate format

cocobas

Processes tokenized TRS-80 Color Computer (CoCo) BASIC programs

dragonbas

Processes tokenized Tano/Dragon Data Dragon 32/64 BASIC programs

macbinary

Processes Apple MacBinary-formatted (merged forks) files

vzsnapshot

[todo: VZ Snapshot? Find out what this is...]

vzbas

Processes Laser/VZ Tokenized Basic Files

thombas5

Thomson MO5 w/ BASIC 1.0, Tokenized Files (read-only, auto-decrypt)

thombas7

Thomson TO7 w/ BASIC 1.0, Tokenized Files (read-only, auto-decrypt)

thombas128

Thomson w/ BASIC 128/512, Tokenized Files (read-only, auto-decrypt)

thomcrypt

Thomson BASIC, Protected file encryption (no tokenization)

bm13bas

Basic Master Level 3 Tokenized Basic Files

10.3 Imgtool Format Info

10.3.1 Amiga floppy disk image (OFS/FFS format) - (*amiga_floppy*)

Driver specific options for module 'amiga_floppy':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--density	dd/hd	Density
--filesystem	ofs/ffs	File system
--mode	none/intl/dirc	File system options

10.3.2 Apple][DOS order disk image (ProDOS format) - (*apple2_do_prodos_525*)

Driver specific options for module 'apple2_do_prodos_525':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1	Heads
--tracks	35	Tracks
--sectors	16	Sectors
--sectorlength	256	Sector Bytes
--firstsectorid	0	First Sector

10.3.3 Apple][Nibble order disk image (ProDOS format) - (*apple2_nib_prodos_525*)

Driver specific options for module 'apple2_nib_prodos_525':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1	Heads
--tracks	35	Tracks
--sectors	16	Sectors
--sectorlength	256	Sector Bytes
--firstsectorid	0	First Sector

10.3.4 Apple][ProDOS order disk image (ProDOS format) - (*apple2_po_prodos_525*)

Driver specific options for module 'apple2_po_prodos_525':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1	Heads
--tracks	35	Tracks
--sectors	16	Sectors
--sectorlength	256	Sector Bytes
--firstsectorid	0	First Sector

10.3.5 Apple][gs 2IMG disk image (ProDOS format) - (*apple35_2img_prodos_35*)

Driver specific options for module 'apple35_2img_prodos_35':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1-2	Heads
--tracks	80	Tracks
--sectorlength	512	Sector Bytes
--firstsectorid	0	First Sector

10.3.6 Apple DiskCopy disk image (Mac HFS Floppy) - (*apple35_dc_mac_hfs*)

Driver specific options for module 'apple35_dc_mac_hfs':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1-2	Heads
--tracks	80	Tracks
--sectorlength	512	Sector Bytes
--firstsectorid	0	First Sector

10.3.7 Apple DiskCopy disk image (Mac MFS Floppy) - (*apple35_dc_mac_mfs*)

Driver specific options for module 'apple35_dc_mac_mfs':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1-2	Heads
--tracks	80	Tracks
--sectorlength	512	Sector Bytes
--firstsectorid	0	First Sector

10.3.8 Apple DiskCopy disk image (ProDOS format) - (*apple35_dc_prodos_35*)

Driver specific options for module 'apple35_dc_prodos_35':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1-2	Heads
--tracks	80	Tracks
--sectorlength	512	Sector Bytes
--firstsectorid	0	First Sector

10.3.9 Apple raw 3.5" disk image (Mac HFS Floppy) - (*apple35_raw_mac_hfs*)

Driver specific options for module 'apple35_raw_mac_hfs':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1-2	Heads
--tracks	80	Tracks
--sectorlength	512	Sector Bytes
--firstsectorid	0	First Sector

10.3.10 Apple raw 3.5" disk image (Mac MFS Floppy) - (*apple35_raw_mac_mfs*)

Driver specific options for module 'apple35_raw_mac_mfs':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1-2	Heads
--tracks	80	Tracks
--sectorlength	512	Sector Bytes
--firstsectorid	0	First Sector

10.3.11 Apple raw 3.5" disk image (ProDOS format) - (*apple35_raw_prodos_35*)

Driver specific options for module 'apple35_raw_prodos_35':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1-2	Heads
--tracks	80	Tracks
--sectorlength	512	Sector Bytes
--firstsectorid	0	First Sector

10.3.12 CoCo DMK disk image (OS-9 format) - (*coco_dmk_os9*)

Driver specific options for module 'coco_dmk_os9':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1-2	Heads
--tracks	35-255	Tracks
--sectors	1-18	Sectors
--sectorlength	128/256/512/1024/2048/4096/8192	Sector Bytes
--interleave	0-17	Interleave
--firstsectorid	0-1	First Sector

10.3.13 CoCo DMK disk image (RS-DOS format) - (*coco_dmk_rsdos*)

Driver specific options for module 'coco_dmk_rsdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/data/binary/assembler	File type
--ascii	ascii/binary	ASCII flag

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1-2	Heads
--tracks	35-255	Tracks
--sectors	1-18	Sectors
--sectorlength	128/256/512/1024/2048/4096/8192	Sector Bytes
--interleave	0-17	Interleave
--firstsectorid	0-1	First Sector

10.3.14 CoCo JVC disk image (OS-9 format) - (*coco_jvc_os9*)

Driver specific options for module 'coco_jvc_os9':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1-2	Heads
--tracks	35-255	Tracks
--sectors	1-255	Sectors
--sectorlength	128/256/512/1024	Sector Bytes
--firstsectorid	0-1	First Sector

10.3.15 CoCo JVC disk image (RS-DOS format) - (*coco_jvc_rsdos*)

Driver specific options for module 'coco_jvc_rsdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/data/binary/assembler	File type
--ascii	ascii/binary	ASCII flag

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1-2	Heads
--tracks	35-255	Tracks
--sectors	1-255	Sectors
--sectorlength	128/256/512/1024	Sector Bytes
--firstsectorid	0-1	First Sector

10.3.16 CoCo OS-9 disk image (OS-9 format) - (*coco_os9_os9*)

Driver specific options for module 'coco_os9_os9':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1-2	Heads
--tracks	35-255	Tracks
--sectors	1-255	Sectors
--sectorlength	128/256/512/1024	Sector Bytes
--firstsectorid	1	First Sector

10.3.17 CoCo VDK disk image (OS-9 format) - (*coco_vdk_os9*)

Driver specific options for module 'coco_vdk_os9':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1-2	Heads
--tracks	35-255	Tracks
--sectors	18	Sectors
--sectorlength	256	Sector Bytes
--firstsectorid	1	First Sector

10.3.18 CoCo VDK disk image (RS-DOS format) - (*coco_vdk_rsdos*)

Driver specific options for module 'coco_vdk_rsdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/data/binary/assembler	File type
--ascii	ascii/binary	ASCII flag

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1-2	Heads
--tracks	35-255	Tracks
--sectors	18	Sectors
--sectorlength	256	Sector Bytes
--firstsectorid	1	First Sector

10.3.19 Concept floppy disk image - (*concept*)

Driver specific options for module 'concept':

No image specific file options

No image specific creation options

10.3.20 CopyQM floppy disk image (Basic Master Level 3 format) - (*cqm_bml3*)

Driver specific options for module 'cqm_bml3':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/data/binary/assembler	File type
--ascii	ascii/binary	ASCII flag

No image specific creation options

10.3.21 CopyQM floppy disk image (FAT format) - (*cqm_fat*)

Driver specific options for module 'cqm_fat':

No image specific file options

No image specific creation options

10.3.22 CopyQM floppy disk image (Mac HFS Floppy) - (*cqm_mac_hfs*)

Driver specific options for module 'cqm_mac_hfs':

No image specific file options

No image specific creation options

10.3.23 CopyQM floppy disk image (Mac MFS Floppy) - (*cqm_mac_mfs*)

Driver specific options for module 'cqm_mac_mfs':

No image specific file options

No image specific creation options

10.3.24 CopyQM floppy disk image (OS-9 format) - (*cqm_os9*)

Driver specific options for module 'cqm_os9':

No image specific file options

No image specific creation options

10.3.25 CopyQM floppy disk image (ProDOS format) - (*cqm_prodos_35*)

Driver specific options for module 'cqm_prodos_35':

No image specific file options

No image specific creation options

10.3.26 CopyQM floppy disk image (ProDOS format) - (*cqm_prodos_525*)

Driver specific options for module 'cqm_prodos_525':

No image specific file options

No image specific creation options

10.3.27 CopyQM floppy disk image (RS-DOS format) - (*cqm_rsdos*)

Driver specific options for module 'cqm_rsdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/data/binary/assembler	File type
--ascii	ascii/binary	ASCII flag

No image specific creation options

10.3.28 CopyQM floppy disk image (VZ-DOS format) - (*cqm_vzdos*)

Driver specific options for module 'cqm_vzdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/binary/data	File type
--fname	intern/extern	Filename

No image specific creation options

10.3.29 Cybiko Classic File System - (*cybiko*)

Driver specific options for module 'cybiko':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--flash	AT45DB041/AT45DB081/AT45DB161	Flash Type

10.3.30 Cybiko Xtreme File System - (*cybikoxt*)

Driver specific options for module 'cybikoxt':

No image specific file options

No image specific creation options

10.3.31 D88 Floppy Disk image (Basic Master Level 3 format) - (*d88_bml3*)

Driver specific options for module 'd88_bml3':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/data/binary/assembler	File type
--ascii	ascii/binary	ASCII flag

No image specific creation options

10.3.32 D88 Floppy Disk image (FAT format) - (*d88_fat*)

Driver specific options for module 'd88_fat':

No image specific file options

No image specific creation options

10.3.33 D88 Floppy Disk image (Mac HFS Floppy) - (*d88_mac_hfs*)

Driver specific options for module 'd88_mac_hfs':

No image specific file options

No image specific creation options

10.3.34 D88 Floppy Disk image (Mac MFS Floppy) - (*d88_mac_mfs*)

Driver specific options for module 'd88_mac_mfs':

No image specific file options

No image specific creation options

10.3.35 D88 Floppy Disk image (OS-9 format) - (*d88_os9*)

Driver specific options for module 'd88_os9':

No image specific file options

No image specific creation options

10.3.36 D88 Floppy Disk image (OS-9 format) - (*d88_os9*)

Driver specific options for module 'd88_prodos_35':

No image specific file options

No image specific creation options

10.3.37 D88 Floppy Disk image (ProDOS format) - (*d88_prodos_525*)

Driver specific options for module 'd88_prodos_525':

No image specific file options

No image specific creation options

10.3.38 D88 Floppy Disk image (RS-DOS format) - (*d88_rsdos*)

Driver specific options for module 'd88_rsdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/data/binary/assembler	File type
--ascii	ascii/binary	ASCII flag

No image specific creation options

10.3.39 D88 Floppy Disk image (VZ-DOS format) - (*d88_vzdos*)

Driver specific options for module 'd88_vzdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/binary/data	File type
--fname	intern/extern	Filename

No image specific creation options

10.3.40 DSK floppy disk image (Basic Master Level 3 format) - (*dsk_bml3*)

Driver specific options for module 'dsk_bml3':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/data/binary/assembler	File type
--ascii	ascii/binary	ASCII flag

No image specific creation options

10.3.41 DSK floppy disk image (FAT format) - (*dsk_fat*)

Driver specific options for module 'dsk_fat':

No image specific file options

No image specific creation options

10.3.42 DSK floppy disk image (Mac HFS Floppy) - (*dsk_mac_hfs*)

Driver specific options for module 'dsk_mac_hfs':

No image specific file options

No image specific creation options

10.3.43 DSK floppy disk image (Mac MFS Floppy) - (*dsk_mac_mfs*)

Driver specific options for module 'dsk_mac_mfs':

No image specific file options

No image specific creation options

10.3.44 DSK floppy disk image (OS-9 format) - (*dsk_os9*)

Driver specific options for module 'dsk_os9':

No image specific file options

No image specific creation options

10.3.45 DSK floppy disk image (ProDOS format) - (*dsk_prodos_35*)

Driver specific options for module 'dsk_prodos_35':

No image specific file options

No image specific creation options

10.3.46 DSK floppy disk image (ProDOS format) - (*dsk_prodos_525*)

Driver specific options for module 'dsk_prodos_525':

No image specific file options

No image specific creation options

10.3.47 DSK floppy disk image (RS-DOS format) - (*dsk_rsdos*)

Driver specific options for module 'dsk_rsdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/data/binary/assembler	File type
--ascii	ascii/binary	ASCII flag

No image specific creation options

10.3.48 DSK floppy disk image (VZ-DOS format) - (*dsk_vzdos*)

Driver specific options for module 'dsk_vzdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/binary/data	File type
--fname	intern/extern	Filename

No image specific creation options

10.3.49 Formatted Disk Image (Basic Master Level 3 format) - (*fdi_bml3*)

Driver specific options for module 'fdi_bml3':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/data/binary/assembler	File type
--ascii	ascii/binary	ASCII flag

No image specific creation options

10.3.50 Formatted Disk Image (FAT format) - (*fdi_fat*)

Driver specific options for module 'fdi_fat':

No image specific file options

No image specific creation options

10.3.51 Formatted Disk Image (Mac HFS Floppy) - (*fdi_mac_hfs*)

Driver specific options for module 'fdi_mac_hfs':

No image specific file options

No image specific creation options

10.3.52 Formatted Disk Image (Mac MFS Floppy) - (*fdi_mac_mfs*)

Driver specific options for module 'fdi_mac_mfs':

No image specific file options

No image specific creation options

10.3.53 Formatted Disk Image (OS-9 format) - (*fdi_os9*)

Driver specific options for module 'fdi_os9':

No image specific file options

No image specific creation options

10.3.54 Formatted Disk Image (ProDOS format) - (*fdi_prodos_35*)

Driver specific options for module 'fdi_prodos_35':

No image specific file options

No image specific creation options

10.3.55 Formatted Disk Image (ProDOS format) - (*fdi_prodos_525*)

Driver specific options for module 'fdi_prodos_525':

No image specific file options

No image specific creation options

10.3.56 Formatted Disk Image (RS-DOS format) - (*fdi_rsdos*)

Driver specific options for module 'fdi_rsdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/data/binary/assembler	File type
--ascii	ascii/binary	ASCII flag

No image specific creation options

10.3.57 Formatted Disk Image (VZ-DOS format) - (*fdi_vzdos*)

Driver specific options for module 'fdi_vzdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/binary/data	File type
--fname	intern/extern	Filename

No image specific creation options

10.3.58 HP48 SX/GX memory card - (*hp48*)

Driver specific options for module 'hp48':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--size	32/64/128/256/512/1024/2048/4096	Size in KB

10.3.59 IMD floppy disk image (Basic Master Level 3 format) - (*imd_bml3*)

Driver specific options for module 'imd_bml3':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/data/binary/assembler	File type
--ascii	ascii/binary	ASCII flag

No image specific creation options

10.3.60 IMD floppy disk image (FAT format) - (*imd_fat*)

Driver specific options for module 'imd_fat':

No image specific file options

No image specific creation options

10.3.61 IMD floppy disk image (Mac HFS Floppy) - (*imd_mac_hfs*)

Driver specific options for module 'imd_mac_hfs':

No image specific file options

No image specific creation options

10.3.62 IMD floppy disk image (Mac MFS Floppy) - (*imd_mac_mfs*)

Driver specific options for module 'imd_mac_mfs':

No image specific file options

No image specific creation options

10.3.63 IMD floppy disk image (OS-9 format) - (*imd_os9*)

Driver specific options for module 'imd_os9':

No image specific file options

No image specific creation options

10.3.64 IMD floppy disk image (ProDOS format) - (*imd_prodos_35*)

Driver specific options for module 'imd_prodos_35':

No image specific file options

No image specific creation options

10.3.65 IMD floppy disk image (ProDOS format) - (*imd_prodos_525*)

Driver specific options for module 'imd_prodos_525':

No image specific file options

No image specific creation options

10.3.66 IMD floppy disk image (RS-DOS format) - (*imd_rsdos*)

Driver specific options for module 'imd_rsdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/data/binary/assembler	File type
--ascii	ascii/binary	ASCII flag

No image specific creation options

10.3.67 IMD floppy disk image (VZ-DOS format) - (*imd_vzdos*)

Driver specific options for module 'imd_vzdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/binary/data	File type
--fname	intern/extern	Filename

No image specific creation options

10.3.68 MESS hard disk image - (*mess_hd*)

Driver specific options for module 'mess_hd':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--blocksize	1-2048	Sectors Per Block
--cylinders	1-65536	Cylinders
--heads	1-64	Heads
--sectors	1-4096	Total Sectors
--seclen	128/256/512/1024/2048/4096/8192/16384/32768/65536	Sector Bytes

10.3.69 TI99 Diskette (PC99 FM format) - (*pc99fm*)

Driver specific options for module 'pc99fm':

No image specific file options

No image specific creation options

10.3.70 TI99 Diskette (PC99 MFM format) - (*pc99mfm*)

Driver specific options for module 'pc99mfm':

No image specific file options

No image specific creation options

10.3.71 PC CHD disk image - (*pc_chd*)

Driver specific options for module 'pc_chd':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--cylinders	10/20/30/40/50/60/70/80/90/100/110/120/130/140/150/160/170/180/190/200	Cylinders
--heads	1-16	Heads
--sectors	1-63	Sectors

10.3.72 PC floppy disk image (FAT format) - (*pc_dsk_fat*)

Driver specific options for module 'pc_dsk_fat':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1-2	Heads
--tracks	40/80	Tracks
--sectors	8/9/10/15/18/36	Sectors

10.3.73 Psion Organiser II Datapack - (*psionpack*)

Driver specific options for module 'psionpack':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--type	OB3/OPL/ODB	file type
--id	0/145-255	File ID

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--size	8k/16k/32k/64k/128k	datapack size
--ram	0/1	EPROM/RAM datapack
--paged	0/1	linear/paged datapack
--protect	0/1	write-protected datapack
--boot	0/1	bootable datapack
--copy	0/1	copyable datapack

10.3.74 Teledisk floppy disk image (Basic Master Level 3 format) - (*td0_bml3*)

Driver specific options for module 'td0_bml3':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/data/binary/assembler	File type
--ascii	ascii/binary	ASCII flag

No image specific creation options

10.3.75 Teledisk floppy disk image (FAT format) - (*td0_fat*)

Driver specific options for module 'td0_fat':

No image specific file options

No image specific creation options

10.3.76 Teledisk floppy disk image (Mac HFS Floppy) - (*td0_mac_hfs*)

Driver specific options for module 'td0_mac_hfs':

No image specific file options

No image specific creation options

10.3.77 Teledisk floppy disk image (Mac MFS Floppy) - (*td0_mac_mfs*)

Driver specific options for module 'td0_mac_mfs':

No image specific file options

No image specific creation options

10.3.78 Teledisk floppy disk image (OS-9 format) - (*td0_os9*)

Driver specific options for module 'td0_os9':

No image specific file options

No image specific creation options

10.3.79 Teledisk floppy disk image (ProDOS format) - (*td0_prodos_35*)

Driver specific options for module 'td0_prodos_35':

No image specific file options

No image specific creation options

10.3.80 Teledisk floppy disk image (ProDOS format) - (*td0_prodos_525*)

Driver specific options for module 'td0_prodos_525':

No image specific file options

No image specific creation options

10.3.81 Teledisk floppy disk image (RS-DOS format) - (*td0_rsdos*)

Driver specific options for module 'td0_rsdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/data/binary/assembler	File type
--ascii	ascii/binary	ASCII flag

No image specific creation options

10.3.82 Teledisk floppy disk image (VZ-DOS format) - (*td0_vzdos*)

Driver specific options for module 'td0_vzdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/binary/data	File type
--fname	intern/extern	Filename

No image specific creation options

10.3.83 Thomson .fd disk image, BASIC format - (*thom_fd*)

Driver specific options for module 'thom_fd':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	auto/B/D/M/A	File type
--format	auto/B/A	Format flag
--comment	(string)	Comment

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1-2	Heads
--tracks	40/80	Tracks
--density	SD/DD	Density
--name	(string)	Floppy name

10.3.84 Thomson .qd disk image, BASIC format - (*thom_qd*)

Driver specific options for module 'thom_qd':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	auto/B/D/M/A	File type
--format	auto/B/A	Format flag
--comment	(string)	Comment

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1-2	Heads
--tracks	25	Tracks
--density	SD/DD	Density
--name	(string)	Floppy name

10.3.85 Thomson .sap disk image, BASIC format - (*thom_sap*)

Driver specific options for module 'thom_sap':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	auto/B/D/M/A	File type
--format	auto/B/A	Format flag
--comment	(string)	Comment

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1	Heads
--tracks	40/80	Tracks
--density	SD/DD	Density
--name	(string)	Floppy name

10.3.86 TI990 Hard Disk - (*ti990hd*)

Driver specific options for module 'ti990hd':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--cylinders	1-2047	Cylinders
--heads	1-31	Heads
--sectors	1-256	Sectors
--bytes per sector	(typically 25256-512 256-512	Bytes Per Sector [Todo: This section is glitched in imgtool]

10.3.87 TI99 Diskette (old MESS format) - (*ti99_old*)

Driver specific options for module 'ti99_old':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--sides	1-2	Sides
--tracks	1-80	Tracks
--sectors	1-36	Sectors (1->9 for SD, 1->18 for DD, 1->36 for HD)
--protection	0-1	Protection (0 for normal, 1 for protected)
--density	Auto/SD/DD/HD	Density

10.3.88 TI99 Harddisk - (*ti99hd*)

Driver specific options for module 'ti99hd':

No image specific file options

No image specific creation options

10.3.89 TI99 Diskette (V9T9 format) - (*v9t9*)

Driver specific options for module 'v9t9':

No image specific file options

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--sides	1-2	Sides
--tracks	1-80	Tracks
--sectors	1-36	Sectors (1->9 for SD, 1->18 for DD, 1->36 for HD)
--protection	0-1	Protection (0 for normal, 1 for protected)
--density	Auto/SD/DD/HD	Density

10.3.90 Laser/VZ disk image (VZ-DOS format) - (*vtech1_vzdos*)

Driver specific options for module 'vtech1_vzdos':

Image specific file options (usable on the 'put' command):

Option	Allowed values	Description
--ftype	basic/binary/data	File type
--fname	intern/extern	Filename

Image specific creation options (usable on the 'create' command):

Option	Allowed values	Description
--heads	1	Heads
--tracks	40	Tracks
--sectors	16	Sectors
--sectorlength	154	Sector Bytes
--firstsectorid	0	First Sector

[todo: fill out the command structures, describe commands better. These descriptions came from the imgtool.txt file and are barebones]

10.4 Castool - *A generic cassette image manipulation tool for MAME*

Castool is a tool for the maintenance and manipulation of cassette images that MAME users need to deal with. MAME directly supports .WAV audio formatted images, but many of the existing images out there may come in forms such as .TAP for Commodore 64 tapes, .CAS for Tandy Color Computer tapes, and so forth. Castool will convert these other formats to .WAV for use in MAME.

Castool is part of the MAME project. It shares large portions of code with MAME, and its existence would not be if it were not for MAME. As such, the distribution terms are the same as MAME. Please read the MAME license thoroughly.

10.4.1 Using Castool

Castool is a command line program that contains a simple set of instructions. Commands are invoked in a manner along the lines of this:

castool convert <format> <inputfile> <outputfile>

- **<format>** is the format of the image
- **<inputfile>** is the filename of the image you're converting from
- **<outputfile>** is the filename of the output WAV file

Example usage:

castool convert coco zaxxon.cas zaxxon.wav

castool convert cbm arkanoid.tap arkanoid.wav

castool convert ddp mybasicprogram.ddp mybasicprogram.wav

10.4.2 Castool Formats

These are the formats supported by Castool for conversion to .WAV files.

A26

Atari 2600 SuperCharger image

File extension: a26

APF

APF Imagination Machine

File extensions: cas, cpf, apt

ATOM

Acorn Atom

File extensions: tap, csw, uef

BBC

Acorn BBC & Electron

File extensions: csw, uef

CBM

Commodore 8-bit series

File extensions: tap

CDT

Amstrad CPC

File extensions: cdt

CGENIE

EACA Colour Genie

File extensions: cas

COCO

Tandy Radio Shack Color Computer

File extensions: cas

CSW

Compressed Square Wave

File extensions: csw

DDP

Coleco ADAM

File extensions: ddp

FM7

Fujitsu FM-7

File extensions: t77

FMSX

MSX

File extensions: tap, cas

GTP

Elektronika inzenjering Galaksija

File extensions: gtp

HECTOR

Micronique Hector & Interact Family Computer

File extensions: k7, cin, for

JUPITER

Jupiter Cantab Jupiter Ace

File extensions: tap

KC85

VEB Mikroelektronik KC 85

File extensions: kcc, kcb, tap, 853, 854, 855, tp2, kcm, sss

KIM1

MOS KIM-1

File extensions: kim, kim1

LVIV

PK-01 Lviv

File extensions: lvt, lvr, lv0, lv1, lv2, lv3

MO5

Thomson MO-series

File extensions: k5, k7

MZ

Sharp MZ-700

File extensions: m12, mzf, mzt

ORAO

PEL Varazdin Orao

File extensions: tap

ORIC

Tangerine Oric

File extensions: tap

PC6001

NEC PC-6001

File extensions: cas

PHC25

Sanyo PHC-25

File extensions: phc

PMD85

Tesla PMD-85

File extensions: pmd, tap, ptp

PRIMO

Microkey Primo

File extensions: ptp

RKU

UT-88

File extensions: rku

RK8

Mikro-80

File extensions: rk8

RKS

Specialist

File extensions: rks

RKO

Orion

File extensions: rko

RKR

Radio-86RK

File extensions: rk, rkr, gam, g16, pki

RKA

Zavod BRA Apogee BK-01

File extensions: rka

RKM

Mikrosha

File extensions: rkm

RKP

SAM SKB VM Partner-01.01

File extensions: rkp

SC3000

Sega SC-3000

File extensions: bit

SOL20

PTC SOL-20

File extensions: svt

SORCERER

Exidy Sorcerer

File extensions: tape

SORDM5

Sord M5

File extensions: cas

SPC1000

Samsung SPC-1000

File extensions: tap, cas

SVI

Spectravideo SVI-318 & SVI-328

File extensions: cas

TO7

Thomson TO-series

File extensions: k7

TRS8012

TRS-80 Level 2

File extensions: cas

TVC64

Videoton TVC 64

File extensions: cas

TZX

Sinclair ZX Spectrum

File extensions: tzx, tap, blk

VG5K

Philips VG 5000

File extensions: k7

VTECH1

Video Technology Laser 110-310

File extensions: cas

VTECH2

Video Technology Laser 350-700

File extensions: cas

X07

Canon X-07

File extensions: k7, lst, cas

X1

Sharp X1

File extensions: tap

ZX80_O

Sinclair ZX80

File extensions: o, 80

ZX81_P

Sinclair ZX81

File extensions: p, 81

10.5 Floptool - *A generic floppy image manipulation tool for MAME*

Floptool is a tool for the maintenance and manipulation of floppy images that MAME users need to deal with. MAME directly supports .WAV audio formatted images, but many of the existing images out there may come in forms such as .TAP for Commodore 64 tapes, .CAS for Tandy Color Computer tapes, and so forth. Castool will convert these other formats to .WAV for use in MAME.

Floptool is part of the MAME project. It shares large portions of code with MAME, and its existence would not be if it were not for MAME. As such, the distribution terms are the same as MAME. Please read the MAME license thoroughly.

10.5.1 Using Floptool

Floptool is a command line program that contains a simple set of instructions. Commands are invoked in a manner along the lines of this:

floptool identify <inputfile> [<inputfile> ...] floptool convert [input_format|auto] output_format <inputfile> <outputile>

- **<format>** is the format of the image
- **<input_format>** is the format of the inputfile, use auto if not known
- **<output_format>** is the format of the converted file
- **<inputfile>** is the filename of the image you're identifying/converting from
- **<outputfile>** is the filename of the converted file

Example usage:

floptool convert coco zaxxon.cas zaxxon.wav

floptool convert cbm arkanoid.tap arkanoid.wav

floptool convert ddp mybasicprogram.ddp mybasicprogram.wav

10.5.2 Floptool Formats

These are the formats supported by Floptool for conversion to other formats.

MFI

MAME floppy image

File extension: mfi

DFI

DiscFerret flux dump format

File extensions: dfi

IPF

SPS floppy disk image

File extensions: ipf

MFM

HxC Floppy Emulator floppy disk image

File extensions: mfm

ADF

Amiga ADF floppy disk image

File extensions: adf

ST

Atari ST floppy disk image

File extensions: st

MSA

Atari MSA floppy disk image

File extensions: msa

PASTI

Atari PASTI floppy disk image

File extensions: stx

DSK

CPC DSK format

File extensions: dsk

D88

D88 disk image

File extensions: d77, d88, 1dd

IMD

IMD disk image

File extensions: imd

TD0

Teledisk disk image

File extensions: td0

CQM

CopyQM disk image

File extensions: cqm, cqi, dsk

PC

PC floppy disk image

File extensions: dsk, ima, img, ufi, 360

NASLITE

NASLite disk image

File extensions: img

DC42

DiskCopy 4.2 image

File extensions: dc42

A2_16SECT

Apple II 16-sector disk image

File extensions: dsk, do, po

A2_EDD

Apple II EDD image

File extensions: edd

ATOM

Acorn Atom disk image

File extensions: 40t, dsk

SSD

Acorn SSD disk image

File extensions: ssd, bbc, img

DSD

Acorn DSD disk image

File extensions: dsd

DOS

Acorn DOS disk image

File extensions: img

ADFS_O

Acorn ADFS (OldMap) disk image

File extensions: adf, ads, adm, adl

ADFS_N

Acorn ADFS (NewMap) disk image

File extensions: adf

ORIC_DSK

Oric disk image

File extensions: dsk

APPLIX

Applix disk image

File extensions: raw

HPI

HP9845A floppy disk image

File extensions: hpi

10.6 Other tools included with MAME

10.6.1 ledutil.exe/ledutil.sh

On Microsoft Windows, ledutil.exe can take control of your keyboard LEDs to mirror those that were present on some early arcade games (e.g. Asteroids)

Start **ledutil.exe** from the command line to enable LED handling. Run **ledutil.exe -kill** to stop the handler.

On SDLMAME platforms such as Mac OS X and Linux, **ledutil.sh** can be used. Use **ledutil.sh -a** to have it automatically close when you exit SDLMAME.

10.7 Developer-focused tools included with MAME

10.7.1 pngcmp

This tool is used in regression testing to compare PNG screenshot results with the runtest.cmd script found in the source archive. This script works only on Microsoft Windows.

10.7.2 nltool

Discrete component conversion tool.

10.7.3 nlwav

Discrete component conversion and testing tool.

10.7.4 jedutil

PAL/PLA/PLD/GAL dump handling tool. It can convert between the industry-standard JED format and MAME's proprietary packed binary format and it can show logic equations for the types of devices it knows the internal logic of.

10.7.5 ldresample

This tool recompresses video data for laserdisc and VHS dumps.

10.7.6 ldverify

This tool is used for comparing laserdisc or VHS CHD images with the source AVI.

10.7.7 romcmp

This tool is used to perform basic data comparisons and integrity checks on binary dumps. With the -h switch, it can also be used to calculate hash functions.

10.7.8 unidasm

Universal disassembler for many of the architectures supported in MAME.

CHAPTER

ELEVEN

CONTRIBUTING TO MAME

So you want to contribute to MAME but aren't sure where to start? Well the great news is that there's always plenty to do for people with a variety of skill sets.

11.1 Testing and reporting bugs

One thing MAME can always do with is more testing and bug reports. If you're familiar with a system that MAME emulates and notice something wrong, or if you find a bug in MAME's user interface, you can head over to MAME Testers and, assuming it isn't already reported, register an account and open an issue. Be sure to read the FAQ and rules first to ensure you start out on the right foot. Please note that MAME Testers only accepts user-facing bugs in tagged release versions.

For other kinds of issues, we have GitHub Issues. There's a bit more leeway here. For example we accept developer-facing issues (e.g. bugs in internal APIs, or build system inadequacies), feature requests, and major regressions before they make it into a released version. Please respect the fact that the issue tracker is *not* a discussion or support forum, it's only for reporting reproducible issues. Don't open issues to ask questions or request support. Also, keep in mind that the `master` branch is unstable. If the current revision doesn't compile at all or is completely broken, we're probably already aware of it and don't need issues opened for that. Wait a while and see if there's an update. You might want to comment on the commit in question with the compiler error message, particularly if you're compiling in an unorthodox but supported configuration.

When opening an issue, remember to provide as much information as possible to help others understand, reproduce, and diagnose the issue. Things that are helpful to include:

- The incorrect behaviour, and expected or correct behaviour. Be specific: just saying it "doesn't work" usually isn't enough detail.
- Environment details, including your operating system, CPU architecture, system locale, and display language, if applicable. For video output bugs, note your video hardware (GPU), driver version, and the MAME video output module you're using. For input handling bugs, include the input peripherals and MAME input modules you're using.
- The exact version of MAME you're using, including a Git commit digest if it isn't a tagged release version, and any non-standard build options.
- The exact system and software being emulated (may not be applicable for issues with parts of MAME's UI, like the system selection menu). Include things like the selected BIOS version, and emulated peripheral (slot device) configuration.
- Steps to reproduce the issue. Assume the person reading is familiar with MAME itself, but not necessarily familiar with the emulated system and software in question. For emulation issues, input recordings and/or saved state files for reproducing the issue can be invaluable.

- An original reference for the correct behaviour. If you have access to the original hardware for the emulated system, it helps to make a recording of the correct behaviour for comparison.

11.2 Contributing to MAME's source code

MAME itself is written in C++, but that isn't the sum total of the source code. The source code also includes:

- The documentation hosted on this site (and also included in releases as a PDF), written in reStructuredText markup.
- The supplied *plugins*, written in Lua 5.4.
- Internal layouts for emulated machines that need to display more than a simple video screen. These are an XML application *described here*.
- The software lists, describing known software media for systems that MAME emulates. MAME software lists are an XML application.
- The user interface translations, in GNU gettext PO format. They can be edited with a good text editor, or a dedicated tool like Poedit.

Our primary source code repository is hosted on GitHub. We prefer to receive source code contributions in the form of pull requests. You'll need to learn the basics of Git distributed version control and familiarise yourself with the Git tools. The basic process for creating a pull request is as follows:

- Sign up for an account on GitHub.
- Create a fork of the mamedev/mame repository.
- Create a new branch off the `master` branch in your forked repository.
- Clone your forked repository, and check out your new branch.
- Make your changes, and build and test them locally.
- Commit your changes, and push your branch to GitHub.
- Optionally enable GitHub Actions for your forked repository, allowing your changes to be built on Windows, macOS and Linux.
- Open a pull request to merge your changes into the `master` branch in the mamedev/mame repository.

Please keep the following in mind (note that not all points are relevant to all kinds of changes):

- Make your commit messages descriptive. Please include what the change affects, and what it's supposed to achieve. A person reading the commit log shouldn't need to resort to examining the diff to get a basic idea of what a commit is supposed to do. The default commit messages provided by GitHub are completely useless, as they don't give any indication of what a change is supposed to do.
- Test your changes. Ensure that a full build of MAME completes, and that the code you changed works. It's a good idea to build with `DEBUG=1` to check that assertions compile and don't trigger.
- Use an enlightening pull request title and description. The title should give a one-line summary of what the overall change affects and what it's supposed to do. The description should contain more detail. Don't leave the description empty and describe the change in comments, as this makes searching and filtering more difficult.
- Be aware that GitHub Actions has opaque resource limits. It isn't clear when you're close to the limits, and we've had contributors banned from GitHub Actions for violating the limits. Even if you appeal the ban, they still won't tell you what the actual limits are, justifying this by saying that if you know the limits, you can take steps to evade them. If you enable GitHub Actions, consider not pushing individual commits if you don't need them to be automatically built, or cancelling workflow runs when you don't need the results.

- If your submission is a computer or other device such as a synthesizer or sampler which requires a disk, tape, cartridge, or other media to start up and run, please consider creating a software list containing at least one example of that media. This helps everyone making changes to shared MAME components to easily verify if the changes negatively impact your code.
- When submitting any new non-arcade machine, but especially a machine which does not auto-boot and requires some interaction to start up and be usable, consider adding usage instructions to the System-Specific Setup and Information page of the MAME Wiki. Anyone can edit the wiki after creating an account, and sub-pages for your system which discuss technical details of the system are also welcome.

We have guidelines for specific parts of the source:

11.2.1 C++ Coding Guidelines

- *Introduction*
- *Definitions*
- *Source file format*
- *Naming conventions*
- *Variables and literals*
- *Bracing and indentation*
- *Spacing*
- *Scoping*
- *Const Correctness*
- *Comments*
- *MAME-Specific Helpers*
- *Logging*
- *Structural organization*

Introduction

In terms of coding conventions, the style present within an existing source file should be favoured over the standards found below.

When a new source file is being created, the following coding conventions should be observed if creating a new file within the MAME core (`src/emu` and `src/lib`). If the source file is outside the core, deference can be given to a contributor's preferred style, although it is strongly encouraged to code with the understanding that the file may need to be comprehensible by more than one person as time marches forward.

Definitions

Snake case
: All lowercase letters with words separated by underscores: `this_is_snake_case`

Screaming snake case
: All uppercase letters with words separated by underscores: `SCREAMING_SNAKE_CASE`

Camel case:
: Lowercase initial letter, first letter of each subsequent word capitalised, with no separators between words: `exampleCamelCase`

Llama case:
: Uppercase initial letter, first letter of each subsequent word capitalised, with no separators between words: `LlamaCaseSample`

Source file format

MAME C++ source files are encoded as UTF-8 text, assuming fixed-width characters, with tab stops at four-space intervals. Source files should end with a terminating end-of-line. Any valid printable Unicode text is permitted in comments. Outside comments and strings, only the printable ASCII subset of Unicode is permitted.

The `srcclean` tool is used to enforce file format rules before each release. You can build this tool and apply it to the files you modify before opening a pull request to avoid conflicts or surprising changes later.

Naming conventions

Preprocessor macros
: Macro names should use screaming snake case. Macros are always global and name conflicts can cause confusing errors – think carefully about what macros really need to be in headers and name them carefully.

Include guards
: Include guard macros should begin with `MAME_`, and end with a capitalised version of the file name, with separators replaced by underscores.

Constants
: Constants should use screaming snake case, whether they are constant globals, constant data members, enumerators or preprocessor constants.

Functions
: Free functions names should use snake case. (There are some utility function that were previously implemented as preprocessor macros that still use screaming snake case.)

Classes
: Class names should use snake case. Abstract class names should end in `_base`. Public member functions (including static member functions) should use snake case.

Device classes
: Concrete driver `driver_device` implementation names conventionally end in `_state`, while other concrete device class names end in `_device`. Concrete `device_interface` names conventionally begin with `device_` and end with `_interface`.

Device types
: Device types should use screaming snake case. Remember that device types are names in the global namespace, so choose explicit, unambiguous names.

Enumerations
: The enumeration name should use snake case. The enumerators should use screaming snake case.

Template parameters
: Template parameters should use llama case (both type and value parameters).

Identifiers containing two consecutive underscores or starting with an underscore followed by an uppercase letter are always reserved and should not be used.

Type names and other identifiers with a leading underscore should be avoided within the global namespace, as they are explicitly reserved according to the C++ standard. Additionally, identifiers suffixed with `_t` should be avoided within the global namespace, as they are also reserved according to POSIX standards. While MAME violates this policy occasionally – most notably with `device_t` – it's considered to be an unfortunate legacy decision that should be avoided in any new code.

Variables and literals

Octal literals are discouraged from use outside of specific cases. They lack the obvious letter-based prefixes found in hexadecimal and binary literals, and therefore can be difficult to distinguish at a glance from a decimal literal to coders who are unfamiliar with octal notation.

Lower-case hexadecimal literals are preferred, e.g. `0xbadc0de` rather than `0xBADC0DE`. For clarity, try not to exceed the bit width of the variable which will be used to store it.

Binary literals have rarely been used in the MAME source code due to the `0b` prefix not being standardised until C++14, but there is no policy to avoid their use.

Integer suffix notation should be used when specifying 64-bit literals, but is not strictly required in other cases. It can, however, clarify the intended use of a given literal at a glance. Uppercase long integer literal suffixes should be used to avoid confusion with the digit 1, e.g. `7LL` rather than `7ll`.

Digit grouping should be used for longer numeric literals, as it aids in recognising order of magnitude or bit field positions at a glance. Decimal literals should use groups of three digits, and hexadecimal literals should use groups of four digits, outside of specific situations where different grouping would be easier to understand, e.g. `4'433'619` or `0xfff8'1fff`.

Types that do not have a specifically defined size should be avoided if they are to be registered with MAME's save-state system, as it harms portability. In general, this means avoiding the use of `int` for these members.

It's encouraged, but not required, for class data members to be prefixed with `m_` for non-static instance members and `s_` for static members. This does not apply to nested classes or structs.

Bracing and indentation

Tabs are used for initial indentation of lines, with one tab used per nested scope level. Statements split across multiple lines should be indented by two tabs. Spaces are used for alignment at other places within a line.

Either K&R or Allman-style bracing is preferred. There is no specific preference for bracing on single-line statements, although bracing should be consistent for a given `if/else` block, as shown:

```
if (x == 0)
{
    return;
}
else
{
    call_some_function();
    x--;
}
```

When using a series of `if`/`else` or `if`/`else if`/`else` blocks with comments at the top indentation level, avoid extraneous newlines. The use of additional newlines may lead to `else if` or `else` blocks being missed due to the newlines pushing the blocks outside the visible editor height:

```
// Early-out if our hypothetical counter has run out.
if (x == 0)
{
    return;
}
// We should do something if the counter is running.
else
{
    call_some_function();
    x--;
}
```

Indentation for `case` statements inside a `switch` body can either be on the same level as the `switch` statement or inward by one level. There is no specific style which is used across all core files, although indenting by one level appears to be used most often.

Spacing

Consistent single-spacing between binary operators, variables, and literals is strongly preferred. The following examples exhibit reasonably consistent spacing:

```
uint8_t foo = (((bar + baz) + 3) & 7) << 1;
uint8_t foo = ((bar << 1) + baz) & 0x0e;
uint8_t foo = bar ? baz : 5;
```

The following examples exhibit extremes in either direction, although having extra spaces is less difficult to read than having too few:

```
uint8_t foo = ( ( ( bar + baz ) + 3 ) & 7 ) << 1;
uint8_t foo = ((bar<<1)+baz)&0x0e;
uint8_t foo = (bar?baz:5);
```

A space should be used between a fundamental C++ statement and its opening parenthesis, e.g.:

```
switch (value) ...
if (a != b) ...
for (int i = 0; i < foo; i++) ...
```

Scoping

Variables should be scoped as narrowly as is reasonably possible. There are many instances of C89-style local variable declaration in the MAME codebase, but this is largely a hold-over from MAME's early days, which pre-date the C99 specification.

The following two snippets exhibit the legacy style of local variable declaration, followed by the more modern and preferred style:

```
void example_device::some_function()
{
```

(continues on next page)

(continued from previous page)

```
    int i;
    uint8_t data;

    for (i = 0; i < std::size(m_buffer); i++)
    {
        data = m_buffer[i];
        if (data)
        {
            some_other_function(data);
        }
    }
}
```

```
void example_device::some_function()
{
    for (int i = 0; i < std::size(m_buffer); i++)
    {
        const uint8_t data = m_buffer[i];
        if (data)
        {
            some_other_function(data);
        }
    }
}
```

Enumerated values, structs, and classes used only by one specific device should be declared within the device's class itself. This avoids pollution of the global namespace and makes the device-specific use of them more obvious at a glance.

Const Correctness

Const-correctness has not historically been a strict requirement of code that goes into MAME, but there's increasing value in it as the amount of code refactoring increases and technical debt decreases.

When writing new code, it's worth taking the time to determine if a local variable can be declared `const`. Similarly, it's encouraged to consider which member functions of a new class can be `const` qualified.

In a similar vein, arrays of constants should be declared `constexpr` and should use screaming snake case, as outlined towards the top of this document. Lastly, arrays of C-style strings should be declared as both a const array of const strings, as so:

```
static const char *const EXAMPLE_NAMES[4] =
{
    "1-bit",
    "2-bit",
    "4-bit",
    "Invalid"
};
```

Comments

While `/* ANSI C comments */` are often found in the codebase, there has been a gradual shift towards `// C++-style comments` for single-line comments. This is very much a guideline, and coders are encouraged to use whichever style is most comfortable.

Unless specifically quoting content from a machine or ancillary materials, comments should be in English so as to match the predominant language that the MAME team shares worldwide.

Commented-out code should typically be removed prior to authoring a pull request, as it has a tendency to rot due to the fast-moving nature of MAME's core API. If there is a desire known beforehand for the code to eventually be included, it should be bookended in `if (0)` or `if (false)`, as code removed through a preprocessor macro will rot at the same rate.

MAME-Specific Helpers

When at all possible, use helper functions and macros for bit manipulation operations.

The `BIT(value, bit)` helper can be used to extract the state of a bit at a given position from an integer value. The resulting value will be aligned to the least significant bit position, i.e. will be either 0 or 1.

An overload of the same function, `BIT(value, bit, width)` can be used to extract a bit field of a specified width from an integer value, starting at the specified bit position. The result will also be right-justified and will be of the same type as the incoming value.

For documentation on helpers related to multiplication and division, refer to `src/osd/eminline.h`.

Logging

MAME has multiple logging function for different purposes. Two of the most frequently used logging functions are `logerror` and `osd_printf_verbose`:

- Devices inherit a `logerror` member function. This automatically includes the fully-qualified tag of the invoking device in log messages. Output is sent to MAME's debugger's rotating log buffer if the debugger is enabled. If the *-log option* is enabled, it's also written to the file `error.log` in the working directory. If the *-oslog option* is enabled, it's additionally sent to the OS diagnostic output (the host debugger diagnostic log on Windows if a host debugger is attached, or standard error otherwise).
- The output of the `osd_printf_verbose` function is sent to standard error if the *-verbose option* is enabled.

The `osd_printf_verbose` function should be used for logging that is useful for diagnosing user issues, while `logerror` should be used for messages more relevant to developers (either developing MAME itself, or developing software for emulated systems using MAME's debugger).

For debug logging, a channel-based logging system exists via the header `logmacro.h`. It can be used as a generic logging system as follows, without needing to make use of its ability to mask out specific channels:

```
// All other headers in the .cpp file should be above this line.
#define VERBOSE (1)
#include "logmacro.h"
...
void some_device::some_reg_write(u8 data)
{
    LOG("%s: some_reg_write: %02x\n", machine().describe_context(), data);
}
```

The above example also makes use of a helper function which is available in all derivatives of `device_t`: `machine().describe_context()`. This function will return a string that describes the emulation context in which the function is being run. This includes the fully-qualified tag of the currently executing device (if any). If the relevant device implements `device_state_interface`, it will also include the current program-counter value reported by the device.

For more fine-grained control, specific bit masks can be defined and used via the `LOGMASKED` macro:

```
// All other headers in the .cpp file should be above this line.
#define LOG_FOO (1 << 1U)
#define LOG_BAR (1 << 2U)

#define VERBOSE (LOG_FOO | LOG_BAR)
#include "logmacro.h"
...
void some_device::some_reg_write(u8 data)
{
    LOGMASKED(LOG_FOO, "some_reg_write: %02x\n", data);
}

void some_device::another_reg_write(u8 data)
{
    LOGMASKED(LOG_BAR, "another_reg_write: %02x\n", data);
}
```

Note that the least significant bit position for user-supplied masks is 1, as bit position 0 is reserved for `LOG_GENERAL`.

By default, `LOG` and `LOGMASKED` will use the device-supplied `logerror` function. However, this can be redirected as desired. The most common use case would be to direct output to the standard output instead, which can be accomplished by explicitly defining `LOG_OUTPUT_FUNC` as so:

```
#define LOG_OUTPUT_FUNC osd_printf_info
```

A developer should always ensure that `VERBOSE` is set to 0 and that any definition of `LOG_OUTPUT_FUNC` is commented out prior to opening a pull request.

Structural organization

All C++ source files must begin with a two comments listing the distribution license and copyright holders in a standard format. Licenses are specified by their SPDX short identifier if available. Here is an example of the standard format:

```
// license:BSD-3-Clause
// copyright-holders:David Haywood, Tomasz Slanina
```

Header includes should generally be grouped from most-dependent to least-dependent, and sorted alphabetically within said groups:

- The project prefix header, `emu.h`, must be the first thing in a translation unit
- Local project headers (i.e. headers found in the same source directory)
- Headers in `src/devices`
- Headers in `src/emu`
- Headers in `src/lib/formats`
- Headers in `src/lib/util`

- Headers from the OSD layer
- C++ standard library headers
- C standard library headers
- OS-specific headers
- Layout headers

Finally, task-specific headers such as `logmacro.h` - described in the previous section - should be included last. A practical example follows:

```
#include "emu.h"

#include "cpu/m68000/m68000.h"
#include "machine/mc68328.h"
#include "machine/ram.h"
#include "sound/dac.h"
#include "video/mc68328lcd.h"
#include "video/sed1375.h"

#include "emupal.h"
#include "screen.h"
#include "speaker.h"

#include "pilot1k.lh"

#define VERBOSE (0)
#include "logmacro.h"
```

In most cases, the class declaration for a system driver should be within the corresponding source file along with the implementation. In such cases, the class declaration and all contents of the source file, excluding the `GAME`, `COMP`, or `CONS` macro, should be enclosed in an anonymous namespace (this produces better compiler diagnostics, allows more aggressive optimisation, reduces the chance of duplicate symbols, and reduces linking time).

Within a class declaration, there should be one section for each member access level (`public`, `protected` and `private`) if practical. This may not be possible in cases where private constants and/or types need to be declared before public members. Members should use the least public access level necessary. Overridden virtual member functions should generally use the same access level as the corresponding member function in the base class.

Class member declarations should be grouped to aid understanding:

- Within a member access level section, constants, types, data members, instance member functions and static member functions should be grouped.
- In device classes, configuration member functions should be grouped separately from live signal member functions.
- Overridden virtual member functions should be grouped according to the base classes they are inherited from.

For classes with multiple overloaded constructors, constructor delegation should be used where possible to avoid repeated member initialiser lists.

Constants which are used by a device or machine driver should be in the form of explicitly-sized enumerated values within the class declaration, or be relegated to `#define` macros within the source file. This helps avoid polluting the preprocessor.

11.2.2 Guidelines for Software Lists

- *Introduction*
- *Items and parts*
- *Metadata*

Introduction

MAME's software lists describe known software media for emulated systems in a form that can be used to identify media image files for known software, verify media image file integrity, and load media image files for emulation. Software lists are implemented as XML files in the `hash` folder. The XML structure is described in the file `hash/softwarelist.dtd`.

Philosophically, software list items should represent the original media, rather than a specific dump of the media. Ideally, it should be possible for anyone with the media to dump it and produce the same image file. Of course, this isn't always possible in practice – in particular it's problematic for inherently analog media, like home computer software stored on audio tape cassettes.

MAME strives to document the best available media images. It is not our intention to propagate corrupted, truncated, defaced, watermarked, or otherwise bad media images. Where possible, file structures matching the original media structure are preferred. For example we prefer individual files for separate ROM chips in cartridge media, and we use disk images rather than archives of files extracted from the original disks.

Items and parts

A software list is a collection of *items* and each item may have multiple *parts*. An item represents a piece of software, as distributed as a complete package. A part represents a single piece of media within the package. Parts can be mounted individually in emulated media devices. For example a piece of software distributed on three floppy disks will be a single item, while each floppy disk will be one part within that item.

Sometimes, logically separate parts of a single physical piece of media are represented as separate parts within a software item. For example each side of an audio tape cassette is represented as a separate part. However individual ROM chips within a cartridge may be separate files, but they are *not* separate parts, as the cartridge is mounted as a whole.

Each item is a `software` element. The `software` element may have the following attributes:

name (required)
: The short name identifying the item. This is used for file names, command line arguments, database keys, URL fragments, and many other purposes. It should be terse but still recognisable. It must be unique within the software list. Valid characters are lowercase English letters, decimal digits and underscores. The maximum allowed length is sixteen characters.

cloneof (optional)
: The short name of the parent item if the item is a clone. The parent must be within the same software list – parent/clone relationships spanning multiple software lists are not supported.

supported (optional)
: One of the values `yes` (fully usable in emulation), `no` (not usable in emulation), or `partial` (usable in emulation with limitations). If the attribute is not present, it is equivalent to `yes`. Examples of partially supported software include games that are playable with graphical glitches, and office software where some but not all functionality works.

Each part is a `part` element within the `software` element. The `part` element must have the following attributes:

name (required)
: The short name identifying the part. This is used for command line arguments, database keys, URL fragments, and many other purposes. It must be unique within the item. It is also used as the display name if a separate display name is not provided. Valid characters are lowercase English letters, decimal digits and underscores. The maximum allowed length is sixteen characters.

interface (required)
: This attribute is used to identify suitable emulated media devices for mounting the software part. Applicable values depend on the emulated system.

Metadata

Software lists support various kinds of metadata. All software list items require the following metadata elements to be present:

description
: This is the primary display name for the software item. It should be the original name of the software, transliterated into English Latin script if necessary. It must be unique within the software list. If extra text besides the title itself is required for disambiguation, use lowercase outside of proper nouns, initialisms and verbatim quotes.

year
: The year of release or copyright year for the software. If unknown, use an estimate with a question mark. Items can be filtered by year in the software selection menu.

publisher
: The publisher of the software. This may be the same as the developer if the software was self-published. Items can be filtered by published in the software selection menu.

Most user-visible software item metadata is provided using `info` elements. Each `info` element must have a `name` attribute and a `value` attribute. The `name` attribute identifies the type of metadata, and the `value` attribute is the metadata value itself. Note that `name` attributes do not need to be unique within an item. Multiple `info` elements with the same `name` may be present if appropriate. This is frequently seen for software sold using different titles in different regions.

Prefer multiple `info` elements with the same `name` attribute over combining multiple values into a single element. For example if a piece of software supports multiple user interface languages, use multiple `info` elements with `name="language"` attributes. This makes filtering and database queries more practical.

MAME displays metadata from `info` elements in the software selection menu. The following `name` attributes are recognised specifically, and can show localised names:

alt_title
: Used for alternate titles. Examples are different tiles used in different languages, scripts or regions, or different titles used on the title screen and packaging. MAME searches alternate titles as well as the description.

author
: Author of the software. Items can be filtered by author in the software selection menu.

barcode
: Barcode number identifying the software package (typically an EAN).

developer
: Developer responsible for implementing the software. Items can be filtered by developer in the software selection menu.

distributor
: Party responsible for distributing the software to retailers (or customers in the case of direct sales). Items can be filtered by distributor in the software selection menu.

install
Installation instructions.

isbn
ISBN for software included with a commercial book.

language
User interface language supported by the software.

oem
Original equipment manufacturer, typically used with customised versions of software distributed by a hardware vendor.

original_publisher
The original publisher, for items representing software re-released by a different publisher.

partno
Distributor's part number for the software.

pcb
Printed circuit board identifier, typically for cartridge media.

programmer
Programmer who wrote the code for the software.

release
Fine-grained release date for the software, if known. Use YYYYMMDD format with no punctuation. If only the month is known, use "xx" for the day digits. For example `199103xx` or `19940729`.

serial
Number identifying the software within a series of releases.

usage
Usage instructions.

version
Version number of the software.

11.3 Use of AI

Use of AI for assistance by team members is to be mentioned in the relevant commit message when committing directly.

All pull requests using AI assistance, for both team members and other contributors, must mention it as part of the initial pull request description. This must include the model and version used.

The use of AI for assistance will not be used to reject a pull request or a change out-of-hand, however it must be mentioned so that additional scrutiny can be given to the change or changes.

As the contributor, you are responsible for understanding and describing your changes.

CHAPTER

TWELVE

TECHNICAL SPECIFICATIONS

This section covers technical specifications useful to programmers working on MAME's source or working on scripts that run within the MAME framework.

12.1 MAME Naming Conventions

- *Introduction*
- *Transliteration*
- *Titles and descriptions*
- *C++ naming conventions*

12.1.1 Introduction

To promote consistency and readability in MAME source code, we have some naming conventions for various elements.

12.1.2 Transliteration

For better or worse, the most broadly recognised script in the world is English Latin. Conveniently, it's also included in almost all character encodings. To make MAME more globally accessible, we require Latin transliterations of titles and other metadata from other scripts. Do not use translations in metadata – translations are inherently subjective and error-prone. Translations may be included in comments if they may be helpful.

If general, if an official Latin script name is known, it should be used in favour of a naïve transliteration. For titles containing foreign loanwords, the conventional Latin spelling should be used for the loanwords (the most obvious example of this is the use of "Mahjong" in Japanese titles rather than "Maajan").

Chinese
: Where the primary audience was Mandarin-speaking, Hanyu Pinyin should be used. In contexts where diacritics are not permitted (e.g. when limited to ASCII), tone numbers should be omitted. When tones are being indicated using diacritics, tone sandhi rules should be applied. Where the primary audience was Cantonese-speaking (primarily Hong Kong and Guandong), Jyutping should be used with tone numbers omitted. If in doubt, use Hanyu Pinyin.

Greek
: Use ISO 843:1997 type 2 (TR) rules. Do not use traditional English spellings for Greek names (people or places).

Japanese
: Modified Hepburn rules should generally be used. Use an apostrophe between syllabic N and a following vowel (including iotised vowels). Do not use hyphens to transliterate prolonged vowels.

Korean
: Use Revised Romanisation of Korean (RR) rules with traditional English spelling for Korean surnames. Do not use ALA-LC rules for word division and use of hyphens.

Vietnamese
: When diacritics can't be used, omit the tones and replace the vowels with single English vowels – do not use VIQR or TELEX conventions ("an chuot nuong" rather than "a(n chuo^.t nu*o*'ng" or "awn chuootj nuowngs").

12.1.3 Titles and descriptions

Try to reproduce the original title faithfully where possible. Try to preserve the case convention used by the manufacturer/publisher. If no official English Latin title is known, use a standard transliteration. For software list entries where a transliteration is used for the `description` element, put the title in an `info` element with a `name="alt_title"` attribute.

For software items that have multiple titles (for example different regional titles with the same installation media), use the most widespread English Latin title for the `description` element, and put the other titles in `info` elements with `name="alt_title"` attributes.

If disambiguation is needed, try to be descriptive as possible. For example, use the manufacturer's version number, regional licensee's name, or terse description of hardware differences in preference to arbitrary set numbers. Surround the disambiguation text with parentheses, preserve original case for names and version text, but use lowercase for anything else besides proper nouns and initialisms.

12.1.4 C++ naming conventions

For C++ naming conventions, see the relevant section in the C++ Coding Guidelines: *Naming conventions*

12.2 MAME Layout Files

- *Introduction*
- *Core concepts*
  - *Numbers*
  - *Coordinates*
  - *Colours*
  - *Parameters*
  - *Pre-defined parameters*
- *Parts of a layout*
  - *Elements*
  - *Views*
  - *Collections*

  - *Reusable groups*
  - *Repeating blocks*
- *Interactivity*
  - *Clickable items*
  - *Element state*
  - *View item animation*
- *Error handling*
- *Automatically-generated views*
- *Using complay.py*
- *Example layout files*

12.2.1 Introduction

Layout files are used to tell MAME what to display when running an emulated system, and how to arrange it. MAME can render emulated screens, images, text, shapes, and specialised objects for common output devices. Elements can be static, or dynamically update to reflect the state of inputs and outputs. Layouts may be automatically generated based on the number/type of emulated screens, built and linked into the MAME binary, or provided externally. MAME layout files are an XML application, using the `.lay` filename extension.

12.2.2 Core concepts

Numbers

There are two kinds of numbers in MAME layouts: integers and floating-point numbers.

Integers may be supplied in decimal or hexadecimal notation. A decimal integer consists of an optional # (hash) prefix, an optional +/- (plus or minus) sign character, and a sequence of digits 0-9. A hexadecimal number consists of one of the prefixes $ (dollar sign) or 0x (zero ex) followed by a sequence of hexadecimal digits 0-9 and A-F. Hexadecimal numbers are case-insensitive for both the prefix and digits.

Floating-point numbers may be supplied in decimal fixed-point or scientific notation. Note that integer prefixes and hexadecimal values are *not* accepted where a floating-point number is expected.

For a few attributes, both integers and floating-point numbers are allowed. In these cases, the presence of a # (hash), $ (dollar sign) or 0x (zero ex) prefix causes the value to be interpreted as an integer. If no recognised integer prefix is found and the value contains a decimal point or the letter E (uppercase or lowercase) introducing an exponent, it is interpreted as a floating-point number. If no integer prefix, decimal point or letter E is found, the number will be interpreted as an integer.

Numbers are parsed using the "C" locale for portability.

Coordinates

Layout coordinates are internally represented as IEEE754 32-bit binary floating-point numbers (also known as "single precision"). Coordinates increase in the rightward and downward directions. The origin (0,0) has no particular significance, and you may freely use negative coordinates in layouts. Coordinates are supplied as floating-point numbers.

MAME assumes that view coordinates have the same aspect ratio as pixels on the output device (host screen or window). Assuming square pixels and no rotation, this means equal distances in X and Y axes correspond to equal horizontal and vertical distances in the rendered output.

Views, groups and elements all have their own internal coordinate systems. When an element or group is referenced from a view or another group, its coordinates are scaled as necessary to fit the specified bounds.

Objects are positioned and sized using `bounds` elements. The horizontal position and size may be specified in three ways: left edge and width using `x` and `width` attributes, horizontal centre and width using `xc` and `width` attributes, or left and right edges using `left` and `right` attributes. Similarly, the vertical position and size may be specified in terms of the top edge and height using `y` and `height` attributes, vertical centre and height using `yc` and `height` attributes, or top and bottom edges using `top` and `bottom` attributes.

These three `bounds` elements are equivalent:

```
<bounds x="455" y="120" width="12" height="8" />
<bounds xc="461" yc="124" width="12" height="8" />
<bounds left="455" top="120" right="467" bottom="128" />
```

It's possible to use different schemes in the horizontal and vertical directions. For example, these equivalent `bounds` elements are also valid:

```
<bounds x="455" top="120" width="12" bottom="128" />
<bounds left="455" yc="124" right="467" height="8" />
```

The `width`/`height` or `right`/`bottom` default to 1.0 if not supplied. It is an error if `width` or `height` are negative, if `right` is less than `left`, or if `bottom` is less than `top`.

Colours

Colours are specified in RGBA space. MAME is not aware of colour profiles and gamuts, so colours will typically be interpreted as sRGB with your system's target gamma (usually 2.2). Channel values are specified as floating-point numbers. Red, green and blue channel values range from 0.0 (off) to 1.0 (full intensity). Alpha ranges from 0.0 (fully transparent) to 1.0 (opaque). Colour channel values are not pre-multiplied by the alpha value.

Component and view item colour is specified using `color` elements. Meaningful attributes are `red`, `green`, `blue` and `alpha`. This example `color` element specifies all channel values:

```
<color red="0.85" green="0.4" blue="0.3" alpha="1.0" />
```

Any omitted channel attributes default to 1.0 (full intensity or opaque). It is an error if any channel value falls outside the range of 0.0 to 1.0 (inclusive).

Parameters

Parameters are named variables that can be used in most attributes. To use a parameter in an attribute, surround its name with tilde (~) characters. If a parameter is not defined, no substitution occurs. Here is an examples showing two instances of parameter use – the values of the `digitno` and `x` parameters will be substituted for `~digitno~` and `~x~`:

```
<element name="digit~digitno~" ref="digit">
    <bounds x="~x~" y="80" width="25" height="40" />
</element>
```

A parameter name is a sequence of uppercase English letters A-Z, lowercase English letters a-z, decimal digits 0-9, and/or underscore (_) characters. Parameter names are case-sensitive. When looking for a parameter, the layout engine starts at the current, innermost scope and works outwards. The outermost scope level corresponds to the top-level `mamelayout` element. Each `repeat`, `group` or `view` element creates a new, nested scope level.

Internally a parameter can hold a string, integer, or floating-point number, but this is mostly transparent. Integers are stored as 64-bit signed twos-complement values, and floating-point numbers are stored as IEEE754 64-bit binary floating-point numbers (also known as "double precision"). Integers are substituted in decimal notation, and floating point numbers are substituted in default format, which may be decimal fixed-point or scientific notation depending on the value). There is no way to override the default formatting of integer and floating-point number parameters.

There are two kinds of parameters: *value parameters* and *generator parameters*. Value parameters keep their assigned value until reassigned. Generator parameters have a starting value and an increment and/or shift to be applied for each iteration.

Value parameters are assigned using a `param` element with `name` and `value` attributes. Value parameters may appear inside the top-level `mamelayout` element, inside `repeat`, and `view` elements, and inside `group` definition elements (that is, `group` elements in the top-level `mamelayout` element, as opposed to `group` reference elements inside `view` elements other `group` definition elements). A value parameter may be reassigned at any point.

Here's an example assigning the value "4" to the value parameter "firstdigit":

```
<param name="firstdigit" value="4" />
```

Generator parameters are assigned using a `param` element with `name` and `start` attributes, and `increment`, `lshift` and/or `rshift` attributes. Generator parameters may only appear inside `repeat` elements (see *Repeating blocks* for details). A generator parameter must not be reassigned in the same scope (an identically named parameter may be defined in a child scope). Here are some example generator parameters:

```
<param name="nybble" start="3" increment="-1" />
<param name="switchpos" start="74" increment="156" />
<param name="mask" start="0x0800" rshift="4" />
```

- The `nybble` parameter generates values 3, 2, 1...
- The `switchpos` parameter generates values 74, 230, 386...
- The `mask` parameter generates values 2048, 128, 8...

The `increment` attribute must be an integer or floating-point number to be added to the parameter's value. The `lshift` and `rshift` attributes must be non-negative integers specifying numbers of bits to shift the parameter's value to the left or right. The increment and shift are applied at the end of the repeating block before the next iteration starts. The parameter's value will be interpreted as an integer or floating-point number before the increment and/or shift are applied. If both an increment and shift are supplied, the increment is applied before the shift.

If the `increment` attribute is present and is a floating-point number, the parameter's value will be converted to a floating-point number if necessary before the increment is added. If the `increment` attribute is present and is an integer while the parameter's value is a floating-point number, the increment will be converted to a floating-point number before the addition.

If the `lshift` and/or `rshift` attributes are present and not equal, the parameter's value will be converted to an integer if necessary, and shifted accordingly. Shifting to the left is defined as shifting towards the most significant bit. If both `lshift` and `rshift` are supplied, they are netted off before being applied. This means you cannot, for example, use equal `lshift` and `rshift` attributes to clear bits at one end of a parameter's value after the first iteration.

It is an error if a `param` element has neither `value` nor `start` attributes, and it is an error if a `param` element has both a `value` attribute and any of the `start`, `increment`, `lshift`, or `rshift` attributes.

A `param` element defines a parameter or reassigns its value in the current, innermost scope. It is not possible to define or reassign parameters in a containing scope.

Pre-defined parameters

A number of pre-defined value parameters are available providing information about the running machine:

devicetag
: The full tag path of the device that caused the layout to be loaded, for example `:` for the root driver device, or `:tty:ie15` for a terminal connected to a port. This parameter is a string defined at layout (global) scope.

devicebasetag
: The base tag of the device that caused the layout to be loaded, for example `root` for the root driver device, or `ie15` for a terminal connected to a port. This parameter is a string defined at layout (global) scope.

devicename
: The full name (description) of the device that caused the layout to be loaded, for example `AIM-65/40` or `IE15 Terminal`. This parameter is a string defined at layout (global) scope.

deviceshortname
: The short name of the device that caused the layout to be loaded, for example `aim65_40` or `ie15_terminal`. This parameter is a string defined at layout (global) scope.

scr0physicalxaspect
: The horizontal part of the physical aspect ratio of the first screen (if present). The physical aspect ratio is provided as a reduced improper fraction. Note that this is the horizontal component *before* rotation is applied. This parameter is an integer defined at layout (global) scope.

scr0physicalyaspect
: The vertical part of the physical aspect ratio of the first screen (if present). The physical aspect ratio is provided as a reduced improper fraction. Note that this is the vertical component *before* rotation is applied. This parameter is an integer defined at layout (global) scope.

scr0nativexaspect
: The horizontal part of the pixel aspect ratio of the first screen's visible area (if present). The pixel aspect ratio is provided as a reduced improper fraction. Note that this is the horizontal component *before* rotation is applied. This parameter is an integer defined at layout (global) scope.

scr0nativeyaspect
: The vertical part of the pixel aspect ratio of the first screen's visible area (if present). The pixel aspect ratio is provided as a reduced improper fraction. Note that this is the vertical component *before* rotation is applied. This parameter is an integer defined at layout (global) scope.

scr0width
: The width of the first screen's visible area (if present) in emulated pixels. Note that this is the width *before* rotation is applied. This parameter is an integer defined at layout (global) scope.

scr0height
: The height of the first screen's visible area (if present) in emulated pixels. Note that this is the height *before* rotation is applied. This parameter is an integer defined at layout (global) scope.

scr1physicalxaspect
The horizontal part of the physical aspect ratio of the second screen (if present). This parameter is an integer defined at layout (global) scope.

scr1physicalyaspect
The vertical part of the physical aspect ratio of the second screen (if present). This parameter is an integer defined at layout (global) scope.

scr1nativexaspect
The horizontal part of the pixel aspect ratio of the second screen's visible area (if present). This parameter is an integer defined at layout (global) scope.

scr1nativeyaspect
The vertical part of the pixel aspect ratio of the second screen's visible area (if present). This parameter is an integer defined at layout (global) scope.

scr1width
The width of the second screen's visible area (if present) in emulated pixels. This parameter is an integer defined at layout (global) scope.

scr1height
The height of the second screen's visible area (if present) in emulated pixels. This parameter is an integer defined at layout (global) scope.

scr*N*physicalxaspect
The horizontal part of the physical aspect ratio of the (zero-based) *N*th screen (if present). This parameter is an integer defined at layout (global) scope.

scr*N*physicalyaspect
The vertical part of the physical aspect ratio of the (zero-based) *N*th screen (if present). This parameter is an integer defined at layout (global) scope.

scr*N*nativexaspect
The horizontal part of the pixel aspect ratio of the (zero-based) *N*th screen's visible area (if present). This parameter is an integer defined at layout (global) scope.

scr*N*nativeyaspect
The vertical part of the pixel aspect ratio of the (zero-based) *N*th screen's visible area (if present). This parameter is an integer defined at layout (global) scope.

scr*N*width
The width of the (zero-based) *N*th screen's visible area (if present) in emulated pixels. This parameter is an integer defined at layout (global) scope.

scr*N*height
The height of the (zero-based) *N*th screen's visible area (if present) in emulated pixels. This parameter is an integer defined at layout (global) scope.

viewname
The name of the current view. This parameter is a string defined at view scope. It is not defined outside a view.

For screen-related parameters, screens are numbered from zero in the order they appear in machine configuration, and all screens are included (not just subdevices of the device that caused the layout to be loaded). X/width and Y/height refer to the horizontal and vertical dimensions of the screen *before* rotation is applied. Values based on the visible area are calculated at the end of configuration. Values are not updated and layouts are not recomputed if the system reconfigures the screen while running.

12.2.3 Parts of a layout

A *view* specifies an arrangement graphical object to display. A MAME layout file can contain multiple views. Views are built up from *elements* and *screens*. To simplify complex layouts, reusable groups and repeating blocks are supported.

The top-level element of a MAME layout file must be a `mamelayout` element with a `version` attribute. The `version` attribute must be an integer. Currently MAME only supports version 2, and will not load any other version. This is an example opening tag for a top-level `mamelayout` element:

```
<mamelayout version="2">
```

In general, children of the top-level `mamelayout` element are processed in reading order from top to bottom. Elements and groups must be defined before they can be used.

The following elements are allowed inside the top-level `mamelayout` element:

param
: Defines or reassigns a value parameter. See *Parameters* for details.

element
: Defines an element – one of the basic objects that can be arranged in a view. See *Elements* for details.

group
: Defines a reusable group of elements/screens that may be referenced from views or other groups. See *Reusable groups* for details.

repeat
: A repeating group of elements – may contain `param`, `element`, `group`, and `repeat` elements. See *Repeating blocks* for details.

view
: An arrangement of elements and/or screens that can be displayed on an output device (a host screen/window). See *Views* for details.

script
: Allows Lua script to be supplied for enhanced interactive layouts. See *MAME Layout Scripting* for details.

Elements

Elements are one of the basic visual objects that may be arranged, along with screens, to make up a view. Elements may be built up of one or more *components*, but an element is treated as a single surface when building the scene graph and rendering. An element may be used in multiple views, and may be used multiple times within a view.

An element's appearance depends on its *state*. The state is an integer which usually comes from an I/O port field or an emulated output (see *Element state* for information on connecting an element to an emulated I/O port or output). Any component of an element may be restricted to only drawing when the element's state is a particular value. Some components (e.g. multi-segment displays) use the state directly to determine their appearance.

Each element has its own internal coordinate system. The bounds of the element's coordinate system are computed as the union of the bounds of the individual components it's composed of.

Every element must have a `name` attribute specifying its name. Elements are referred to by name when instantiated in groups or views. It is an error for a layout file to contain multiple elements with identical `name` attributes. Elements may optionally supply a default state value with a `defstate` attribute, to be used if not connected to an emulated output or I/O port. If present, the `defstate` attribute must be a non-negative integer.

Child elements of the `element` element instantiate components, which are drawn into the element texture in reading order from first to last using alpha blending (components draw over and may obscure components that come before them). All components support a few common features:

- Components may be conditionally drawn depending on the element's state by supplying `state` and/or `statemask` attributes. If present, these attributes must be non-negative integers. If only the `state` attribute is present, the component will only be drawn when the element's state matches its value. If only the `statemask` attribute is present, the component will only be drawn when all the bits that are set in its value are set in the element's state.

  If both the `state` and `statemask` attributes are present, the component will only be drawn when the bits in the element's state corresponding to the bits that are set in the `statemask` attribute's value match the value of the corresponding bits in the `state` attribute's value.

  (The component will always be drawn if neither `state` nor `statemask` attributes are present, or if the `statemask` attribute's value is zero.)

- Each component may have a `bounds` child element specifying its position and size (see *Coordinates*). If no such element is present, the bounds default to a unit square (width and height of 1.0) with the top left corner at (0,0).

  A component's position and/or size may be animated according to the element's state by supplying multiple `bounds` child elements with `state` attributes. The `state` attribute of each `bounds` child element must be a non-negative integer. The `state` attributes must not be equal for any two `bounds` elements within a component.

  If the element's state is lower than the `state` value of any `bounds` child element, the position/size specified by the `bounds` child element with the lowest `state` value will be used. If the element's state is higher than the `state` value of any `bounds` child element, the position/size specified by the `bounds` child element with the highest `state` value will be used. If the element's state is between the `state` values of two `bounds` child elements, the position/size will be interpolated linearly.

- Each component may have a `color` child element specifying an RGBA colour (see *Colours* for details). This can be used to control the colour of geometric, algorithmically drawn, or textual components. For `image` components, the colour of the image pixels is multiplied by the specified colour. If no such element is present, the colour defaults to opaque white.

  A component's color may be animated according to the element's state by supplying multiple `color` child elements with `state` attributes. The `state` attributes must not be equal for any two `color` elements within a component.

  If the element's state is lower than the `state` value of any `color` child element, the colour specified by the `color` child element with the lowest `state` value will be used. If the element's state is higher than the `state` value of any `color` child element, the colour specified by the `color` child element with the highest `state` value will be used. If the element's state is between the `state` values of two `color` child elements, the RGBA colour components will be interpolated linearly.

The following components are supported:

rect
: Draws a uniform colour rectangle filling its bounds.

disk
: Draws a uniform colour ellipse fitted to its bounds.

image
: Draws an image loaded from a PNG, JPEG, Windows DIB (BMP) or SVG file. The name of the file to load (including the file name extension) is supplied using the `file` attribute. Additionally, an optional `alphafile` attribute may be used to specify the name of a PNG file (including the file name extension) to load into the alpha channel of the image.

  Alternatively, image data may be supplied in the layout file itself using a `data` child element. This can be useful for supplying simple, human-readable SVG graphics. A `file` attribute or `data` child element must be supplied; it is an error if neither or both are supplied.

  If the `alphafile` attribute refers to a file, it must have the same dimensions (in pixels) as the file referred to by the `file` attribute, and must have a bit depth no greater than eight bits per channel per pixel. The intensity from this

image (brightness) is copied to the alpha channel, with full intensity (white in a greyscale image) corresponding to fully opaque, and black corresponding to fully transparent. The `alphafile` attribute will be ignored if the `file` attribute refers to an SVG image or the `data` child element contains SVG data; it is only used in conjunction with bitmap images.

The image file(s) should be placed in the same directory/archive as the layout file. Image file formats are detected by examining the content of the files, file name extensions are ignored.

text
Draws text in the specified colour, using the artwork font. The text to draw must be supplied using a `string` attribute. An `align` attribute may be supplied to set text alignment. If present, the `align` attribute must be an integer, where 0 (zero) means centred, 1 (one) means left-aligned, 2 (two) means right-aligned, and 3 (three) means that the text will be stretched horizontally to fill its bounds. If the `align` attribute is absent, the text will be centred.

led7seg
Draws a standard seven-segment (plus decimal point) digital LED/fluorescent display in the specified colour. The low eight bits of the element's state control which segments are lit. Starting from the least significant bit, the bits correspond to the top segment, the upper right-hand segment, continuing clockwise to the upper left segment, the middle bar, and the decimal point. Unlit segments are drawn at low intensity (0x20/0xff).

led14seg
Draws a standard fourteen-segment alphanumeric LED/fluorescent display in the specified colour. The low fourteen bits of the element's state control which segments are lit. Starting from the least significant bit, the bits correspond to the top segment, the upper right-hand segment, continuing clockwise to the upper left segment, the left-hand and right-hand halves of the horizontal middle bar, the upper and lower halves of the vertical middle bar, and the diagonal bars clockwise from lower left to lower right. Unlit segments are drawn at low intensity (0x20/0xff).

led14segsc
Draws a standard fourteen-segment alphanumeric LED/fluorescent display with decimal point/comma in the specified colour. The low sixteen bits of the element's state control which segments are lit. The low fourteen bits correspond to the same segments as in the `led14seg` component. Two additional bits correspond to the decimal point and comma tail. Unlit segments are drawn at low intensity (0x20/0xff).

led16seg
Draws a standard sixteen-segment alphanumeric LED/fluorescent display in the specified colour. The low sixteen bits of the element's state control which segments are lit. Starting from the least significant bit, the bits correspond to the left-hand half of the top bar, the right-hand half of the top bar, continuing clockwise to the upper left segment, the left-hand and right-hand halves of the horizontal middle bar, the upper and lower halves of the vertical middle bar, and the diagonal bars clockwise from lower left to lower right. Unlit segments are drawn at low intensity (0x20/0xff).

led16segsc
Draws a standard sixteen-segment alphanumeric LED/fluorescent display with decimal point/comma in the specified colour. The low eighteen bits of the element's state control which segments are lit. The low sixteen bits correspond to the same segments as in the `led16seg` component. Two additional bits correspond to the decimal point and comma tail. Unlit segments are drawn at low intensity (0x20/0xff).

simplecounter
Displays the numeric value of the element's state in the specified colour, using the artwork font. The value is formatted in decimal notation. A `digits` attribute may be supplied to specify the minimum number of digits to display. If present, the `digits` attribute must be a positive integer; if absent, a minimum of two digits will be displayed. A `maxstate` attribute may be supplied to specify the maximum state value to display. If present, the `maxstate` attribute must be a non-negative number; if absent it defaults to 999. An `align` attribute may be supplied to set text alignment. If present, the `align` attribute must be an integer, where 0 (zero) means centred, 1 (one) means left-aligned, and 3 (three) means that the text will be stretched horizontally to fill its bounds. If the `align` attribute is absent, the text will be centred.

An example element that draws a static left-aligned text string:

```
<element name="label_reset_cpu">
    <text string="CPU" align="1"><color red="1.0" green="1.0" blue="1.0" /></text>
</element>
```

An example element that displays a circular LED where the intensity depends on the state of an active-high output:

```
<element name="led" defstate="0">
    <disk state="0"><color red="0.43" green="0.35" blue="0.39" /></disk>
    <disk state="1"><color red="1.0" green="0.18" blue="0.20" /></disk>
</element>
```

An example element for a button that gives visual feedback when clicked:

```
<element name="btn_rst">
    <rect state="0"><bounds x="0.0" y="0.0" width="1.0" height="1.0" /><color red="0.2"␣
↪green="0.2" blue="0.2" /></rect>
    <rect state="1"><bounds x="0.0" y="0.0" width="1.0" height="1.0" /><color red="0.1"␣
↪green="0.1" blue="0.1" /></rect>
    <rect state="0"><bounds x="0.1" y="0.1" width="0.9" height="0.9" /><color red="0.1"␣
↪green="0.1" blue="0.1" /></rect>
    <rect state="1"><bounds x="0.1" y="0.1" width="0.9" height="0.9" /><color red="0.2"␣
↪green="0.2" blue="0.2" /></rect>
    <rect><bounds x="0.1" y="0.1" width="0.8" height="0.8" /><color red="0.15" green="0.
↪15" blue="0.15" /></rect>
    <text string="RESET"><bounds x="0.1" y="0.4" width="0.8" height="0.2" /><color red=
↪"1.0" green="1.0" blue="1.0" /></text>
</element>
```

An example of an element that draws a seven-segment LED display using external segment images:

```
<element name="digit_a" defstate="0">
    <image file="a_off.png" />
    <image file="a_a.png" statemask="0x01" />
    <image file="a_b.png" statemask="0x02" />
    <image file="a_c.png" statemask="0x04" />
    <image file="a_d.png" statemask="0x08" />
    <image file="a_e.png" statemask="0x10" />
    <image file="a_f.png" statemask="0x20" />
    <image file="a_g.png" statemask="0x40" />
    <image file="a_dp.png" statemask="0x80" />
</element>
```

An example of a bar graph that grows vertically and changes colour from green, through yellow, to red as the state increases:

```
<element name="pedal">
    <rect>
        <bounds state="0x000" left="0.0" top="0.9" right="1.0" bottom="1.0" />
        <bounds state="0x610" left="0.0" top="0.0" right="1.0" bottom="1.0" />
        <color state="0x000" red="0.0" green="1.0" blue="0.0" />
        <color state="0x184" red="1.0" green="1.0" blue="0.0" />
        <color state="0x610" red="1.0" green="0.0" blue="0.0" />
```

(continues on next page)

(continued from previous page)

```
    </rect>
</element>
```

An example of a bar graph that grows horizontally to the left or right and changes colour from green, through yellow, to red as the state changes from the neutral position:

```
<element name="wheel">
    <rect>
        <bounds state="0x800" left="0.475" top="0.0" right="0.525" bottom="1.0" />
        <bounds state="0x280" left="0.0" top="0.0" right="0.525" bottom="1.0" />
        <bounds state="0xd80" left="0.475" top="0.0" right="1.0" bottom="1.0" />
        <color state="0x800" red="0.0" green="1.0" blue="0.0" />
        <color state="0x3e0" red="1.0" green="1.0" blue="0.0" />
        <color state="0x280" red="1.0" green="0.0" blue="0.0" />
        <color state="0xc20" red="1.0" green="1.0" blue="0.0" />
        <color state="0xd80" red="1.0" green="0.0" blue="0.0" />
    </rect>
</element>
```

Views

A view defines an arrangement of elements and/or emulated screen images that can be displayed in a window or on a screen. Views also connect elements to emulated I/O ports and/or outputs. A layout file may contain multiple views. If a view references a non-existent screen, it will be considered *unviable*. MAME will print a warning message, skip over the unviable view, and continue to load views from the layout file. This is particularly useful for systems where a screen is optional, for example computer systems with front panel controls and an optional serial terminal.

Views are identified by name in MAME's user interface and in command-line options. For layouts files associated with devices other than the root driver device, view names are prefixed with the device's tag (with the initial colon omitted) – for example a view called "Keyboard LEDs" loaded for the device `:tty:ie15` will be called "tty:ie15 Keyboard LEDs" in MAME's user interface. Views are listed in the order they are loaded. Within a layout file, views are loaded in the order they appear, from top to bottom.

Views are created with `view` elements inside the top-level `mamelayout` element. Each `view` element must have a `name` attribute, supplying its human-readable name for use in the user interface and command-line options. This is an example of a valid opening tag for a `view` element:

```
<view name="Control panel">
```

A view creates a nested parameter scope inside the parameter scope of the top-level `mamelayout` element.

A `view` element may have a `showpointers` attribute to set whether mouse and pen pointers should be shown for the view. If present, the value must be either `yes` or `no`. If the `showpointers` attribute is not present, pen and mouse pointers are shown for views that contain items bound to I/O ports.

The following child elements are allowed inside a `view` element:

bounds
: Sets the origin and size of the view's internal coordinate system if present. See *Coordinates* for details. If absent, the bounds of the view are computed as the union of the bounds of all screens and elements within the view. It only makes sense to have one `bounds` as a direct child of a view element. Any content outside the view's bounds is cropped, and the view is scaled proportionally to fit the output window or screen.

param
: Defines or reassigns a value parameter in the view's scope. See *Parameters* for details.

element
: Adds an element to the view (see *Elements*). The name of the element to add is specified using the required `ref` attribute. It is an error if no element with this name is defined in the layout file. Within a view, elements are drawn in the order they appear in the layout file, from front to back. See below for more details.

  May optionally be connected to an emulated I/O port using `inputtag` and `inputmask` attributes, and/or an emulated output using a `name` attribute. See *Clickable items* for details. See *Element state* for details on supplying a state value to the instantiated element.

screen
: Adds an emulated screen image to the view. The screen must be identified using either an `index` attribute or a `tag` attribute (it is an error for a `screen` element to have both `index` and `tag` attributes). If present, the `index` attribute must be a non-negative integer. Screens are numbered by the order they appear in machine configuration, starting at zero (0). If present, the `tag` attribute must be the tag path to the screen relative to the device that causes the layout to be loaded. Screens are drawn in the order they appear in the layout file, from front to back.

  May optionally be connected to an emulated I/O port using `inputtag` and `inputmask` attributes, and/or an emulated output using a `name` attribute. See *Clickable items* for details.

collection
: Adds screens and/or items in a collection that can be shown or hidden by the user (see *Collections*). The name of the collection is specified using the required `name` attribute. There is a limit of 32 collections per view.

group
: Adds the content of the group to the view (see *Reusable groups*). The name of the group to add is specified using the required `ref` attribute. It is an error if no group with this name is defined in the layout file. See below for more details on positioning.

repeat
: Repeats its contents the number of times specified by the required `count` attribute. The `count` attribute must be a positive integer. A `repeat` element in a view may contain `element`, `screen`, `group`, and further `repeat` elements, which function the same way they do when placed in a view directly. See *Repeating blocks* for discussion on using `repeat` elements.

Screens (`screen` elements) and layout elements (`element` elements) may have an `id` attribute. If present, the `id` attribute must not be empty, and must be unique within a view, including screens and elements instantiated via reusable groups and repeating blocks. Screens and layout elements with `id` attributes can be looked up by Lua scripts (see *MAME Layout Scripting*).

Screens (`screen` elements), layout elements (`element` elements) and groups (`group` elements) may have their orientation altered using an `orientation` child element. For screens, the orientation modifiers are applied in addition to the orientation modifiers specified on the screen device and on the machine. The `orientation` element supports the following attributes, all of which are optional:

rotate
: If present, applies clockwise rotation in ninety degree increments. Must be an integer equal to 0, 90, 180 or 270.

swapxy
: Allows the screen, element or group to be mirrored along a line at forty-five degrees to vertical from upper left to lower right. Must be either `yes` or `no` if present. Mirroring applies logically after rotation.

flipx
: Allows the screen, element or group to be mirrored around its vertical axis, from left to right. Must be either `yes` or `no` if present. Mirroring applies logically after rotation.

flipy
: Allows the screen, element or group to be mirrored around its horizontal axis, from top to bottom. Must be either `yes` or `no` if present. Mirroring applies logically after rotation.

Screens (`screen` elements) and layout elements (`element` elements) may have a `blend` attribute to set the blending mode. Supported values are `none` (no blending), `alpha` (alpha blending), `multiply` (RGB multiplication), and `add` (additive blending). The default for screens is to allow the driver to specify blending per layer; the default blending mode for layout elements is alpha blending.

Screens (`screen` elements), layout elements (`element` elements) and groups (`group` elements) may be positioned and sized using a `bounds` child element (see *Coordinates* for details). In the absence of a `bounds` child element, screens' and layout elements' bounds default to a unit square (origin at 0,0 and height and width both equal to 1). In the absence of a `bounds` child element, groups are expanded with no translation/scaling (note that groups may position screens/elements outside their bounds). This example shows a view instantiating and positioning a screen, an individual layout element, and two element groups:

```
<view name="LED Displays, Terminal and Keypad">
    <screen index="0"><bounds x="0" y="132" width="320" height="240" /></screen>
    <element ref="beige"><bounds x="320" y="0" width="172" height="372" /></element>
    <group ref="displays"><bounds x="0" y="0" width="320" height="132" /></group>
    <group ref="keypad"><bounds x="336" y="16" width="140" height="260" /></group>
</view>
```

Screens (`screen` elements), layout elements (`element` elements) and groups (`group` elements) may have a `color` child element (see *Colours*) specifying a modifier colour. The component colours of the screen or layout element(s) are multiplied by this colour.

Screens (`screen` elements) and layout elements (`element` elements) may have their colour and position/size animated by supplying multiple `color` and/or `bounds` child elements with `state` attributes. See *View item animation* for details.

Layout elements (`element` elements) may be configured to show only part of the element's width or height using `xscroll` and/or `yscroll` child elements. This can be used for devices like slot machine reels. The `xscroll` and `yscroll` elements support the same attributes:

size
: The size of the horizontal or vertical scroll window, as a proportion of the element's width or height, respectively. Must be in the range 0.01 to 1.0, inclusive, if present (1% of the width/height to the full width/height). By default, the entire width and height of the element is shown.

wrap
: Whether the element should wrap horizontally or vertically. Must be either `yes` or `no` if present. By default, items do not wrap horizontally or vertically.

inputtag
: If present, the horizontal or vertical scroll position will be taken from the value of the corresponding I/O port. Specifies the tag path of an I/O port relative to the device that caused the layout file to be loaded. The raw value from the input port is used, active-low switch values are not normalised.

name
: If present, the horizontal or vertical scroll position will be taken from the correspondingly named output.

mask
: If present, the horizontal or vertical scroll position will be masked with the value and shifted to the right to remove trailing zeroes (for example a mask of 0x05 will result in no shift, while a mask of 0x68 will result in the value being shifted three bits to the right). Note that this applies to output values (specified with the `name` attribute) as well as input port values (specified with the `inputtag` attribute). Must be an integer value if present. If not present, it is equivalent to all 32 bits being set.

min
: Minimum horizontal or vertical scroll position value. When the horizontal or vertical scroll position has this value, the left or top edge or the scroll window will be aligned with the left or top edge of the element. Must be an integer value if present. Defaults to zero.

max
: Maximum horizontal or vertical scroll position value. Must be an integer value if present. Defaults to the `mask` value shifted to the right to remove trailing zeroes.

Collections

Collections of screens and/or layout elements can be shown or hidden by the user as desired. For example, a single view could include both displays and a clickable keypad, and allow the user to hide the keypad leaving only the displays visible. Collections are created using `collection` elements inside `view`, `group` and other `collection` elements.

A collection element must have a `name` attribute providing the display name for the collection. Collection names must be unique within a view. The initial visibility of a collection may be specified by providing a `visible` attribute. Set the `visible` attribute to `yes` if the collection should be initially visible, or `no` if it should be initially hidden. Collections are initially visible by default.

Here is an example demonstrating the use of collections to allow parts of a view to be hidden by the user:

```
<view name="LED Displays, CRT and Keypad">
    <collection name="LED Displays">
        <group ref="displays"><bounds x="240" y="0" width="320" height="47" /></group>
    </collection>
    <collection name="Keypad">
        <group ref="keypad"><bounds x="650" y="57" width="148" height="140" /></group>
    </collection>
    <screen tag="screen"><bounds x="0" y="57" width="640" height="480" /></screen>
</view>
```

A collection creates a nested parameter scope. Any `param` elements inside the collection element set parameters in the local scope for the collection. See *Parameters* for more detail on parameters. (Note that the collection's name and default visibility are not part of its content, and any parameter references in the `name` and `visible` attributes themselves will be substituted using parameter values from the collection's parent's scope.)

Reusable groups

Groups allow an arrangement of screens and/or layout elements to be used multiple times in views or other groups. Groups can be beneficial even if you only use the arrangement once, as they can be used to encapsulate part of a complex layout. Groups are defined using `group` elements inside the top-level `mamelayout` element, and instantiated using `group` elements inside `view` and other `group` elements.

Each group definition element must have a `name` attribute providing a unique identifier. It is an error if a layout file contains multiple group definitions with identical `name` attributes. The value of the `name` attribute is used when instantiating the group from a view or another group. This is an example opening tag for a group definition element inside the top-level `mamelayout` element:

```
<group name="panel">
```

This group may then be instantiated in a view or another group element using a group reference element, optionally supplying destination bounds, orientation, and/or modifier colour. The `ref` attribute identifies the group to instantiate – in this example, destination bounds are supplied:

```
<group ref="panel"><bounds x="87" y="58" width="23" height="23.5" /></group>
```

Group definition elements allow all the same child elements as views. Positioning and orienting screens, layout elements and nested groups works the same way as for views. See *Views* for details. A group may instantiate other groups, but recursive loops are not permitted. It is an error if a group directly or indirectly instantiates itself.

Groups have their own internal coordinate systems. If a group definition element has no `bounds` element as a direct child, its bounds are computed as the union of the bounds of all the screens, layout elements and/or nested groups it instantiates. A `bounds` child element may be used to explicitly specify group bounds (see *Coordinates* for details). Note that groups' bounds are only used for the purpose of calculating the coordinate transform when instantiating a group. A group may position screens and/or elements outside its bounds, and they will not be cropped.

To demonstrate how bounds calculation works, consider this example:

```
<group name="autobounds">
    <!-- bounds automatically calculated with origin at (5,10), width 30, and height 15 -
↪->
    <element ref="topleft"><bounds x="5" y="10" width="10" height="10" /></element>
    <element ref="bottomright"><bounds x="25" y="15" width="10" height="10" /></element>
</group>

<view name="Test">
    <!--
        group bounds translated and scaled to fit - 2/3 scale horizontally and double␣
↪vertically
        element topleft positioned at (0,0) with width 6.67 and height 20
        element bottomright positioned at (13.33,10) with width 6.67 and height 20
        view bounds calculated with origin at (0,0), width 20, and height 30
    -->
    <group ref="autobounds"><bounds x="0" y="0" width="20" height="30" /></group>
</view>
```

This is relatively straightforward, as all elements inherently fall within the group's automatically computed bounds. Now consider what happens if a group positions elements outside its explicit bounds:

```
<group name="periphery">
    <!-- elements are above the top edge and to the right of the right edge of the␣
↪bounds -->
    <bounds x="10" y="10" width="20" height="25" />
    <element ref="topleft"><bounds x="10" y="0" width="10" height="10" /></element>
    <element ref="bottomright"><bounds x="30" y="20" width="10" height="10" /></element>
</group>

<view name="Test">
    <!--
        group bounds translated and scaled to fit - 3/2 scale horizontally and unity␣
↪vertically
        element topleft positioned at (5,-5) with width 15 and height 10
        element bottomright positioned at (35,15) with width 15 and height 10
        view bounds calculated with origin at (5,-5), width 45, and height 30
    -->
    <group ref="periphery"><bounds x="5" y="5" width="30" height="25" /></group>
</view>
```

The group's elements are translated and scaled as necessary to distort the group's internal bounds to the destination bounds in the view. The group's content is not restricted to its bounds. The view considers the bounds of the actual layout elements when computing its bounds, not the destination bounds specified for the group.

When a group is instantiated, it creates a nested parameter scope. The logical parent scope is the parameter scope of the view, group or repeating block where the group is instantiated (*not* its lexical parent, the top-level `mamelayout` element). Any `param` elements inside the group definition element set parameters in the local scope for the group

instantiation. Local parameters do not persist across multiple instantiations. See *Parameters* for more detail on parameters. (Note that the group's name is not part of its content, and any parameter references in the `name` attribute itself will be substituted at the point where the group definition appears in the top-level `mamelayout` element's scope.)

Repeating blocks

Repeating blocks provide a concise way to generate or arrange large numbers of similar elements. Repeating blocks are generally used in conjunction with generator parameters (see *Parameters*). Repeating blocks may be nested for more complex arrangements.

Repeating blocks are created with `repeat` elements. Each `repeat` element requires a `count` attribute specifying the number of iterations to generate. The `count` attribute must be a positive integer. Repeating blocks are allowed inside the top-level `mamelayout` element, inside `group` and `view` elements, and insider other `repeat` elements. The exact child elements allowed inside a `repeat` element depend on where it appears:

- A repeating block inside the top-level `mamelayout` element may contain `param`, `element`, `group` (definition), and `repeat` elements.
- A repeating block inside a `group` or `view` element may contain `param`, `element` (reference), `screen`, `group` (reference), and `repeat` elements.

A repeating block effectively repeats its contents the number of times specified by its `count` attribute. See the relevant sections for details on how the child elements are used (*Parts of a layout*, *Reusable groups*, and *Views*). A repeating block creates a nested parameter scope inside the parameter scope of its lexical (DOM) parent element.

Generating white number labels from zero to eleven named `label_0`, `label_1`, and so on (inside the top-level `mamelayout` element):

```
<repeat count="12">
    <param name="labelnum" start="0" increment="1" />
    <element name="label_~labelnum~">
        <text string="~labelnum~"><color red="1.0" green="1.0" blue="1.0" /></text>
    </element>
</repeat>
```

A horizontal row of forty digital displays, with five units space between them, controlled by outputs `digit0` to `digit39` (inside a `group` or `view` element):

```
<repeat count="40">
    <param name="i" start="0" increment="1" />
    <param name="x" start="5" increment="30" />
    <element name="digit~i~" ref="digit">
        <bounds x="~x~" y="5" width="25" height="50" />
    </element>
</repeat>
```

Eight five-by-seven dot matrix displays in a row, with pixels controlled by outputs `Dot_000` to `Dot_764` (inside a `group` or `view` element):

```
<repeat count="8"> <!-- 8 digits -->
    <param name="digitno" start="1" increment="1" />
    <param name="digitx" start="0" increment="935" /> <!-- distance between digits ((111␣
↪* 5) + 380) -->
    <repeat count="7"> <!-- 7 rows in each digit -->
        <param name="rowno" start="1" increment="1" />
        <param name="rowy" start="0" increment="114" /> <!-- vertical distance between␣
```

(continues on next page)

(continued from previous page)

```
↪LEDs -->
        <repeat count="5"> <!-- 5 columns in each digit -->
            <param name="colno" start="1" increment="1" />
            <param name="colx" start="~digitx~" increment="111" /> <!-- horizontal␣
↪distance between LEDs -->
            <element name="Dot_~digitno~~rowno~~colno~" ref="Pixel" state="0">
                <bounds x="~colx~" y="~rowy~" width="100" height="100" /> <!-- size of␣
↪each LED -->
            </element>
        </repeat>
    </repeat>
</repeat>
```

Two horizontally separated, clickable, four-by-four keypads (inside a `group` or `view` element):

```
<repeat count="2">
    <param name="group" start="0" increment="4" />
    <param name="padx" start="10" increment="530" />
    <param name="mask" start="0x01" lshift="4" />
    <repeat count="4">
        <param name="row" start="0" increment="1" />
        <param name="y" start="100" increment="110" />
        <repeat count="4">
            <param name="col" start="~group~" increment="1" />
            <param name="btnx" start="~padx~" increment="110" />
            <param name="mask" start="~mask~" lshift="1" />
            <element ref="btn~row~~col~" inputtag="row~row~" inputmask="~mask~">
                <bounds x="~btnx~" y="~y~" width="80" height="80" />
            </element>
        </repeat>
    </repeat>
</repeat>
```

The buttons are drawn using elements `btn00` in the top left, `bnt07` in the top right, `btn30` in the bottom left, and `btn37` in the bottom right, counting in between. The four rows are connected to I/O ports `row0`, `row1`, `row2`, and `row3`, from top to bottom. The columns are connected to consecutive I/O port bits, starting with the least significant bit on the left. Note that the `mask` parameter in the innermost `repeat` element takes its initial value from the correspondingly named parameter in the enclosing scope, but does not modify it.

Generating a chequerboard pattern with alternating alpha values 0.4 and 0.2 (inside a `group` or `view` element):

```
<repeat count="4">
    <param name="pairy" start="3" increment="20" />
    <param name="pairno" start="7" increment="-2" />
    <repeat count="2">
        <param name="rowy" start="~pairy~" increment="10" />
        <param name="rowno" start="~pairno~" increment="-1" />
        <param name="lalpha" start="0.4" increment="-0.2" />
        <param name="ralpha" start="0.2" increment="0.2" />
        <repeat count="4">
            <param name="lx" start="3" increment="20" />
            <param name="rx" start="13" increment="20" />
            <param name="lmask" start="0x01" lshift="2" />
```

(continues on next page)

(continued from previous page)

```
            <param name="rmask" start="0x02" lshift="2" />
            <element ref="hl" inputtag="board:IN.~rowno~" inputmask="~lmask~">
                <bounds x="~lx~" y="~rowy~" width="10" height="10" />
                <color alpha="~lalpha~" />
            </element>
            <element ref="hl" inputtag="board:IN.~rowno~" inputmask="~rmask~">
                <bounds x="~rx~" y="~rowy~" width="10" height="10" />
                <color alpha="~ralpha~" />
            </element>
        </repeat>
    </repeat>
</repeat>
```

The outermost `repeat` element generates a group of two rows on each iteration; the next `repeat` element generates an individual row on each iteration; the innermost `repeat` element produces two horizontally adjacent tiles on each iteration. Rows are connected to I/O ports `board:IN.7` at the top to `board.IN.0` at the bottom.

12.2.4 Interactivity

Interactive views are supported by allowing items to be bound to emulated outputs and I/O ports. Five kinds of interactivity are supported:

Clickable items
: If an item in a view is bound to an I/O port switch field, clicking the item will activate the emulated switch.

State-dependent components
: Some components will be drawn differently depending on the containing element's state. These include the dot matrix, multi-segment LED display and simple counter elements. See *Elements* for details.

Conditionally-drawn components
: Components may be conditionally drawn or hidden depending on the containing element's state by supplying `state` and/or `statemask` attributes. See *Elements* for details.

Component parameter animation
: Components' colour and position/size within their containing element may be animated according the element's state by providing multiple `color` and/or `bounds` elements with `state` attributes. See *Elements* for details.

Item parameter animation
: Items' colour and position/size within their containing view may be animated according to their animation state.

Clickable items

If a view item (`element` or `screen` element) has `inputtag` and `inputmask` attribute values that correspond to a digital switch field in the emulated system, clicking the element will activate the switch. The switch will remain active as long as the primary button is held down and the pointer is within the item's current bounds. (Note that the bounds may change depending on the item's animation state, see *View item animation*).

The `inputtag` attribute specifies the tag path of an I/O port relative to the device that caused the layout file to be loaded. The `inputmask` attribute must be an integer specifying the bits of the I/O port field that the item should activate. This sample shows instantiation of clickable buttons:

The `clickthrough` attribute controls whether clicks can pass through the view item to other view items drawn below it. The `clickthrough` attribute must be `yes` or `no` if present. The default is `no` (clicks do not pass through) for view items with `inputtag` and `inputmask` attributes, and `yes` (clicks pass through) for other view items.

```
<element ref="btn_3" inputtag="X2" inputmask="0x10">
    <bounds x="2.30" y="4.325" width="1.0" height="1.0" />
</element>
<element ref="btn_0" inputtag="X0" inputmask="0x20">
    <bounds x="0.725" y="5.375" width="1.0" height="1.0" />
</element>
<element ref="btn_rst" inputtag="RESET" inputmask="0x01">
    <bounds x="1.775" y="5.375" width="1.0" height="1.0" />
</element>
```

When handling pointer input, MAME treats all layout elements as being rectangular.

Element state

A view item that instantiates an element (`element` element) may supply a state value to the element from an emulated I/O port or output. See *Elements* for details on how an element's state affects its appearance.

If the `element` element has a `name` attribute, the element state value will be taken from the value of the correspondingly named emulated output. Note that output names are global, which can become an issue when a machine uses multiple instances of the same type of device. This example shows how digital displays may be connected to emulated outputs:

```
<element name="digit6" ref="digit"><bounds x="16" y="16" width="48" height="80" /></
↪element>
<element name="digit5" ref="digit"><bounds x="64" y="16" width="48" height="80" /></
↪element>
<element name="digit4" ref="digit"><bounds x="112" y="16" width="48" height="80" /></
↪element>
<element name="digit3" ref="digit"><bounds x="160" y="16" width="48" height="80" /></
↪element>
<element name="digit2" ref="digit"><bounds x="208" y="16" width="48" height="80" /></
↪element>
<element name="digit1" ref="digit"><bounds x="256" y="16" width="48" height="80" /></
↪element>
```

If the `element` element has `inputtag` and `inputmask` attributes but lacks a `name` attribute, the element state value will be taken from the value of the corresponding I/O port, masked with the `inputmask` value. The `inputtag` attribute specifies the tag path of an I/O port relative to the device that caused the layout file to be loaded. The `inputmask` attribute must be an integer specifying the bits of the I/O port field to use.

If the `element` element has no `inputraw` attribute, or if the value of the `inputraw` attribute is `no`, the I/O port's value is masked with the `inputmask` value and XORed with the I/O port default field value. If the result is non-zero, the element state is 1, otherwise it's 0. This is often used or provide visual feedback for clickable buttons, as values for active-high and active-low switches are normalised.

If the `element` element has an `inputraw` attribute with the value `yes`, the element state will be taken from the I/O port's value masked with the `inputmask` value and shifted to the right to remove trailing zeroes (for example a mask of 0x05 will result in no shift, while a mask of 0xb0 will result in the value being shifted four bits to the right). This is useful for obtaining the value of analog or positional inputs.

View item animation

Items' colour and position/size within their containing view may be animated. This is achieved by supplying multiple `color` and/or `bounds` child elements with `state` attributes. The `state` attribute of each `color` or `bounds` child element must be a non-negative integer. Within a view item, no two `color` elements may have equal state `state` attributes, and no two `bounds` elements may have equal `state` attributes.

If the item's animation state is lower than the `state` value of any `bounds` child element, the position/size specified by the `bounds` child element with the lowest `state` value will be used. If the item's animation state is higher than the `state` value of any `bounds` child element, the position/size specified by the `bounds` child element with the highest `state` value will be used. If the item's animation state is between the `state` values of two `bounds` child elements, the position/size will be interpolated linearly.

If the item's animation state is lower than the `state` value of any `color` child element, the colour specified by the `color` child element with the lowest `state` value will be used. If the item's animation state is higher than the `state` value of any `color` child element, the colour specified by the `color` child element with the highest `state` value will be used. If the item's animation state is between the `state` values of two `color` child elements, the RGBA colour components will be interpolated linearly.

An item's animation state may be bound to an emulated output or input port by supplying an `animate` child element. If present, the `animate` element must have either an `inputtag` attribute or a `name` attribute (but not both). If the `animate` child element is not present, the item's animation state is the same as its element state (see *Element state*).

If the `animate` child element is present and has an `inputtag` attribute, the item's animation state will be taken from the value of the corresponding I/O port. The `inputtag` attribute specifies the tag path of an I/O port relative to the device that caused the layout file to be loaded. The raw value from the input port is used, active-low switch values are not normalised.

If the `animate` child element is present and has a `name` attribute, the item's animation state will be taken from the value of the correspondingly named emulated output. Note that output names are global, which can become an issue when a machine uses multiple instances of the same type of device.

If the `animate` child element has a `mask` attribute, the item's animation state will be masked with the `mask` value and shifted to the right to remove trailing zeroes (for example a mask of 0x05 will result in no shift, while a mask of 0xb0 will result in the value being shifted four bits to the right). Note that the `mask` attribute applies to output values (specified with the `name` attribute) as well as input port values (specified with the `inputtag` attribute). If the `mask` attribute is present, it must be an integer value. If the `mask` attribute is not present, it is equivalent to all 32 bits being set.

This example shows elements with independent element state and animation state, using the animation state taken from emulated outputs to control their position:

```
<repeat count="5">
    <param name="x" start="10" increment="9" />
    <param name="i" start="0" increment="1" />
    <param name="mask" start="0x01" lshift="1" />

    <element name="cg_sol~i~" ref="cosmo">
        <animate name="cg_count~i~" />
        <bounds state="0" x="~x~" y="10" width="6" height="7" />
        <bounds state="255" x="~x~" y="48.5" width="6" height="7" />
    </element>

    <element ref="nothing" inputtag="FAKE1" inputmask="~mask~">
        <animate name="cg_count~i~" />
        <bounds state="0" x="~x~" y="10" width="6" height="7" />
        <bounds state="255" x="~x~" y="48.5" width="6" height="7" />
```

(continues on next page)

(continued from previous page)

```
    </element>
</repeat>
```

This example shows elements with independent element state and animation state, using the animation state taken from an emulated positional input to control their positions:

```
<repeat count="4">
    <param name="y" start="1" increment="3" />
    <param name="n" start="0" increment="1" />
    <element ref="ledr" name="~n~.7">
        <animate inputtag="IN.1" mask="0x0f" />
        <bounds state="0" x="0" y="~y~" width="1" height="1" />
        <bounds state="11" x="16.5" y="~y~" width="1" height="1" />
    </element>
</repeat>
```

12.2.5 Error handling

- For internal (developer-supplied) layout files, errors detected by the `complay.py` script result in a build failure.
- MAME will stop loading a layout file if a syntax error is encountered. No views from the layout will be available. Examples of syntax errors include undefined element or group references, invalid bounds, invalid colours, recursively nested groups, and redefined generator parameters.
- When loading a layout file, if a view references a non-existent screen, MAME will print a warning message and continue. Views referencing non-existent screens are considered unviable and not available to the user.

12.2.6 Automatically-generated views

After loading internal (developer-supplied) and external (user-supplied) layouts, MAME automatically generates views based on the machine configuration. The following views will be automatically generated:

- If the system has no screens and no viable views were found in the internal and external layouts, MAME will load a view that shows the message "No visual output."
- For each emulated screen, MAME will generate a view showing the screen at its physical aspect ratio with rotation applied.
- For each emulated screen where the configured pixel aspect ratio doesn't match the physical aspect ratio, MAME will generate a view showing the screen at an aspect ratio that produces square pixels, with rotation applied.
- If the system has a single emulated screen, MAME will generate a view showing two copies of the screen image above each other with a small gap between them. The upper copy will be rotated by 180 degrees. This view can be used in a "cocktail table" cabinet for simultaneous two-player games, or alternating play games that don't automatically rotate the display for the second player. The screen will be displayed at its physical aspect ratio, with rotation applied.
- If the system has exactly two emulated screens, MAME will generate a view showing the second screen above the first screen with a small gap between them. The second screen will be rotated by 180 degrees. This view can be used to play a dual-screen two-player game on a "cocktail table" cabinet with a single screen. The screens will be displayed at their physical aspect ratios, with rotation applied.
- If the system has exactly two emulated screens and no view in the internal or external layouts shows all screens, or if the system has more than two emulated screens, MAME will generate views with the screens arranged

horizontally from left to right and vertically from top to bottom, both with and without small gaps between them. The screens will be displayed at physical aspect ratio, with rotation applied.

- If the system has three or more emulated screens, MAME will generate views tiling the screens in grid patterns, in both row-major (left-to-right then top-to-bottom) and column-major (top-to-bottom then left-to-right) order. Views are generated with and without gaps between the screens. The screens will be displayed at physical aspect ratio, with rotation applied.

12.2.7 Using complay.py

The MAME source contains a Python script called `complay.py`, found in the `scripts/build` subdirectory. This script is used as part of MAME's build process to reduce the size of data for internal layouts and convert it to a form that can be built into the executable. However, it can also detect many common layout file format errors, and generally provides better error messages than MAME does when loading a layout file. Note that it doesn't actually run the whole layout engine, so it can't detect errors like undefined element references when parameters are used, or recursively nested groups. The `complay.py` script is compatible with both Python 2.7 and Python 3 interpreters.

The `complay.py` script takes three parameters – an input file name, an output file name, and a base name for variables in the output:

python scripts/build/complay.py <*input*> [<*output*> [<*varname*>]]

The input file name is required. If no output file name is supplied, `complay.py` will parse and check the input, reporting any errors found, without producing output. If no base variable name is provided, `complay.py` will generate one based on the input file name. This is not guaranteed to produce valid identifiers. The exit status is 0 (zero) on success, 1 on an error in the command invocation, 2 if error are found in the input file, or 3 in case of an I/O error. If an output file name is specified, the file will be created/overwritten on success or removed on failure.

To check a layout file for common errors, run the script with the path to the file to check and no output file name or base variable name. For example:

python scripts/build/complay.py artwork/dino/default.lay

12.2.8 Example layout files

These layout files demonstrate various artwork system features. They are all internal layouts included in MAME.

sstrangr.lay
: A simple case of using translucent colour overlays to visually separate and highlight elements on a black and white screen.

seawolf.lay
: This system uses lamps for key gameplay elements. Blending modes are used for the translucent colour overlay placed over the monitor, and the lamps reflected in front of the monitor. Also uses collections to allow parts of the layout to be disabled selectively.

armora.lay
: This game's monitor is viewed directly through a translucent colour overlay rather than being reflected from inside the cabinet. This means the overlay reflects ambient light as well as affecting the colour of the video image. The shapes on the overlay are drawn using embedded SVG images.

tranz330.lay
: A multi-segment alphanumeric display and keypad. The keys are clickable, and provide visual feedback when pressed.

esq2by16.lay
: Builds up a multi-line dot matrix character display. Repeats are used to avoid repetition for the rows in a character, characters in a line, and lines in a page. Group colors allow a single element to be used for all four display colours.

cgang.lay
Animates the position of element items to simulate an electromechanical shooting gallery game. Also demonstrates effective use of components to build up complex graphics.

minspace.lay
Shows the position of a slider control with LEDs on it.

md6802.lay
Effectively using groups as a procedural programming language to build up an image of a trainer board.

beena.lay
Using event-based scripting to dynamically position elements and draw element content programmatically.

12.3 MAME Layout Scripting

- *Introduction*
- *Practical examples*
  - *Espial: joystick split across ports*
  - *Star Wars: animation on two axes*
- *The layout script environment*
- *Layout events*
  - *Layout file events*
  - *Layout view events*
  - *Layout view item events*
  - *Layout element events*

12.3.1 Introduction

MAME layout files can embed Lua script to provide enhanced functionality. Although there's a lot you can do with conditionally drawn components and parameter animation, some things can only be done with scripting. MAME uses an event-based model. Scripts can supply functions that will be called after certain events, or when certain data is required.

Layout scripting requires the *layout plugin* to be enabled. For example, to run BWB Double Take with the Lua script in the layout enabled, you might use this command:

```
mame -plugins -plugin layout v4dbltak
```

You may want to add the settings to enable the layout plugin to an INI file to save having to enable it every time you start a system. See *Plugins* for more information about using plugins with MAME.

12.3.2 Practical examples

Before diving into the technical details of how it works, we'll start with some example layout files using Lua script for enhancement. It's assumed that you're familiar with MAME's artwork system and have a basic understanding of Lua scripting. For details on MAME's layout file, see *MAME Layout Files*; for detailed descriptions of MAME's Lua interface, see *Lua Scripting Interface*.

Espial: joystick split across ports

Take a look at the player input definitions for Espial:

```
PORT_START("IN1")
PORT_BIT( 0x01, IP_ACTIVE_HIGH, IPT_START1 )
PORT_BIT( 0x02, IP_ACTIVE_HIGH, IPT_START2 )
PORT_BIT( 0x04, IP_ACTIVE_HIGH, IPT_JOYSTICK_LEFT )  PORT_8WAY PORT_COCKTAIL
PORT_BIT( 0x08, IP_ACTIVE_HIGH, IPT_JOYSTICK_RIGHT ) PORT_8WAY PORT_COCKTAIL
PORT_BIT( 0x10, IP_ACTIVE_HIGH, IPT_JOYSTICK_UP )    PORT_8WAY PORT_COCKTAIL
PORT_BIT( 0x20, IP_ACTIVE_HIGH, IPT_JOYSTICK_DOWN )  PORT_8WAY
PORT_BIT( 0x40, IP_ACTIVE_HIGH, IPT_JOYSTICK_DOWN )  PORT_8WAY PORT_COCKTAIL
PORT_BIT( 0x80, IP_ACTIVE_HIGH, IPT_BUTTON2 ) PORT_COCKTAIL

PORT_START("IN2")
PORT_BIT( 0x01, IP_ACTIVE_HIGH, IPT_UNKNOWN )
PORT_BIT( 0x02, IP_ACTIVE_HIGH, IPT_COIN1 )
PORT_BIT( 0x04, IP_ACTIVE_HIGH, IPT_UNKNOWN )
PORT_BIT( 0x08, IP_ACTIVE_HIGH, IPT_JOYSTICK_RIGHT ) PORT_8WAY
PORT_BIT( 0x10, IP_ACTIVE_HIGH, IPT_JOYSTICK_UP )    PORT_8WAY
PORT_BIT( 0x20, IP_ACTIVE_HIGH, IPT_BUTTON1 ) PORT_COCKTAIL
PORT_BIT( 0x40, IP_ACTIVE_HIGH, IPT_BUTTON1 )
PORT_BIT( 0x80, IP_ACTIVE_HIGH, IPT_JOYSTICK_LEFT )  PORT_8WAY
```

There are two joysticks, one used for both players on an upright cabinet or the first player on a cocktail cabinet, and one used for the second player on a cocktail cabinet. Notice that the switches for the first joystick are split across the two I/O ports.

There's no layout file syntax to build the element state using bits from multiple I/O ports. It's also inconvenient if each joystick needs to be defined as a separate element because the bits for the switches aren't arranged the same way.

We can overcome these limitations using a script to read the player inputs and set the items' element state:

```
<?xml version="1.0"?>
<mamelayout version="2">

    <!-- element for drawing a joystick -->
    <!-- up = 1 (bit 0), down = 2 (bit 1), left = 4 (bit 2), right = 8 (bit 3) -->
    <element name="stick" defstate="0">
        <image state="0x0" file="stick_c.svg" />
        <image state="0x1" file="stick_u.svg" />
        <image state="0x9" file="stick_ur.svg" />
        <image state="0x8" file="stick_r.svg" />
        <image state="0xa" file="stick_dr.svg" />
        <image state="0x2" file="stick_d.svg" />
        <image state="0x6" file="stick_dl.svg" />
        <image state="0x4" file="stick_l.svg" />
```

(continues on next page)

(continued from previous page)

```
        <image state="0x5" file="stick_ul.svg" />
    </element>

    <!-- we'll warn the user if the layout plugin isn't enabled -->
    <!-- draw only when state is 1, and set the default state to 1 so warning is visible␣
↪initially -->
    <element name="warning" defstate="1">
        <text state="1" string="This view requires the layout plugin." />
    </element>

    <!-- view showing the screen and joysticks on a cocktail cabinet -->
    <view name="Joystick Display">
        <!-- draw the screen with correct aspect ratio -->
        <screen index="0">
            <bounds x="0" y="0" width="4" height="3" />
        </screen>

        <!-- first joystick, id attribute allows script to find item -->
        <!-- no bindings, state will be set by the script -->
        <element id="joy_p1" ref="stick">
            <!-- position below the screen -->
            <bounds xc="2" yc="3.35" width="0.5" height="0.5" />
        </element>

        <!-- second joystick, id attribute allows script to find item -->
        <!-- no bindings, state will be set by the script -->
        <element id="joy_p2" ref="stick">
            <!-- screen is flipped for second player, so rotate by 180 degrees -->
            <orientation rotate="180" />
            <!-- position above the screen -->
            <bounds xc="2" yc="-0.35" width="0.5" height="0.5" />
        </element>

        <!-- warning text item also has id attribute so the script can find it -->
        <element id="warning" ref="warning">
            <!-- position over the screen near the bottom -->
            <bounds x="0.2" y="2.6" width="3.6" height="0.2" />
        </element>
    </view>

    <!-- the content of the script element will be called as a function by the layout␣
↪plugin -->
    <!-- use CDATA block to avoid the need to escape angle brackets and ampersands -->
    <script><![CDATA[
        -- file is the layout file object
        -- set a function to call after resolving tags
        file:set_resolve_tags_callback(
                function ()
                    -- file.device is the device that caused the layout to be loaded
                    -- in this case, it's the root machine driver for espial
                    -- look up the two I/O ports we need to be able to read
                    local in1 = file.device:ioport("IN1")
```

(continues on next page)

(continued from previous page)

```
                    local in2 = file.device:ioport("IN2")

                    -- look up the view items for showing the joystick state
                    local p1_stick = file.views["Joystick Display"].items["joy_p1"]
                    local p2_stick = file.views["Joystick Display"].items["joy_p2"]

                    -- set a function to call before adding the view items to the render␣
↪target
                    file.views["Joystick Display"]:set_prepare_items_callback(
                            function ()
                                -- read the two player input I/O ports
                                local in1_val = in1:read()
                                local in2_val = in2:read()

                                -- set element state for first joystick
                                p1_stick:set_state(
                                        ((in2_val & 0x10) >> 4) |   -- shift up from IN2␣
↪bit 4 to bit 0
                                        ((in1_val & 0x20) >> 4) |   -- shift down from␣
↪IN1 bit 5 to bit 1
                                        ((in2_val & 0x80) >> 5) |   -- shift left from␣
↪IN2 bit 7 to bit 2
                                        (in2_val & 0x08))           -- right is in IN2␣
↪bit 3

                                -- set element state for second joystick
                                p2_stick:set_state(
                                        ((in1_val & 0x10) >> 4) |   -- shift up from IN1␣
↪bit 4 to bit 0
                                        ((in1_val & 0x40) >> 5) |   -- shift down from␣
↪IN1 bit 6 to bit 1
                                        (in1_val & 0x04) |          -- left is in IN1␣
↪bit 2
                                        (in1_val & 0x08))           -- right is in IN1␣
↪bit 3
                            end)

                    -- hide the warning, since if we got here the script is running
                    file.views["Joystick Display"].items["warning"]:set_state(0)
                end)
    ]]></script>

</mamelayout>
```

The layout has a `script` element containing the Lua script. This is called as a function by the layout plugin when the layout file is loaded. The layout views have been built at this point, but the emulated system has not finished starting. In particular, it's not safe to access inputs and outputs at this time. The key variable in the script environment is `file`, which gives the script access to its *layout file*.

We supply a function to be called after tags in the layout file have been resolved. At this point, the emulated system will have completed starting. This function does the following tasks:

- Looks up the two *I/O ports* used for player input. I/O ports can be looked up by tag relative to the device that caused the layout file to be loaded.

- Looks up the two *view items* used to display joystick state. Views can be looked up by name (i.e. value of the `name` attribute), and items within a view can be looked up by ID (i.e. the value of the `id` attribute).
- Supplies a function to be called before view items are added to the render target when drawing a frame.
- Hides the warning that reminds the user to enable the layout plugin by setting the element state for the item to 0 (the text component is only drawn when the element state is 1).

The function called before view items are added to the render target reads the player inputs, and shuffles the bits into the order needed by the joystick element.

Star Wars: animation on two axes

We'll make a layout that shows the position of the flight yoke for Atari Star Wars. The input ports are straightforward – each analog axis produces a value in the range from 0x00 (0) to 0xff (255), inclusive:

```
PORT_START("STICKY")
PORT_BIT( 0xff, 0x80, IPT_AD_STICK_Y ) PORT_SENSITIVITY(70) PORT_KEYDELTA(30)

PORT_START("STICKX")
PORT_BIT( 0xff, 0x80, IPT_AD_STICK_X ) PORT_SENSITIVITY(50) PORT_KEYDELTA(30)
```

Here's our layout file:

```
<?xml version="1.0"?>
<mamelayout version="2">

    <!-- a square with a white outline 1% of its width -->
    <element name="outline">
        <rect><bounds x="0.00" y="0.00" width="1.00" height="0.01" /></rect>
        <rect><bounds x="0.00" y="0.99" width="1.00" height="0.01" /></rect>
        <rect><bounds x="0.00" y="0.00" width="0.01" height="1.00" /></rect>
        <rect><bounds x="0.99" y="0.00" width="0.01" height="1.00" /></rect>
    </element>

    <!-- a rectangle with a vertical line 10% of its width down the middle -->
    <element name="line">
        <!-- use a transparent rectangle to force element dimensions -->
        <rect>
            <bounds x="0" y="0" width="0.1" height="1" />
            <color alpha="0" />
        </rect>
        <!-- this is the visible white line -->
        <rect><bounds x="0.045" y="0" width="0.01" height="1" /></rect>
    </element>

    <!-- an outlined square inset by 20% with lines 10% of the element width/height -->
    <element name="box">
        <!-- use a transparent rectangle to force element dimensions -->
        <rect>
            <bounds x="0" y="0" width="0.1" height="0.1" />
            <color alpha="0" />
        </rect>
        <!-- draw the outline of a square -->
```

(continues on next page)

(continued from previous page)

```
        <rect><bounds x="0.02" y="0.02" width="0.06" height="0.01" /></rect>
        <rect><bounds x="0.02" y="0.07" width="0.06" height="0.01" /></rect>
        <rect><bounds x="0.02" y="0.02" width="0.01" height="0.06" /></rect>
        <rect><bounds x="0.07" y="0.02" width="0.01" height="0.06" /></rect>
    </element>

    <!-- we'll warn the user if the layout plugin isn't enabled -->
    <!-- draw only when state is 1, and set the default state to 1 so warning is visible␣
↪initially -->
    <element name="warning" defstate="1">
        <text state="1" string="This view requires the layout plugin." />
    </element>

    <!-- view showing the screen and flight yoke position -->
    <view name="Analog Control Display">
        <!-- draw the screen with correct aspect ratio -->
        <screen index="0">
            <bounds x="0" y="0" width="4" height="3" />
        </screen>

        <!-- draw the white outlined square to the right of the screen near the bottom --
↪>
        <!-- the script uses the size of this item to determine movement ranges -->
        <element id="outline" ref="outline">
            <bounds x="4.1" y="1.9" width="1.0" height="1.0" />
        </element>

        <!-- vertical line for displaying X axis input -->
        <element id="vertical" ref="line">
            <!-- element draws a vertical line, no need to rotate it -->
            <orientation rotate="0" />
            <!-- centre it in the square horizontally, using the full height -->
            <bounds x="4.55" y="1.9" width="0.1" height="1" />
        </element>

        <!-- horizontal line for displaying Y axis input -->
        <element id="horizontal" ref="line">
            <!-- rotate the element by 90 degrees to get a horizontal line -->
            <orientation rotate="90" />
            <!-- centre it in the square vertically, using the full width -->
            <bounds x="4.1" y="2.35" width="1" height="0.1" />
        </element>

        <!-- draw a small box at the intersection of the vertical and horizontal lines --
↪>
        <element id="box" ref="box">
            <bounds x="4.55" y="2.35" width="0.1" height="0.1" />
        </element>

        <!-- draw the warning text over the screen near the bottom -->
        <element id="warning" ref="warning">
            <bounds x="0.2" y="2.6" width="3.6" height="0.2" />
```

(continues on next page)

(continued from previous page)

```
        </element>
    </view>

    <!-- the content of the script element will be called as a function by the layout␣
↪plugin -->
    <!-- use CDATA block to avoid the need to escape angle brackets and ampersands -->
    <script><![CDATA[
        -- file is the layout file object
        -- set a function to call after resolving tags
        file:set_resolve_tags_callback(
                function ()
                    -- file.device is the device that caused the layout to be loaded
                    -- in this case, it's the root machine driver for starwars
                    -- find the analog axis inputs
                    local x_input = file.device:ioport("STICKX")
                    local y_input = file.device:ioport("STICKY")

                    -- find the outline item
                    local outline_item = file.views["Analog Control Display"].items[
↪"outline"]

                    -- variables for keeping state across callbacks
                    local outline_bounds    -- bounds of the outlined square
                    local width, height     -- width and height for animated items
                    local x_scale, y_scale  -- ratios of axis units to render coordinates
                    local x_pos, y_pos      -- display positions for the animated items

                    -- set a function to call when view dimensions have been recalculated
                    -- this can happen when when the window is resized or scaling␣
↪options are changed
                    file.views["Analog Control Display"]:set_recomputed_callback(
                            function ()
                                -- get the bounds of the outlined square
                                outline_bounds = outline_item.bounds
                                -- animated items use 10% of the width/height of the␣
↪square
                                width = outline_bounds.width * 0.1
                                height = outline_bounds.height * 0.1
                                -- calculate ratios of axis units to render coordinates
                                -- animated items leave 90% of the width/height for the␣
↪movement range
                                -- the end of the range of each axis is at 0xff
                                x_scale = outline_bounds.width * 0.9 / 0xff
                                y_scale = outline_bounds.height * 0.9 / 0xff
                            end)

                    -- set a function to call before adding the view items to the render␣
↪target
                    file.views["Analog Control Display"]:set_prepare_items_callback(
                            function ()
                                -- read analog axes, reverse Y axis as zero is at the␣
↪bottom
```

(continues on next page)

(continued from previous page)

```
                                local x = x_input:read() & 0xff
                                local y = 0xff - (y_input:read() & 0xff)
                                -- convert the input values to layout coordinates
                                -- use the top left corner of the outlined square as the␣
↪origin
                                x_pos = outline_bounds.x0 + (x * x_scale)
                                y_pos = outline_bounds.y0 + (y * y_scale)
                            end)

                    -- set a function to supply the bounds for the vertical line
                    file.views["Analog Control Display"].items["vertical"]:set_bounds_
↪callback(
                            function ()
                                -- create a new render bounds object (starts as a unit␣
↪square)
                                local result = emu.render_bounds()
                                -- set left, top, width and height
                                result:set_wh(
                                        x_pos,                  -- calculated X position␣
↪for animated items
                                        outline_bounds.y0,      -- top of outlined square
                                        width,                  -- 10% of width of␣
↪outlined square
                                        outline_bounds.height)  -- full height of␣
↪outlined square
                                return result
                            end)

                    -- set a function to supply the bounds for the horizontal line
                    file.views["Analog Control Display"].items["horizontal"]:set_bounds_
↪callback(
                            function ()
                                -- create a new render bounds object (starts as a unit␣
↪square)
                                local result = emu.render_bounds()
                                -- set left, top, width and height
                                result:set_wh(
                                        outline_bounds.x0,      -- left of outlined␣
↪square
                                        y_pos,                  -- calculated Y position␣
↪for animated items
                                        outline_bounds.width,   -- full width of␣
↪outlined square
                                        height)                 -- 10% of height of␣
↪outlined square
                                return result
                            end)

                    -- set a function to supply the bounds for the box at the␣
↪intersection of the lines
                    file.views["Analog Control Display"].items["box"]:set_bounds_
↪callback(
```

(continues on next page)

(continued from previous page)

```
                            function ()
                                -- create a new render bounds object (starts as a unit␣
↪square)
                                local result = emu.render_bounds()
                                -- set left, top, width and height
                                result:set_wh(
                                        x_pos,                  -- calculated X position␣
↪for animated items
                                        y_pos,                  -- calculated Y position␣
↪for animated items
                                        width,                  -- 10% of width of␣
↪outlined square
                                        height)                 -- 10% of height of␣
↪outlined square
                                return result
                            end)

                    -- hide the warning, since if we got here the script is running
                    file.views["Analog Control Display"].items["warning"]:set_state(0)
                end)
    ]]></script>

</mamelayout>
```

The layout has a `script` element containing the Lua script, to be called as a function by the layout plugin when the layout file is loaded. This happens after the layout views have been build, but before the emulated system has finished starting. The *layout file* object is supplied to the script in the `file` variable.

We supply a function to be called after tags in the layout file have been resolved. This function does the following:

- Looks up the analog axis *inputs*.
- Looks up the *view item* that draws the outline of area where the yoke position is displayed.
- Declares some variables to hold calculated values across function calls.
- Supplies a function to be called when the view's dimensions have been recomputed.
- Supplies a function to be called before adding view items to the render container when drawing a frame.
- Supplies functions that will supply the bounds for the animated items.
- Hides the warning that reminds the user to enable the layout plugin by setting the element state for the item to 0 (the text component is only drawn when the element state is 1).

The view is looked up by name (value of its `name` attribute), and items within the view are looked up by ID (values of their `id` attributes).

Layout view dimensions are recomputed in response to several events, including the window being resized, entering/leaving full screen mode, toggling visibility of item collections, and changing the zoom to screen area setting. When this happens, we need to update our size and animation scale factors. We get the bounds of the square where the yoke position is displayed, calculate the size for the animated items, and calculate the ratios of axis units to render target coordinates in each direction. It's more efficient to do these calculations only when the results may change.

Before view items are added to the render target, we read the analog axis inputs and convert the values to coordinates positions for the animated items. The Y axis input uses larger values to aim higher, so we need to reverse the value by subtracting it from 0xff (255). We add in the coordinates of the top left corner of the square where we're displaying

the yoke position. We do this once each time the layout is drawn for efficiency, since we can use the values for all three animated items.

Finally, we supply bounds for the animated items when required. These functions need to return `render_bounds` objects giving the position and size of the items in render target coordinates.

(Since the vertical and horizontal line elements each only move on a single axis, it would be possible to animate them using the layout file format's item animation features. Only the box at the intersection of the line actually requires scripting. It's done entirely using scripting here for illustrative purposes.)

12.3.3 The layout script environment

The Lua environment is provided by the layout plugin. It's fairly minimal, only providing what's needed:

- `file` giving the script's *layout file* object. Has a `device` property for obtaining the *device* that caused the layout file to be loaded, and a `views` property for obtaining the layout's *views* (indexed by name).
- `machine` giving MAME's current *running machine*.
- `emu.device_enumerator`, `emu.palette_enumerator`, `emu.screen_enumerator`, `emu.cassette_enumerator`, `emu.image_enumerator` and `emu.slot_enumerator` functions for obtaining specific device interfaces.
- `emu.attotime`, `emu.render_bounds` and `emu.render_color` functions for creating *attotime*, *bounds* and *colour* objects.
- `emu.bitmap_ind8`, `emu.bitmap_ind16`, `emu.bitmap_ind32`, `emu.bitmap_ind64`, `emu.bitmap_yuy16`, `emu.bitmap_rgb32` and `emu.bitmap_argb32` objects for creating *bitmaps*.
- `emu.print_verbose`, `emu.print_error`, `emu.print_warning`, `emu.print_info` and `emu.print_debug` functions for diagnostic output.
- Standard Lua `tonumber`, `tostring`, `pairs` and `ipairs` functions, and `math`, `table` and `string` objects for manipulating numbers, strings, tables and other containers.
- Standard Lua `print` function for text output to the console.

12.3.4 Layout events

MAME layout scripting uses an event-based model. Scripts can supply functions to be called after events occur, or when data is needed. There are three levels of events: layout file events, layout view events, and layout view item events.

Layout file events

Layout file events apply to the file as a whole, and not to an individual view.

Resolve tags

`file:set_resolve_tags_callback(cb)`

Called after the emulated system has finished starting, input and output tags in the layout have been resolved, and default item callbacks have been set up. This is a good time to look up inputs and set up view item event handlers.

The callback function has no return value and takes no parameters. Call with `nil` as the argument to remove the event handler.

Layout view events

Layout view events apply to an individual view.

Prepare items

`view:set_prepare_items_callback(cb)`

Called before the view's items are added to the render target in preparation for drawing a video frame.

The callback function has no return value and takes no parameters. Call with `nil` as the argument to remove the event handler.

Preload

`view:set_preload_callback(cb)`

Called after pre-loading visible view elements. This can happen when the view is selected for the first time in a session, or when the user toggles visibility of an element collection on. Be aware that this can be called multiple times in a session and avoid repeating expensive tasks.

The callback function has no return value and takes no parameters. Call with `nil` as the argument to remove the event handler.

Dimensions recomputed

`view:set_recomputed_callback(cb)`

Called after view dimensions are recomputed. This happens in several situations, including the window being resized, entering or leaving full screen mode, toggling visibility of item collections, and changes to the rotation and zoom to screen area settings. If you're animating the position of view items, this is a good time to calculate positions and scale factors.

The callback function has no return value and takes no parameters. Call with `nil` as the argument to remove the event handler.

Pointer updated

`view:set_pointer_updated_callback(cb)`

Called when a pointer enters, moves or changes button state over the view.

The callback function is passed nine arguments:

- The pointer type as a string. This will be `mouse`, `pen`, `touch` or `unknown`, and will not change for the lifetime of a pointer.
- The pointer ID. This will be a non-negative integer that will not change for the lifetime of a pointer. Pointer ID values are recycled aggressively.
- The device ID. This will be a non-negative integer that can be used to group pointers for recognising multi-touch gestures.
- The horizontal position of the pointer in layout coordinates.
- The vertical position of the pointer in layout coordinates.
- A bit mask representing the currently pressed buttons. The primary button is the least significant bit.
- A bit mask representing the buttons that were pressed in this update. The primary button is the least significant bit.
- A bit mask representing the buttons that were released in this update. The primary button is the least significant bit.
- The click count. This is positive for multi-click actions, or negative if a click is turned into a hold or drag. This only applies to the primary button.

The callback function has no return value. Call with `nil` as the argument to remove the event handler.

Pointer left

`view:set_pointer_left_callback(cb)`

Called when a pointer leaves the view normally. After receiving this event, the pointer ID may be reused for a new pointer.

The callback function is passed seven arguments:

- The pointer type as a string. This will be `mouse`, `pen`, `touch` or `unknown`, and will not change for the lifetime of a pointer.
- The pointer ID. This will be a non-negative integer that will not change for the lifetime of a pointer. Pointer ID values are recycled aggressively.
- The device ID. This will be a non-negative integer that can be used to group pointers for recognising multi-touch gestures.
- The horizontal position of the pointer in layout coordinates.
- The vertical position of the pointer in layout coordinates.
- A bit mask representing the buttons that were released in this update. The primary button is the least significant bit.
- The click count. This is positive for multi-click actions, or negative if a click is turned into a hold or drag. This only applies to the primary button.

The callback function has no return value. Call with `nil` as the argument to remove the event handler.

Pointer aborted

`view:set_pointer_aborted_callback(cb)`

Called when a pointer leaves the view abnormally. After receiving this event, the pointer ID may be reused for a new pointer.

The callback function is passed seven arguments:

- The pointer type as a string. This will be `mouse`, `pen`, `touch` or `unknown`, and will not change for the lifetime of a pointer.
- The pointer ID. This will be a non-negative integer that will not change for the lifetime of a pointer. Pointer ID values are recycled aggressively.
- The device ID. This will be a non-negative integer that can be used to group pointers for recognising multi-touch gestures.
- The horizontal position of the pointer in layout coordinates.
- The vertical position of the pointer in layout coordinates.
- A bit mask representing the buttons that were released in this update. The primary button is the least significant bit.
- The click count. This is positive for multi-click actions, or negative if a click is turned into a hold or drag. This only applies to the primary button.

The callback function has no return value. Call with `nil` as the argument to remove the event handler.

Forget pointers

`view:set_forget_pointers_callback(cb)`

Called when the view should stop processing pointer input. This can happen in a number of situations, including:

- The user activated a menu.
- The view configuration will change.

- The view will be deactivated.

The callback function has no return value and takes no parameters. Call with `nil` as the argument to remove the event handler.

Layout view item events

Layout view item callbacks apply to individual items within a view. They are used to override items' default element state, animation state, bounds and colour behaviour.

Get element state

`item:set_element_state_callback(cb)`

Set callback for getting the item's element state. This controls how the item's element is drawn, for components that change appearance depending on state, conditionally-drawn components, and component bounds/colour animation. Do not attempt to access the item's `element_state` property from the callback, as it will result in infinite recursion.

The callback function must return an integer, and takes no parameters. Call with `nil` as the argument to restore the default element state handler (based on the item's XML attributes).

Get animation state

`item:set_animation_state_callback(cb)`

Set callback for getting the item's animation state. This is used for item bounds/colour animation. Do not attempt to access the item's `animation_state` property from the callback, as it will result in infinite recursion.

The callback function must return an integer, and takes no parameters. Call with `nil` as the argument to restore the default animation state handler (based on the item's XML attributes and `animate` child element).

Get item bounds

`item:set_bounds_callback(cb)`

Set callback for getting the item's bounds (position and size). Do not attempt to access the item's `bounds` property from the callback, as it will result in infinite recursion.

The callback function must return a render bounds object representing the item's bounds in render target coordinates (usually created by calling `emu.render_bounds`), and takes no parameters. Call with `nil` as the argument to restore the default bounds handler (based on the item's animation state and `bounds` child elements).

Get item colour

`item:set_color_callback(cb)`

Set callback for getting the item's colour (the element texture's colours multiplied by this colour). Do not attempt to access the item's `color` property from the callback, as it will result in infinite recursion.

The callback function must return a render colour object representing the ARGB colour (usually created by calling `emu.render_color`), and takes no parameters. Call with `nil` as the argument to restore the default colour handler (based on the item's animation state and `color` child elements).

Get item horizontal scroll window size

`item:set_scroll_size_x_callback(cb)`

Set callback for getting the item's horizontal scroll window size. This allows the script to control how much of the element is displayed by the item. Do not attempt to access the item's `scroll_size_x` property from the callback, as it will result in infinite recursion.

The callback function must return a floating-point number representing the horizontal window size as a proportion of the associated element's width, and takes no parameters. A value of 1.0 will display the entire width of the element; smaller values will display proportionally smaller parts of the element. Call with `nil` as the argument to restore the default horizontal scroll window size handler (based on the `xscroll` child element).

Get item vertical scroll window size

`item:set_scroll_size_y_callback(cb)`

Set callback for getting the item's vertical scroll window size. This allows the script to control how much of the element is displayed by the item. Do not attempt to access the item's `scroll_size_y` property from the callback, as it will result in infinite recursion.

The callback function must return a floating-point number representing the vertical window size as a proportion of the associated element's height, and takes no parameters. A value of 1.0 will display the entire height of the element; smaller values will display proportionally smaller parts of the element. Call with `nil` as the argument to restore the default vertical scroll window size handler (based on the `xscroll` child element).

Get item horizontal scroll position

`item:set_scroll_pos_x_callback(cb)`

Set callback for getting the item's horizontal scroll position. This allows the script to control which part of the element is displayed by the item. Do not attempt to access the item's `scroll_pos_x` property from the callback, as this will result in infinite recursion.

The callback must return a floating-point number, and takes no parameters. A value of 0.0 aligns the left edge of the element with the left edge of the item; larger values pan right. Call with `nil` as the argument to restore the default horizontal scroll position handler (based on bindings in the `xscroll` child element).

Get item vertical scroll position

`item:set_scroll_pos_y_callback(cb)`

Set callback for getting the item's vertical scroll position. This allows the script to control which part of the element is displayed by the item. Do not attempt to access the item's `scroll_pos_y` property from the callback, as this will result in infinite recursion.

The callback must return a floating-point number, and takes no parameters. A value of 0.0 aligns the top edge of the element with the top edge of the item; larger values pan down. Call with `nil` as the argument to restore the default vertical scroll position handler (based on bindings in the `yscroll` child element).

Layout element events

Layout element events apply to an individual visual element definition.

Draw

`element:set_draw_callback(cb)`

Set callback for additional drawing after the element's components have been drawn. This gives the script direct control over the final texture when an element item is drawn.

The callback is passed two arguments: the element state (an integer) and the 32-bit ARGB bitmap at the required size. The callback must not attempt to resize the bitmap. Call with `nil` as the argument to remove the event handler.

12.4 Object Finders

- *Introduction*
- *Types of object finder*
- *Finding resources*

- *Connections between devices*
- *Object finder arrays*
- *Optional object finders*
  - *Optional system components*
  - *Optional resources*
- *Object finder types in more detail*
  - *Device finders*
  - *Memory system object finders*
  - *I/O port finders*
  - *Address space finders*
  - *Memory pointer finders*
- *Output finders*

12.4.1 Introduction

Object finders are an important part of the glue MAME provides to tie the devices that make up an emulated system together. Object finders are used to specify connections between devices, to efficiently access resources, and to check that necessary resources are available on validation.

Object finders search for a target object by tag relative to a base device. Some types of object finder require additional parameters.

Most object finders have required and optional versions. The required versions will raise an error if the target object is not found. This will prevent a device from starting or cause a validation error. The optional versions will log a verbose message if the target object is not found, and provide additional members for testing whether the target object was found or not.

Object finder classes are declared in the header src/emu/devfind.h and have Doxygen format API documentation.

12.4.2 Types of object finder

required_device<DeviceClass>, optional_device<DeviceClass>
: Finds a device. The template argument `DeviceClass` should be a class derived from `device_t` or `device_interface`.

required_memory_region, optional_memory_region
: Finds a memory region, usually from ROM definitions. The target is the `memory_region` object.

required_memory_bank, optional_memory_bank
: Finds a memory bank instantiated in an address map. The target is the `memory_bank` object.

memory_bank_creator
: Finds a memory bank instantiated in an address map, or creates it if it doesn't exist. The target is the `memory_bank` object. There is no optional version, because the target object will always be found or created.

required_ioport, optional_ioport
: Finds an I/O port from a device's input port definitions. The target is the `ioport_port` object.

required_address_space, optional_address_space
: Finds a device's address space. The target is the `address_space` object.

required_region_ptr<PointerType>, optional_region_ptr<PointerType>
: Finds the base pointer of a memory region, usually from ROM definitions. The template argument `PointerType` is the target type (usually an unsigned integer type). The target is the first element in the memory region.

required_shared_ptr<PointerType>, optional_shared_ptr<PointerType>
: Finds the base pointer of a memory share instantiated in an address map. The template argument `PointerType` is the target type (usually an unsigned integer type). The target is the first element in the memory share.

memory_share_creator<PointerType>
: Finds the base pointer of a memory share instantiated in an address map, or creates it if it doesn't exist. The template argument `PointerType` is the target type (usually an unsigned integer type). The target is the first element in the memory share. There is no optional version, because the target object will always be found or created.

12.4.3 Finding resources

We'll start with a simple example of a device that uses object finders to access its own child devices, inputs and ROM region. The code samples here are based on the Apple II Parallel Printer Interface card, but a lot of things have been removed for clarity.

Object finders are declared as members of the device class:

```
class a2bus_parprn_device : public device_t, public device_a2bus_card_interface
{
public:
    a2bus_parprn_device(machine_config const &mconfig, char const *tag, device_t *owner,␣
↪u32 clock);

    virtual void write_c0nx(u8 offset, u8 data) override;
    virtual u8 read_cnxx(u8 offset) override;

protected:
    virtual tiny_rom_entry const *device_rom_region() const override;
    virtual void device_add_mconfig(machine_config &config) override;
    virtual ioport_constructor device_input_ports() const override;

private:
    required_device<centronics_device>      m_printer_conn;
    required_device<output_latch_device>    m_printer_out;
    required_ioport                         m_input_config;
    required_region_ptr<u8>                 m_prom;
};
```

We want to find a `centronics_device`, an `output_latch_device`, an I/O port, and an 8-bit memory region.

In the constructor, we set the initial target for the object finders:

```
a2bus_parprn_device::a2bus_parprn_device(machine_config const &mconfig, char const *tag,␣
↪device_t *owner, u32 clock) :
    device_t(mconfig, A2BUS_PARPRN, tag, owner, clock),
    device_a2bus_card_interface(mconfig, *this),
    m_printer_conn(*this, "prn"),
```

(continues on next page)

(continued from previous page)

```
    m_printer_out(*this, "prn_out"),
    m_input_config(*this, "CFG"),
    m_prom(*this, "prom")
{
}
```

Each object finder takes a base device and tag as constructor arguments. The base device supplied at construction serves two purposes. Most obviously, the tag is specified relative to this device. Possibly more importantly, the object finder registers itself with this device so that it will be called to perform validation and object resolution.

Note that the object finders *do not* copy the tag strings. The caller must ensure the tag string remains valid until after validation and/or object resolution is complete.

The memory region and I/O port come from the ROM definition and input definition, respectively:

```
namespace {

ROM_START(parprn)
    ROM_REGION(0x100, "prom", 0)
    ROM_LOAD( "prom.b4", 0x0000, 0x0100, BAD_DUMP CRC(00b742ca)␣
↪SHA1(c67888354aa013f9cb882eeeed924e292734e717) )
ROM_END

INPUT_PORTS_START(parprn)
    PORT_START("CFG")
    PORT_CONFNAME(0x01, 0x00, "Acknowledge latching edge")
    PORT_CONFSETTING(   0x00, "Falling (/Y-B)")
    PORT_CONFSETTING(   0x01, "Rising (Y-B)")
    PORT_CONFNAME(0x06, 0x02, "Printer ready")
    PORT_CONFSETTING(   0x00, "Always (S5-C-D)")
    PORT_CONFSETTING(   0x02, "Acknowledge latch (Z-C-D)")
    PORT_CONFSETTING(   0x04, "ACK (Y-C-D)")
    PORT_CONFSETTING(   0x06, "/ACK (/Y-C-D)")
    PORT_CONFNAME(0x08, 0x00, "Strobe polarity")
    PORT_CONFSETTING(   0x00, "Negative (S5-A-/X, GND-X)")
    PORT_CONFSETTING(   0x08, "Positive (S5-X, GND-A-/X)")
    PORT_CONFNAME(0x10, 0x10, "Character width")
    PORT_CONFSETTING(   0x00, "7-bit")
    PORT_CONFSETTING(   0x10, "8-bit")
INPUT_PORTS_END

} // anonymous namespace

tiny_rom_entry const *a2bus_parprn_device::device_rom_region() const
{
    return ROM_NAME(parprn);
}

ioport_constructor a2bus_parprn_device::device_input_ports() const
{
    return INPUT_PORTS_NAME(parprn);
}
```

Note that the tags `"prom"` and `"CFG"` match the tags passed to the object finders on construction.

Child devices are instantiated in the device's machine configuration member function:

```
void a2bus_parprn_device::device_add_mconfig(machine_config &config)
{
    CENTRONICS(config, m_printer_conn, centronics_devices, "printer");
    m_printer_conn->ack_handler().set(FUNC(a2bus_parprn_device::ack_w));

    OUTPUT_LATCH(config, m_printer_out);
    m_printer_conn->set_output_latch(*m_printer_out);
}
```

Object finders are passed to device types to provide tags when instantiating child devices. After instantiating a child device in this way, the object finder can be used like a pointer to the device until the end of the machine configuration member function. Note that to use an object finder like this, its base device must be the same as the device being configured (the `this` pointer of the machine configuration member function).

After the emulated machine has been started, the object finders can be used in much the same way as pointers:

```
void a2bus_parprn_device::write_c0nx(u8 offset, u8 data)
{
    ioport_value const cfg(m_input_config->read());

    m_printer_out->write(data & (BIT(cfg, 8) ? 0xffU : 0x7fU));
    m_printer_conn->write_strobe(BIT(~cfg, 3));
}


u8 a2bus_parprn_device::read_cnxx(u8 offset)
{
    offset ^= 0x40U;
    return m_prom[offset];
}
```

For convenience, object finders that target the base pointer of memory regions and shares can be indexed like arrays.

12.4.4 Connections between devices

Devices need to be connected together within a system. For example the Sun SBus device needs access to the host CPU and address space. Here's how we declare the object finders in the device class (with all distractions removed):

```
DECLARE_DEVICE_TYPE(SBUS, sbus_device)

class sbus_device : public device_t, public device_memory_interface
{
    template <typename T, typename U>
    sbus_device(
            machine_config const &mconfig, char const *tag, device_t *owner, u32 clock,
            T &&cpu_tag,
            U &&space_tag, int space_num) :
        sbus_device(mconfig, tag, owner, clock)
    {
        set_cpu(std::forward<T>(cpu_tag));
        set_type1space(std::forward<U>(space_tag), space_num);
```

(continues on next page)

(continued from previous page)

```
    }

    sbus_device(machine_config const &mconfig, char const *tag, device_t *owner, u32␣
↪clock) :
        device_t(mconfig, SBUS, tag, owner, clock),
        device_memory_interface(mconfig, *this),
        m_maincpu(*this, finder_base::DUMMY_TAG),
        m_type1space(*this, finder_base::DUMMY_TAG, -1)
    {
    }

    template <typename T> void set_cpu(T &&tag) { m_maincpu.set_tag(std::forward<T>
↪(tag)); }
    template <typename T> void set_type1space(T &&tag, int num) { m_type1space.set_
↪tag(std::forward<T>(tag), num); }

protected:
    required_device<sparc_base_device> m_maincpu;
    required_address_space m_type1space;
};
```

There are several things to take note of here:

- Object finder members are declared for the things the device needs to access.
- The device doesn't know how it will fit into a larger system, the object finders are constructed with dummy arguments.
- Configuration member functions are provided to set the tag for the host CPU, and the tag and index for the type 1 address space.
- In addition to the standard device constructor, a constructor with additional parameters for setting the CPU and type 1 address space is provided.

The constant `finder_base::DUMMY_TAG` is guaranteed to be invalid and will not resolve to an object. This makes it easy to detect incomplete configuration and report an error. Address spaces are numbered from zero, so a negative address space number is invalid.

The member functions for configuring object finders take a universal reference to a tag-like object (templated type with && qualifier), as well as any other parameters needed by the specific type of object finder. An address space finder needs an address space number in addition to a tag-like object.

So what's a tag-like object? Three things are supported:

- A C string pointer (`char const *`) representing a tag relative to the device being configured. Note that the object finder will not copy the string. The caller must ensure it remains valid until resolution and/or validation is complete.
- Another object finder. The object finder will take on its current target.
- For device finders, a reference to an instance of the target device type, setting the target to that device. Note that this will not work if the device is subsequently replaced in the machine configuration. It's most often used with `*this`.

The additional constructor that sets initial configuration delegates to the standard constructor and then calls the configuration member functions. It's purely for convenience.

When we want to instantiate this device and hook it up, we do this:

```
SPARCV7(config, m_maincpu, 20'000'000);

ADDRESS_MAP_BANK(config, m_type1space);

SBUS(config, m_sbus, 20'000'000);
m_sbus->set_cpu(m_maincpu);
m_sbus->set_type1space(m_type1space, 0);
```

We supply the same object finders to instantiate the CPU and address space devices, and to configure the SBus device.

Note that we could also use literal C strings to configure the SBus device, at the cost of needing to update the tags in multiple places if they change:

```
SBUS(config, m_sbus, 20'000'000);
m_sbus->set_cpu("maincpu");
m_sbus->set_type1space("type1", 0);
```

If we want to use the convenience constructor, we just supply additional arguments when instantiating the device:

```
SBUS(config, m_sbus, 20'000'000, m_maincpu, m_type1space, 0);
```

12.4.5 Object finder arrays

Many systems have multiple similar devices, I/O ports or other resources that can be logically organised as an array. To simplify these use cases, object finder array types are provided. The object finder array type names have `_array` added to them:

required_device	required_device_array
optional_device	optional_device_array
required_memory_region	required_memory_region_array
optional_memory_region	optional_memory_region_array
required_memory_bank	required_memory_bank_array
optional_memory_bank	optional_memory_bank_array
memory_bank_creator	memory_bank_array_creator
required_ioport	required_ioport_array
optional_ioport	optional_ioport_array
required_address_space	required_address_space_array
optional_address_space	optional_address_space_array
required_region_ptr	required_region_ptr_array
optional_region_ptr	optional_region_ptr_array
required_shared_ptr	required_shared_ptr_array
optional_shared_ptr	optional_shared_ptr_array
memory_share_creator	memory_share_array_creator

A common case for an object array finder is a key matrix:

```
class keyboard_base : public device_t, public device_mac_keyboard_interface
{
protected:
    keyboard_base(machine_config const &mconfig, device_type type, char const *tag,␣
↪device_t *owner, u32 clock) :
```

(continues on next page)

(continued from previous page)

```
        device_t(mconfig, type, tag, owner, clock),
        device_mac_keyboard_interface(mconfig, *this),
        m_rows(*this, "ROW%u", 0U)
    {
    }

    u8 bus_r()
    {
        u8 result(0xffU);
        for (unsigned i = 0U; m_rows.size() > i; ++i)
        {
            if (!BIT(m_row_drive, i))
                result &= m_rows[i]->read();
        }
        return result;
    }

    required_ioport_array<10> m_rows;
};
```

Constructing an object finder array is similar to constructing an object finder, except that rather than just a tag you supply a tag format string and index offset. In this case, the tags of the I/O ports in the array will be `ROW0`, `ROW1`, `ROW2`, … `ROW9`. Note that the object finder array allocates dynamic storage for the tags, which remain valid until destruction.

The object finder array is used in much the same way as a `std::array` of the underlying object finder type. It supports indexing, iterators, and range-based `for` loops.

Because an index offset is specified, the tags don't need to use zero-based indices. It's common to use one-based indexing like this:

```
class dooyong_state : public driver_device
{
protected:
    dooyong_state(machine_config const &mconfig, device_type type, char const *tag) :
        driver_device(mconfig, type, tag),
        m_bg(*this, "bg%u", 1U),
        m_fg(*this, "fg%u", 1U)
    {
    }

    optional_device_array<dooyong_rom_tilemap_device, 2> m_bg;
    optional_device_array<dooyong_rom_tilemap_device, 2> m_fg;
};
```

This causes `m_bg` to find devices with tags `bg1` and `bg2`, while `m_fg` finds devices with tags `fg1` and `fg2`. Note that the indexes into the object finder arrays are still zero-based like any other C array.

It's also possible to other format conversions, like hexadecimal (`%x` and `%X`) or character (`%c`):

```
class eurit_state : public driver_device
{
public:
    eurit_state(machine_config const &mconfig, device_type type, char const *tag) :
        driver_device(mconfig, type, tag),
```

(continues on next page)

(continued from previous page)

```
        m_keys(*this, "KEY%c", 'A')
    {
    }

private:
    required_ioport_array<5> m_keys;
};
```

In this case, the key matrix ports use tags `KEYA`, `KEYB`, `KEYC`, `KEYD` and `KEYE`.

When the tags don't follow a simple ascending sequence, you can supply a brace-enclosed initialiser list of tags:

```
class seabattl_state : public driver_device
{
public:
    seabattl_state(machine_config const &mconfig, device_type type, char const *tag) :
        driver_device(mconfig, type, tag),
        m_digits(*this, { "sc_thousand", "sc_hundred", "sc_half", "sc_unity", "tm_half",
↪"tm_unity" })
    {
    }

private:
    required_device_array<dm9368_device, 6> m_digits;
};
```

If the underlying object finders require additional constructor arguments, supply them after the tag format and index offset (the same values will be used for all elements of the array):

```
class dreamwld_state : public driver_device
{
public:
    dreamwld_state(machine_config const &mconfig, device_type type, char const *tag) :
        driver_device(mconfig, type, tag),
        m_vram(*this, "vram_%u", 0U, 0x2000U, ENDIANNESS_BIG)
    {
    }

private:
    memory_share_array_creator<u16, 2> m_vram;
};
```

This finds or creates memory shares with tags `vram_0` and `vram_1`, each of which is 8 KiB organised as 4,096 big-Endian 16-bit words.

12.4.6 Optional object finders

Optional object finders don't raise an error if the target object isn't found. This is useful in two situations: `driver_device` implementations (state classes) representing a family of systems where some components aren't present in all configurations, and devices that can optionally use a resource. Optional object finders provide additional member functions for testing whether the target object was found.

Optional system components

Often a class is used to represent a family of related systems. If a component isn't present in all configurations, it may be convenient to use an optional object finder to access it. We'll use the Sega X-board device as an example:

```
class segaxbd_state : public device_t
{
protected:
    segaxbd_state(machine_config const &mconfig, device_type type, char const *tag,␣
↪device_t *owner, u32 clock) :
        device_t(mconfig, type, tag, owner, clock),
        m_soundcpu(*this, "soundcpu"),
        m_soundcpu2(*this, "soundcpu2"),
        m_segaic16vid(*this, "segaic16vid"),
        m_pc_0(0),
        m_lastsurv_mux(0),
        m_adc_ports(*this, "ADC%u", 0),
        m_mux_ports(*this, "MUX%u", 0)
    {
    }

    optional_device<z80_device> m_soundcpu;
    optional_device<z80_device> m_soundcpu2;
    required_device<mb3773_device> m_watchdog;
    required_device<segaic16_video_device> m_segaic16vid;
    bool m_adc_reverse[8];
    u8 m_pc_0;
    u8 m_lastsurv_mux;
    optional_ioport_array<8> m_adc_ports;
    optional_ioport_array<4> m_mux_ports;
};
```

The `optional_device` and `optional_ioport_array` members are declared and constructed in the usual way. Before accessing the target object, we call an object finder's `found()` member function to check whether it's present in the system (the explicit cast-to-Boolean operator can be used for the same purpose):

```
void segaxbd_state::pc_0_w(u8 data)
{
    m_pc_0 = data;

    m_watchdog->write_line_ck(BIT(data, 6));

    m_segaic16vid->set_display_enable(data & 0x20);

    if (m_soundcpu.found())
        m_soundcpu->set_input_line(INPUT_LINE_RESET, (data & 0x01) ? CLEAR_LINE : ASSERT_
```

(continues on next page)

(continued from previous page)

```
↪LINE);
    if (m_soundcpu2.found())
        m_soundcpu2->set_input_line(INPUT_LINE_RESET, (data & 0x01) ? CLEAR_LINE :␣
↪ASSERT_LINE);
}
```

Optional I/O ports provide a convenience member function called `read_safe` that reads the port value if present, or returns the supplied default value otherwise:

```
u8 segaxbd_state::analog_r()
{
    int const which = (m_pc_0 >> 2) & 7;
    u8 value = m_adc_ports[which].read_safe(0x10);

    if (m_adc_reverse[which])
        value = 255 - value;

    return value;
}

u8 segaxbd_state::lastsurv_port_r()
{
    return m_mux_ports[m_lastsurv_mux].read_safe(0xff);
}
```

The ADC ports return 0x10 (16 decimal) if they are not present, while the multiplexed digital ports return 0xff (255 decimal) if they are not present. Note that `read_safe` is a member of the `optional_ioport` itself, and not a member of the target `ioport_port` object (the `optional_ioport` is not dereferenced when using it).

There are some disadvantages to using optional object finders:

- There's no way to distinguish between the target not being present, and the target not being found due to mismatched tags, making it more error-prone.
- Checking whether the target is present may use CPU branch prediction resources, potentially hurting performance if it happens very frequently.

Consider whether optional object finders are the best solution, or whether creating a derived class for the system with additional components is more appropriate.

Optional resources

Some devices can optionally use certain resources. If the host system doesn't supply them, the device will still function, although some functionality may not be available. For example, the Virtual Boy cartridge slot responds to three address spaces, called EXP, CHIP and ROM. If the host system will never use one or more of them, it doesn't need to supply a place for the cartridge to install the corresponding handlers. (For example a copier may only use the ROM space.)

Let's look at how this is implemented. The Virtual Boy cartridge slot device declares `optional_address_space` members for the three address spaces, `offs_t` members for the base addresses in these spaces, and inline member functions for configuring them:

```
class vboy_cart_slot_device :
        public device_t,
        public device_image_interface,
```

(continues on next page)

(continued from previous page)

```
        public device_single_card_slot_interface<device_vboy_cart_interface>
{
public:
    vboy_cart_slot_device(machine_config const &mconfig, char const *tag, device_t␣
↪*owner, u32 clock = 0U);

    template <typename T> void set_exp(T &&tag, int no, offs_t base)
    {
        m_exp_space.set_tag(std::forward<T>(tag), no);
        m_exp_base = base;
    }
    template <typename T> void set_chip(T &&tag, int no, offs_t base)
    {
        m_chip_space.set_tag(std::forward<T>(tag), no);
        m_chip_base = base;
    }
    template <typename T> void set_rom(T &&tag, int no, offs_t base)
    {
        m_rom_space.set_tag(std::forward<T>(tag), no);
        m_rom_base = base;
    }

protected:
    virtual void device_start() override;

private:
    optional_address_space m_exp_space;
    optional_address_space m_chip_space;
    optional_address_space m_rom_space;
    offs_t m_exp_base;
    offs_t m_chip_base;
    offs_t m_rom_base;

    device_vboy_cart_interface *m_cart;
};

DECLARE_DEVICE_TYPE(VBOY_CART_SLOT, vboy_cart_slot_device)
```

The object finders are constructed with dummy values for the tags and space numbers (`finder_base::DUMMY_TAG` and -1):

```
vboy_cart_slot_device::vboy_cart_slot_device(machine_config const &mconfig, char const␣
↪*tag, device_t *owner, u32 clock) :
    device_t(mconfig, VBOY_CART_SLOT, tag, owner, clock),
    device_image_interface(mconfig, *this),
    device_single_card_slot_interface<device_vboy_cart_interface>(mconfig, *this),
    m_exp_space(*this, finder_base::DUMMY_TAG, -1, 32),
    m_chip_space(*this, finder_base::DUMMY_TAG, -1, 32),
    m_rom_space(*this, finder_base::DUMMY_TAG, -1, 32),
    m_exp_base(0U),
    m_chip_base(0U),
    m_rom_base(0U),
```

(continues on next page)

(continued from previous page)

```
    m_cart(nullptr)
{
}
```

To help detect configuration errors, we'll check for cases where address spaces have been configured but aren't present:

```
void vboy_cart_slot_device::device_start()
{
    if (!m_exp_space && ((m_exp_space.finder_tag() != finder_base::DUMMY_TAG) || (m_exp_
↪space.spacenum() >= 0)))
        throw emu_fatalerror("%s: Address space %d of device %s not found (EXP)\n",␣
↪tag(), m_exp_space.spacenum(), m_exp_space.finder_tag());

    if (!m_chip_space && ((m_chip_space.finder_tag() != finder_base::DUMMY_TAG) || (m_
↪chip_space.spacenum() >= 0)))
        throw emu_fatalerror("%s: Address space %d of device %s not found (CHIP)\n",␣
↪tag(), m_chip_space.spacenum(), m_chip_space.finder_tag());

    if (!m_rom_space && ((m_rom_space.finder_tag() != finder_base::DUMMY_TAG) || (m_rom_
↪space.spacenum() >= 0)))
        throw emu_fatalerror("%s: Address space %d of device %s not found (ROM)\n",␣
↪tag(), m_rom_space.spacenum(), m_rom_space.finder_tag());

    m_cart = get_card_device();
}
```

12.4.7 Object finder types in more detail

All object finders provide configuration functionality:

```
char const *finder_tag() const { return m_tag; }
std::pair<device_t &, char const *> finder_target();
void set_tag(device_t &base, char const *tag);
void set_tag(char const *tag);
void set_tag(finder_base const &finder);
```

The `finder_tag` and `finder_target` member function provides access to the currently configured target. Note that the tag returned by `finder` tag is relative to the base device. It is not sufficient on its own to identify the target.

The `set_tag` member functions configure the target of the object finder. These members must not be called after the object finder is resolved. The first form configures the base device and relative tag. The second form sets the relative tag and also implicitly sets the base device to the device that is currently being configured. This form must only be called from machine configuration functions. The third form sets the base object and relative tag to the current target of another object finder.

Note that the `set_tag` member functions **do not** copy the relative tag. It is the caller's responsibility to ensure the C string remains valid until the object finder is resolved (or reconfigured with a different tag). The base device must also be valid at resolution time. This may not be the case if the device could be removed or replaced later.

All object finders provide the same interface for accessing the target object:

```
ObjectClass *target() const;
operator ObjectClass *() const;
ObjectClass *operator->() const;
```

These members all provide access to the target object. The `target` member function and cast-to-pointer operator will return `nullptr` if the target has not been found. The pointer member access operator asserts that the target has been found.

Optional object finders additionally provide members for testing whether the target object has been found:

```
bool found() const;
explicit operator bool() const;
```

These members return `true` if the target was found, on the assumption that the target pointer will be non-null if the target was found.

Device finders

Device finders require one template argument for the expected device class. This should derive from either `device_t` or `device_interface`. The target device object must either be an instance of this class, an instance of a class that derives from it. A warning message is logged if a matching device is found but it is not an instance of the expected class.

Device finders provide an additional `set_tag` overload:

```
set_tag(DeviceClass &object);
```

This is equivalent to calling `set_tag(object, DEVICE_SELF)`. Note that the device object must not be removed or replaced before the object finder is resolved.

Memory system object finders

The memory system object finders, `required_memory_region`, `optional_memory_region`, `required_memory_bank`, `optional_memory_bank` and `memory_bank_creator`, do not have any special functionality. They are often used in place of literal tags when installing memory banks in an address space.

Example using memory bank finders in an address map:

```
class qvt70_state : public driver_device
{
public:
    qvt70_state(machine_config const &mconfig, device_type type, char const *tag) :
        driver_device(mconfig, type, tag),
        m_rombank(*this, "rom"),
        m_rambank(*this, "ram%d", 0U),
    { }

private:
    required_memory_bank m_rombank;
    required_memory_bank_array<2> m_rambank;

    void mem_map(address_map &map);
```

(continues on next page)

(continued from previous page)

```
    void rombank_w(u8 data);
};

void qvt70_state::mem_map(address_map &map)
{
    map(0x0000, 0x7fff).bankr(m_rombank);
    map(0x8000, 0x8000).w(FUNC(qvt70_state::rombank_w));
    map(0xa000, 0xbfff).ram();
    map(0xc000, 0xdfff).bankrw(m_rambank[0]);
    map(0xe000, 0xffff).bankrw(m_rambank[1]);
}
```

Example using a memory bank creator to install a memory bank dynamically:

```
class vegaeo_state : public eolith_state
{
public:
    vegaeo_state(machine_config const &mconfig, device_type type, char const *tag) :
        eolith_state(mconfig, type, tag),
        m_qs1000_bank(*this, "qs1000_bank")
    {
    }

    void init_vegaeo();

private:
    memory_bank_creator m_qs1000_bank;
};

void vegaeo_state::init_vegaeo()
{
    // Set up the QS1000 program ROM banking, taking care not to overlap the internal RAM
    m_qs1000->cpu().space(AS_IO).install_read_bank(0x0100, 0xffff, m_qs1000_bank);
    m_qs1000_bank->configure_entries(0, 8, memregion("qs1000:cpu")->base() + 0x100,␣
↪0x10000);

    init_speedup();
}
```

I/O port finders

Optional I/O port finders provide an additional convenience member function:

```
ioport_value read_safe(ioport_value defval);
```

This will read the port's value if the target I/O port was found, or return `defval` otherwise. It is useful in situations where certain input devices are not always present.

Address space finders

Address space finders accept an additional argument for the address space number to find. A required data width can optionally be supplied to the constructor.

```
address_space_finder(device_t &base, char const *tag, int spacenum, u8 width = 0);
void set_tag(device_t &base, char const *tag, int spacenum);
void set_tag(char const *tag, int spacenum);
void set_tag(finder_base const &finder, int spacenum);
template <bool R> void set_tag(address_space_finder<R> const &finder);
```

The base device and tag must identify a device that implements `device_memory_interface`. The address space number is a zero-based index to one of the device's address spaces.

If the width is non-zero, it must match the target address space's data width in bits. If the target address space exists but has a different data width, a warning message will be logged, and it will be treated as not being found. If the width is zero (the default argument value), the target address space's data width won't be checked.

Member functions are also provided to get the configured address space number and set the required data width:

```
int spacenum() const;
void set_data_width(u8 width);
```

Memory pointer finders

The memory pointer finders, `required_region_ptr`, `optional_region_ptr`, `required_shared_ptr`, `optional_shared_ptr` and `memory_share_creator`, all require one template argument for the element type of the memory area. This should usually be an explicitly-sized unsigned integer type (`u8`, `u16`, `u32` or `u64`). The size of this type is compared to the width of the memory area. If it doesn't match, a warning message is logged and the region or share is treated as not being found.

The memory pointer finders provide an array access operator, and members for accessing the size of the memory area:

```
PointerType &operator[](int index) const;
size_t length() const;
size_t bytes() const;
```

The array access operator returns a non-`const` reference to an element of the memory area. The index is in units of the element type; it must be non-negative and less than the length of the memory area. The `length` member returns the number of elements in the memory area. The `bytes` member returns the size of the memory area in bytes. These members should not be called if the target region/share has not been found.

The `memory_share_creator` requires additional constructor arguments for the size and Endianness of the memory share:

```
memory_share_creator(device_t &base, char const *tag, size_t bytes, endianness_t␣
↪endianness);
```

The size is specified in bytes. If an existing memory share is found, it is an error if its size does not match the specified size. If the width is wider than eight bits and an existing memory share is found, it is an error if its Endianness does not match the specified Endianness.

The `memory_share_creator` provides additional members for accessing properties of the memory share:

```
endianness_t endianness() const;
u8 bitwidth() const;
u8 bytewidth() const;
```

These members return the Endianness, width in bits and width in bytes of the memory share, respectively. They must not be called if the memory share has not been found.

12.4.8 Output finders

Output finders are used for exposing outputs that can be used by the artwork system, or by external programs. A common application using an external program is a control panel or cabinet lighting controller.

Output finders are not really object finders, but they're described here because they're used in a similar way. There are a number of important differences to be aware of:

- Output finders always create outputs if they do not exist.
- Output finders are array-like, and support an arbitrary number of dimensions.

Output finders take a variable number of template arguments corresponding to the number of array dimensions you want. Let's look at an example that uses zero-, one- and two-dimensional output finders:

```
class mmd2_state : public driver_device
{
public:
    mmd2_state(machine_config const &mconfig, device_type type, char const *tag) :
        driver_device(mconfig, type, tag),
        m_digits(*this, "digit%u", 0U),
        m_p(*this, "p%u_%u", 0U, 0U),
        m_led_halt(*this, "led_halt"),
        m_led_hold(*this, "led_hold")
    { }

protected:
    virtual void machine_start() override;

private:
    void round_leds_w(offs_t, u8);
    void digit_w(u8 data);
    void status_callback(u8 data);

    u8 m_digit;

    output_finder<9> m_digits;
    output_finder<3, 8> m_p;
    output_finder<> m_led_halt;
    output_finder<> m_led_hold;
};
```

The `m_led_halt` and `m_led_hold` members are zero-dimensional output finders. They find a single output each. The `m_digits` member is a one-dimensional output finder. It finds nine outputs organised as a single-dimensional array. The `m_p` member is a two-dimensional output finder. It finds 24 outputs organised as three rows of eight columns each. Larger numbers of dimensions are supported.

The output finder constructor takes a base device reference, a format string, and an index offset for each dimension. In this case, all the offsets are zero. The one-dimensional output finder `m_digits` will find outputs `digit0`, `digit1`, `digit2`, … `digit8`. The two-dimensional output finder `m_p` will find the outputs `p0_0`, `p0_1`, … `p0_7` for the first row, `p1_0`, `p1_1`, … `p1_7` for the second row, and `p2_0`, `p2_1`, … `p2_7` for the third row.

The output names may also be specified by providing an array of strings with the same dimensions as the output finder (i.e. a one-dimensional output finder requires a one-dimensional array of strings, a two-dimensional output finder requires a two-dimensional array of strings,

The names may be changed during machine configuration by calling `set_name` for a zero-dimensional output finder or `set_names` for any other kind of output finder. Names are specified the same way as during construction, i.e. as a format string and an index offset for each dimension, or as an array of strings with the same dimensions as the output finder. Names will be interpreted relative to the device currently being configured.

Note that output finders *do not* copy name format strings or arrays of output names. The caller must ensure that the name format string or array of names remains valid until after validation and/or object resolution is complete.

Output finders provide operators allowing them to be assigned from or cast to 32-bit signed integers. The assignment operator will send a notification if the new value is different to the output's current value.

```
operator s32() const;
s32 operator=(s32 value);
```

To set output values, assign through the output finders, as you would with an array of the same rank:

```
void mmd2_state::round_leds_w(offs_t offset, u8 data)
{
    for (u8 i = 0; i < 8; i++)
        m_p[offset][i] = BIT(~data, i);
}

void mmd2_state::digit_w(u8 data)
{
    if (m_digit < 9)
        m_digits[m_digit] = data;
}

void mmd2_state::status_callback(u8 data)
{
    m_led_halt = (~data & i8080_cpu_device::STATUS_HLTA) ? 1 : 0;
    m_led_hold = (data & i8080_cpu_device::STATUS_WO) ? 1 : 0;
}
```

12.5 Input System

- *Introduction*
- *Components*
  - *Input device*
  - *Input device item*
  - *I/O port field*

  - *Input manager*
  - *I/O port manager*
- *Structures and data types*
  - *Input code*
  - *Input sequence*
- *Input provider modules*
- *Player positions*
- *Updating I/O port fields*
  - *Updating digital fields*
  - *Updating absolute analog fields*
  - *Updating relative analog fields*

12.5.1 Introduction

The variety of systems MAME emulates, as well as the variation in host systems and peripherals, necessitates a flexible, configurable input system.

Note that the input system is concerned with low-level user input. High-level user interaction, involving things like text input and pointing devices, is handled separately.

12.5.2 Components

From the emulated system's point of view, the input system has the following conceptual components.

Input device

Input devices supply input values. An input device typically corresponds to a physical device in the host system, for example a keyboard, mouse or game controller. However, there isn't always a one-to-one correspondence between input devices and physical devices. For example the SDL keyboard provider module aggregates all keyboards into a single input device, and the Win32 lightgun provider module can present two input devices using input from a single mouse.

Input devices are identified by their device class (keyboard, mouse, joystick or lightgun) and device number within the class. Input provider modules can also supply an implementation-dependent identifier to allow the user to configure stable device numbering.

Note that input devices are unrelated to emulated devices (`device_t` implementations) despite the similar name.

Input device item

Also known as a **control**, and input device item corresponds to a input source that produces a single value. This usually corresponds to a physical control or sensor, for example a joystick axis, a button or an accelerometer.

MAME supports three kinds of controls: **switches**, **absolute axes** and **relative axes**:

- Switches produce the value 0 when inactive (released or off) or 1 when active (pressed or on).
- Absolute axes produce a value normalised to the range -65,536 to 65,536 with zero corresponding to the neutral position.
- Relative axes produce a value corresponding to the movement since the previous input update. Mouse-like devices scale values to approximately 512 per nominal 100 DPI pixel.

Negative axis values should correspond to directions up, to the left, away from the player, or anti-clockwise. For single-ended axes (e.g. pedals or displacement-sensitive triggers and buttons), only zero and the negative portion of the range should be used.

Switches are used to represent controls that naturally have two distinct states, like buttons and toggle switches.

Absolute axes are used to represent controls with a definite range and/or neutral position. Examples include steering wheels with limit stops, joystick axes, and displacement-sensitive triggers.

Relative axes are used to represent controls with an effectively infinite range. Examples include mouse/trackball axes, incremental encoder dials, and gyroscopes.

Accelerometers and force sensing joystick axes should be represented as absolute axes, even though the range is theoretically open-ended. In practice, there is a limit to the range the transducers can report, which is usually substantially larger than needed for normal operation.

Input device items are identified by their associated device's class and device number along with an **input item ID**. MAME supplies item IDs for common types of controls. Additional controls or controls that do not correspond to a common type are dynamically assigned item IDs. MAME supports hundreds to items per input device.

I/O port field

An I/O port field represents an input source in an emulated device or system. Most types of I/O port fields can be assigned one or more combinations of controls, allowing the user to control the input to the emulated system.

Similarly to input device items, there are multiple types of I/O port fields:

- **Digital fields** function as switches that produce one of two distinct values. They are used for keyboard keys, eight-way joystick direction switches, toggle switches, photointerruptors and other emulated inputs that function as two-position switches.
- **Absolute analog fields** have a range with defined minimum, maximum and neutral positions. They are used for analog joystick axes, displacement-sensitive pedals, paddle knobs, and other emulated inputs with a defined range.
- **Relative analog fields** have a range with defined minimum, maximum and starting positions. On each update, the value accumulates and wraps when it passes either end of the range. Functionally, this is like the output of an up/down counter connected to an incremental encoder. They are used for mouse/trackball axes, steering wheels without limit stops, and other emulated inputs that have no range limits.
- DIP switch, configuration and adjuster fields allow the user to set the value through MAME's user interface.
- Additional special field types are used to produce fixed or programmatically generated values.

A digital field appears to the user as a single assignable input, which accepts switch values.

An analog field appears to the user as three assignable inputs: an **axis input**, which accepts axis values; and an **increment input** and a **decrement input** which accept switch values.

Input manager

The input manager has several responsibilities:

- Tracking the available input devices in the system.
- Reading input values.
- Converting between internal identifier values, configuration token strings and display strings.

In practice, emulated devices and systems rarely interact with the input manager directly. The most common reason to access the input manager is implementing special debug controls, which should be disabled in release builds. Plugins that respond to input need to call the input manager to read inputs.

I/O port manager

The I/O port manager's primary responsibilities include:

- Managing assignments of controls to I/O port fields and user interface actions.
- Reading input values via the input manager and updating I/O port field values.

Like the input manager, the I/O port manager is largely transparent to emulated devices and systems. You just need to set up your I/O ports and fields, and the I/O port manager handles the rest.

12.5.3 Structures and data types

The following data types are used for dealing with input.

Input code

An input code specifies an input device item and how it should be interpreted. It is a tuple consisting of the following values: **device class**, **device number**, **item class**, **item modifier** and **item ID**:

- The device class, device number and item ID together identify the input device item to read.
- The item class specifies the type of output value desired: switch, absolute axis or relative axis. Axis values can be converted to switch values by specifying an appropriate modifier.
- The modifier specifies how a value should be interpreted. Valid options depend on the type of input device item and the specified item class.

If the specified input item is a switch, it can only be read using the switch class, and no modifiers are supported. Attempting to read a switch as an absolute or relative axis always returns zero.

If the specified input item is an absolute axis, it can be read as an absolute axis or as a switch:

- Reading an absolute axis item as an absolute axis returns the current state of the control, potentially transformed if a modifier is specified. Supported modifiers are **reverse** to reverse the range of the control, **positive** to map the positive range of the control onto the output (zero corresponding to -65,536 and 65,536 corresponding to 65,536), and **negative** to map the negative range of the control onto the output (zero corresponding to -65,536 and -65,536 corresponding to 65,536).

- Reading an absolute axis item as a switch returns zero or 1 depending on whether the control is past a threshold in the direction specified by the modifier. Use the **negative** modifier to return 1 when the control is beyond the threshold in the negative direction (up or left), or the **positive** modifier to return 1 when the control is beyond the threshold in the positive direction (down or right). There are two special pairs of modifiers, **left/right** and **up/down** that are only applicable to the primary X/Y axes of joystick devices. The user can specify a *joystick map* to control how these modifiers interpret joystick movement.
- Attempting to read an absolute axis item as a relative axis always returns zero.

If the specified input item is a relative axis, it can be read as a relative axis or as a switch:

- Reading a relative axis item as a relative axis returns the change in value since the last input update. The only supported modifier is **reverse**, which negates the value, reversing the direction.
- Reading a relative axis as a switch returns 1 if the control moved in the direction specified by the modifier since the last input update. Use the **negative/left/up** modifiers to return 1 when the control has been moved in the negative direction (up or left), or the **positive/right/down** modifiers to return 1 when the control has moved in the positive direction (down or right).
- Attempting to read a relative axis item as an absolute axis always returns zero.

There are also special input codes used for specifying how multiple controls are to be combined in an input sequence.

The most common place you'll encounter input codes in device and system driver code is when specifying initial assignments for I/O port fields that don't have default assignments supplied by the core. The `PORT_CODE` macro is used for this purpose.

MAME provides macros and helper functions for producing commonly used input codes, including standard keyboard keys and mouse/joystick/lightgun axes and buttons.

Input sequence

An input sequence specifies a combination controls that can be assigned to an input. The name refers to the fact that it is implemented as a sequence container with input codes as elements. It is somewhat misleading, as input sequences are interpreted using instantaneous control values. Input sequences are interpreted differently for switch and axis input.

Input sequences for switch input must only contain input codes with the item class set to switch along with the special **or** and **not** input codes. The input sequence is interpreted using sum-of-products logic. A **not** code causes the value returned by the immediately following code to be inverted. The conjunction of values returned by successive codes is evaluated until an **or** code is encountered. If the current value is 1 when an **or** code is encountered it is returned, otherwise evaluation continues.

Input sequences for axis input can contain input codes with the item class set to switch, absolute axis or relative axis along with the special **or** and **not** codes. It's helpful to think of the input sequence as containing one or more groups of input codes separated by **or** codes:

- A **not** code causes the value returned by an immediately following switch code to be inverted. It has no effect on absolute or relative axis codes.
- Within a group, the conjunction of the values returned by switch codes is evaluated. If it is zero, the group is ignored.
- Within a group, multiple axis values of the same type are summed. Values returned by absolute axis codes are summed, and values returned by relative axis codes are summed.
- If any absolute axis code in a group returns a non-zero value, the sum of relative axes in the group is ignored. Any non-zero absolute axis value takes precedence over relative axis values.
- The same logic is applied when combining group values: group values produced from the same axis type are summed, and values produced from absolute axes take precedence over values produced from relative axes.

- After the group values are summed, if the value was produced from absolute axes it is clamped to the range -65,536 to 65,536 (values produced from relative axes are not clamped).

Emulation code rarely needs to deal with input sequences directly, as they're handled internally between the I/O port manager and input manager. The input manager also converts input sequences to and from the token strings stored in configuration files and produces text for displaying input sequences to users.

Plugins with controls or hotkeys need to use input sequences to allow configuration. Utility classes are provided to allow input sequences to be entered by the user in a consistent way, and the input manager can be used for conversions to and from configuration and display strings. It is very rare to need to directly manipulate input sequences.

12.5.4 Input provider modules

Input provider modules are part of the OS-dependent layer (OSD), and are not directly exposed to emulation and user interface code. Input provider modules are responsible for detecting available host input devices, setting up input devices for the input manager, and providing callbacks to read the current state of input device items. Input provider modules may also provide additional default input assignments suitable for host input devices that are present.

The user is given a choice of input modules to use. One input provider module is used for each of the four input device classes (keyboard, mouse, joystick and lightgun). The available modules depend on the host operating system and OSD implementation. Different modules may use different APIs, support different kinds of devices, or present devices in different ways.

12.5.5 Player positions

MAME uses a concept called *player positions* to help manage input assignments. The number of player positions supported depends on the I/O port field type:

- Ten player positions are supported for common game inputs, including joystick, pedal, paddle, dial, trackball, lightgun and mouse.
- Four player positions are supported for mahjong and hanafuda inputs.
- One player position is supported for gambling system inputs.
- Other inputs do not use player positions. This includes coin slots, arcade start buttons, tilt switches, service switches and keyboard/keypad keys.

The user can configure default input assignments per player position for supported I/O port field types which are saved in the file **default.cfg**. These assignments are used for all systems unless the device/system driver supplies its own default assignments, or the user configures system-specific input assignments.

In order to facilitate development of reusable emulated devices with inputs, particularly slot devices, the I/O port manager automatically renumbers player positions when setting up the emulated system:

- The I/O port manager starts at player position 1 and begins iterating the emulated device tree in depth first order, starting from the root device.
- If a device has I/O port fields that support player positions, they are renumbered to start from the I/O port manager's current player position.
- Before advancing to the next device, the I/O port manager sets its current player position to the last seen player position plus one.

For a simple example, consider what happens when you run a Sega Mega Drive console with two game pads connected:

- The I/O port manager starts at player position 1 at the root device.

- The first device encountered with I/O port fields that support player positions is the first game pad. The inputs are renumbered to start at player position 1. This has no visible effect, as the I/O port fields are initially numbered starting at player position 1.
- Before moving to the next device, the I/O port manager sets its current player position to 2 (the last player position seen plus one).
- The next device encountered with I/O port fields that support player positions is the second game pad. The inputs are renumbered to start at player position 2. This avoids I/O port field type conflicts with the first game pad.
- Before moving to the next device, the I/O port manager sets its current player position to 3 (the last player position seen plus one).
- No more devices with I/O port fields that support player positions are encountered.

12.5.6 Updating I/O port fields

The I/O port manager updates I/O port fields once for each video frame produced by the first emulated screen in the system. How a field is updated depends on whether it is a digital or analog field.

Updating digital fields

Updating digital I/O port fields is simple:

- The I/O port manager reads the current value for the field's assigned input sequence (via the input manager).
- If the value is zero, the field's default value is set.
- If the value is non-zero, the binary complement of the field's default value is set.

Updating absolute analog fields

Updating absolute analog I/O port fields is more complex due to the need to support a variety of control setups:

- The I/O port manager reads the current value for the field's assigned axis input sequence (via the input manager).
- If the current value changed since the last update and the input device item that produced the current value was an absolute axis, the field's value is set to the current value scaled to the correct range, and no further processing is performed.
- If the current value is non-zero and the input device item that produced the current value was a relative axis, the current value is added to the field's value, scaled by the field's sensitivity setting.
- The I/O port manager reads the current value for the field's assigned increment input sequence (via the input manager); if this value is non-zero, the field's increment/decrement speed setting value is added to its value, scaled by its sensitivity setting.
- The I/O port manager reads the current value for the field's assigned decrement input sequence (via the input manager); if this value is non-zero, the field's increment/decrement speed setting value is subtracted from its value, scaled by its sensitivity setting.
- If the current axis input, increment input and decrement input values are all zero, but either or both of the increment input and decrement input values were non-zero the last time the field's value changed in response to user input, the field's auto-centring speed setting value is added to or subtracted from its value to move it toward its default value.

Note that the sensitivity setting value for absolute analog fields affects the response to relative axis input device items and increment/decrement inputs, but it does not affect the response to absolute axis input device items or the auto-centring speed.

Updating relative analog fields

Relative analog I/O port fields also need special handling to cater for multiple control setups, but they are a little simpler than absolute analog fields:

- The I/O port manager reads the current value for the field's assigned axis input sequence (via the input manager).
- If the current value is non-zero and the input device item that produced the current value was an absolute axis, the current value is added to the field's value, scaled by the field's sensitivity setting, and no further processing is performed.
- If the current value is non-zero and the input device item that produced the current value was a relative axis, the current value is added to the field's value, scaled by the field's sensitivity setting.
- The I/O port manager reads the current value for the field's assigned increment input sequence (via the input manager); if this value is non-zero, the field's increment/decrement speed setting value is added to its value, scaled by its sensitivity setting.
- The I/O port manager reads the current value for the field's assigned decrement input sequence (via the input manager); if this value is non-zero, the field's increment/decrement speed setting value is subtracted from its value, scaled by its sensitivity setting.

Note that the sensitivity setting value for relative analog fields affects the response to all user input.

12.6 The device_memory_interface

- *1. Capabilities*
- *2. Setup*
- *3. Associating maps to spaces*
- *4. Accessing the spaces*
- *5. MMU support for disassembler*

12.6.1 1. Capabilities

The device memory interface provides devices with the capability of creating address spaces, to which address maps can be associated. It's used for any device that provides a (logical) address/data bus that other devices can be connected to. That's mainly, but not solely, CPUs.

The interface allows for an unlimited set of address spaces, numbered with small, non-negative values. The IDs index vectors, so they should stay small to keep the lookup fast. Spaces numbered 0-3 have associated constant name:

ID	Name
0	AS_PROGRAM
1	AS_DATA
2	AS_IO
3	AS_OPCODES

Spaces 0 and 3, i.e. `AS_PROGRAM` and `AS_OPCODES`, are special for the debugger and some CPUs. `AS_PROGRAM` is use by the debugger and the CPUs as the space from which the CPU reads its instructions for the disassembler. When present,

AS_OPCODES is used by the debugger and some CPUs to read the opcode part of the instruction. What opcode means is device-dependant, for instance for the Z80 it's the initial byte(s) which are read with the M1 signal asserted, while for the 68000 is means every instruction word plus PC-relative accesses. The main, but not only, use of AS_OPCODES is to implement hardware decryption of instructions separately from data.

12.6.2 2. Setup

```
std::vector<std::pair<int, const address_space_config *>> memory_space_config() const;
```

The device must override that method to provide a vector of pairs comprising of a space number and an associated address_space_config describing its configuration. Some examples to look up when needed:

- Standard two-space vector: v60_device
- Conditional AS_OPCODES: z80_device
- Inherit configuration and add a space: hd647180x_device
- Inherit configuration and modify a space: tmpz84c011_device

```
bool has_configured_map(int index = 0) const;
```

The has_configured_map method allows to test whether an address_map has been associated with a given space in the memory_space_config method. That allows optional memory spaces to be implemented, such as AS_OPCODES in certain CPU cores.

12.6.3 3. Associating maps to spaces

Associating maps to spaces is done at the machine configuration level, after the device is instantiated.

```
void set_addrmap(int spacenum, T &obj, Ret (U::*func)(Params...));
void set_addrmap(int spacenum, Ret (T::*func)(Params...));
void set_addrmap(int spacenum, address_map_constructor map);
```

These function associate a map with a given space. Address maps associated with non-existent spaces are ignored (no warning given). The first form takes a reference to an object and a method to call on that object. The second form takes a method to call on the current device being configured. The third form takes an address_map_constructor to copy. In each case, the function must be callable with reference to an address_map object as an argument.

To remove a previously configured address map, call set_addrmap with a default-constructed address_map_constructor (useful for removing a map for an optional space in a derived machine configuration).

As an example, here's the address map configuration for the main CPU in the Hana Yayoi and Hana Fubuki machines, with all distractions removed:

```
class hnayayoi_state : public driver_device
{
public:
    void hnayayoi(machine_config &config);
    void hnfubuki(machine_config &config);

private:
    required_device<cpu_device> m_maincpu;
```

(continues on next page)

(continued from previous page)

```
    void hnayayoi_map(address_map &map);
    void hnayayoi_io_map(address_map &map);
    void hnfubuki_map(address_map &map);
};

void hnayayoi_state::hnayayoi(machine_config &config)
{
    Z80(config, m_maincpu, 20000000/4);
    m_maincpu->set_addrmap(AS_PROGRAM, &hnayayoi_state::hnayayoi_map);
    m_maincpu->set_addrmap(AS_IO, &hnayayoi_state::hnayayoi_io_map);
}

void hnayayoi_state::hnfubuki(machine_config &config)
{
    hnayayoi(config);

    m_maincpu->set_addrmap(AS_PROGRAM, &hnayayoi_state::hnfubuki_map);
    m_maincpu->set_addrmap(AS_IO, address_map_constructor());
}
```

12.6.4 4. Accessing the spaces

```
address_space &space(int index = 0) const;
```

Returns the specified address space post-initialization. The specified address space must exist.

```
bool has_space(int index = 0) const;
```

Indicates whether a given space actually exists.

12.6.5 5. MMU support for disassembler

```
bool translate(int spacenum, int intention, offs_t &address, address_space *&target_
↪space);
```

Does a logical to physical address translation through the device's MMU. spacenum gives the space number, intention for the type of the future access (`TR_(READ\|WRITE\|FETCH)`), address is an in/out parameter holding the address to translate on entry and the translated version on return, and finally target_space is the actual space the access would end up in, which may be in a different device. Should return `true` if the translation went correctly, or `false` if the address is unmapped. The call must not change the state of the device.

Note that for some historical reason, the device itself must override the virtual method `memory_translate` with the same signature.

12.7 The device_rom_interface

- *1. Capabilities*
- *2. Setup*
- *3. ROM access*
- *4. ROM banking*
- *5. Caveats*

12.7.1 1. Capabilities

This interface is designed for devices that expect to have a ROM connected to them on a dedicated bus. It's mostly designed for sound chips. Other devices types may be interested but other considerations may make it impractical (graphics decode caching, for instance). The interface provides the capability to connect a ROM region, connect an address map, or dynamically set up a block of memory as ROM. In the region/memory block cases, banking is handled automatically.

12.7.2 2. Setup

```
device_rom_interface<AddrWidth, DataWidth=0, AddrShift=0, Endian=ENDIANNESS_LITTLE>
```

The interface is a template that takes the address width of the dedicated bus as a parameter. In addition the data bus width (if not byte), address shift (if non-zero) and Endianness (if not little Endian or byte-sized bus) can be provided. Data bus width is 0 for byte, 1 for word, etc.

```
void set_map(map);
```

Use that method at machine configuration time to provide an address map for the bus to connect to. It has priority over a ROM region if one is also present.

```
void set_device_rom_tag(tag);
```

Used to specify a ROM region to use if a device address map is not given. Defaults to `DEVICE_SELF`, i.e. the device's tag.

```
ROM_REGION(length, tag, flags)
```

If a ROM region with the tag specified using `set_device_rom_tag` if present, or identical to the device tag otherwise, is provided in the ROM definitions for the system, it will be automatically picked up as the connected ROM. An address map has priority over the region if present in the machine configuration.

```
void override_address_width(u8 width);
```

This method allows the address bus width to be overridden. It must be called from within the device before **config_complete** time.

```
void set_rom(const void *base, u32 size);
```

At any time post-`interface_pre_start`, a memory block can be set up as the connected ROM with that method. It overrides any previous setup that may have been provided. It can be done multiple times.

12.7.3 3. ROM access

```
u8 read_byte(offs_t addr);
u16 read_word(offs_t addr);
u32 read_dword(offs_t addr);
u64 read_qword(offs_t addr);
```

These methods provide read access to the connected ROM. Out-of-bounds access results in standard unmapped read `logerror` messages.

12.7.4 4. ROM banking

If the ROM region or the memory block in `set_rom` is larger than the address bus can access, banking is automatically set up.

```
void set_rom_bank(int bank);
```

That method selects the current bank number.

12.7.5 5. Caveats

Using that interface makes the device derive from `device_memory_interface`. If the device wants to actually use the memory interface for itself, remember that space zero (0, or `AS_PROGRAM`) is used by the ROM interface, and don't forget to call the base `memory_space_config` method.

For devices which have outputs that can be used to address ROMs but only to forward the data to another device for processing, it may be helpful to disable the interface when it is not required. This can be done by overriding `memory_space_config` to return an empty vector.

12.8 The device_disasm_interface and the disassemblers

12.8.1 1. Capabilities

The disassemblers are classes that provide disassembly and opcode meta-information for the cpu cores and **unidasm**. The **device_disasm_interface** connects a cpu core with its disassembler.

12.8.2 2. The disassemblers

2.1. Definition

A disassembler is a class that derives from **util::disasm_interface**. It then has two required methods to implement, **opcode_alignment** and **disassemble**, and 6 optional, **interface_flags**, **page_address_bits**, **pc_linear_to_real**, **pc_real_to_linear**, and one with four possible variants, **decrypt8/16/32/64**.

2.2. opcode_alignment

u32 **opcode_alignment**() const

Returns the required alignment of opcodes by the cpu, in PC-units. In other words, the required alignment for the PC register of the cpu. Tends to be 1 (almost everything), 2 (68000...), 4 (mips, ppc...), which an exceptional 8 (tms 32082 parallel processor) and 16 (tms32010, instructions are 16-bits aligned and the PC targets bits). It must be a power-of-two or things will break.

Note that processors like the tms32031 which have 32-bits instructions but where the PC targets 32-bits values have an alignment of 1.

2.3. disassemble

offs_t **disassemble**(std::ostream &stream, offs_t pc, const data_buffer &opcodes, const data_buffer &params)

This is the method where the real work is done. This method must disassemble the instruction at address *pc* and write the result to *stream*. The values to decode are retrieved from the *opcode* buffer. A **data_buffer** object offers four accessor methods:

u8 util::disasm_interface::data_buffer::**r8** (offs_t pc) const
u16 util::disasm_interface::data_buffer::**r16**(offs_t pc) const
u32 util::disasm_interface::data_buffer::**r32**(offs_t pc) const
u64 util::disasm_interface::data_buffer::**r64**(offs_t pc) const

They read the data at a given address and take endianness and nonlinear PCs for larger-than-bus-width accesses. The debugger variant also caches the read data in one block, so for that reason one should not read data too far from the base pc (e.g. stay within 16K or so, careful when trying to follow indirect accesses).

A number of CPUs have an external signal that splits fetches into an opcode part and a parameter part. This is for instance the M1 signal of the z80 or the SYNC signal of the 6502. Some systems present different values to the cpu depending on whether that signal is active, usually for protection purposes. On these cpus the opcode part should be read from the *opcode* buffer, and the parameter part from the *params* buffer. They will or will not be the same buffer depending on the system itself.

The method returns the size of the instruction in PC units, with a maximum of 65535. In addition, if possible, the disassembler should give some meta-information about the opcode by OR-ing in into the result:

- **STEP_OVER** for subroutine calls or auto-decrementing loops. If there is some delay slots, also OR with **step_over_extra**(n) where n is the number of instruction slots.
- **STEP_OUT** for the return-from-subroutine instructions

In addition, to indicated that these flags are supported, OR the result with **SUPPORTED**. An annoying number of disassemblers lies about that support (e.g. they do a or with **SUPPORTED** without even generating the **STEP_OVER** or **STEP_OUT** information). Don't do that, it breaks the step over/step out functionality of the debugger.

2.4. interface_flags

u32 **interface_flags**() const

That optional method indicates specifics of the disassembler. Default of zero is correct most of the time. Possible flags, which need to be OR-ed together, are:

- **NONLINEAR_PC**: stepping to the next opcode or the next byte of the opcode is not adding one to pc. Used for old LFSR-based PCs.
- **PAGED**: PC wraps at a page boundary
- **PAGED2LEVEL**: not only PC wraps at some kind of page boundary, but there are two levels of paging
- **INTERNAL_DECRYPTION**: there is some decryption tucked between reading from AS_PROGRAM and the actual disassembler
- **SPLIT_DECRYPTION**: there is some decryption tucked between reading from AS_PROGRAM and the actual disassembler, and that decryption is different for opcodes and parameters

Note that in practice non-linear pc systems are also paged, that **PAGED2LEVEL** implies **PAGED**, and that **SPLIT_DECRYPTION** implies **DECRYPTION**.

2.5. pc_linear_to_real and pc_real_to_linear

offs_t **pc_linear_to_real**(offs_t pc) const
offs_t **pc_real_to_linear**(offs_t pc) const

These methods should be present only when **NONLINEAR_PC** is set in the interface flags. They must convert pc to and from a value to a linear domain where the instruction parameters and next instruction are reached by incrementing the value. **pc_real_to_linear** converts to that domain, **pc_linear_to_real** converts back from that domain.

2.6. page_address_bits

u32 **page_address_bits**() const

Present on when **PAGED** or **PAGED2LEVEL** is set, gives the number of address bits in the lowest page.

2.7. page2_address_bits

u32 **page2_address_bits**() const

Present on when **PAGED2LEVEL** is set, gives the number of address bits in the upper page.

2.8. decryptnn

u8 **decrypt8** (u8 value, offs_t pc, bool opcode) const
u16 **decrypt16**(u16 value, offs_t pc, bool opcode) const
u32 **decrypt32**(u32 value, offs_t pc, bool opcode) const
u64 **decrypt64**(u64 value, offs_t pc, bool opcode) const

One of these must be defined when **INTERNAL_DECRYPTION** or **SPLIT_DECRYPTION** is set. The chosen one is the one which takes what **opcode_alignment** represents in bytes.

That method decrypts a given value read from address pc (from AS_PROGRAM) and gives the result which will be passed to the disassembler. In the split decryption case, opcode indicates whether we're in the opcode (true) or parameter (false) part of the instruction.

12.8.3 3. Disassembler interface, device_disasm_interface

3.1. Definition

A CPU core derives from **device_disasm_interface** through **cpu_device**. One method has to be implemented, **create_disassembler**.

3.2. create_disassembler

util::disasm_interface ***create_disassembler**()

That method must return a pointer to a newly allocated disassembler object. The caller takes ownership and handles the lifetime.

This method will be called at most one in the lifetime of the cpu object.

12.8.4 4. Disassembler configuration and communication

Some disassemblers need to be configured. Configuration can be unchanging (static) for the duration of the run (cpu model type for instance) or dynamic (state of a flag or a user preference). Static configuration can be done through either (a) parameter(s) to the disassembler constructor, or through deriving a main disassembler class. If the information is short and its semantics obvious (like a model name), feel free to use a parameter. Otherwise derive the class.

Dynamic configuration must be done by first defining a nested public struct called `config` in the disassembler, with virtual destructor and pure virtual methods to pull the required information. A pointer to that struct should be passed to the disassembler constructor. The cpu core should then add a derivation from that config struct and implement the methods. Unidasm will have to derive a small class from the config class to give the information.

12.8.5 5. Missing stuff

There currently is no way for the debugger GUI to add per-core configuration. In particular, it is needed for the s2650 and Saturn cores. It should go through the cpu core class itself, since it's pulled from the config struct.

There is support missing in unidasm for per-cpu configuration. That's needed for a lot of things, see the unidasm source code for the current list ("Configuration missing" comments).

12.9 The device_sound_interface

- *1. The sound system*
- *2. Devices using device_sound_interface*
  - *2.1 Initialisation*
  - *2.2 Sound input/output*
  - *2.3 Stream information*
  - *2.4 Gain management*
  - *2.5 Routing setup*
- *3. Streams*
  - *3.1 Generalities*
  - *3.2 Characteristics*
  - *3.3 Sample access*
  - *3.4 Timing*
  - *3.5 Gain management*
  - *3.6 Misc. actions*
- *4. Devices using device_mixer_interface*

12.9.1 1. The sound system

The device sound interface is the entry point for devices that handle sound input and/or output. The sound system is built on the concept of *streams* which connect devices together with resampling and mixing applied transparently as needed. Microphones (audio input) and speakers (audio output) are specific known devices which use the same interface.

12.9.2 2. Devices using device_sound_interface

2.1 Initialisation

Sound streams must be created in the device_start (or interface_pre_start) method.

```
sound_stream *stream_alloc(int inputs, int outputs, int sample_rate, sound_stream_flags␣
↪flags = STREAM_DEFAULT_FLAGS);
```

A stream is created with `stream_alloc`. It takes the number of input and output channels, the sample rate and optionally flags.

The sample rate can be SAMPLE_RATE_INPUT_ADAPTIVE, SAMPLE_RATE_OUTPUT_ADAPTIVE or SAMPLE_RATE_ADAPTIVE. In that case the chosen sample rate is the highest one among the inputs, outputs or both respectively. In case of loop, the chosen sample rate is the configured global sample rate.

The only available non-default flag is STREAM_SYNCHRONOUS. When set, the sound generation method will be called for every sample individually. It's necessary for DSPs that run a program on every sample. but on the other hand it's expensive, so only to be used when required.

Devices can create multiple streams. It's rare though. Some Yamaha chips should but don't. Inputs and outputs are numbered from 0 and arrange all streams in the order they are created.

2.2 Sound input/output

```
virtual void sound_stream_update(sound_stream &stream);
```

This method is required to be implemented to consume the input samples and/or compute the output ones. The stream to update for is passed as the parameter. See the streams section, specifically sample access, to see how to write the method.

2.3 Stream information

```
int inputs() const;
int outputs() const;
std::pair<sound_stream *, int> input_to_stream_input(int inputnum) const;
std::pair<sound_stream *, int> output_to_stream_output(int outputnum) const;
```

The method `inputs` returns the total number of inputs in the streams created by the device. The method `outputs` similarly counts the outputs. The other two methods allow to grab the stream and channel number for the device corresponding to the global input or output number.

2.4 Gain management

```
float input_gain(int inputnum) const;
float output_gain(int outputnum) const;
void set_input_gain(int inputnum, float gain);
void set_output_gain(int outputnum, float gain);
void set_route_gain(int source_channel, device_sound_interface *target, int target_
↪channel, float gain);

float user_output_gain() const;
float user_output_gain(int outputnum) const;
void set_user_output_gain(float gain);
void set_user_output_gain(int outputnum, float gain);
```

Those methods allow to set the gain on every step of the routes between streams. All gains are multipliers, with default value 1.0. The steps are, from samples output in `sound_stream_update` to samples read in the next device's `sound_stream_update`:

- Per-channel output gain
- Per-channel user output gain
- Per-device user output gain
- Per-route gain
- Per-channel input gain

The user gains must not be set from the driver, they're for use by the user interface (the sliders) and are saved in the game configuration. The other gains are for driver/device use, and are saved in save states.

2.5 Routing setup

```
device_sound_interface &add_route(u32 output, const device_finder<T, R> &target, double␣
↪gain, u32 channel = 0)
device_sound_interface &add_route(u32 output, const char *target, double gain, u32␣
↪channel = 0);
device_sound_interface &add_route(u32 output, device_sound_interface &target, double␣
↪gain, u32 channel = 0);

device_sound_interface &reset_routes();
```

Routes between devices, e.g. between streams, are set at configuration time. The method `add_route` must be called on the source device and gives the channel on the source device, the target device, the gain, and optionally the channel on the target device. The constant `ALL_OUTPUTS` can be used to add a route from every channel of the source to a given channel of the destination.

The method `reset_routes` is used to remove all the routes setup on a given source device.

```
u32 get_sound_requested_inputs() const;
u32 get_sound_requested_outputs() const;
u64 get_sound_requested_inputs_mask() const;
u64 get_sound_requested_outputs_mask() const;
```

Those methods are useful for devices which want to behave differently depending on what routes are set up on them. You get either the max number of requested channel plus one (which is the number of channels when all channels are

routed, but is more useful when there are gaps) or a mask of use for channels 0-63. Note that `ALL_OUTPUTS` does not register any specific output or output count.

12.9.3 3. Streams

3.1 Generalities

Streams are endpoints associated with devices and, when connected together, ensure the transmission of audio data between them. A stream has a number of inputs (which can be zero) and outputs (same) and one sample rate which is common to all inputs and outputs. The connections are set up at the machine configuration level and the sound system ensures mixing and resampling is done transparently.

Samples in streams are encoded as sample_t. In the current implementation, this is a float. Nominal values are between -1 and 1, but clamping at the device level is not recommended (unless that's what happens in hardware of course) because the gain values, volume and effects can easily avoid saturation.

They are implemented in the class `sound_stream`.

3.2 Characteristics

```
device_t &device() const;
bool input_adaptive() const;
bool output_adaptive() const;
bool synchronous() const;
u32 input_count() const;
u32 output_count() const;
u32 sample_rate() const;
attotime sample_period() const;
```

3.3 Sample access

```
s32 samples() const;

void put(s32 output, s32 index, sample_t sample);
void put_clamp(s32 output, s32 index, sample_t sample, sample_t clamp = 1.0);
void put_int(s32 output, s32 index, s32 sample, s32 max);
void put_int_clamp(s32 output, s32 index, s32 sample, s32 maxclamp);
void add(s32 output, s32 index, sample_t sample);
void add_int(s32 output, s32 index, s32 sample, s32 max);
void fill(s32 output, sample_t value, s32 start, s32 count);
void fill(s32 output, sample_t value, s32 start);
void fill(s32 output, sample_t value);
void copy(s32 output, s32 input, s32 start, s32 count);
void copy(s32 output, s32 input, s32 start);
void copy(s32 output, s32 input);
sample_t get(s32 input, s32 index) const;
sample_t get_output(s32 output, s32 index) const;
```

Those are the methods used to implement `sound_stream_update`. First `samples` tells how many samples to consume and/or generate. The to-generate samples, if any, are pre-cleared (e.g. set to zero).

Input samples are retrieved with `get`, where `input` is the stream channel number and `index` the sample number.

Generated samples are written with the put variants. `put` sets a sample_t in channel `output` at position `index`. `put_clamp` does the same but first clamps the value to +/-`clamp`. `put_int` does it with an integer `sample` but pre-divides it by `max`. `put_int_clamp` does the same but also pre-clamps within -`maxclamp` and `maxclamp`-1, which is the normal range for a 2-complement value.

`add` and `add_int` are similar but add the value of the sample to what's there instead of replacing. `get_output` gets the currently stored output value.

`fill` sets a range of an output channel to a given `value`. `start` tells where to start (default index 0), `count` how many (default up to the end of the buffer).

`copy` does the same as fill but gets its value from the identical position in an input channel.

Note that clamping should not be used unless it actually happens in hardware. Between gains and effects there is a fair chance saturation can be avoided later in the chain.

3.4 Timing

```
u32 sample_rate() const;
attotime sample_period() const;

u64 start_index() const;
u64 end_index() const;
attotime start_time() const;
attotime end_time() const;

attotime sample_to_time(u64 index) const;
```

`sample_rate` gives the current sample rate of the stream and `sample_period` the corresponding duration.

Within a call to the update callback, `start_index` gives the number (starting at zero at system power on) and `start_time` the time of the first sample to compute in the update. `end_index` and `end_time` correspondingly indicate one past the last sample to update, or in other words the first sample of the next update call. Outside of an update callback, they all point to the first sample of the next update call.

Finally `sample_to_time` allows to convert from a sample number to a time.

Note that in case of change of sample rate sample numbers are recalculated to end up as if the stream had had the new rate from the start. And the times will still be such that sample 0 is at time 0.

3.5 Gain management

```
float user_output_gain() const;
void set_user_output_gain(float gain);
float user_output_gain(s32 output) const;
void set_user_output_gain(s32 output, float gain);

float input_gain(s32 input) const;
void set_input_gain(s32 input, float gain);
void apply_input_gain(s32 input, float gain);
float output_gain(s32 output) const;
void set_output_gain(s32 output, float gain);
void apply_output_gain(s32 output, float gain);
```

This is similar to the device gain control, with a twist: apply multiplies the current gain by the given value.

3.6 Misc. actions

```
void set_sample_rate(u32 sample_rate);
void update();
```

The method `set_sample_rate` allows to change the sample rate of the stream. The method `update` triggers a call of `sound_stream_update` on the stream and the ones it depends on to reach the current time in terms of samples.

12.9.4 4. Devices using device_mixer_interface

The device mixer interface is used for devices that want to relay sound in the device tree without acting on it. It's very useful on for instance slot devices connectors, where the slot device may have an audio connection with the main system. They are routed like every other sound device, create the streams automatically and copy input to output. Nothing needs to be done in the device.

12.10 Emulated system memory and address spaces management

- *1. Overview*
- *2. Basic concepts*
  - *2.1 Address spaces*
  - *2.2 Address maps*
  - *2.3 Shares, banks and regions*
  - *2.4 Views*
- *3. Memory objects*
  - *3.1 Shares - memory_share*
  - *3.2 Banks - memory_bank*
  - *3.3 Regions - memory_region*
  - *3.4 Views - memory_view*
  - *3.5 Bus contention handling*
- *4. Address maps API*
  - *4.1 General API structure*
  - *4.2 Global configurations*
    - *4.2.1 Global masking*
    - *4.2.2 Returned value on unmapped/nop-ed read*
  - *4.3 Handler setting*
    - *4.3.1 Method on the current device*
    - *4.3.2 Method on a different device*
    - *4.3.3 Lambda function*

        * *4.3.4 Direct memory access*
        * *4.3.5 Bank access*
        * *4.3.6 Port access*
        * *4.3.7 Dropped access*
        * *4.3.8 Unmapped access*
        * *4.3.9 Subdevice mapping*
    - *4.4 Range qualifiers*
        * *4.4.1 Mirroring*
        * *4.4.2 Masking*
        * *4.4.3 Selection*
        * *4.4.4 Sub-unit selection*
        * *4.4.5 Chip select handling on sub-unit*
        * *4.4.6 User flags*
    - *4.5 Contention*
    - *4.6 View setup*
- *5. Address space dynamic mapping API*
    - *5.1 General API structure*
    - *5.2 Handler mapping*
    - *5.3 Direct memory range mapping*
    - *5.4 Bank mapping*
    - *5.5 Port mapping*
    - *5.6 Dropped accesses*
    - *5.7 Unmapped accesses*
    - *5.8 Device map installation*
    - *5.9 Contention*
    - *5.10 View installation*
    - *5.11 Taps*

12.10.1 1. Overview

The memory subsystem (emumem and addrmap) combines multiple functions useful for system emulation:

- address bus decoding and dispatching with caching
- static descriptions of an address map
- RAM allocation and registration for state saving
- interaction with memory regions to access ROM

Devices create address spaces, e.g. decodable buses, through the `device_memory_interface`. The machine configuration sets up address maps to put in the address spaces, then the device can do read and writes through the bus.

12.10.2 2. Basic concepts

2.1 Address spaces

An address space, implemented in the class **address_space**, represents an addressable bus with potentially multiple sub-devices connected requiring a decode. It has a number of data lines (8, 16, 32 or 64) called data width, a number of address lines (1 to 32) called address width and an Endianness. In addition an address shift allows for buses that have an atomic granularity different than a byte.

Address space objects provide a series of methods for read and write access, and a second series of methods for dynamically changing the decode.

2.2 Address maps

An address map is a static description of the decode expected when using a bus. It connects to memory, other devices and methods, and is installed, usually at startup, in an address space. That description is stored in an **address_map** structure which is filled programmatically.

2.3 Shares, banks and regions

Memory shares are allocated memory zones that can be put in multiple places in the same or different address spaces, and can also be directly accessed from devices.

Memory banks are zones that indirect memory access, giving the possibility to dynamically and efficiently change where a zone actually points to.

Memory regions are read-only memory zones in which ROMs are loaded.

All of these have names allowing to access them.

2.4 Views

Views are a way to multiplex different submaps over a memory range with fast switching. It is to be used when multiple devices map at the same addresses and are switched in externally. They must be created as an object of the device and then setup either statically in a memory map or dynamically through `install_*` calls.

Switchable submaps, aka variants, are named through an integer. An internal indirection through a map ensures that any integer value can be used.

12.10.3 3. Memory objects

3.1 Shares - memory_share

```
class memory_share {
    const std::string &name() const;
    void *ptr() const;
    size_t bytes() const;
    endianness_t endianness() const;
```

(continues on next page)

(continued from previous page)

```
    u8 bitwidth() const;
    u8 bytewidth() const;
};
```

A memory share is a named allocated memory zone that is automatically saved in save states and can be mapped in address spaces. It is the standard container for memory that is shared between spaces, but also shared between an emulated CPU and a driver. As such one has easy access to its contents from the driver class.

```
required_shared_ptr<uNN> m_share_ptr;
optional_shared_ptr<uNN> m_share_ptr;
required_shared_ptr_array<uNN, count> m_share_ptr_array;
optional_shared_ptr_array<uNN, count> m_share_ptr_array;

[device constructor] m_share_ptr(*this, "name"),
[device constructor] m_share_ptr_array(*this, "name%u", 0U),
```

At the device level, a pointer to the memory zone can easily be retrieved by building one of these four finders. Note that like for every finder calling `target()` on the finder gives you the base pointer of the `memory_share` object.

```
memory_share_creator<uNN> m_share;

[device constructor] m_share(*this, "name", size, endianness),
```

A memory share can be created if it doesn't exist in a memory map through that creator class. If it already exists it is just retrieved. That class behaves like a pointer but also has the `target()`, `length()`, `bytes()`, `endianness()`, `bitwidth()` and `bytewidth()` methods for share information. The desired size is specified in bytes.

```
memory_share *memshare(string tag) const;
```

The `memshare` device method retrieves a memory share by name. Beware that the lookup can be expensive, prefer finders instead.

3.2 Banks - memory_bank

```
class memory_bank {
    const std::string &tag() const;
    int entry() const;
    void set_entry(int entrynum);
    void configure_entry(int entrynum, void *base);
    void configure_entries(int startentry, int numentry, void *base, offs_t stride);
    void set_base(void *base);
    void *base() const;
};
```

A memory bank is a named memory zone indirection that can be mapped in address spaces. It points to `nullptr` when created. `configure_entry` associates an entry number and a base pointer. `configure_entries` does the same for multiple consecutive entries spanning a memory zone.

`set_base` sets the base address for the active entry. If there are no entries, entry 0 (zero) is automatically created and selected. Use of `set_base` should be avoided in favour of pre-configured entries unless there are an impractically large number of possible base addresses.

`set_entry` dynamically and efficiently selects the active entry, `entry()` returns the active entry number, and `base()` gets the associated base pointer.

```
required_memory_bank m_bank;
optional_memory_bank m_bank;
required_memory_bank_array<count> m_bank_array;
optional_memory_bank_array<count> m_bank_array;

[device constructor] m_bank(*this, "name"),
[device constructor] m_bank_array(*this, "name%u", 0U),
```

At the device level, a pointer to the memory bank object can easily be retrieved by building one of these four finders.

```
memory_bank_creator m_bank;

[device constructor] m_bank(*this, "name"),
```

A memory bank can be created if it doesn't exist in a memory map through that creator class. If it already exists it is just retrieved.

```
memory_bank *membank(string tag) const;
```

The `membank` device method retrieves a memory bank by name. Beware that the lookup can be expensive, prefer finders instead.

3.3 Regions - memory_region

```
class memory_region {
    u8 *base();
    u8 *end();
    u32 bytes() const;
    const std::string &name() const;
    endianness_t endianness() const;
    u8 bitwidth() const;
    u8 bytewidth() const;
    u8 &as_u8(offs_t offset = 0);
    u16 &as_u16(offs_t offset = 0);
    u32 &as_u32(offs_t offset = 0);
    u64 &as_u64(offs_t offset = 0);
}
```

A region is used to store read-only data like ROMs or the result of fixed decryptions. Their contents are not saved, which is why they should not being written to from the emulated system. They don't really have an intrinsic width (`base()` returns an `u8 *` always), which is historical and pretty much unfixable at this point. The `as_*` methods allow for accessing them at a given width.

```
required_memory_region m_region;
optional_memory_region m_region;
required_memory_region_array<count> m_region_array;
optional_memory_region_array<count> m_region_array;

[device constructor] m_region(*this, "name"),
[device constructor] m_region_array(*this, "name%u", 0U),
```

At the device level, a pointer to the memory region object can easily be retrieved by building one of these four finders.

```
memory_region *memregion(string tag) const;
```

The `memregion` device method retrieves a memory region by name. Beware that the lookup can be expensive, prefer finders instead.

3.4 Views - memory_view

```
class memory_view {
    memory_view(device_t &device, std::string name);
    memory_view_entry &operator[](int slot);

    void select(int entry);
    void disable();

    const std::string &name() const;
}
```

A view allows to switch part of a memory map between multiple possibilities, or even disable it entirely to see what was there before. It is created as an object of the device.

```
memory_view m_view;

[device constructor] m_view(*this, "name"),
```

It is then setup through the address map API or dynamically. At runtime, a numbered variant can be selected using the `select` method, or the view can be disabled using the `disable` method. A disabled view can be re-enabled at any time.

3.5 Bus contention handling

Some specific CPUs have been upgraded to be interruptible which allows to add bus contention and wait states capabilities. Being interruptible means, in practice, that an instruction can be interrupted at any time and the execute_run method of the core exited. Other devices can then run, then eventually controls returns to the core and the instruction continues from the point it was started. Importantly, this can be triggered from a handler and even be used to interrupt just before the access that is currently done (e.g. continuation will redo the access).

The CPUs supporting that declare their capability by overriding the method `cpu_is_interruptible` to return true.

Three intermediate contention handlers can be added to accesses:

- `before_delay`: wait a number of cycles before doing the access.
- `after_delay`: wait a number of cycles after doing the access.
- `before_time`: wait for a given time before doing the access.

For the delay handlers, a method or lambda is called which returns the number of cycles to wait (as a u32).

The `before_time` is special. First, the time is compared to the current value of cpu->total_cycles(). That value is the number of cycles elapsed since the last reset of the cpu. It is passed as a parameter to the method as a u64 and must return the earliest time as a u64 when the access can be done, which can be equal to the passed-in time. From there two things can happen: either the running cpu has enough cycles left to consume to reach that time. In that case, the necessary number of cycles is consumed, and the access is done. Otherwise, when there isn't enough, the remaining cycles are consumed, the access aborted, scheduling happens, and eventually the access is redone. In that case the

method is called again with the new current time, and must return the (probably same) earliest time again. This will happen until enough cycles to consume are available to directly do the access.

This approach allows to for instance handle consecutive DMAs. A first DMA grabs the bus for a transfer. This shows up as the method answering for the earliest time for access the time of the end of the dma. If no timer happens until that time the access will then happen just after the dma finishes. But if a timer elapses before that and as a consequence another dma is queued while the first is running, the cycle will be aborted for lack of remaining time, and the method will eventually be called again. It will then give the time of when the second dma will finish, and all will be well.

It can also allow to reduce said earlier time when circumstances require it. For instance a PIO latch that waits up to 64 cycles that data arrives can indicate that current time + 64 as a target (which will trigger a bus error for instance) but if a timer elapses and fills the latch meanwhile the method will be called again and that time can just return the current time to let the access pass though. Beware that if the timer elapsing did not fill the latch then the method must return the time it returned previously, e.g. the initial access time + 64, otherwise irrelevant timers happening or simply scheduling quantum effects will delay the timeout, possibly to infinity if the quantum is small enough.

Contention handlers on the same address are taken into account in the `before_time`, `before_delay` then `after_delay` order. Contention handlers of the same type on the same address at last-one-wins. Installing any non-contention handler on a range where a contention handler was removes it.

12.10.4 4. Address maps API

4.1 General API structure

An address map is a method of a device which fills an **address_map** structure, usually called **map**, passed by reference. The method then can set some global configuration through specific methods and then provide address range-oriented entries which indicate what should happen when a specific range is accessed.

The general syntax for entries uses method chaining:

```
map(start, end).handler(...).handler_qualifier(...).range_qualifier().contention();
```

The values start and end define the range, the handler() block determines how the access is handled, the handler_qualifier() block specifies some aspects of the handler (memory sharing for instance) and the range_qualifier() block refines the range (mirroring, masking, lane selection, etc.). The contention methods handle bus contention and wait states for cpus supporting them.

The map follows a "last one wins" principle, where the handler specified last is selected when multiple handlers match a given address.

4.2 Global configurations

4.2.1 Global masking

```
map.global_mask(offs_t mask);
```

Specifies a mask to be applied to all addresses when accessing the space that map is installed in.

4.2.2 Returned value on unmapped/nop-ed read

```
map.unmap_value_low();
map.unmap_value_high();
map.unmap_value(u8 value);
```

Sets the value to return on reads to an unmapped or nopped-out address. Low means 0, high ~0.

4.3 Handler setting

4.3.1 Method on the current device

```
(...).r(FUNC(my_device::read_method))
(...).w(FUNC(my_device::write_method))
(...).rw(FUNC(my_device::read_method), FUNC(my_device::write_method))

uNN my_device::read_method(address_space &space, offs_t offset, uNN mem_mask)
uNN my_device::read_method(address_space &space, offs_t offset)
uNN my_device::read_method(address_space &space)
uNN my_device::read_method(offs_t offset, uNN mem_mask)
uNN my_device::read_method(offs_t offset)
uNN my_device::read_method()

void my_device::write_method(address_space &space, offs_t offset, uNN data, uNN mem_mask)
void my_device::write_method(address_space &space, offs_t offset, uNN data)
void my_device::write_method(address_space &space, uNN data)
void my_device::write_method(offs_t offset, uNN data, uNN mem_mask)
void my_device::write_method(offs_t offset, uNN data)
void my_device::write_method(uNN data)
```

Sets a method of the current device or driver to read, write or both for the current entry. The prototype of the method can take multiple forms making some elements optional. `uNN` represents `u8`, `u16`, `u32` or `u64` depending on the data width of the handler. The handler can be narrower than the bus itself (for instance an 8-bit device on a 32-bit bus).

The offset passed in is built from the access address. It starts at zero at the start of the range, and increments for each `uNN` unit. An `u8` handler will get an offset in bytes, an `u32` one in double words. The `mem_mask` has its bits set where the accessors actually drive the bit. It's usually built in byte units, but in some cases of I/O chips ports with per-bit direction registers the resolution can be at the bit level.

4.3.2 Method on a different device

```
(...).r(m_other_device, FUNC(other_device::read_method))
(...).r("other-device-tag", FUNC(other_device::read_method))
(...).w(m_other_device, FUNC(other_device::write_method))
(...).w("other-device-tag", FUNC(other_device::write_method))
(...).rw(m_other_device, FUNC(other_device::read_method), FUNC(other_device::write_
↪method))
(...).rw("other-device-tag", FUNC(other_device::read_method), FUNC(other_device::write_
↪method))
```

Sets a method of another device, designated by an object finder (usually `required_device` or `optional_device`) or its tag, to read, write or both for the current entry.

4.3.3 Lambda function

```
(...).lr{8,16,32,64}(NAME([...](address_space &space, offs_t offset, uNN mem_mask) ->␣
↪uNN { ... }))
(...).lr{8,16,32,64}([...](address_space &space, offs_t offset, uNN mem_mask) -> uNN { ..
↪. }, "name")
(...).lw{8,16,32,64}(NAME([...](address_space &space, offs_t offset, uNN data, uNN mem_
↪mask) -> void { ... }))
(...).lw{8,16,32,64}([...](address_space &space, offs_t offset, uNN data, uNN mem_mask) -
↪> void { ... }, "name")
(...).lrw{8,16,32,64}(NAME(read), NAME(write))
(...).lrw{8,16,32,64}(read, "name_r", write, "name_w")
```

Sets a lambda called on read, write or both. The lambda prototype can be any of the six available for methods. One can either use `NAME()` over the whole lambda, or provide a name after the lambda definition. The number is the data width of the access, e.g. the NN.

4.3.4 Direct memory access

```
(...).rom()
(...).writeonly()
(...).ram()
```

Selects the range to access a memory zone as read-only, write-only or read/write respectively. Specific handler qualifiers specify the location of this memory zone. There are two cases when no qualifier is acceptable:

- `ram()` gives an anonymous RAM zone not accessible outside of the address space.
- `rom()` when the memory map is used in an `AS_PROGRAM` space of a (CPU) device which names is also the name of a region. Then the memory zone points to that region at the offset corresponding to the start of the zone.

```
(...).rom().region("name", offset)
```

The `region` qualifier causes a read-only zone point to the contents of a given region at a given offset.

```
(...).rom().share("name")
(...).writeonly.share("name")
(...).ram().share("name")
```

The `share` qualifier causes the zone point to a shared memory region identified by its name. If the share is present in multiple spaces, the size, bus width, and, if the bus is more than byte-wide, the Endianness must match.

4.3.5 Bank access

```
(...).bankr("name")
(...).bankw("name")
(...).bankrw("name")
```

Sets the range to point at the contents of a memory bank in read, write or read/write mode.

4.3.6 Port access

```
(...).portr("name")
(...).portw("name")
(...).portrw("name")
```

Sets the range to point at an I/O port.

4.3.7 Dropped access

```
(...).nopr()
(...).nopw()
(...).noprw()
```

Sets the range to drop the access without logging. When reading, the unmap value is returned.

4.3.8 Unmapped access

```
(...).unmapr()
(...).unmapw()
(...).unmaprw()
```

Sets the range to drop the access with logging. When reading, the unmap value is returned.

4.3.9 Subdevice mapping

```
(...).m(m_other_device, FUNC(other_device::map_method))
(...).m("other-device-tag", FUNC(other_device::map_method))
```

Includes a device-defined submap. The start of the range indicates where the address zero of the submap ends up, and the end of the range clips the submap if needed. Note that range qualifiers (defined later) apply.

Currently, only handlers are allowed in submaps and not memory zones or banks.

4.4 Range qualifiers

4.4.1 Mirroring

```
(...).mirror(mask)
```

Duplicate the range on the addresses reachable by setting any of the 1 bits present in mask. For instance, a range 0-0x1f with mirror 0x300 will be present on 0-0x1f, 0x100-0x11f, 0x200-0x21f and 0x300-0x31f. The addresses passed in to the handler stay in the 0-0x1f range, the mirror bits are not seen by the handler.

4.4.2 Masking

```
(...).mask(mask)
```

Only valid with handlers, the address will be masked with the mask before being passed to the handler.

4.4.3 Selection

```
(...).select(mask)
```

Only valid with handlers, the range will be mirrored as with mirror, but the mirror address bits are preserved in the offset passed to the handler when it is called. This is useful for devices like sound chips where the low bits of the address select a function and the high bits a voice number.

4.4.4 Sub-unit selection

```
(...).umask16(16-bits mask)
(...).umask32(32-bits mask)
(...).umask64(64-bits mask)
```

Only valid with handlers and submaps, selects which data lines of the bus are actually connected to the handler or the device. The mask value should be a multiple of a byte, e.g. the mask is a series of 00 and ff. The offset will be adjusted accordingly, so that a difference of 1 means the next handled unit in the access.

If the mask is narrower than the bus width, the mask is replicated in the upper lines.

4.4.5 Chip select handling on sub-unit

```
(...).cselect(16/32/64)
```

When a device is connected to part of the bus, like a byte on a 16-bits bus, the target handler is only activated when that part is actually accessed. In some cases, very often byte access on a 68000 16-bits bus, the actual hardware only checks the word address and not if the correct byte is accessed. `cswidth` tells the memory system to trigger the handler if a wider part of the bus is accessed. The parameter is that trigger width (would be 16 in the 68000 case).

4.4.6 User flags

```
(...).flags(16-bits mask)
```

This parameter allows to set user-defined flags on the handler which can then be retrieved by an accessing device to change their behaviour. An example of use the i960 which marks burstable zones that way (they have a specific hardware-level support).

4.5 Contention

```
(...).before_time(method).(...)
(...).before_delay(method).(...)
(...).after_delay(method).(...)
```

These three methods allow to add the contention methods to a handler. See section *3.5*. Multiple methods can be handler to one handler.

4.6 View setup

```
map(start, end).view(m_view);
m_view[0](start1, end1).[...];
```

A view is setup in a address map with the view method. The only qualifier accepted is mirror. The "disabled" version of the view will include what was in the range prior to the view setup.

The different variants are setup by indexing the view with the variant number and setting up an entry in the usual way. The entries within a variant must of course stay within the range. There are no other additional constraints. The contents of a variant, by default, are what was there before, i.e. the contents of the disabled view, and setting it up allows part or all of it to be overridden.

Variants can only be setup once the view itself has been setup with the `view` method.

A view can only be put in one address map and in only one position. If multiple views have identical or similar contents, remember that setting up a map is nothing more than a method call, and creating a second method to setup a view is perfectly reasonable. A view is of type `memory_view` and an indexed entry (e.g. a variant to setup) is of type `memory_view::memory_view_entry &`.

A view can be installed in another view, but don't forget that a view can be installed only once. A view can also be part of "what was there before".

12.10.5 5. Address space dynamic mapping API

5.1 General API structure

A series of methods allow the bus decoding of an address space to be changed on-the-fly. They're powerful but have some issues:

- changing the mappings repeatedly can be slow
- the address space state is not saved in the saved states, so it has to be rebuilt after state load
- they can be hidden anywhere rather than be grouped in an address map, which can be less readable

The methods, rather than decomposing the information in handler, handler qualifier and range qualifier, put them all together as method parameters. To make things a little more readable, lots of them are optional.

5.2 Handler mapping

```
uNN my_device::read_method(address_space &space, offs_t offset, uNN mem_mask)
uNN my_device::read_method_m(address_space &space, offs_t offset)
uNN my_device::read_method_mo(address_space &space)
uNN my_device::read_method_s(offs_t offset, uNN mem_mask)
uNN my_device::read_method_sm(offs_t offset)
uNN my_device::read_method_smo()

void my_device::write_method(address_space &space, offs_t offset, uNN data, uNN mem_mask)
void my_device::write_method_m(address_space &space, offs_t offset, uNN data)
void my_device::write_method_mo(address_space &space, uNN data)
void my_device::write_method_s(offs_t offset, uNN data, uNN mem_mask)
void my_device::write_method_sm(offs_t offset, uNN data)
void my_device::write_method_smo(uNN data)

readNN_delegate   (device, FUNC(read_method))
readNNm_delegate  (device, FUNC(read_method_m))
readNNmo_delegate (device, FUNC(read_method_mo))
readNNs_delegate  (device, FUNC(read_method_s))
readNNsm_delegate (device, FUNC(read_method_sm))
readNNsmo_delegate(device, FUNC(read_method_smo))

writeNN_delegate   (device, FUNC(write_method))
writeNNm_delegate  (device, FUNC(write_method_m))
writeNNmo_delegate (device, FUNC(write_method_mo))
writeNNs_delegate  (device, FUNC(write_method_s))
writeNNsm_delegate (device, FUNC(write_method_sm))
writeNNsmo_delegate(device, FUNC(write_method_smo))
```

To be added to a map, a method call and the device it is called onto have to be wrapped in the appropriate delegate type. There are twelve types, for read and for write and for all six possible prototypes. Note that as all delegates, they can also wrap lambdas.

```
space.install_read_handler(addrstart, addrend, read_delegate, unitmask, cswidth, flags)
space.install_read_handler(addrstart, addrend, addrmask, addrmirror, addrselect, read_
↪delegate, unitmask, cswidth, flags)
space.install_write_handler(addrstart, addrend, write_delegate, unitmask, cswidth, flags)
space.install_write_handler(addrstart, addrend, addrmask, addrmirror, addrselect, write_
↪delegate, unitmask, cswidth, flags)
space.install_readwrite_handler(addrstart, addrend, read_delegate, write_delegate,␣
↪unitmask, cswidth, flags)
space.install_readwrite_handler(addrstart, addrend, addrmask, addrmirror, addrselect,␣
↪read_delegate, write_delegate, unitmask, cswidth, flags)
```

These six methods allow to install delegate-wrapped handlers in a live address space. Either plain or with mask, mirror and select. In the read/write case both delegates must be of the same flavor (`smo` stuff) to avoid a combinatorial explosion of method types. The `unitmask`, `cswidth` and `flags` arguments are optional.

5.3 Direct memory range mapping

```
space.install_rom(addrstart, addrend, void *pointer)
space.install_rom(addrstart, addrend, addrmirror, void *pointer)
space.install_rom(addrstart, addrend, addrmirror, flags, void *pointer)
space.install_writeonly(addrstart, addrend, void *pointer)
space.install_writeonly(addrstart, addrend, addrmirror, void *pointer)
space.install_writeonly(addrstart, addrend, addrmirror, flags, void *pointer)
space.install_ram(addrstart, addrend, void *pointer)
space.install_ram(addrstart, addrend, addrmirror, void *pointer)
space.install_ram(addrstart, addrend, addrmirror, flags, void *pointer)
```

Installs a memory block in an address space, with or without mirror and flags. `_rom` is read-only, `_ram` is read/write, `_writeonly` is write-only. The pointer must be non-null, this method will not allocate the memory.

5.4 Bank mapping

```
space.install_read_bank(addrstart, addrend, memory_bank *bank)
space.install_read_bank(addrstart, addrend, addrmirror, memory_bank *bank)
space.install_read_bank(addrstart, addrend, addrmirror, flags, memory_bank *bank)
space.install_write_bank(addrstart, addrend, memory_bank *bank)
space.install_write_bank(addrstart, addrend, addrmirror, memory_bank *bank)
space.install_write_bank(addrstart, addrend, addrmirror, flags, memory_bank *bank)
space.install_readwrite_bank(addrstart, addrend, memory_bank *bank)
space.install_readwrite_bank(addrstart, addrend, addrmirror, memory_bank *bank)
space.install_readwrite_bank(addrstart, addrend, addrmirror, flags, memory_bank *bank)
```

Install an existing memory bank for reading, writing or both in an address space.

5.5 Port mapping

```
space.install_read_port(addrstart, addrend, const char *rtag)
space.install_read_port(addrstart, addrend, addrmirror, const char *rtag)
space.install_read_port(addrstart, addrend, addrmirror, flags, const char *rtag)
space.install_write_port(addrstart, addrend, const char *wtag)
space.install_write_port(addrstart, addrend, addrmirror, const char *wtag)
space.install_write_port(addrstart, addrend, addrmirror, flags, const char *wtag)
space.install_readwrite_port(addrstart, addrend, const char *rtag, const char *wtag)
space.install_readwrite_port(addrstart, addrend, addrmirror, const char *rtag, const␣
↪char *wtag)
space.install_readwrite_port(addrstart, addrend, addrmirror, flags, const char *rtag,␣
↪const char *wtag)
```

Install ports by name for reading, writing or both.

5.6 Dropped accesses

```
space.nop_read(addrstart, addrend, addrmirror, flags)
space.nop_write(addrstart, addrend, addrmirror, flags)
space.nop_readwrite(addrstart, addrend, addrmirror, flags)
```

Drops the accesses for a given range with an optional mirror and flags;

5.7 Unmapped accesses

```
space.unmap_read(addrstart, addrend, addrmirror, flags)
space.unmap_write(addrstart, addrend, addrmirror, flags)
space.unmap_readwrite(addrstart, addrend, addrmirror, flags)
```

Unmaps the accesses (e.g. logs the access as unmapped) for a given range with an optional mirror and flags.

5.8 Device map installation

```
space.install_device(addrstart, addrend, device, map, unitmask, cswidth, flags)
```

Install a device address with an address map in a space. The `unitmask`, `cswidth` and `flags` arguments are optional.

5.9 Contention

```
using ws_time_delegate  = device_delegate<u64 (offs_t, u64)>;
using ws_delay_delegate = device_delegate<u32 (offs_t)>;

space.install_read_before_time(addrstart, addrend, addrmirror, ws_time_delegate)
space.install_write_before_time(addrstart, addrend, addrmirror, ws_time_delegate)
space.install_readwrite_before_time(addrstart, addrend, addrmirror, ws_time_delegate)

space.install_read_before_delay(addrstart, addrend, addrmirror, ws_delay_delegate)
space.install_write_before_delay(addrstart, addrend, addrmirror, ws_delay_delegate)
space.install_readwrite_before_delay(addrstart, addrend, addrmirror, ws_delay_delegate)

space.install_read_after_delay(addrstart, addrend, addrmirror, ws_delay_delegate)
space.install_write_after_delay(addrstart, addrend, addrmirror, ws_delay_delegate)
space.install_readwrite_after_delay(addrstart, addrend, addrmirror, ws_delay_delegate)
```

Install a contention handler in the decode path. The addrmirror parameter is optional.

5.10 View installation

```
space.install_view(addrstart, addrend, view)
space.install_view(addrstart, addrend, addrmirror, view)

view[0].install...
```

Installs a view in a space. This can be only done once and in only one space, and the view must not have been setup through the address map API before. Once the view is installed, variants can be selected by indexing to call a dynamic mapping method on it.

A view can be installed into a variant of another view without issues, with only the usual constraint of single installation.

5.11 Taps

```
using tap = std::function<void (offs_t offset, uNN &data, uNN mem_mask)

memory_passthrough_handler mph = space.install_read_tap(addrstart, addrend, name, read_
↪tap, &mph);
memory_passthrough_handler mph = space.install_write_tap(addrstart, addrend, name, write_
↪tap, &mph);
memory_passthrough_handler mph = space.install_readwrite_tap(addrstart, addrend, name,␣
↪read_tap, write_tap, &mph);

mph.remove();
```

A tap is a method that is be called when a specific range of addresses is accessed without overriding the actual access. Taps can change the data passed around. A write tap happens before the access, and can change the value to be written. A read tap happens after the access, and can change the value returned.

Taps must be of the same width and alignment as the bus. Multiple taps can act over the same addresses.

The `memory_passthrough_handler` object collates a number of taps and allow to remove them all in one call. The `mph` parameter is optional and a new one will be created if absent.

Taps are lost when a new handler is installed at the same addresses (under the usual principle of last one wins). If they need to be preserved, one should install a change notifier on the address space, and remove + reinstall the taps when notified.

12.11 CPU devices

- *1. Overview*
- *2. DRC*
- *3. Interruptibility*
  - *3.1 Definition*
  - *3.2 Implementation requirements*
  - *3.3 Example implementation with generators*

- *3.4 Bus contention cpu_device interface*
- *3.5 Interaction with DRC*

12.11.1 1. Overview

CPU devices derivatives are used, unsurprisingly, to implement the emulation of CPUs, MCUs and SOCs. A CPU device is first a combination of `device_execute_interface`, `device_memory_interface`, `device_state_interface` and `device_disasm_interface`. Refer to the associated documentations when they exist.

Two more functionalities are specific to CPU devices which are the DRC and the interruptibility support.

12.11.2 2. DRC

TODO.

12.11.3 3. Interruptibility

3.1 Definition

An interruptible CPU is defined as a core which is able to suspend the execution of one instruction at any time, exit execute_run, then at the next call of `execute_run` keep going from where it was. This includes being able to abort an issued memory access, quit execute_run, then upon the next call of execute_run reissue the exact same access.

3.2 Implementation requirements

Memory accesses must be done with `read_interruptible` or `write_interruptible` on a `memory_access_specific` or a `memory_access_cache`. The access must be done as bus width and bus alignment.

After each access the core must test whether `icount <= 0`. This test should be done after `icount` is decremented of the time taken by the access itself, to limit the number of tests. When `icount` reaches 0 or less it means that the instruction emulation needs to be suspended.

To know whether the access needs to be re-issued, `access_to_be_redone()` needs to be called. If it returns true then the time taken by the access needs to be credited back, since it hasn't yet happened, and the access will need to be re-issued. The call to `access_to_be_redone()` clears the reissue flag. If you need to check the flag without clearing it use `access_to_be_redone_noclear()`.

The core needs to do enough bookkeeping to eventually restart the instruction execution just before the access or just after the test, depending on the need of reissue.

Finally, to indicate to the rest of the infrastructure the support, it must override cpu_is_interruptible() to return true.

3.3 Example implementation with generators

To ensure decent performance, the current implementations (h8, 6502 and 68000) use a python generator to generate two versions of each instruction interpreter, one for the normal emulation, and one for restarting the instruction.

The restarted version looks like that (for a 4-cycles per access cpu):

```
void device::execute_inst_restarted()
{
    switch(m_inst_substate) {
    case 0:
        [...]

        m_address = [...];
        m_mask = [...];
        [[fallthrough]];
    case 42:
        m_result = specific.read_interruptible(m_address, m_mask);
        m_icount -= 4;
        if(m_icount <= 0) {
            if(access_to_be_redone()) {
                m_icount += 4;
                m_inst_substate = 42;
            } else
                m_inst_substate = 43;
            return;
        }
        [[fallthrough]];
    case 43:
        [...] = m_result;
        [...]
    }
    m_inst_substate = 0;
    return;
}
```

The non-restarted version is the same thing with the switch and the final `m_inst_substate` clearing removed.

```
void device::execute_inst_non_restarted()
{
    [...]
    m_address = [...];
    m_mask = [...];
    m_result = specific.read_interruptible(m_address, m_mask);
    m_icount -= 4;
    if(m_icount <= 0) {
        if(access_to_be_redone()) {
            m_icount += 4;
            m_inst_substate = 42;
        } else
            m_inst_substate = 43;
        return;
    }
    [...] = m_result;
```

(continues on next page)

(continued from previous page)

```
    [...]
    return;
}
```

The main loop then looks like this:

```
void device::execute_run()
{
    if(m_inst_substate)
        call appropriate restarted instruction handler
    while(m_icount > 0) {
        debugger_instruction_hook(m_pc);
        call appropriate non-restarted instruction handler
    }
}
```

The idea is thus that `m_inst_substate` indicates where in an instruction one is, but only when an interruption happens. It otherwise stays at 0 and is essentially never looked at. Having two versions of the interpretation allows to remove the overhead of the switch and the end-of-instruction substate clearing.

It is not a requirement to use a generator-based that method, but a different one which does not have unacceptable performance implications has not yet been found.

3.4 Bus contention cpu_device interface

The main way to setup bus contention is through the memory maps. Lower-level access can be obtained through some methods on cpu_device though.

```
bool cpu_device::access_before_time(u64 access_time, u64 current_time) noexcept;
```

The method `access_before_time` allows to try to run an access at a given time in cpu cycles. It takes the current time (`total_cycles()`) and the expected time for the access. If there aren't enough cycles to reach that time the remaining cycles are eaten and the method returns true to tell not to do the access and call the method again eventually. Otherwise enough cycles are eaten to reach the access time and false is returned to tell to do the access.

```
bool cpu_device::access_before_delay(u32 cycles, const void *tag) noexcept;
```

The method `access_before_delay` allows to try to run an access after a given delay. The tag is an opaque, non-nullptr value used to characterize the source of the delay, so that the delay is not applied multiple times. Similarly to the previous method cycles are eaten and true is returned to abort the access, false to execute it.

```
void cpu_device::access_after_delay(u32 cycles) noexcept;
```

The method `access_after_delay` allows to add a delay after an access is done. There is no abort possible, hence no return boolean.

```
void cpu_device::defer_access() noexcept;
```

The method `defer_access` tells the cpu that we need to wait for an external event. It marks the access as to be redone, and eats all the remaining cycles of the timeslice. The idea is then that the access will be retried after time advances up to the next global system synchronisation event (sync, timer timeout or set_input_line). This is the method to use when for instance waiting on a magic latch for data expected from scsi transfers, which happen on timer timeouts.

```
void cpu_device::retry_access() noexcept;
```

The method `retry_access` tells the cpu that the access will need to be retried, and nothing else. This can easily reach a situation of livelock, so be careful. It is used for instance to simulate a wait line (for the z80 for instance) which is controlled through set_input_line. The idea is that the device setting wait does the set_input_line and a retry_access. The cpu core, as long as the wait line is set just eats cycles. Then, when the line is cleared the core will retry the access.

3.5 Interaction with DRC

At this point, interruptibility and DRC are entirely incompatible. We do not have a method to quit the generated code before or after an access. It's theoretically possible but definitely non-trivial.

12.12 The new floppy subsystem

12.12.1 1. Introduction

The new floppy subsystem aims at emulating the behaviour of floppies and floppy controllers at a level low enough that protections work as a matter of course. It reaches its goal by following the real hardware configuration:

- a floppy image class keeps in memory the magnetic state of the floppy surface and its physical characteristics
- an image handler class talks with the floppy image class to simulate the floppy drive, providing all the signals you have on a floppy drive connector
- floppy controller devices talk with the image handler and provide the register interfaces to the host we all know and love
- format handling classes are given the task of statelessly converting to and from an on-disk image format to the in-memory magnetic state format the floppy image class manages

12.12.2 2. Floppy storage 101

2.1. Floppy disk

A floppy disk is a disc that stores magnetic orientations on their surface disposed in a series on concentric circles called tracks or cylinders[1]. Its main characteristics are its size (goes from a diameter of around 2.8" to 8") , its number of writable sides (1 or 2) and its magnetic resistivity. The magnetic resistivity indicates how close magnetic orientation changes can happen and the information kept. That's one third of what defines the term "density" that is so often used for floppies (the other two are floppy drive head size and bit-level encoding).

The magnetic orientations are always binary, e.g. they're one way or the opposite, there's no intermediate state. Their direction can either be tangentially to the track, i.e. in the direction of or opposite to the rotation, or in the case of perpendicular recording the direction is perpendicular to the disc surface (hence the name). Perpendicular recording allows for closer orientation changes by writing the magnetic information more deeply, but arrived late in the technology lifetime. 2.88Mb disks and the floppy children (Zip drives, etc.) used perpendicular recording. For simulation purposes the direction is not important, only the fact that only two orientations are possible is. Two more states are possible though: a portion of a track can be demagnetized (no orientation) or damaged (no orientation and can't be written to).

A specific position in the disk rotation triggers an index pulse. That position can be detected through a hole in the surface (very visible in 5.25" and 3" floppies for instance) or through a specific position of the rotating center (3.5"

[1] Cylinder is a hard-drive term somewhat improperly used for floppies. It comes from the fact that hard-drives are similar to floppies but include a series of stacked disks with a read/write head on each. The heads are physically linked and all point to the same circle on every disk at a given time, making the accessed area look like a cylinder. Hence the name.

floppies, perhaps others). This index pulse is used to designate the beginning of the track, but is not used by every system. Older 8" floppies have multiple index holes used to mark the beginning of sectors (called hard sectoring) but one of them is positioned differently to be recognized as the track start, and the others are at fixed positions relative to the origin one.

2.2. Floppy drive

A floppy drive is what reads and writes a floppy disk. It includes an assembly capable of rotating the disk at a fixed speed and one or two magnetic heads tied to a positioning motor to access the tracks.

The head width and positioning motor step size decides how many tracks are written on the floppy. Total number of tracks goes from 32 to 84 depending on the floppy and drive, with the track 0 being the most exterior (longer) one of the concentric circles, and the highest numbered the smallest interior circle. As a result the tracks with the lowest numbers have the lowest physical magnetic orientation density, hence the best reliability. Which is why important and/or often changed structures like the boot block or the fat allocation table are at track 0. That is also where the terminology "stepping in" to increase the track number and "stepping out" to decrease it comes from. The number of tracks available is the second part of what is usually behind the term "density".

A sensor detects when the head is on track 0 and the controller is not supposed to try to go past it. In addition physical blocks prevent the head from going out of the correct track range. Some systems (Apple II, some C64) do not take the track 0 sensor into account and just wham the head against the track 0 physical block, giving a well-known crash noise and eventually damaging the head alignment.

Also, some systems (Apple II and C64 again) have direct access to the phases of the head positioning motor, allowing to trick the head into going between tracks, in middle or even quarter positions. That was not usable to write more tracks, since the head width did not change, but since reliable reading was only possible with the correct position it was used for some copy protection systems.

The disk rotates at a fixed speed for a given track. The most usual speed is 300 rpm for every track, with 360 rpm found for HD 5.25" floppies and most 8" ones, and a number of different values like 90 rpm for the earlier floppies or 150 rpm for an HD floppy in an Amiga. Having a fixed rotational speed for the whole disk is called Constant Angular Velocity (CAV, almost everybody) or Zoned Constant Angular Velocity (ZCAV, C64) depending on whether the read/write bitrate is constant or track-dependant. Some systems (Apple II, Mac) vary the rotational speed depending on the track (something like 394 rpm up to 590 rpm) to end up with a Constant Linear Velocity (CLV). The idea behind ZCAV/CLV is to get more bits out of the media by keeping the minimal spacing between magnetic orientation transitions close to the best the support can do. It seems that the complexity was not deemed worth it since almost no system does it.

Finally, after the disc rotates and the head is over the proper track reading happens. The reading is done through an inductive head, which gives it the interesting characteristic of not reading the magnetic orientation directly but instead of being sensitive to orientation inversions, called flux transitions. This detection is weak and somewhat uncalibrated, so an amplifier with Automatic Gain Calibration (AGC) and a peak detector are put behind the head to deliver clean pulses. The AGC slowly increases the amplification level until a signal goes over the threshold, then modulates its gain so that said signal is at a fixed position over the threshold. Afterwards the increase happens again. This makes the amplifier calibrate itself to the signals read from the floppy as long as flux transitions happen often enough. Too long and the amplification level will reach a point where the random noise the head picks from the environment is amplified over the threshold, creating a pulse where none should be. Too long in our case happens to be around 16-20us with no transitions. That means a long enough zone with a fixed magnetic orientation or no orientation at all (demagnetized or damaged) is going to be read as a series of random pulses after a brief delay. This is used by protections and is known as "weak bits", which read differently each time they're accessed.

A second level of filtering happens after the peak detector. When two transitions are a little close (but still over the media threshold) a bouncing effect happens between them giving two very close pulses in the middle in addition to the two normal pulses. The floppy drive detects when pulses are too close and filter them out, leaving the normal ones. As a result, if one writes a train of high-frequency pulses to the floppy they will be read back as a train of too close pulses (weak because they're over the media tolerance, but picked up by the AGC anyway, only somewhat unreliably)

they will be all filtered out, giving a large amount of time without any pulse in the output signal. This is used by some protections since it's not writable with a normally clocked controller.

Writing is symmetrical, with a series of pulses sent which make the write head invert the magnetic field orientation each time a pulse is received.

So, in conclusion, the floppy drive provides inputs to control disk rotation and head position (and choice when double-sided), and the data goes both way as a train of pulses representing magnetic orientation inversions. The absolute value of the orientation itself is never known.

2.3. Floppy controller

The task of the floppy controller is to turn the signals to/from the floppy drive into something the main CPU can digest. The level of support actually done by the controller is extremely variable from one device to the other, from pretty much nothing (Apple II, C64) through minimal (Amiga) to complete (Western Digital chips, uPD765 family). Usual functions include drive selection, motor control, track seeking and of course reading and writing data. Of these only the last two need to be described, the rest is obvious.

The data is structured at two levels: how individual bits (or nibbles, or bytes) are encoded on the surface, and how these are grouped in individually-addressable sectors. Two standards exist for these, called FM and MFM, and in addition a number of systems use their home-grown variants. Moreover, some systems such as the Amiga use a standard bit-level encoding (MFM) but a homegrown sector-level organisation.

2.3.1. Bit-level encodings

2.3.1.1. Cell organization

All floppy controllers, even the wonkiest like the Apple II one, start by dividing the track in equally-sized cells. They're angular sections in the middle of which a magnetic orientation inversion may be present. From a hardware point of view the cells are seen as durations, which combined with the floppy rotation give the section. For instance the standard MFM cell size for a 3" double-density floppy is 2us, which combined with the also standard 300 rpm rotational speed gives an angular size of 1/100000th of a turn. Another way of saying it is that there are 100K cells in a 3" DD track.

In every cell there may or may not be a magnetic orientation transition, e.g. a pulse coming from (reading) or going to (writing) the floppy drive. A cell with a pulse is traditionally noted '1', and one without '0'. Two constraints apply to the cell contents though. First, pulses must not be too close together or they'll blur each-other and/or be filtered out. The limit is slightly better than 1/50000th of a turn for single and double density floppies, half that for HD floppies, and half that again for ED floppies with perpendicular recording. Second, they must not be too away from each other or either the AGC is going to get wonky and introduce phantom pulses or the controller is going to lose sync and get a wrong timing on the cells on reading. Conservative rule of thumb is not to have more than three consecutive '0' cells.

Of course protections play with that to make formats not reproducible by the system controller, either breaking the three-zeroes rule or playing with the cells durations/sizes.

Bit encoding is then the art of transforming raw data into a cell 0/1 configuration that respects the two constraints.

2.3.1.2. FM encoding

The very first encoding method developed for floppies is called Frequency Modulation, or FM. The cell size is set at slightly over the physical limit, e.g. 4us. That means it is possible to reliably have consecutive '1' cells. Each bit is encoded on two cells:

- the first cell, called the clock bit, is '1'
- the second cell, called data bit, is the bit

Since every other cell at least is '1' there is no risk of going over three zeroes.

The name Frequency Modulation simply derives from the fact that a 0 is encoded with one period of a 125Khz pulse train while a 1 is two periods of a 250Khz pulse train.

2.3.1.3. MFM encoding

The FM encoding has been superseded by the Modified Frequency Modulation encoding, which can cram exactly twice as much data on the same surface, hence its other name of "double density". The cell size is set at slightly over half the physical limit, e.g. 2us usually. The constraint means that two '1' cells must be separated by at least one '0' cell. Each bit is once again encoded on two cells:

- the first cell, called the clock bit, is '1' if both the previous and current data bits are 0, '0' otherwise
- the second cell, called data bit, is the bit

The minimum space rule is respected since a '1' clock bit is by definition surrounded by two '0' data bits, and a '1' data bit is surrounded by two '0' clock bits. The longest '0'-cell string possible is when encoding 101 which gives x10001, respecting the maximum of three zeroes.

2.3.1.4. GCR encodings

Group Coded Recording, or GCR, encodings are a class of encodings where strings of bits at least nibble-size are encoded into a given cell stream given by a table. It has been used in particular by the Apple II, the Mac and the C64, and each system has its own table, or tables.

2.3.1.5. Other encodings

Other encodings exist, like M2FM, but they're very rare and system-specific.

2.3.1.6. Reading back encoded data

Writing encoded data is easy: you only need a clock at the appropriate frequency and send or not a pulse on the clock edges. Reading back the data is where the fun is. Cells are a logical construct and not a physical measurable entity. Rotational speeds very around the defined one ($\pm$2% is not rare), and local perturbations (air turbulence, surface distance…) make the instantaneous speed very variable in general. So to extract the cell values stream, the controller must dynamically synchronize with the pulse train that the floppy head picks up. The principle is simple: a cell-sized duration window is built within which the presence of at least one pulse indicates the cell is a '1', and the absence of any a '0'. After reaching the end of the window, the starting time is moved appropriately to try to keep the observed pulse at the exact middle of the window. This allows the phase to be corrected on every '1' cell, making the synchronization work if the rotational speed is not too off. Subsequent generations of controllers used Phase-Locked Loops (PLLs) which vary both phase and window duration to adapt better to inaccurate rotational speeds, usually with a tolerance of $\pm$15%.

Once the cell data stream is extracted, decoding depends on the encoding. In the FM and MFM case the only question is to recognize data bits from clock bits, while in GCR the start position of the first group should be found. That second level of synchronization is handled at a higher level using patterns not found in a normal stream.

2.3.2. Sector-level organization

Floppies have been designed for read/write random access to reasonably sized blocks of data. Track selection allows for a first level of random access and sizing, but the ~6K of a double density track would be too big a block to handle. 256/512 bytes are considered a more appropriate value. To that end data on a track is organized as a series of (sector header, sector data) pairs where the sector header indicates important information like the sector number and size, and the sector data contains the data. Sectors have to be broken in two parts because while reading is easy, read the header then read the data if you want it, writing requires reading the header to find the correct place then once that is done switching on the writing head for the data. Starting writing is not instantaneous and will not be perfectly phase-aligned with the read header, so space for synchronization is required between header and data.

In addition somewhere in the sector header and in the sector data are pretty much always added some kind of checksum allowing to know whether the data was damaged or not.

FM and MFM have (not always used) standard sector layout methods.

2.3.2.1. FM sector layout

The standard "PC" track/sector layout for FM is as such:

- A number of FM-encoded 0xff (usually 40)
- 6 FM-encoded 0x00 (giving a steady 125KHz pulse train)
- The 16-cell stream 1111011101111010 (f77a, clock 0xd7, data 0xfc)
- A number of FM-encoded 0xff (usually 26, very variable)

Then for each sector: - 6 FM-encoded 0x00 (giving a steady 125KHz pulse train)

- The 16-cell stream 1111010101111110 (f57e, clock 0xc7, data 0xfe)
- Sector header, e.g. FM encoded track, head, sector, size code and two bytes of crc
- 11 FM-encoded 0xff
- 6 FM-encoded 0x00 (giving a steady 125KHz pulse train)
- The 16-cell stream 1111010101101111 (f56f, clock 0xc7, data 0xfb)
- FM-encoded sector data followed by two bytes of crc
- A number of FM-encoded 0xff (usually 48, very variable)

The track is finished with a stream of '1' cells.

The 125KHz pulse trains are used to lock the PLL to the signal correctly. The specific 16-cells streams allow to distinguish between clock and data bits by providing a pattern that is not supposed to happen in normal FM-encoded data. In the sector header track numbers start at 0, heads are 0/1 depending on the size, sector numbers usually start at 1 and size code is 0 for 128 bytes, 1 for 256, 2 for 512, etc.

The CRC is a cyclic redundancy check of the data bits starting with the mark just after the pulse train using polynom 0x11021.

The Western Digital-based controllers usually get rid of everything but some 0xff before the first sector and allow a better use of space as a result.

2.3.2.2. MFM sector layout

The standard "PC" track/sector layout for MFM is as such:

- A number of MFM-encoded 0x4e (usually 80)
- 12 FM-encoded 0x00 (giving a steady 250KHz pulse train)
- 3 times the 16-cell stream 0101001000100100 (5224, clock 0x14, data 0xc2)
- The MFM-encoded value 0xfc
- A number of MFM-encoded 0x4e (usually 50, very variable)

Then for each sector:

- 12 FM-encoded 0x00 (giving a steady 250KHz pulse train)
- 3 times the 16-cell stream 0100010010001001 (4489, clock 0x0a, data 0xa1)
- Sector header, e.g. MFM-encoded 0xfe, track, head, sector, size code and two bytes of crc
- 22 MFM-encoded 0x4e
- 12 MFM-encoded 0x00 (giving a steady 250KHz pulse train)
- 3 times the 16-cell stream 0100010010001001 (4489, clock 0x0a, data 0xa1)
- MFM-encoded 0xfb, sector data followed by two bytes of crc
- A number of MFM-encoded 0x4e (usually 84, very variable)

The track is finished with a stream of MFM-encoded 0x4e.

The 250KHz pulse trains are used to lock the PLL to the signal correctly. The cell pattern 4489 does not appear in normal MFM-encoded data and is used for clock/data separation.

As for FM, the Western Digital-based controllers usually get rid of everything but some 0x4e before the first sector and allow a better use of space as a result.

2.3.2.3. Formatting and write splices

To be usable, a floppy must have the sector headers and default sector data written on every track before using it. The controller starts writing at a given place, often the index pulse but on some systems whenever the command is sent, and writes until a complete turn is done. That's called formatting the floppy. At the point where the writing stops there is a synchronization loss since there is no chance the cell stream clock warps around perfectly. This brutal phase change is called a write splice, specifically the track write splice. It is the point where writing should start if one wants to raw copy the track to a new floppy.

Similarly two write splices are created when a sector is written at the start and end of the data block part. They're not supposed to happen on a mastered disk though, even if there are some rare exceptions.

12.12.3 3. The new implementation

3.1. Floppy disk representation

The floppy disk contents are represented by the class floppy_image. It contains information of the media type and a representation of the magnetic state of the surface.

The media type is divided in two parts. The first half indicates the physical form factor, i.e. all medias with that form factor can be physically inserted in a reader that handles it. The second half indicates the variants which are usually detectable by the reader, such as density and number of sides.

Track data consists of a series of 32-bits lsb-first values representing magnetic cells. Bits 0-27 indicate the absolute position of the start of the cell (not the size), and bits 28-31 the type. Type can be:

- 0, MG_A -> Magnetic orientation A
- 1, MG_B -> Magnetic orientation B
- 2, MG_N -> Non-magnetized zone (neutral)
- 3, MG_D -> Damaged zone, reads as neutral but cannot be changed by writing

The position is in angular units of 1/200,000,000th of a turn. It corresponds to one nanosecond when the drive rotates at 300 rpm.

The last cell implicit end position is of course 200,000,000.

Unformatted tracks are encoded as zero-size.

The "track splice" information indicates where to start writing if you try to rewrite a physical disk with the data. Some preservation formats encode that information, it is guessed for others. The write track function of fdcs should set it. The representation is the angular position relative to the index.

3.2. Converting to and from the internal representation

3.2.1. Class and interface

We need to be able to convert on-disk formats of the floppy data to and from the internal representation. This is done through classes derived from floppy_image_format_t. The interface to be implemented includes: - **name()** gives the short name of the on-disk format

- **description()** gives a short description of the format
- **extensions()** gives a comma-separated list of file name extensions found for that format
- **supports_save()** returns true is converting to that external format is supported
- **identify(file, form factor)** gives a 0-100 score for the file to be of that format:
  - **0** = not that format
  - **100** = certainly that format
  - **50** = format identified from file size only
- **load(file, form factor, floppy_image)** loads an image and converts it into the internal representation
- **save(file, floppy_image)** (if implemented) converts from the internal representation and saves an image

All of these methods are supposed to be stateless.

3.2.2. Conversion helper methods

A number of methods are provided to simplify writing the converter classes.

3.2.2.1. Load-oriented conversion methods

generate_track_from_bitstream(track number,
head number,
UINT8 *cell stream,
int cell count,
floppy image)

Takes a stream of cell types (0/1), MSB-first, converts it to the internal format and stores it at the given track and head in the given image.

generate_track_from_levels(track number,
head number,
UINT32 *cell levels,
int cell count,
splice position,
floppy image)

Takes a variant of the internal format where each value represents a cell, the position part of the values is the size of the cell and the level part is MG_0, MG_1 for normal cell types, MG_N, MG_D for unformatted/damaged cells, and MG_W for Dungeon-Master style weak bits. Converts it into the internal format. The sizes are normalized so that they total to a full turn.

normalize_times(UINT32 *levels,
int level_count)

Takes an internal-format buffer where the position part represents angle until the next change and turns it into a normal positional stream, first ensuring that the total size is normalized to a full turn.

3.2.2.2. Save-oriented conversion methods

generate_bitstream_from_track(track number,
head number,
base cell size,
UINT8 *cell stream,
int &cell_stream_size,
floppy image)

Extract a cell 0/1 stream from the internal format using a PLL setup with an initial cell size set to 'base cell size' and a +/- 25% tolerance.

struct desc_xs { int track, head, size; const UINT8 *data }
extract_sectors_from_bitstream_mfm_pc(...)
extract_sectors_from_bitstream_fm_pc(const UINT8 *cell stream,
int cell_stream_size,
desc_xs *sectors,
UINT8 *sectdata,
int sectdata_size)

Extract standard mfm or fm sectors from a regenerated cell stream. Sectors must point to an array of 256 desc_xs.

An existing sector is recognizable by having ->data non-null. Sector data is written in sectdata up to sectdata_size bytes.

get_geometry_mfm_pc(...)
get_geometry_fm_pc(floppy image,
base cell size,
int &track_count,
int &head_count,
int &sector_count)

Extract the geometry (heads, tracks, sectors) from a pc-ish floppy image by checking track 20.

get_track_data_mfm_pc(...)
get_track_data_fm_pc(track number,
head number,
floppy image,
base cell size,
sector size,
sector count,
UINT8 *sector data)

Extract what you'd get by reading in order 'sector size'-sized sectors from number 1 to sector count and put the result in sector data.

3.3. Floppy drive

The class floppy_image_interface simulates the floppy drive. That includes a number of control signals, reading, and writing. Control signal changes must be synchronized, e.g. fired off a timer to ensure the current time is the same for all devices.

3.3.1. Control signals

Due to the way they're usually connected to CPUs (e.g. directly on an I/O port), the control signals work with physical instead of logical values. Which means than in general 0 means active, 1 means inactive. Some signals also have a callback associated called when they change.

mon_w(state) / mon_r()

Motor on signal, rotates on 0.

idx_r() / setup_index_pulse_cb(cb)

Index signal, goes 0 at start of track for about 2ms. Callback is synchronized. Only happens when a disk is in and the motor is running.

ready_r() / setup_ready_cb(cb)

Ready signal, goes to 1 when the disk is removed or the motor stopped. Goes to 0 after two index pulses.

wpt_r() / setup_wpt_cb(cb)

Write protect signal (1 = readonly). Callback is unsynchronized.

dskchg_r()

Disk change signal, goes to 1 when a disk is change, goes to 0 on track change.

dir_w(dir)

Selects track stepping direction (1 = out = decrease track number).

stp_w(state)

Step signal, moves by one track on 1->0 transition.

trk00_r()

Track 0 sensor, returns 0 when on track 0.

ss_w(ss) / ss_r()

Side select

3.3.2. Read/write interface

The read/write interface is designed to work asynchronously, e.g. somewhat independently of the current time.

get_next_transition(from_when)

Returns the attotime of the next flux transition after *from_when*. Returns `attotime::never` when there is no upcoming transition.

write_start(when)

Activates the write gate at time *when*. No effect when no disk is loaded, the motor is off or the media is write-protected.

write_flux_change(when)

Records a single flux transition at time *when*. Buffered until a flush or end. If *when* is not after the most recently buffered transition, later entries are discarded first; this handles speculative re-execution transparently.

write_end(when)

Deactivates the write gate. Buffered transitions earlier than *when* are flushed to the track with the write span ending at *when*.

write_flush(when)

Commits buffered transitions earlier than *when* without deactivating the write gate. Transitions at or after *when* are retained for a subsequent flush or end.

set_write_splice(when)

Sets the track write-splice position to the angular position corresponding to *when*.

12.13 The new SCSI subsystem

12.13.1 Introduction

The **nscsi** subsystem was created to allow an implementation to be closer to the physical reality, making it easier (hopefully) to implement new controller chips from the documentations.

12.13.2 Global structure

Parallel SCSI is built around a symmetric bus to which a number of devices are connected. The bus is composed of 9 control lines (for now, later SCSI versions may have more) and up to 32 data lines (but currently implemented chips only handle 8). All the lines are open collector, which means that either one or multiple chip connect the line to ground and the line, of course, goes to ground, or no chip drives anything and the line stays at Vcc. Also, the bus uses inverted logic, where ground means 1. SCSI chips traditionally work in logical and not physical levels, so the nscsi subsystem also works in logical levels and does a logical-or of all the outputs of the devices.

Structurally, the implementation is done around two main classes: **nscsi_bus_devices** represents the bus, and **nscsi_device** represents an individual device. A device only communicate with the bus, and the bus takes care of transparently handling the device discovery and communication. In addition the **nscsi_full_device** class proposes a SCSI device with the SCSI protocol implemented making building generic SCSI devices like hard drives or CD-ROM readers easier.

12.13.3 Plugging in a SCSI bus in a driver

The nscsi subsystem leverages the slot interfaces and the device naming to allow for a configurable yet simple bus implementation.

First you need to create a list of acceptable devices to plug on the bus. This usually comprises of **cdrom**, **harddisk** and the controller chip. For instance:

```
static SLOT_INTERFACE_START( next_scsi_devices )
    SLOT_INTERFACE("cdrom", NSCSI_CDROM)
    SLOT_INTERFACE("harddisk", NSCSI_HARDDISK)
    SLOT_INTERFACE_INTERNAL("ncr5390", NCR5390)
SLOT_INTERFACE_END
```

The **_INTERNAL** interface indicates a device that is not user-selectable, which is useful for the controller.

Then in the machine config (or in a fragment config) you need to first add the bus, and then the (potential) devices as sub-devices of the bus with the SCSI ID as the name. For instance you can use:

```
MCFG_NSCSI_BUS_ADD("scsibus")
MCFG_NSCSI_ADD("scsibus:0", next_scsi_devices, "cdrom", 0, 0, 0, false)
MCFG_NSCSI_ADD("scsibus:1", next_scsi_devices, "harddisk", 0, 0, 0, false)
MCFG_NSCSI_ADD("scsibus:2", next_scsi_devices, 0, 0, 0, 0, false)
MCFG_NSCSI_ADD("scsibus:3", next_scsi_devices, 0, 0, 0, 0, false)
MCFG_NSCSI_ADD("scsibus:4", next_scsi_devices, 0, 0, 0, 0, false)
MCFG_NSCSI_ADD("scsibus:5", next_scsi_devices, 0, 0, 0, 0, false)
MCFG_NSCSI_ADD("scsibus:6", next_scsi_devices, 0, 0, 0, 0, false)
MCFG_NSCSI_ADD("scsibus:7", next_scsi_devices, "ncr5390", 0, &next_ncr5390_interface, 10000000, true)
```

That configuration puts as default a CD-ROM reader on SCSI ID 0 and a hard drive on SCSI ID 1, and forces the controller on ID 7. The parameters of add are:

- device tag, comprised of bus-tag:scsi-id
- the list of acceptable devices
- the device name as per the list, if one is to be there by default
- the device input config, if any (and there usually isn't one)
- the device configuration structure, usually for the controller only
- the frequency, usually for the controller only
- "**false**" for a user-modifiable slot, "**true**" for a fixed slot

The full device name, for mapping purposes, will be **bus-tag:scsi-id:device-type**, i.e. "*scsibus:7:ncr5390*" for our controller here.

12.13.4 Creating a new SCSI device using nscsi_device

The base class "**nscsi_device**" is to be used for down-to-the-metal devices, i.e. SCSI controller chips. The class provides three variables and one method. The first variable, **scsi_bus**, is a pointer to the **nscsi_bus_device**. The second, **scsi_refid**, is an opaque reference to pass to the bus on some operations. Finally, **scsi_id** gives the SCSI ID as per the device tag. It's written once at startup and never written or read afterwards, the device can do whatever it wants with the value or the variable.

The virtual method **scsi_ctrl_changed** is called when watched-for control lines change. Which lines are watched is defined through the bus.

The bus proposes five methods to access the lines. The read methods are **ctrl_r()** and **data_r()**. The meaning of the control bits are defined in the **S_*** enum of **nscsi_device**. The bottom three bits (**INP**, **CTL** and **MSG**) are setup so that masking with 7 (**S_PHASE_MASK**) gives the traditional numbers for the phases, which are also available with the **S_PHASE_*** enum.

Writing the data lines is done with **data_w(scsi_refid, value)**.

Writing the control lines is done with **ctrl_w(scsi_refid, value, mask-of-lines-to-change)**. To change all control lines in one call use **S_ALL** as the mask.

Of course, what is read is the logical-or of all of what is driven by all devices.

Finally, the method **ctrl_wait_w(scsi_id, value, mask-of-wait-lines-to-change)** allows to select which control lines are watched. The watch mask is per-device, and the device method **scsi_ctrl_changed** is called whenever a control line in the mask changes due to an action of another device (not itself, to avoid an annoying and somewhat useless recursion).

Implementing the controller is then just a matter of following the state machines descriptions, at least if they're available. The only part often not described is the arbitration/selection, which is documented in the SCSI standard though. For an initiator (which is what the controller essentially always is), it goes like this:

- wait for the bus to be idle
- assert the data line which number is your scsi_id (1 << scsi_id)
- assert the busy line
- wait the arbitration time
- check that the of the active data lines the one with the highest number is yours
  - if no, the arbitration was lost, stop driving anything and restart at the beginning
- assert the select line (at that point, the bus is yours)
- wait a short while
- keep your data line asserted, assert the data line which number is the SCSI ID of the target
- wait a short while
- assert the atn line if needed, de-assert busy
- wait for busy to be asserted or timeout
  - timeout means nobody is answering at that id, de-assert everything and stop
- wait a short while for de-skewing
- de-assert the data bus and the select line
- wait a short while

and then you're done, you're connected with the target until the target de-asserts the busy line, either because you asked it to or just to annoy you. The de-assert is called a disconnect.

The **ncr5390** is an example of how to use a two-level state machine to handle all the events.

12.13.5 Creating a new SCSI device using nscsi_full_device

The base class "**nscsi_full_device**" is used to create HLE-d SCSI devices intended for generic uses, like hard drives, CD-ROMs, perhaps scanners, etc. The class provides the SCSI protocol handling, leaving only the command handling and (optionally) the message handling to the implementation.

The class currently only support target devices.

The first method to implement is **scsi_command()**. That method is called when a command has fully arrived. The command is available in **scsi_cmdbuf[]**, and its length is in **scsi_cmdsize** (but the length is generally useless, the command first byte giving it). The 4096-bytes **scsi_cmdbuf** array is then freely modifiable.

In **scsi_command()**, the device can either handle the command or pass it up with **nscsi_full_device::scsi_command()**.

To handle the command, a number of methods are available:

- **get_lun(lua-set-in-command)** will give you the LUN to work on (the in-command one can be overridden by a message-level one).
- **bad_lun()** replies to the host that the specific LUN is unsupported.
- **scsi_data_in(buffer-id, size)** sends size bytes from buffer *buffer-id*
- **scsi_data_out(buffer-id, size)** receives size bytes into buffer *buffer-id*
- **scsi_status_complete(status)** ends the command with a given status.
- **sense(deferred, key)** prepares the sense buffer for a subsequent request-sense command, which is useful when returning a check-condition status.

The **scsi_data_*** and **scsi_status_complete** commands are queued, the command handler should call them all without waiting.

buffer-id identifies a buffer. 0, aka **SBUF_MAIN**, targets the **scsi_cmdbuf** buffer. Other acceptable values are 2 or more. 2+ ids are handled through the **scsi_get_data** method for read and **scsi_put_data** for write.

UINT8 device::scsi_get_data(int id, int pos) must return byte pos of buffer id, upcalling in **nscsi_full_device** for id < 2.

void device::scsi_put_data(int id, int pos, UINT8 data) must write byte pos in buffer id, upcalling in **nscsi_full_device** for id < 2.

scsi_get_data and **scsi_put_data** should do the external reads/writes when needed.

The device can also override **scsi_message** to handle SCSI messages other than the ones generically handled, and it can also override some of the timings (but a lot of them aren't used, beware).

A number of enums are defined to make things easier. The **SS_*** enum gives status returns (with **SS_GOOD** for all's well). The **SC_*** enum gives the scsi commands. The **SM_*** enum gives the SCSI messages, with the exception of identify (which is **80-ff**, doesn't really fit in an enum).

12.13.6 What's missing in scsi_full_device

- **Initiator support** We have no initiator device to HLE at that point.
- **Delays** A scsi_delay command would help giving more realistic timings to the CD-ROM reader in particular.
- **Disconnected operation** Would first require delays and in addition an emulated OS that can handle it.
- **16-bits wide operation** needs an OS and an initiator that can handle it.

12.13.7 What's missing in the ncr5390 (and probably future other controllers)

- **Bus free detection** Right now the bus is considered free if the controllers isn't using it, which is true. This may change once disconnected operation is in.
- **Target commands** We don't emulate (vs. HLE) any target yet.

12.14 The new 6502 family implementation

12.14.1 Introduction

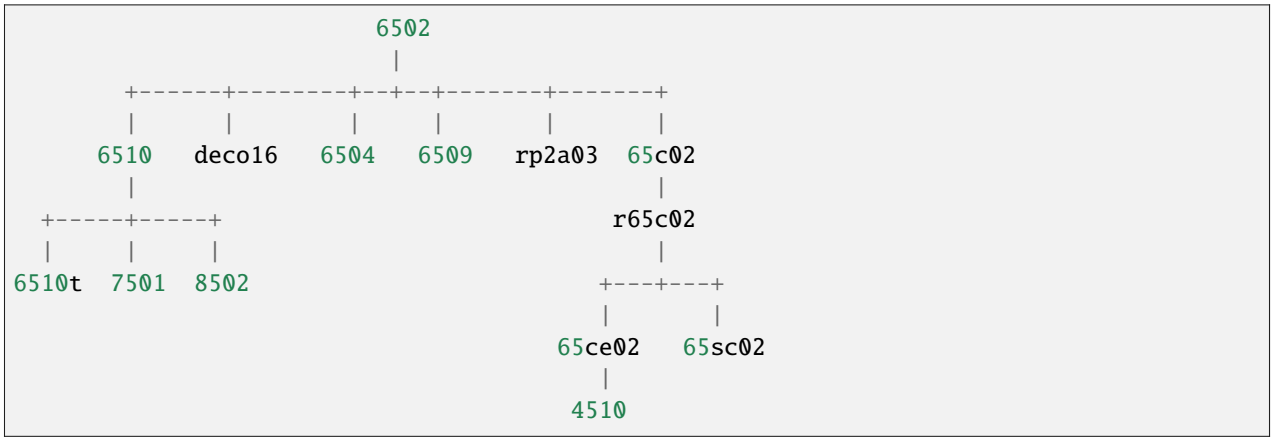
The new 6502 family implementation has been created to reach sub-instruction accuracy in observable behaviour. It is designed with 3 goals in mind:

- every bus cycle must happen at the exact time it would happen in a real CPU, and every access the real CPU does is done
- instructions can be interrupted at any time in the middle then restarted at that point transparently
- instructions can be interrupted even from within a memory handler for bus contention/wait states emulation purposes

Point 1 has been ensured through bisimulation with the gate-level simulation perfect6502. Point 2 has been ensured structurally through a code generator which will be explained in section 8. Point 3 is not done yet due to lack of support on the memory subsystem side, but section 9 shows how it will be handled.

12.14.2 The 6502 family

The MOS 6502 family has been large and productive. A large number of variants exist, varying on bus sizes, I/O, and even opcodes. Some offshoots (g65c816, hu6280) even exist that live elsewhere in the mame tree. The final class hierarchy is this:



The 6510 adds an up to 8 bits I/O port, with the 6510t, 7501 and 8502 being software-identical variants with different pin count (hence I/O count), die process (NMOS, HNMOS, etc.) and clock support.

The deco16 is a Deco variant with a small number of not really understood additional instructions and some I/O.

The 6504 is a pin and address-bus reduced version.

The 6509 adds internal support for paging.

The rp2a03 is the NES variant with the D flag disabled and sound functionality integrated.

The 65c02 is the very first cmos variant with some additional instructions, some fixes, and most of the undocumented instructions turned into nops. The R (Rockwell, but eventually produced by WDC too among others) variant adds a number of bitwise instructions and also stp and wai. The SC variant, used by the Lynx portable console, looks identical to the R variant. The 'S' probably indicates a static-ram-cell process allowing full DC-to-max clock control.

The 65ce02 is the final evolution of the ISA in this hierarchy, with additional instructions, registers, and removals of a lot of dummy accesses that slowed the original 6502 down by at least 25%. The 4510 is a 65ce02 with integrated MMU and GPIO support.

12.14.3 Usage of the classes

All the CPUs are standard modern CPU devices, with all the normal interaction with the device infrastructure. To include one of these CPUs in your driver you need to include "**CPU/m6502/<CPU>.h**" and then do a **MCFG_CPU_ADD("tag", <CPU>, clock)**.

6510 variants port I/O callbacks are setup through:
MCFG_<CPU>_PORT_CALLBACKS(READ8(type, read_method), WRITE8(type, write_method))

And the pullup and floating lines mask is given through:
MCFG_<CPU>_PORT_PULLS(pullups, floating)

In order to see all bus accesses on the memory handlers it is possible to disable accesses through the direct map (at a CPU cost, of course) with:
MCFG_M6502_DISABLE_DIRECT()

In that case, transparent decryption support is also disabled, everything goes through normal memory-map read/write calls. The state of the sync line is given by the CPU method **get_sync()**, making implementing the decryption in the handler possible.

Also, as for every executable device, the CPU method **total_cycles()** gives the current time in cycles since the start of the machine from the point of view of the CPU. Or, in other words, what is usually called the cycle number for the CPU when somebody talks about bus contention or wait states. The call is designed to be fast (no system-wide sync, no call to **machine.time()**) and is precise. Cycle number for every access is exact at the sub-instruction level.

The 4510 special nomap line is accessible through **get_nomap()**.

Other than these specifics, these are perfectly normal CPU classes.

12.14.4 General structure of the emulations

Each variant is emulated through up to 4 files:

- <CPU>.h = header for the CPU class
- <CPU>.c = implementation of most of the CPU class
- d<CPU>.lst = dispatch table for the CPU
- o<CPU>.lst = opcode implementation code for the CPU

The last two are optional. They're used to generate a **<CPU>.inc** file in the object directory which is included by the .c file.

At a minimum, the class must include a constructor and an enum picking up the correct input line ids. See m65sc02 for a minimalist example. The header can also include specific configuration macros (see m8502) and also the class can include specific memory accessors (more on these later, simple example in m6504).

If the CPU has its own dispatch table, the class must also include the declaration (but not definition) of **disasm_entries**, **do_exec_full** and **do_exec_partial**, the declaration and definition of **disasm_disassemble** (identical for all classes but refers to the class-specific **disasm_entries** array) and include the .inc file (which provides the missing definitions). Support for the generation must also be added to CPU.mak.

If the CPU has in addition its own opcodes, their declaration must be done through a macro, see e.g. m65c02. The .inc file will provide the definitions.

12.14.5 Dispatch tables

Each d<CPU>.lst is the dispatch table for the CPU. Lines starting with '#' are comments. The file must include 257 entries, the first 256 being opcodes and the 257th what the CPU should do on reset. In the 6502 irq and nmi actually magically call the "brk" opcode, hence the lack of specific description for them.

Entries 0 to 255, i.e. the opcodes, must have one of these two structures:

- opcode_addressing-mode
- opcode_middle_addressing-mode

Opcode is traditionally a three-character value. Addressing mode must be a 3-letter value corresponding to one of the DASM_* macros in m6502.h. Opcode and addressing mode are used to generate the disassembly table. The full entry text is used in the opcode description file and the dispatching methods, allowing for per-CPU variants for identical-looking opcodes.

An entry of "." was usable for unimplemented/unknown opcodes, generating "???" in the disassembly, but is not a good idea at this point since it will infloop in execute() if encountered.

12.14.6 Opcode descriptions

Each **o<CPU>.lst** file includes the CPU-specific opcodes descriptions. An opcode description is a series of lines starting by an opcode entry by itself and followed by a series of indented lines with code executing the opcode.

For instance the asl <absolute address> opcode looks like this:

```
asl_aba
        TMP = read_pc();
        TMP = set_h(TMP, read_pc());
        TMP2 = read(TMP);
        write(TMP, TMP2);
        TMP2 = do_asl(TMP2);
        write(TMP, TMP2);
        prefetch();
```

First the low part of the address is read, then the high part (**read_pc** is auto-incrementing). Then, now that the address is available the value to shift is read, then re-written (yes, the 6502 does that), shifted then the final result is written (do_asl takes care of the flags). The instruction finishes with a prefetch of the next instruction, as all non-CPU-crashing instructions do.

Available bus-accessing functions are:

read(adr)	standard read
read_direct(adr)	read from program space
read_pc()	read at the PC address and increment it
read_pc_noinc()	read at the PC address
read_9()	6509 indexed-y banked read
write(adr, val)	standard write
prefetch()	instruction prefetch
prefetch_noirq()	instruction prefetch without irq check

Cycle counting is done by the code generator which detects (through string matching) the accesses and generates the appropriate code. In addition to the bus-accessing functions a special line can be used to wait for the next event (irq or whatever). "**eat-all-cycles;**" on a line will do that wait then continue. It is used by wai_imp and stp_imp for the m65c02.

Due to the constraints of the code generation, some rules have to be followed:

- in general, stay with one instruction/expression per line
- there must be no side effects in the parameters of a bus-accessing function
- local variables lifetime must not go past a bus access. In general, it's better to leave them to helper methods (like **do_asl**) which do not do bus accesses. Note that "TMP" and "TMP2" are not local variables, they're variables of the class.
- single-line then or else constructs must have braces around them if they're calling a bus-accessing function

The per-opcode generated code are methods of the CPU class. As such they have complete access to other methods of the class, variables of the class, everything.

12.14.7 Memory interface

For better opcode reuse with the MMU/banking variants, a memory access subclass has been created. It's called **memory_interface**, declared in m6502_device, and provides the following accessors:

UINT8 read(UINT16 adr)	normal read
UINT8 read_sync(UINT16 adr)	opcode read with sync active (first byte of opcode)
UINT8 read_arg(UINT16 adr)	opcode read with sync inactive (rest of opcode)
void write(UINT16 adr, UINT8 val)	normal write

UINT8 read_9(UINT16 adr)	special y-indexed 6509 read, defaults to read()
void write_9(UINT16 adr, UINT8 val);	special y-indexed 6509 write, defaults to write()

Two implementations are given by default, one usual, **mi_default_normal**, one disabling direct access, **mi_default_nd**. A CPU that wants its own interface (see 6504 or 6509 for instance) must override device_start, intialize mintf there then call **init()**.

12.14.8 The generated code

A code generator is used to support interrupting and restarting an instruction in the middle. This is done through a two-level state machine with updates only at the boundaries. More precisely, inst_state tells you which main state you're in. It's equal to the opcode byte when 0-255, and 0xff00 means reset. It's always valid and used by instructions like rmb. inst_substate indicates at which step we are in an instruction, but it set only when an instruction has been interrupted. Let's go back to the asl <abs> code:

```
asl_aba
        TMP = read_pc();
        TMP = set_h(TMP, read_pc());
        TMP2 = read(TMP);
        write(TMP, TMP2);
```

```
        TMP2 = do_asl(TMP2);
        write(TMP, TMP2);
        prefetch();
```

The complete generated code is:

```
void m6502_device::asl_aba_partial()
{
switch(inst_substate) {
case 0:
        if(icount == 0) { inst_substate = 1; return; }
case 1:
        TMP = read_pc();
        icount--;
        if(icount == 0) { inst_substate = 2; return; }
case 2:
        TMP = set_h(TMP, read_pc());
        icount--;
        if(icount == 0) { inst_substate = 3; return; }
case 3:
        TMP2 = read(TMP);
        icount--;
        if(icount == 0) { inst_substate = 4; return; }
case 4:
        write(TMP, TMP2);
        icount--;
        TMP2 = do_asl(TMP2);
        if(icount == 0) { inst_substate = 5; return; }
case 5:
        write(TMP, TMP2);
        icount--;
        if(icount == 0) { inst_substate = 6; return; }
case 6:
        prefetch();
        icount--;
}
        inst_substate = 0;
}
```

One can see that the initial switch() restarts the instruction at the appropriate substate, that icount is updated after each access, and upon reaching 0 the instruction is interrupted and the substate updated. Since most instructions are started from the beginning a specific variant is generated for when inst_substate is known to be 0:

```
void m6502_device::asl_aba_full()
{
	if(icount == 0) { inst_substate = 1; return; }
	TMP = read_pc();
	icount--;
	if(icount == 0) { inst_substate = 2; return; }
	TMP = set_h(TMP, read_pc());
	icount--;
	if(icount == 0) { inst_substate = 3; return; }
	TMP2 = read(TMP);
	icount--;
	if(icount == 0) { inst_substate = 4; return; }
	write(TMP, TMP2);
	icount--;
	TMP2 = do_asl(TMP2);
	if(icount == 0) { inst_substate = 5; return; }
	write(TMP, TMP2);
	icount--;
	if(icount == 0) { inst_substate = 6; return; }
	prefetch();
	icount--;
}
```

That variant removes the switch, avoiding a costly computed branch and also an inst_substate write. There is in addition a fair chance that the decrement-test with zero pair is compiled into something efficient.

All these opcode functions are called through two virtual methods, **do_exec_full** and **do_exec_partial**, which are generated into a 257-entry switch statement. Pointers-to-methods being expensive to call, a virtual function implementing a switch has a fair chance of being better.

The execute main call ends up very simple:

```
void m6502_device::execute_run()
{
	if(inst_substate)
		do_exec_partial();

	while(icount > 0) {
		if(inst_state < 0x100) {
			PPC = NPC;
			inst_state = IR;
			if(machine().debug_flags & DEBUG_FLAG_ENABLED)
				debugger_instruction_hook(this, NPC);
		}
		do_exec_full();
```

```
        }
}
```

If an instruction was partially executed finish it (icount will then be zero if it still doesn't finish). Then try to run complete instructions. The NPC/IR dance is due to the fact that the 6502 does instruction prefetching, so the instruction PC and opcode come from the prefetch results.

12.14.9 Future bus contention/delay slot support

Supporting bus contention and delay slots in the context of the code generator only requires being able to abort a bus access when not enough cycles are available into icount, and restart it when cycles have become available again. The implementation plan is to:

- Have a delay() method on the CPU that removes cycles from icount. If icount becomes zero or less, having it throw a **suspend()** exception.
- Change the code generator to generate this:

```
void m6502_device::asl_aba_partial()
{
switch(inst_substate) {
case 0:
        if(icount == 0) { inst_substate = 1; return; }
case 1:
        try {
        TMP = read_pc();
        } catch(suspend) { inst_substate = 1; return; }
        icount--;
        if(icount == 0) { inst_substate = 2; return; }
case 2:
        try {
        TMP = set_h(TMP, read_pc());
        } catch(suspend) { inst_substate = 2; return; }
        icount--;
        if(icount == 0) { inst_substate = 3; return; }
case 3:
        try {
        TMP2 = read(TMP);
        } catch(suspend) { inst_substate = 3; return; }
        icount--;
        if(icount == 0) { inst_substate = 4; return; }
case 4:
        try {
        write(TMP, TMP2);
        } catch(suspend) { inst_substate = 4; return; }
        icount--;
        TMP2 = do_asl(TMP2);
        if(icount == 0) { inst_substate = 5; return; }
```

```
case 5:
        try {
        write(TMP, TMP2);
        } catch(suspend) { inst_substate = 5; return; }
        icount--;
        if(icount == 0) { inst_substate = 6; return; }
case 6:
        try {
        prefetch();
        } catch(suspend) { inst_substate = 6; return; }
        icount--;
}
        inst_substate = 0;
}
```

A modern try/catch costs nothing if an exception is not thrown. Using this the control will go back to the main loop, which will then look like this:

```
void m6502_device::execute_run()
{
        if(waiting_cycles) {
                icount -= waiting_cycles;
                waiting_cycles = 0;
        }

        if(icount > 0 && inst_substate)
                do_exec_partial();

        while(icount > 0) {
                if(inst_state < 0x100) {
                        PPC = NPC;
                        inst_state = IR;
                        if(machine().debug_flags & DEBUG_FLAG_ENABLED)
                                debugger_instruction_hook(this, NPC);
                }
                do_exec_full();
        }

        waiting_cycles = -icount;
        icount = 0;
}
```

A negative icount means that the CPU won't be able to do anything for some time in the future, because it's either waiting for the bus to be free or for a peripheral to answer. These cycles will be counted until elapsed and then normal processing will go on. It's important to note that the exception path only happens when the contention/wait state goes further than the scheduling slice of the CPU. That should not usually be the case, so the cost should be minimal.

12.14.10 Multi-dispatch variants

Some variants currently in the process of being supported change instruction set depending on an internal flag, either switching to a 16-bit mode or changing some register accesses to memory accesses. This is handled by having multiple dispatch tables for the CPU, the d<CPU>.lst not being 257 entries anymore but 256*n+1. The variable **inst_state_base** must select which instruction table to use at a given time. It must be a multiple of 256, and is in fact simply OR-ed to the first instruction byte to get the dispatch table index (aka inst_state).

12.14.11 Current TO-DO:

- Implement the bus contention/wait states stuff, but that requires support on the memory map side first.
- Integrate the I/O subsystems in the 4510
- Possibly integrate the sound subsystem in the rp2a03
- Add decent hookups for the Apple 3 madness

12.15 UML Instruction Reference

- *Introduction*
- *Special value types*
  - *Conditions*
  - *Flag masks*
  - *Floating point rounding modes*
- *Flow control*
  - *COMMENT*
  - *NOP*
  - *LABEL*
  - *HANDLE*
  - *HASH*
  - *JMP*
  - *CALLH*
  - *EXH*
  - *RET*
  - *HASHJMP*
  - *EXIT*
  - *CALLC*
  - *DEBUG*
  - *BREAK*

- *Status and control*
  - *SET*
  - *CARRY*
  - *SETFLGS*
  - *GETFLGS*
  - *SETFMOD*
  - *GETFMOD*
  - *RESTORE*
  - *SAVE*
- *Data movement*
  - *MOV*
  - *FMOV*
  - *FCOPYI*
  - *ICOPYF*
  - *LOAD*
  - *LOADS*
  - *STORE*
  - *FLOAD*
  - *FSTORE*
  - *GETEXP*
  - *MAPVAR*
  - *RECOVER*
- *Emulated memory access*
  - *READ*
  - *READM*
  - *WRITE*
  - *WRITEM*
  - *FREAD*
  - *FWRITE*
- *Integer arithmetic and logic*
  - *ADD*
  - *ADDC*
  - *SUB*
  - *SUBB*
  - *CMP*

  - *AND*
  - *TEST*
  - *OR*
  - *XOR*
  - *SEXT*
  - *LZCNT*
  - *TZCNT*
- *Integer multiply and divide*
  - *MULLW*
  - *MUL*
  - *DIV*
- *Integer shift and rotate*
  - *SHL*
  - *SHR*
  - *SAR*
  - *ROL*
  - *ROR*
  - *ROLC*
  - *RORC*
  - *BFX*
  - *ROLAND*
  - *ROLINS*
  - *BSWAP*
- *Floating point arithmetic*
  - *FADD*
  - *FSUB*
  - *FCMP*
  - *FMUL*
  - *FDIV*
  - *FNEG*
  - *FABS*
  - *FSQRT*
  - *FRECIP*
  - *FRSQRT*
  - *FRNDS*

- *Floating point conversion*
  - *FTOINT*
  - *FFRINT*
  - *FFRFLT*

12.15.1 Introduction

UML is the instruction set used by MAME's recompiler framework. Code running on emulated guest CPUs is translated to UML instructions, and back-ends convert the UML instructions to a form that can be executed or interpreted on the host system.

Many UML instruction have multiple instruction sizes. Integer instructions default to 32-bit size. Adding a `D` or `d` prefix to the mnemonic changes to 64-bit size (double word). Floating point instructions use the mnemonic prefix/suffix `FS` or `fs` for IEEE 754 32-bit format (single precision) or or the prefix/suffix `FD` or `fd` for IEEE 754 64-bit format (double precision).

12.15.2 Special value types

Conditions

Disassembly	Mnemonic	Usage	Flags tested
z	zero	COND_Z	Z
	equal	COND_E	
nz	not zero	COND_NZ	!Z
	not equal	COND_NE	
s	sign set	COND_S	S
ns	sign not set	COND_NS	!S
c	carry	COND_C	C
	below (unsigned)	COND_B	
nc	no carry	COND_NC	!C
	above or equal (unsigned)	COND_AE	
v	overflow (signed)	COND_V	V
nv	no overflow (signed)	COND_NV	!V
u	unordered	COND_U	U
nu	not unordered	COND_NU	!U
a	above (unsigned)	COND_A	!Z && !C
be	below or equal (unsigned)	COND_BE	Z || C
g	greater than (signed)	COND_G	!Z && (S == V)
le	less than or equal (signed)	COND_LE	Z || (S != V)
l	less than (signed)	COND_L	S != V
ge	greater than or equal (signed)	COND_GE	S == V

Flag masks

Value	Disassembly	Mnemonic	Usage
0x01	C	carry	FLAG_C
0x02	V	overflow	FLAG_V
0x04	Z	zero	FLAG_Z
0x08	S	sign	FLAG_S
0x10	U	unordered	FLAG_U
0x00			FLAGS_NONE
0x1f	USZVC		FLAGS_ALL

Floating point rounding modes

Value	Disassembly	Mnemonic	Usage	Mode
0	trunc	truncate	ROUND_TRUNC	Round toward zero
1	round	round	ROUND_ROUND	Round to nearest, round half to even
2	ceil	ceiling	ROUND_CEIL	Round toward positive infinity
3	floor	floor	ROUND_FLOOR	Round toward negative infinity
	default	default	ROUND_DEFAULT	Use the current default rounding mode

12.15.3 Flow control

COMMENT

Insert a comment into logged UML code.

Disassembly	Usage
comment string	UML_COMMENT(block, string);

Operands

string
: The comment text as a pointer to a NUL-terminated string. This must remain valid until code is generated for the block.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

NOP

No operation.

Disassembly	Usage
`nop`	`UML_NOP(block);`

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

LABEL

Associate a location with a label number local to the current generated code block. Label numbers must not be reused within a generated code block. The *JMP* instruction may be used to transfer control to the location associated with a label number.

Disassembly	Usage
`label  label`	`UML_LABEL(block, label);`

Operands

label (label number)
The label number to associate with the current location. A label number must not be used more than once within a generated code block.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

HANDLE

Mark a location as an entry point of a subroutine. Subroutines may be called using the *CALLH* and *EXH* instructions, and also by the *HASHJMP* <*umlinst-hashjmp*> if no location is associated with the specified mode and emulated program counter.

Disassembly	Usage
`handle  handle`	`UML_HANDLE(block, handle);`

Operands

handle (code handle)
The code handle to bind to the current location. The handle must already be allocated, and must not have been bound since the last generated code reset (all handles are implicitly unbound when resetting the generated code cache).

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

HASH

Associate a location with the specified mode and emulated program counter values. The *HASHJMP* instruction may be used to transfer control to the location associated with a mode and emulated program counter value. HASH instructions are not permitted in invariant code blocks.

This is usually used to mark the location of the generated code for an emulated instruction or sequence of instructions.

Disassembly	Usage
`hash mode,pc`	`UML_HASH(block, mode, pc);`

Operands

mode (32-bit – immediate, map variable)
The mode to associate with the current location in the generated code. Must be greater than or equal to zero and less than the number of modes specified when creating the recompiler context.

pc (32-bit – immediate, map variable)
The emulated program counter value to associate with the current location in the generated code.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

JMP

Jump to the location associated with a label number within the current block.

Disassembly	Usage
`jmp label`	`UML_JMP(block, label);`
`jmp label,cond`	`UML_JMPc(block, cond, label);`

Operands

label (label number)
The label number associated with the location to jump to in the current generated code block. The label number must be associated with a location in the generated code block before the block is finalised.

cond (condition)
If supplied, a condition that must be met to jump to the specified label. If the condition is not met, execution will continue with the following instruction.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

CALLH

Call the subroutine beginning at the specified code handle.

Disassembly	Usage
`callh handle`	`UML_CALLH(block, handle);`
`callh handle,cond`	`UML_CALLHc(block, cond, handle);`

Operands

handle (code handle)
Handle located at the entry point of the subroutine to call. The handle must already be allocated but does not need to be bound until the instruction is executed. Calling a handle that was unbound at code generation time may produce less efficient code than calling a handle that was already bound.

cond (condition)
If supplied, a condition that must be met for the subroutine to be called. If the condition is not met, the subroutine will not be called.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

EXH

Set the `EXP` register and call the subroutine beginning at the specified code handle. The `EXP` register is a 32-bit special function register that may be retrieved with the *GETEXP* instruction.

Disassembly	Usage
`exh handle,arg` `exh handle,arg,cond`	`UML_EXH(block, handle, arg);` `UML_EXHc(block, handle, arg, cond);`

Operands

handle (code handle)
Handle located at the entry point of the subroutine to call. The handle must already be allocated but does not need to be bound until the instruction is executed. Calling a handle that was unbound at code generation time may produce less efficient code than calling a handle that was already bound.

arg (32-bit – memory, integer register, immediate, map variable)
Value to store in the `EXP` register.

cond (condition)
If supplied, a condition that must be met for the subroutine to be called. If the condition is not met, the subroutine will not be called and the `EXP` register will not be modified.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

Simplification rules

- Immediate values for the `arg` operand are truncated to 32 bits.

RET

Return from a subroutine, transferring control to the instruction following the *CALLH* or *EXH* instruction used to call the subroutine. This instruction must only be used within generated code subroutines. The *EXIT* instruction must be used to exit from the generated code.

Disassembly	Usage
`ret` `ret  cond`	`UML_RET(block);` `UML_RETc(block, cond);`

Operands

cond (condition)
If supplied, a condition that must be met to return from the subroutine. If the condition is not met, execution will continue with the following instruction.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

HASHJMP

Unwind all nested generated code subroutine frames and transfer control to the location associated with the specified mode and emulated program counter values. If no location is associated with the specified mode and program counter values, call the subroutine beginning at the specified code handle. Note that all nested generated code subroutine frames are unwound in either case.

This is usually used to jump to the generated code corresponding to the emulated code at a particular address when it is not known to be in the current generated code block or when the mode changes.

Disassembly	Usage
`hashjmp mode,pc,handle`	`UML_HASHJMP(block, mode, pc, handle);`

Operands

mode (32-bit – memory, integer register, immediate, map variable)
The mode associated with the location in the generated code to transfer control to. Must be greater than or equal to zero and less than the number of modes specified when creating the recompiler context.

pc (32-bit – memory, integer register, immediate, map variable)
The emulated program counter value associated with the location in the generated code to transfer control to.

handle (code handle)
Handle located at the entry point of the subroutine to call if no location in the generated code is associated with the specified mode and emulated program counter values. The handle must already be allocated but does not need to be bound until the instruction is executed. Calling a handle that was unbound at code generation time may produce less efficient code than calling a handle that was already bound.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

EXIT

Exit from the generated code, returning control to the caller. May be used from within any level of nested subroutine calls in the generated code.

Disassembly	Usage
`exit arg,` `exit arg,,cond`	`UML_EXIT(block, arg);` `UML_EXITc(block, arg, cond);`

Operands

arg (32-bit – memory, integer register, immediate, map variable)
The value to return to the caller.

cond (condition)
If supplied, a condition that must be met to exit from the generated code. If the condition is not met, execution will continue with the following instruction.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

Simplification rules

- Immediate values for the `arg` operand are truncated to 32 bits.

CALLC

Call a C function with the signature `void (*)(T *)` or `void (*)(T &)` (i.e. a function with a single pointer or reference parameter that does not return a value).

Disassembly	Usage
`callc func,arg` `callc func,arg,cond`	`UML_CALLC(block, func, arg);` `UML_CALLCc(block, func, arg, cond);`

Operands

func (C function)
Function pointer to the function to call.

arg (memory)
Argument to pass to the function.

cond (condition)
If supplied, a condition that must be met for the function to be called. If the condition is not met, the function will not be called.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

DEBUG

Call the debugger instruction hook function if appropriate.

If the debugger is active, this should be executed before each emulated instruction. Any emulated CPU state kept in UML registers should be flushed to memory before executing this instruction and reloaded afterwards to ensure the debugger can display and modify values correctly.

Disassembly	Usage
`debug pc`	`UML_DEBUG(block, pc);`

Operands

pc (32-bit – memory, integer register, immediate, map variable)
The emulated program counter value to supply to the debugger instruction hook function.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

- Immediate values for the pc operand are truncated to 32 bits.

BREAK

Break into the host debugger if attached. Has no effect or crashes if no host debugger is attached depending on the host system and configuration. This is intended as a developer aid and should not be left in final code.

Disassembly	Usage
break	UML_BREAK(block);

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

No simplifications are applied to this instruction.

12.15.4 Status and control

SET

Conditionally set integer to zero or one depending on flags.

Disassembly	Usage
`set dst,cond` `dset dst,cond`	`UML_SETc(block, dst);` `UML_DSETc(block, cond, dst);`

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination that will be set to zero (0) if the condition is not met or one (1) if the condition is met.

cond (condition)
A condition to test. The destination will be set to zero (0) if the condition is not met or one (1) if the condition is met.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

Simplification rules

No simplifications are applied to this instruction.

CARRY

Set the carry flag.

Disassembly	Usage
`carry src,bit` `dcarry src,bit`	`UML_CARRY(block, src, bit);` `UML_DCARRY(block, src, bit);`

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
An integer value containing a bit to be copied to the carry flag.

bit (32-bit or 64-bit – memory, integer register, immediate, map variable)
The index of the bit to copy to the carry flag. Bits are numbered starting at zero for the least significant bit position, ascending toward the most significant bit position. Only the least significant five bits or six bits of this operand are used, depending on the instruction size.

Flags

carry (C)
Set to the value of the selected bit of the `src` operand.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

- Immediate values for the `src` operand are truncated to the instruction size.
- Immediate values for the `bit` operand are truncated to five or six bits for 32-bit or 64-bit operands, respectively.

SETFLGS

Set the flags arbitrarily.

Disassembly	Usage
`setflgs src`	`UML_SETFLGS(block, src);`

The five least significant bits of the value of `src` are copied to the flags. Bits are copied to the **carry (C)**, **overflow (V)**, **zero (Z)**, **sign (S)** and **unordered (U)** flags, starting from the least significant bit position.

Operands

src (32-bit – memory, integer register, immediate, map variable)
: The value to copy to the flags. Only the least significant five bits of this operand are used.

Flags

carry (C)
: Set to the value of bit 0 of the `src` operand, counting from the least significant bit starting from zero.

overflow (V)
: Set to the value of bit 1 of the `src` operand, counting from the least significant bit starting from zero.

zero (Z)
: Set to the value of bit 2 of the `src` operand, counting from the least significant bit starting from zero.

sign (S)
: Set to the value of bit 3 of the `src` operand, counting from the least significant bit starting from zero.

unordered (U)
: Set to the value of bit 4 of the `src` operand, counting from the least significant bit starting from zero.

Simplification rules

No simplifications are applied to this instruction.

GETFLGS

Copy flags.

Disassembly	Usage
`getflgs dst,mask`	`UML_GETFLGS(block, dst, mask);`

The flags corresponding to bit positions that are set in `mask` are copied to the corresponding bit positions in `dst`. Bit positions corresponding in `dst` that do not correspond to flags or that correspond to bit positions that are clear in `mask` are cleared.

Back-ends may be able to generate more efficient code if fewer bit positions are set in `mask`.

Operands

src (32-bit – memory, integer register)
The destination where the flags corresponding to bit positions that are set in the `mask` operand will be copied.

mask (flag mask – immediate, map variable)
The mask to specify which flags to copy. Only the least significant five bits of this operand are used.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

Simplification rules

No simplifications are applied to this instruction.

SETFMOD

Set the default floating point rounding mode. The default rounding mode is used for floating point arithmetic and for floating point to integer conversion when `ROUND_DEFAULT` is specified.

Disassembly	Usage
`setfmod round`	`UML_SETFMOD(block, round);`

Operands

round (32-bit – memory, integer register, immediate, map variable)
The rounding mode to set as the default. Only the two least significant bits of the value are used. Must be 0 (ROUND_TRUNC) to round toward zero, 1 (ROUND_ROUND) to round to nearest, 2 (ROUND_CEIL) to round toward positive infinity, or 3 (ROUND_FLOOR) to round toward negative infinity.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

No simplifications are applied to this instruction.

GETFMOD

Get the current default floating point rounding mode set by the most recent *SETFMOD* or *RESTORE* instruction.

Disassembly	Usage
`getfmod dst`	`UML_GETFMOD(block, dst);`

Note that the result of this instruction may not correspond to the actual effective default rounding mode between entering the generated code and executing the first *SETFMOD* or *RESTORE* instruction.

Operands

dst (32-bit – memory, integer register)
The destination where the current default rounding mode will be stored. Will be set to 0 (ROUND_TRUNC) for round toward zero, 1 (ROUND_ROUND) for round to nearest, 2 (ROUND_CEIL) for round toward positive infinity, or 3 (ROUND_FLOOR) for round toward negative infinity.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

No simplifications are applied to this instruction.

RESTORE

Set the contents of the UML integer and floating point registers, the contents of the `EXP` register, the flags and the default floating point rounding mode from a `drcuml_machine_state` structure.

Disassembly	Usage
`restore src`	`UML_RESTORE(block, src);`

Restores program-visible UML state from a structure in memory. The subroutine call stack and current instruction pointer are not changed. Execution continues with the following UML instruction.

src (`drcuml_machine_state` structure – memory)
The source that will be used to set program-visible UML machine state. This may be any host memory location accessible by the application. It is not restricted to the recompiler cache.

Flags

carry (C)
Set from the `src` operand.

overflow (V)
Set from the `src` operand.

zero (Z)
Set from the `src` operand.

sign (S)
Set from the `src` operand.

unordered (U)
Set from the `src` operand.

Simplification rules

No simplifications are applied to this instruction.

SAVE

Copy the contents of the UML integer and floating point registers, the contents of the `EXP` register, the flags and the default floating point rounding mode to a `drcuml_machine_state` structure.

Disassembly	Usage
`save dst`	`UML_SAVE(block, dst);`

Saves program-visible UML state to a structure in memory that can subsequently be restored using the *RESTORE* instruction. The subroutine call stack and current instruction pointer are not saved.

Note that the saved floating point rounding mode may not correspond to the actual effective default rounding mode between entering the generated code and executing the first *SETFMOD* or *RESTORE* instruction.

Operands

dst (`drcuml_machine_state` structure – memory)
: The destination where program-visible UML machine state will be saved. This may be any host memory location accessible by the application. It is not restricted to the recompiler cache.

Flags

carry (C)
: Undefined.

overflow (V)
: Undefined.

zero (Z)
: Undefined.

sign (S)
: Undefined.

unordered (U)
: Undefined.

Simplification rules

No simplifications are applied to this instruction.

12.15.5 Data movement

MOV

Copy an integer value.

Disassembly	Usage
`mov dst,src` `mov dst,src,cond` `dmov dst,src` `dmov dst,src,cond`	`UML_MOV(block, dst, src);` `UML_MOVc(block, cond, dst, src);` `UML_DMOV(block, dst, src);` `UML_DMOVc(block, cond, dst, src);`

Operands

dst (32-bit or 64-bit – memory, integer register)
: The destination where the value will be copied to.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
: The source value to copy.

cond (condition)
: If supplied, a condition that must be met to copy the value. If the condition is not met, the instruction will have no effect.

Flags

carry (C)
: Unchanged.

overflow (V)
: Unchanged.

zero (Z)
: Unchanged.

sign (S)
: Unchanged.

unordered (U)
: Unchanged.

Simplification rules

- Immediate values for the `src` operand are truncated to the instruction size.
- Converted to *NOP* if the `src` and `dst` operands refer to the same memory location or register and the instruction size is no larger than the destination size.

FMOV

Copy a floating point value. The binary value will be preserved even if it is not a valid representation of a floating point number.

Disassembly	Usage
`fsmov dst,src` `fsmov dst,src,cond` `fdmov dst,src` `fdmov dst,src,cond`	`UML_FSMOV(block, dst, src);` `UML_FSMOVc(block, cond, dst, src);` `UML_FDMOV(block, dst, src);` `UML_FDMOVc(block, cond, dst, src);`

Operands

dst (32-bit or 64-bit – memory, floating point register)
The destination where the value will be copied to.

src (32-bit or 64-bit – memory, floating point register)
The source value to copy.

cond (condition)
If supplied, a condition that must be met to copy the value. If the condition is not met, the instruction will have no effect.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

Simplification rules

- Converted to *NOP* if the `src` and `dst` operands refer to the same memory location or register.

FCOPYI

Reinterpret an integer value as a floating point value. The binary value will be preserved even if it is not a valid representation of a floating point number.

Disassembly	Usage
`fscopyi dst,src` `fdcopyi dst,src`	`UML_FSCOPYI(block, dst, src);` `UML_FDCOPYI(block, dst, src);`

Operands

dst (32-bit or 64-bit – memory, floating point register)
The destination where the value will be copied to.

src (32-bit or 64-bit – memory, integer register)
The source value to copy.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

ICOPYF

Reinterpret a floating point value as an integer value. The binary value will be preserved even if it is not a valid representation of a floating point number.

Disassembly	Usage
`icopyfs dst,src` `icopyfd dst,src`	`UML_ICOPYFS(block, dst, src);` `UML_ICOPYFD(block, dst, src);`

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the value will be copied to.

src (32-bit or 64-bit – memory, floating point register)
The source value to copy.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

LOAD

Load an unsigned integer value from a memory location with variable displacement. The value is zero-extended to the size of the destination. Host system rules for integer alignment must be followed.

Disassembly	Usage
`load dst,base,index,size_scale` `dload dst,base,index,size_scale`	`UML_LOAD(block, dst, base, index, size, ↪scale);` `UML_DLOAD(block, dst, base, index, size, ↪scale);`

This instruction can be used to read a value from any host memory location accessible by the application. It is not restricted to the recompiler cache.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the value read from memory will be stored.

base (memory)
The base address of the area of memory to read from.

index (32-bit – memory, integer register, immediate, map variable)
The displacement value added to the base address to calculate the address to read from. This value may be scaled by a factor of 1, 2, 4 or 8 depending on the `scale` operand. Note that this is always a 32-bit operand interpreted as a signed integer, irrespective of the instruction size.

size (access size)
The size of the value to read. Must be `SIZE_BYTE` (8-bit), `SIZE_WORD` (16-bit), `SIZE_DWORD` (32-bit) or `SIZE_QWORD` (64-bit). Note that this operand controls the size of the value read from memory while the instruction size sets the size of the `dst` operand.

scale (index scale)
The scale factor to apply to the `index` operand. Must be `SCALE_x1`, `SCALE_x2`, `SCALE_x4` or `SCALE_x8` to multiply by 1, 2, 4 or 8, respectively (shift left by 0, 1, 2 or 3 bits).

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

LOADS

Load a signed integer value from a memory location with variable displacement. The value is sign-extended to the size of the destination. Host system rules for integer alignment must be followed.

Disassembly	Usage
`loads dst,base,index,size_scale` `dloads dst,base,index,size_scale`	`UML_LOADS(block, dst, base, index, size, scale);` `UML_DLOADS(block, dst, base, index, size, scale);`

This instruction can be used to read a value from any host memory location accessible by the application. It is not restricted to the recompiler cache.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the value read from memory will be stored.

base (memory)
The base address of the area of memory to read from.

index (32-bit – memory, integer register, immediate, map variable)
The displacement value added to the base address to calculate the address to read from. This value may be scaled by a factor of 1, 2, 4 or 8 depending on the `scale` operand. Note that this is always a 32-bit operand interpreted as a signed integer, irrespective of the instruction size.

size (access size)
The size of the value to read. Must be `SIZE_BYTE` (8-bit), `SIZE_WORD` (16-bit), `SIZE_DWORD` (32-bit) or `SIZE_QWORD` (64-bit). Note that this operand controls the size of the value read from memory while the instruction size sets the size of the `dst` operand.

scale (index scale)
The scale factor to apply to the `index` operand. Must be `SCALE_x1`, `SCALE_x2`, `SCALE_x4` or `SCALE_x8` to multiply by 1, 2, 4 or 8, respectively (shift left by 0, 1, 2 or 3 bits).

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

STORE

Store an integer value to a location in memory with variable displacement. Host system rules for integer alignment must be followed.

Disassembly	Usage
`store base,index,src,size_scale` `dstore base,index,src,size_scale`	`UML_STORE(block, base, index, src, size, scale);` `UML_DSTORE(block, base, index, src, size, scale);`

This instruction can be used to write a value to any host memory location accessible by the application. It is not restricted to the recompiler cache.

Operands

base (memory)
The base address of the area of memory to write to.

index (32-bit – memory, integer register, immediate, map variable)
The displacement value added to the base address to calculate the address to write to. This value may be scaled by a factor of 1, 2, 4 or 8 depending on the `scale` operand. Note that this is always a 32-bit operand interpreted as a signed integer, irrespective of the instruction size.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
The value to write to memory.

size (access size)
The size of the value to write. Must be `SIZE_BYTE` (8-bit), `SIZE_WORD` (16-bit), `SIZE_DWORD` (32-bit) or `SIZE_QWORD` (64-bit). Note that this operand controls the size of the value written to memory while the instruction size sets the size of the `src` operand.

scale (index scale)
The scale factor to apply to the `index` operand. Must be `SCALE_x1`, `SCALE_x2`, `SCALE_x4` or `SCALE_x8` to multiply by 1, 2, 4 or 8, respectively (shift left by 0, 1, 2 or 3 bits).

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

FLOAD

Load a floating point value from a memory location with variable displacement. The binary value will be preserved even if it is not a valid representation of a floating point number. Host system rules for memory access alignment must be followed.

Disassembly	Usage
`fsload dst,base,index` `fdload dst,base,index`	`UML_FSLOAD(block, dst, base, index);` `UML_FDLOAD(block, dst, base, index);`

Operands

dst (32-bit or 64-bit – memory, floating point register)
The destination where the value read from memory will be stored.

base (memory)
The base address of the area of memory to read from.

index (32-bit – memory, integer register, immediate, map variable)
The displacement value added to the base address to calculate the address to read from. This value will be scaled by the instruction size (multiplied by 4 or 8). Note that this is always a 32-bit operand interpreted as a signed integer, irrespective of the instruction size.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

FSTORE

Store a floating point value to a memory location with variable displacement. The binary value will be preserved even if it is not a valid representation of a floating point number. Host system rules for memory access alignment must be followed.

Disassembly	Usage
`fsstore base,index,src` `fdstore base,index,src`	`UML_FSSTORE(block, base, index, src);` `UML_FDSTORE(block, base, index, src);`

Operands

base (memory)
The base address of the area of memory to write to.

index (32-bit – memory, integer register, immediate, map variable)
The displacement value added to the base address to calculate the address to write to. This value will be scaled by the instruction size (multiplied by 4 or 8). Note that this is always a 32-bit operand interpreted as a signed integer, irrespective of the instruction size.

src (32-bit or 64-bit – memory, floating point register)
The value to write to memory.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

GETEXP

Copy the value of the EXP register. The EXP register can be set using the *EXH* instruction.

Disassembly	Usage
`getexp  dst`	`UML_GETEXP(block, dst);`

Operands

dst (32-bit – memory, integer register)
The destination to copy the value of the EXP register to. Note that the EXP register can only hold a 32-bit value.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

MAPVAR

Set the value of a map variable starting at the current location in the current generated code block.

Disassembly	Usage
`mapvar  mapvar,value`	`UML_MAPVAR(block, mapvar, value);`

Operands

mapvar (map variable)
The map variable to set the value of.

value (32-bit – immediate, map variable)
The value to set the map variable to. Note that map variables can only hold 32-bit values.

Flags

carry (C)
Unchanged.

overflow (V)
Unchanged.

zero (Z)
Unchanged.

sign (S)
Unchanged.

unordered (U)
Unchanged.

RECOVER

Retrieve the value of a map variable at the location of the call instruction in the outermost generated code frame. This instruction should only be used from within a generated code subroutine. Results are undefined if this instruction is executed from outside any generated code subroutines.

Disassembly	Usage
`recover dst,mapvar`	`UML_RECOVER(block, dst, mapvar);`

Operands

dst (32-bit – memory, integer register)
The destination to copy the value of the map variable to. Note that map variables can only hold 32-bit values.

mapvar (map variable)
The map variable to retrieve the value of from the outermost generated code frame.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

12.15.6 Emulated memory access

READ

Read from an emulated address space. The access mask is implied to have all bits set.

Disassembly	Usage
`read dst,addr,space_size` `dread dst,addr,space_size`	`UML_READ(block, dst, addr, size, space);` `UML_DREAD(block, dst, addr, size, space);`

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the value read from the emulated address space will be stored.

addr (32-bit – memory, integer register, immediate, map variable)
The address to read from in the emulated address space. Note that this is always a 32-bit operand, irrespective of the instruction size.

size (access size)
The size of the emulated memory access. Must be `SIZE_BYTE` (8-bit), `SIZE_WORD` (16-bit), `SIZE_DWORD` (32-bit) or `SIZE_QWORD` (64-bit). Note that this operand controls the size of the emulated memory access while the instruction size sets the size of the `dst` operand.

space (address space number)
An integer identifying the address space to read from. May be `SPACE_PROGRAM`, `SPACE_DATA`, `SPACE_IO` or `SPACE_OPCODES` for one of the common CPU address spaces, or a non-negative integer cast to `memory_space`.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

- Immediate values for the `addr` operand are truncated to 32 bits.

READM

Read from an emulated address space with access mask specified.

Disassembly	Usage
`readm dst,addr,mask,space_size` `dreadm dst,addr,mask,space_size`	`UML_READM(block, dst, addr, mask, size,␣ ↪space);` `UML_DREADM(block, dst, addr, mask, size,␣ ↪space);`

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the value read from the emulated address space will be stored.

addr (32-bit – memory, integer register, immediate, map variable)
The address to read from in the emulated address space. Note that this is always a 32-bit operand, irrespective of the instruction size.

mask (32-bit or 64-bit – memory, integer register, immediate, map variable)
The access mask for the emulated memory access.

size (access size)
The size of the emulated memory access. Must be `SIZE_BYTE` (8-bit), `SIZE_WORD` (16-bit), `SIZE_DWORD` (32-bit) or `SIZE_QWORD` (64-bit). Note that this operand controls the size of the emulated memory access while the instruction size sets the size of the `dst` and `mask` operands.

space (address space number)
An integer identifying the address space to read from. May be `SPACE_PROGRAM`, `SPACE_DATA`, `SPACE_IO` or `SPACE_OPCODES` for one of the common CPU address spaces, or a non-negative integer cast to `memory_space`.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

- Immediate values for the `addr` operand are truncated to 32 bits.
- Immediate values for the `mask` operand are truncated to the access size.
- Converted to *READ* if the `mask` operand is an immediate value with all bits set.

WRITE

Write to an emulated address space. The access mask is implied to have all bits set.

Disassembly	Usage
`write addr,src,space_size` `dwrite addr,src,space_size`	`UML_WRITE(block, addr, src, size, space);` `UML_DWRITE(block, addr, src, size, space);`

Operands

addr (32-bit – memory, integer register, immediate, map variable)
The address to write to in the emulated address space. Note that this is always a 32-bit operand, irrespective of the instruction size.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
The value to write to the emulated address space.

size (access size)
The size of the emulated memory access. Must be `SIZE_BYTE` (8-bit), `SIZE_WORD` (16-bit), `SIZE_DWORD` (32-bit) or `SIZE_QWORD` (64-bit). Note that this operand controls the size of the emulated memory access while the instruction size sets the size of the `src` operand.

space (address space number)
An integer identifying the address space to read from. May be `SPACE_PROGRAM`, `SPACE_DATA`, `SPACE_IO` or `SPACE_OPCODES` for one of the common CPU address spaces, or a non-negative integer cast to `memory_space`.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

- Immediate values for the `addr` operand are truncated to 32 bits.
- Immediate values for the `src` operand are truncated to the access size.

WRITEM

Write to an emulated address space with access mask specified.

Disassembly	Usage
`writem  addr,src,mask,space_size` `dwritem addr,src,mask,space_size`	`UML_WRITEM(block, addr, src, mask, size,␣` `↪space);` `UML_DWRITEM(block, addr, src, mask, size,␣` `↪space);`

Operands

addr (32-bit – memory, integer register, immediate, map variable)
The address to write to in the emulated address space. Note that this is always a 32-bit operand, irrespective of the instruction size.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
The value to write to the emulated address space.

mask (32-bit or 64-bit – memory, integer register, immediate, map variable)
The access mask for the emulated memory access.

size (access size)
The size of the emulated memory access. Must be `SIZE_BYTE` (8-bit), `SIZE_WORD` (16-bit), `SIZE_DWORD` (32-bit) or `SIZE_QWORD` (64-bit). Note that this operand controls the size of the emulated memory access while the instruction size sets the size of the `src` and `mask` operands.

space (address space number)
An integer identifying the address space to read from. May be `SPACE_PROGRAM`, `SPACE_DATA`, `SPACE_IO` or `SPACE_OPCODES` for one of the common CPU address spaces, or a non-negative integer cast to `memory_space`.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

- Immediate values for the `addr` operand are truncated to 32 bits.
- Immediate values for the `src` and `mask` operands are truncated to the access size.
- Converted to *WRITE* if the `mask` operand is an immediate value with all bits set.

FREAD

Read a floating point value from an emulated address space. The binary value will be preserved even if it is not a valid representation of a floating point number. The access mask is implied to have all bits set.

Disassembly	Usage
`fsread dst,addr,space_size` `fdread dst,addr,space_size`	`UML_FSREAD(block, dst, addr, space);` `UML_FDREAD(block, dst, addr, space);`

Operands

dst (32-bit or 64-bit – memory, floating point register)
The destination where the value read from the emulated address space will be stored.

addr (32-bit – memory, integer register, immediate, map variable)
The address to read from in the emulated address space. Note that this is always a 32-bit operand, irrespective of the instruction size.

space (address space number)
An integer identifying the address space to read from. May be `SPACE_PROGRAM`, `SPACE_DATA`, `SPACE_IO` or `SPACE_OPCODES` for one of the common CPU address spaces, or a non-negative integer cast to `memory_space`.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

- Immediate values for the `addr` operand are truncated to 32 bits.

FWRITE

Write a floating point value to an emulated address space. The binary value will be preserved even if it is not a valid representation of a floating point number. The access mask is implied to have all bits set.

Disassembly	Usage
`fswrite addr,src,space_size` `fdwrite addr,src,space_size`	`UML_FSWRITE(block, addr, src, space);` `UML_FDWRITE(block, addr, src, space);`

Operands

addr (32-bit – memory, integer register, immediate, map variable)
The address to write to in the emulated address space. Note that this is always a 32-bit operand, irrespective of the instruction size.

src (32-bit or 64-bit – memory, floating point register)
The value to write to the emulated address space. will be stored.

space (address space number)
An integer identifying the address space to read from. May be `SPACE_PROGRAM`, `SPACE_DATA`, `SPACE_IO` or `SPACE_OPCODES` for one of the common CPU address spaces, or a non-negative integer cast to `memory_space`.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

- Immediate values for the `addr` operand are truncated to 32 bits.

12.15.7 Integer arithmetic and logic

ADD

Add two integers.

Disassembly	Usage
`add dst,src1,src2` `dadd dst,src1,src2`	`UML_ADD(block, dst, src1, src2);` `UML_DADD(block, dst, src1, src2);`

Calculates `dst = src1 + src2`.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the sum will be stored.

src1 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The first addend.

src2 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The second addend.

Flags

carry (C)
Set in the case of arithmetic carry out of the most significant bit, or cleared otherwise (unsigned overflow).

overflow (V)
Set in the case of signed two's complement overflow, or cleared otherwise.

zero (Z)
Set if the result is zero, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV*, *AND* or *OR* if the `src1` and `src2` operands are both immediate values and the carry and overflow flags are not required.
- Converted to *MOV* or *AND* if the `src1` operand or `src2` operand is the immediate value zero and the carry and overflow flags are not required.
- Immediate values for the `src1` and `src2` operands are truncated to the instruction size.
- If the `src2` and `dst` operands refer to the same register or memory location, the `src1` and `src2` operands are exchanged.
- If the `src1` operand is an immediate value and the `src2` operand is not an immediate value, the `src1` and `src2` operands are exchanged.

ADDC

Add two integers and the carry flag.

Disassembly	Usage
`addc dst,src1,src2` `daddc dst,src1,src2`	`UML_ADDC(block, dst, src1, src2);` `UML_DADDC(block, dst, src1, src2);`

Calculates `dst = src1 + src2 + C`.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the sum will be stored.

src1 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The first addend.

src2 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The second addend.

Flags

carry (C)
Set in the case of arithmetic carry out of the most significant bit, or cleared otherwise (unsigned overflow).

overflow (V)
Set in the case of signed two's complement overflow, or cleared otherwise.

zero (Z)
Set if the result is zero, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Immediate values for the `src1` and `src2` operands are truncated to the instruction size.
- If the `src2` and `dst` operands refer to the same register or memory location, the `src1` and `src2` operands are exchanged.
- If the `src1` operand is an immediate value and the `src2` operand is not an immediate value, the `src1` and `src2` operands are exchanged.

SUB

Subtract an integer from another integer.

Disassembly	Usage
`sub dst,src1,src2` `dsub dst,src1,src2`	`UML_SUB(block, dst, src1, src2);` `UML_DSUB(block, dst, src1, src2);`

Calculates `dst = src1 - src2`.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the difference will be stored.

src1 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The minuend (the value to subtract from).

src2 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The subtrahend (the value to subtract).

Flags

carry (C)
Set if the subtrahend is a larger unsigned value than the minuend, or cleared otherwise (unsigned overflow, or arithmetic borrow).

overflow (V)
Set in the case of signed two's complement overflow, or cleared otherwise.

zero (Z)
Set if the result is zero, or cleared otherwise (set if the minuend and subtrahend are equal, or cleared otherwise).

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV*, *AND* or *OR* if the `src1` and `src2` operands are both immediate values and the carry and overflow flags are not required.
- Converted to *MOV* or *AND* if the `src2` operand is the immediate value zero and the carry and overflow flags are not required.
- Immediate values for the `src1` and `src2` operands are truncated to the instruction size.

SUBB

Subtract an integer and the carry flag from another integer.

Disassembly	Usage
`subb dst,src1,src2` `dsubb dst,src1,src2`	`UML_SUBB(block, dst, src1, src2);` `UML_DSUBB(block, dst, src1, src2);`

Calculates `dst = src1 - src2 - C`.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the difference will be stored.

src1 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The minuend (the value to subtract from).

src2 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The subtrahend (the value to subtract).

Flags

carry (C)
Set if the subtrahend plus the carry flag is a larger unsigned value than the minuend, or cleared otherwise (unsigned overflow, or arithmetic borrow).

overflow (V)
Set in the case of signed two's complement overflow, or cleared otherwise.

zero (Z)
Set if the result is zero, or cleared otherwise (set if the minuend is equal to the subtrahend plus the carry flag, or cleared otherwise).

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Immediate values for the `src1` and `src2` operands are truncated to the instruction size.

CMP

Compare two integers and set the flags as though they were subtracted.

Disassembly	Usage
`cmp src1,src2` `dcmp src1,src2`	`UML_CMP(block, src1, src2);` `UML_DCMP(block, src1, src2);`

Sets the flags based on calculating `src1 - src2` but discards the result of the subtraction.

Operands

src1 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The left-hand side value to compare, or the minuend (the value to subtract from).

src2 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The right-hand side value to compare, or the subtrahend (the value to subtract).

Flags

carry (C)
Set if the unsigned value of the `src1` operand is smaller than the unsigned value of the `src2` operand, or cleared otherwise.

overflow (V)
Set if subtracting the value of the `src2` operand from the value of the `src1` operand would result in two's complement overflow, or cleared otherwise.

zero (Z)
Set if the values of the `src1` and `src2` operands are equal, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result of subtracting the value of the `src2` operand from the value of the `src1` operand (set if the result would be a negative signed integer, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *NOP* if no flags are required.
- Immediate values for the `src1` and `src2` operands are truncated to the instruction size.

AND

Calculate the bitwise logical conjunction of two integers (result bits will be set if the corresponding bits are set in both inputs).

Disassembly	Usage
`and dst,src1,src2` `dand dst,src1,src2`	`UML_AND(block, dst, src1, src2);` `UML_DAND(block, dst, src1, src2);`

Calculates `dst = src1 & src2`.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the logical conjunction will be stored.

src1 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The first input.

src2 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The second input.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Set if the result is zero, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV* if the `src1` and `src2` operands refer to the same memory location or register, the `src1` and `src2` operands are both immediate values or one of them is an immediate value with all bits set or no bits set and flags are not required.
- Converted to *OR* if the `src1` and `src2` operands are both immediate values with all bits set and flags are required.
- Converted to *TEST* if the instruction size is 64 bits or the `dst` operand refers to a memory location, one of the `src1` and `src2` operands refer to the same memory location or register as `dst`, the other source operand refers to the same memory location or register or is an immediate value with all bits set, and flags are required.
- If the `src1` and `src2` operands are both immediate values, the conjunction is not zero and flags are required, `src1` is replaced with the conjunction and `src2` is set to an immediate value with all bits set.
- If the `src1` and `src2` operands are both immediate values and the conjunction is zero or either the `src1` or `src2` operand is the immediate value zero and flags are required, `src1` is set to refer to the same memory location or register as `dst` and `src2` is set to the immediate value zero.
- Immediate values for the `src1` and `src2` operands are truncated to the instruction size.
- If the `src2` and `dst` operands refer to the same register or memory location, the `src1` and `src2` operands are exchanged.
- If the `src1` operand is an immediate value and the `src2` operand is not an immediate value, the `src1` and `src2` operands are exchanged.

TEST

Set the flags based on the bitwise logical conjunction of two integers.

Disassembly	Usage
`test src1,src2` `dtest src1,src2`	`UML_TEST(block, src1, src2);` `UML_DTEST(block, src1, src2);`

Sets the flags based on calculating `src1 & src2` but discards the result of the conjunction.

Operands

src1 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The first input.

src2 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The second input.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Set if the result of the conjunction is zero, or cleared otherwise.

sign (S)
Set if the most significant bit is set in both inputs, or cleared otherwise (set if the both inputs are negative signed integers, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *NOP* if flags are not required.
- If the `src1` and `src2` operands are both immediate values and the bitwise logical conjunction is not zero, the `src1` operand is set to the conjunction and the `src2` operand is set to an immediate value with all bits set.
- If either of the `src1` and `src2` operands is the immediate value zero or the `src1` and `src2` operands are both immediate values and the bitwise logical conjunction is zero, the `src1` and `src2` operands are both set to the immediate value zero.
- If the `src1` and `src2` operands refer to the same memory location or register, the `src2` operand is set to an immediate value with all bits set. * Immediate values for the `src1` and `src2` operands are truncated to the instruction size.

- If the `src1` operand is an immediate value and the `src2` operand is not an immediate value, the `src1` and `src2` operands are exchanged.

OR

Calculate the bitwise logical inclusive disjunction of two integers (result bits will be set if the corresponding bits are set in either input).

Disassembly	Usage
`or dst,src1,src2` `dor dst,src1,src2`	`UML_OR(block, dst, src1, src2);` `UML_DOR(block, dst, src1, src2);`

Calculates `dst = src1 | src2`.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the logical inclusive disjunction will be stored.

src1 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The first input.

src2 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The second input.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Set if the result is zero, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV* if the `src1` and `src2` operands are both immediate values or one of the `src1` or `src2` operands is an immediate value with all bits set and flags are not required.
- Converted to *AND* if the `src1` and `src2` operands are both immediate values and the inclusive disjunction does not have all bits set and flags are required.
- Converted to *MOV*, *AND* or *TEST* if the `src1` and `src2` operands refer to the same memory location or register or if one of the `src1` and `src2` operands is the immediate value zero.
- If one of the `src1` and `src2` operands is an immediate value with all bits set or the `src1` and `src2` operands are both immediate values and the inclusive disjunction has all bits set and flags are required, `src1` is set to refer to the same memory location or register as `dst` and `src2` is set to an immediate value with all bits set.
- Immediate values for the `src1` and `src2` operands are truncated to the instruction size.
- If the `src2` and `dst` operands refer to the same register or memory location, the `src1` and `src2` operands are exchanged.
- If the `src1` operand is an immediate value and the `src2` operand is not an immediate value, the `src1` and `src2` operands are exchanged.

XOR

Calculate the bitwise logical exclusive disjunction of two integers (result bits will be set if the corresponding bit is set in one input and unset in the other input).

Disassembly	Usage
`xor dst,src1,src2` `dxor dst,src1,src2`	`UML_XOR(block, dst, src1, src2);` `UML_DXOR(block, dst, src1, src2);`

Calculates `dst = src1 ^ src2`.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the logical exclusive disjunction will be stored.

src1 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The first input.

src2 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The second input.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Set if the result is zero, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV*, *AND*, *TEST* or *OR* if the `src1` and `src2` operands are both immediate values, if one of the `src1` and `src2` operands is the immediate value zero or if the `src1` and `src2` operands refer to the same memory location or register.

SEXT

Sign extend an integer value.

Disassembly	Usage
`sext dst,src,size` `dsext dst,src,size`	`UML_SEXT(block, dst, src, size);` `UML_DSEXT(block, dst, src, size);`

Sets `dst` to the value of `src` sign extended from the size specified by the `size` operand to the instruction size.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the sign extended value will be stored.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
The value to sign extend.

size (access size)
The size of the value to sign extend. Must be `SIZE_BYTE` (8-bit), `SIZE_WORD` (16-bit) or `SIZE_DWORD` (32-bit).

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Set if the result is zero, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV*, *AND* or *OR* if the `src` operand is an immediate value or if the `size` operand specifies a size no smaller than the instruction size.

LZCNT

Count the number of contiguous left-aligned zero bits in an integer (count leading zeroes).

Disassembly	Usage
`lzcnt dst,src` `dlzcnt dst,src`	`UML_LZCNT(block, dst, src);` `UML_DLZCNT(block, dst, src);`

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the result will be stored.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
The input value in which to count left-aligned zero bits.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Set if the result is zero, or cleared otherwise (set to the most significant bit of the input).

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV* or *AND* if the `src` operand is an immediate value.

TZCNT

Count the number of contiguous right-aligned zero bits in an integer (count trailing zeroes).

Disassembly	Usage
`tzcnt dst,src` `dtzcnt dst,src`	`UML_TZCNT(block, dst, src);` `UML_DTZCNT(block, dst, src);`

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the result will be stored.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
The input value in which to count right-aligned zero bits.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Set if the result is zero, or cleared otherwise (set to the least significant bit of the input).

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV* or *AND* if the `src` operand is an immediate value.

12.15.8 Integer multiply and divide

MULLW

Multiply two integer values.

Disassembly	Usage
`mululw  dst,src1,src2` `mulslw  dst,src1,src2` `dmululw dst,src1,src2` `dmulslw dst,src1,src2`	`UML_MULULW(block, dst, src1, src2);` `UML_MULSLW(block, dst, src1, src2);` `UML_DMULULW(block, dst, src1, src2);` `UML_DMULSLW(block, dst, src1, src2);`

Calculates `dst = src1 * src2` producing a result the same size as the inputs. MULULW and DMULULW take unsigned integer values as inputs and produce an unsigned integer value as a result, while MULSLW and DMULSLW take signed integer values as inputs and produce a signed integer value as a result. Note that the distinction between signed and unsigned values only affects the calculation of the overflow flag for this instruction. It does not affect the result of the multiplication.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the product will be stored.

src1 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The multiplicand (the value to multiply).

src2 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The multiplier (the value to multiply by).

Flags

carry (C)
Undefined.

overflow (V)
Set if the full result of the multiplication cannot be represented within the instruction size.

zero (Z)
Set if the result is zero, or cleared otherwise. Note that this is based on the possibly truncated result value, not the full result of the multiplication.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise). Note that this is based on the possibly truncated result value, not the full result of the multiplication.

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV*, *AND* or *OR* if the `src1` and `src2` operands are both immediate values or either the `src1` or `src2` operand is the immediate value zero or one and the overflow flag is not required.
- Immediate values for the `src1` and `src2` operands are truncated to the instruction size.
- If the `src2` and `dst` operands refer to the same register or memory location, the `src1` and `src2` operands are exchanged.
- If the `src1` operand is an immediate value and the `src2` operand is not an immediate value, the `src1` and `src2` operands are exchanged.

MUL

Multiply two integer values, possibly producing an extended result.

Disassembly	Usage
`mulu dst,edst,src1,src2` `muls dst,edst,src1,src2` `dmulu dst,edst,src1,src2` `dmuls dst,edst,src1,src2`	`UML_MULU(block, dst, edst, src1, src2);` `UML_MULS(block, dst, edst, src1, src2);` `UML_DMULU(block, dst, edst, src1, src2);` `UML_DMULS(block, dst, edst, src1, src2);`

Calculates `edst:dst = src1 * src2` if the `dst` and `edst` operands do not refer to the same register or memory location, or `dst = src1 * src2` if the `dst` and `edst` operands refer to the same register or memory location. MULU and DMULU take unsigned integer values as inputs and produce an unsigned integer value as a result, while MULS and DMULS take signed integer values as inputs and produce a signed integer value as a result.

Operands

dst (32-bit or 64-bit – memory, integer register)
: The destination where the least significant half of the full product will be stored.

edst (32-bit or 64-bit – memory, integer register)
: The destination where the most significant half of the full product will be stored if this operand does not refer to the same memory location or register as the `dst` operand. If this operand refers to the same memory location or register as the `dst` operand, the most significant half of the full product will be discarded, producing a result the same size as the inputs.

src1 (32-bit or 64-bit – memory, integer register, immediate, map variable)
: The multiplicand (the value to multiply).

src2 (32-bit or 64-bit – memory, integer register, immediate, map variable)
: The multiplier (the value to multiply by).

Flags

carry (C)
Undefined.

overflow (V)
Set if the full result of the multiplication cannot be represented within the instruction size.

zero (Z)
Set if the full result of the multiplication is zero, or cleared otherwise. Note that this is based on the full result of the multiplication even when the `dst` and `edst` operands refer to the same memory location or register, causing the result to be truncated.

sign (S)
Set to the value of the most significant bit of the full result of the multiplication (set if the result is a negative signed integer value, or cleared otherwise). Note that this is based on the full result of the multiplication even when the `dst` and `edst` operands refer to the same memory location or register, causing the result to be truncated.

unordered (U)
Undefined.

Simplification rules

- Converted to *MULLW* if the `dst` and `edst` operands refer to the same memory location or register and the zero and sign flags are not required.
- Converted to *MOV*, *AND* or *OR* if the `dst` and `edst` operands refer to the same memory location or register, the `src1` and `src2` operands are both immediate values or either the `src1` or `src2` operand is the immediate value zero, the most significant half of the full result of the multiplication is the sign extension of the least significant half or the sign flag is not required, and the overflow flag is not required.
- Converted to *MOV* or *AND* if the `dst` and `edst` operands refer to the same memory location or register, either the `src1` or `src2` operand is the immediate value one, signed multiplication is being performed or the sign flag is not required, and the overflow flag is not required.
- Immediate values for the `src1` and `src2` operands are truncated to the instruction size.
- If the `src1` operand is an immediate value and the `src2` operand is not an immediate value, the `src1` and `src2` operands are exchanged.

DIV

Divide an integer value by another integer value.

Disassembly	Usage
`divu dst,edst,src1,src2`	`UML_DIVU(block, dst, edst, src1, src2);`
`divs dst,edst,src1,src2`	`UML_DIVS(block, dst, edst, src1, src2);`
`ddivu dst,edst,src1,src2`	`UML_DDIVU(block, dst, edst, src1, src2);`
`ddivs dst,edst,src1,src2`	`UML_DDIVS(block, dst, edst, src1, src2);`

If the value of `src2` is not zero, the value of `src1` is divided by the value of `src2`, the quotient is stored in the memory location or register referred to by `dst`, and the remainder is stored in the memory location or register referred to by `edst` if the `dst` and `edst` operands do not refer to the same memory location or register.

If the value of `src2` is zero, the overflow flag is set and the values of the memory locations or registers referred to by the `dst` and `edst` operands are undefined.

DIVU and DDIVU take unsigned integer values as inputs and produce unsigned integer values as results, while DIVS and DDIVS take signed integer values as inputs and produce signed integer values as results.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the quotient will be stored if the value of the `src2` operand is not zero.

edst (32-bit or 64-bit – memory, integer register)
The destination where the value of the remainder will be stored if this operand does not refer to the same memory location or register as the `dst` operand and the value of the `src2` operand is not zero.

src1 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The dividend (the value to divide).

src2 (32-bit or 64-bit – memory, integer register, immediate, map variable)
The divisor (the value to divide by).

Flags

carry (C)
Undefined.

overflow (V)
Set if the divisor (the value of the `src2` operand) is zero, or cleared otherwise.

zero (Z)
Set if the divisor (the value of the `src2` operand) is not zero and the quotient is zero, or cleared otherwise.

sign (S)
Set to the most significant bit of the quotient (set if the quotient is a negative signed integer value) if the divisor (the value of the `src2` operand) is not zero, or cleared otherwise.

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV*, *AND* or *OR* if the `dst` and `edst` operands refer to the same memory location or register, the `src1` and `src2` operands are both immediate values or `src2` operand is the immediate value one, the `src2` operand is not the immediate value zero, and the overflow flag is not required.
- Immediate values for the `src1` and `src2` operands are truncated to the instruction size.

12.15.9 Integer shift and rotate

SHL

Shift an integer value to the left (toward the most significant bit position), shifting zeroes into the least significant bit.

Disassembly	Usage
`shl dst,src,count` `dshl dst,src,count`	`UML_SHL(block, dst, src, count);` `UML_DSHL(block, dst, src, count);`

Sets `dst` to the value of `src` shifted left by `count` bit positions modulo the operand size in bits. Zeroes are shifted into the least significant bit position.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the shifted value will be stored.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
The value to shift.

count (32-bit or 64-bit – memory, integer register, immediate, map variable)
The number of bit positions to shift by. Only the least significant five bits or six bits of this operand are used, depending on the instruction size.

Flags

carry (C)
Set to the value of the last bit shifted out of the most significant bit position if the shift count modulo the operand size in bits is non-zero, or cleared if the shift count modulo the operand size in bits is zero.

overflow (V)
Undefined.

zero (Z)
Set if the result is zero, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV*, *AND* or *OR* if the `src` and `count` operands are both immediate values or the `count` operand is the immediate value zero and the carry flag is not required.
- Immediate values for the `src` operand are truncated to the instruction size.
- Immediate values for the `count` operand are truncated to five or six bits for 32-bit or 64-bit operands, respectively.

SHR

Shift an integer value to the right (toward the least significant bit position), shifting zeroes into the most significant bit.

Disassembly	Usage
`shr dst,src,count` `dshr dst,src,count`	`UML_SHR(block, dst, src, count);` `UML_DSHR(block, dst, src, count);`

Sets `dst` to the value of `src` shifted right by `count` bit positions modulo the operand size in bits. Zeroes are shifted into the most significant bit position.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the shifted value will be stored.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
The value to shift.

count (32-bit or 64-bit – memory, integer register, immediate, map variable)
The number of bit positions to shift by. Only the least significant five bits or six bits of this operand are used, depending on the instruction size.

Flags

carry (C)
Set to the value of the last bit shifted out of the least significant bit position if the shift count modulo the operand size in bits is non-zero, or cleared if the shift count modulo the operand size in bits is zero.

overflow (V)
Undefined.

zero (Z)
Set if the result is zero, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV*, *AND* or *OR* if the `src` and `count` operands are both immediate values or the `count` operand is the immediate value zero and the carry flag is not required.
- Immediate values for the `src` operand are truncated to the instruction size.
- Immediate values for the `count` operand are truncated to five or six bits for 32-bit or 64-bit operands, respectively.

SAR

Shift an integer value to the right (toward the least significant bit position), preserving the value of the most significant bit.

Disassembly	Usage
`sar dst,src,count` `dsar dst,src,count`	`UML_SAR(block, dst, src, count);` `UML_DSAR(block, dst, src, count);`

Sets `dst` to the value of `src` shifted right by `count` bit positions modulo the operand size in bits. The value of the most significant bit is preserved after each shift step.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the shifted value will be stored.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
The value to shift.

count (32-bit or 64-bit – memory, integer register, immediate, map variable)
The number of bit positions to shift by. Only the least significant five bits or six bits of this operand are used, depending on the instruction size.

Flags

carry (C)
Set to the value of the last bit shifted out of the least significant bit position if the shift count modulo the operand size in bits is non-zero, or cleared if the shift count modulo the operand size in bits is zero.

overflow (V)
Undefined.

zero (Z)
Set if the result is zero, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV*, *AND* or *OR* if the `src` and `count` operands are both immediate values or the `count` operand is the immediate value zero and the carry flag is not required.
- Immediate values for the `src` operand are truncated to the instruction size.
- Immediate values for the `count` operand are truncated to five or six bits for 32-bit or 64-bit operands, respectively.

ROL

Rotate an integer value to the left (toward the most significant bit position). Bits shifted out of the most significant bit position are shifted into the least significant bit position.

Disassembly	Usage
`rol dst,src,count` `drol dst,src,count`	`UML_ROL(block, dst, src, count);` `UML_DROL(block, dst, src, count);`

Sets `dst` to the value of `src` rotated left by `count` bit positions.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the rotated value will be stored.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
The value to rotated.

count (32-bit or 64-bit – memory, integer register, immediate, map variable)
The number of bit positions to rotate by. Only the least significant five bits or six bits of this operand are used, depending on the instruction size.

Flags

carry (C)
Set to the value of the last bit rotated out of the most significant bit position (set to the least significant bit of the result) if the shift count modulo the operand size in bits is non-zero, or cleared if the shift count modulo the operand size in bits is zero.

overflow (V)
Undefined.

zero (Z)
Set if the result is zero, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV*, *AND* or *OR* if the `src` and `count` operands are both immediate values or the `count` operand is the immediate value zero and the carry flag is not required.
- Immediate values for the `src` operand are truncated to the instruction size.
- Immediate values for the `count` operand are truncated to five or six bits for 32-bit or 64-bit operands, respectively.

ROR

Rotate an integer value to the right (toward the least significant bit position). Bits shifted out of the least significant bit position are shifted into the most significant bit position.

Disassembly	Usage
`ror dst,src,count` `dror dst,src,count`	`UML_ROR(block, dst, src, count);` `UML_DROR(block, dst, src, count);`

Sets `dst` to the value of `src` rotated right by `count` bit positions.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the rotated value will be stored.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
The value to rotated.

count (32-bit or 64-bit – memory, integer register, immediate, map variable)
The number of bit positions to rotate by. Only the least significant five bits or six bits of this operand are used, depending on the instruction size.

Flags

carry (C)
Set to the value of the last bit rotated out of the least significant bit position (set to the most significant bit of the result) if the shift count modulo the operand size in bits is non-zero, or cleared if the shift count modulo the operand size in bits is zero.

overflow (V)
Undefined.

zero (Z)
Set if the result is zero, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV*, *AND* or *OR* if the `src` and `count` operands are both immediate values or the `count` operand is the immediate value zero and the carry flag is not required.
- Immediate values for the `src` operand are truncated to the instruction size.
- Immediate values for the `count` operand are truncated to five or six bits for 32-bit or 64-bit operands, respectively.

ROLC

Rotate an integer value concatenated with the carry flag to the left (toward the most significant bit position). For each step, the carry flag is shifted into the least significant bit position, and the carry flag is set to the bit shifted out of the most significant bit position.

Disassembly	Usage
`rolc dst,src,count` `drolc dst,src,count`	`UML_ROLC(block, dst, src, count);` `UML_DROLC(block, dst, src, count);`

Sets `dst` to the value of `src` concatenated with the carry flag rotated left by `count` bit positions modulo the operand size in bits. For each shift step, the current value of the carry flag is shifted into the least significant bit position, and the carry flag is set to the value of the bit shifted out of the most significant bit position.

Note that although this instruction rotates a 33-bit or 65-bit value (including the carry flag), the shift count is interpreted modulo 32 or 64.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the rotated value will be stored.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
The value to rotated.

count (32-bit or 64-bit – memory, integer register, immediate, map variable)
The number of bit positions to rotate by. Only the least significant five bits or six bits of this operand are used, depending on the instruction size.

Flags

carry (C)
Set to the value of the last bit shifted out of the least significant bit position if the shift count modulo the operand size in bits is non-zero, or unchanged if the shift count modulo the operand size in bits is zero.

overflow (V)
Undefined.

zero (Z)
Set if the result is zero, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV* or *NOP* if the `count` operand is the immediate value zero and the zero and sign flags are not required.
- Immediate values for the `src` operand are truncated to the instruction size.
- Immediate values for the `count` operand are truncated to five or six bits for 32-bit or 64-bit operands, respectively.

RORC

Rotate an integer value concatenated with the carry flag to the right (toward the least significant bit position). For each step, the carry flag is shifted into the most significant bit position, and the carry flag is set to the bit shifted out of the least significant bit position.

Disassembly	Usage
`rorc dst,src,count` `drorc dst,src,count`	`UML_RORC(block, dst, src, count);` `UML_DRORC(block, dst, src, count);`

Sets `dst` to the value of `src` concatenated with the carry flag rotated right by `count` bit positions modulo the operand size in bits. For each shift step, the current value of the carry flag is shifted into the most significant bit position, and the carry flag is set to the value of the bit shifted out of the least significant bit position.

Note that although this instruction rotates a 33-bit or 65-bit value (including the carry flag), the shift count is interpreted modulo 32 or 64.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the rotated value will be stored.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
The value to rotated.

count (32-bit or 64-bit – memory, integer register, immediate, map variable)
The number of bit positions to rotate by. Only the least significant five bits or six bits of this operand are used, depending on the instruction size.

Flags

carry (C)
Set to the value of the last bit shifted out of the least significant bit position if the shift count modulo the operand size in bits is non-zero, or unchanged if the shift count modulo the operand size in bits is zero.

overflow (V)
Undefined.

zero (Z)
Set if the result is zero, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV* or *NOP* if the `count` operand is the immediate value zero and the zero and sign flags are not required.
- Immediate values for the `src` operand are truncated to the instruction size.
- Immediate values for the `count` operand are truncated to five or six bits for 32-bit or 64-bit operands, respectively.

BFX

Extract a contiguous bit field from an integer value.

Disassembly	Usage
`bfxu dst,src,shift,width`	`UML_BFXU(block, dst, src, shift, width);`
`bfxs dst,src,shift,width`	`UML_BFXS(block, dst, src, shift, width);`
`dbfxu dst,src,shift,width`	`UML_DBFXU(block, dst, src, shift, width);`
`dbfxs dst,src,shift,width`	`UML_DBFXS(block, dst, src, shift, width);`

Extracts and right-aligns a contiguous bit field from the value of `src`, specified by its least significant bit position and width in bits. The field must be narrower than the `src` operand, but it may wrap around from the most significant bit position to the least significant bit position. BFXU and DBFXU zero-extend an unsigned field, while BFXS and DBFXS sign-extend a signed field.

Back-ends may be able to optimise some forms of this instruction for example when the `shift` and `width` operands are both immediate values.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the extracted field will be stored.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
The value to extract a contiguous bit field from.

shift (32-bit or 64-bit – memory, integer register, immediate, map variable)
The position of the least significant bit of the field to extract, where zero is the least significant bit position, and bit numbers increase toward the most significant bit position. Only the least significant five bits or six bits of this operand are used, depending on the instruction size.

width (32-bit or 64-bit – memory, integer register, immediate, map variable)
The width of the field to extract in bits. Only the least significant five bits or six bits of this operand are used, depending on the instruction size. The result is undefined if the width modulo the instruction size in bits is zero.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Set if the result is zero, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV*, *AND* or *OR* if the `src`, `shift` and `width` operands are all immediate values, or if the `width` operand is the immediate value zero.
- Converted to *SHR* or *SAR* if the `src` operand is not an immediate value, the `shift` and `width` operands are both immediate values, and the sum of the value of the `shift` operand and the value of the `width` operand is equal to the instruction size in bits.
- BFXU and DBFXU are converted to *AND* if the `shift` operand is the immediate value zero and `width` operand is an immediate value.
- BFXS and DBFXS are converted to *SEXT* if the `shift` operand is the immediate value zero and `width` operand is the immediate value 8, 16 or 32.
- Immediate values for the `src` operand are truncated to the instruction size.
- Immediate values for the `shift` and `width` operands are truncated to five or six bits for 32-bit or 64-bit operands, respectively.

ROLAND

Rotate and mask an integer value. The value is rotated to the left (toward the most significant bit position). Bits shifted out of the most significant bit position are shifted into the least significant bit position. The logical conjunction of the rotated value and the mask is then calculated.

Disassembly	Usage
`roland  dst,src,count,mask` `droland dst,src,count,mask`	`UML_ROLAND(block, dst, src, count, mask);` `UML_DROLAND(block, dst, src, count, mask);`

Sets `dst` to the logical conjunction of the value of `src` rotated left by `count` bit positions and the value of `mask`.

Back-ends may be able to optimise some forms of this instruction, for example when the `count` and `mask` operands are both immediate values.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the rotated and masked value will be stored.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
The value to be rotated and masked.

count (32-bit or 64-bit – memory, integer register, immediate, map variable)
The number of bit positions to rotate by. Only the least significant five bits or six bits of this operand are used, depending on the instruction size.

mask (32-bit or 64-bit – memory, integer register, immediate, map variable)
The mask to calculate the logical conjunction with.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Set if the result is zero, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV*, *AND* or *OR* if the `src` and `count` operands are both immediate values, or if either the `count` or `mask` operand is the immediate value zero.
- Converted to *ROL* if the `mask` operand is an immediate value with all bits set.
- Converted to *SHL* if the `count` operand is an immediate value and the `mask` operand is an immediate value containing a single contiguous left-aligned sequence of set bits of the appropriate length for the value of the `count` operand.
- Converted to *SHR* or *BFX* if the `count` operand is an immediate value and the `mask` operand is an immediate value containing a single contiguous right-aligned sequence of set bits.
- Immediate values for the `src` and `mask` operands are truncated to the instruction size.
- Immediate values for the `count` operand are truncated to five or six bits for 32-bit or 64-bit operands, respectively.

ROLINS

Rotate an integer value and combine with the destination value using a mask. The value is rotated to the left (toward the most significant bit position). Bits shifted out of the most significant bit position are shifted into the least significant bit position. The rotated value is then combined with the destination value: for bit positions that are set in the mask, the corresponding bit position values are copied from the rotated value to the destination; for bit positions that are clear in the mask, the corresponding bit position values in the destination are preserved.

Disassembly	Usage
`rolins  dst,src,count,mask` `drolins dst,src,count,mask`	`UML_ROLINS(block, dst, src, count, mask);` `UML_DROLINS(block, dst, src, count, mask);`

Rotates the value of `src` left by `count` bit positions and then copies bit values to `dst` where the corresponding bit positions are set in `mask`.

Back-ends may be able to optimise some forms of this instruction, for example when the `count` and `mask` operands are both immediate values.

Operands

dst (32-bit or 64-bit – memory, integer register)
: The destination that will be combined with the rotated value.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
: The value to be rotated and masked.

count (32-bit or 64-bit – memory, integer register, immediate, map variable)
: The number of bit positions to rotate by. Only the least significant five bits or six bits of this operand are used, depending on the instruction size.

mask (32-bit or 64-bit – memory, integer register, immediate, map variable)
: The mask to control which bit positions are copied from the rotated value and which bit positions are preserved in the destination.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Set if the result is zero, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *ROL*, *AND*, *OR* or *MOV* if the `mask` operand is an immediate value with all bits set or the immediate value zero.
- Immediate values for the `src` and `mask` operands are truncated to the instruction size.
- Immediate values for the `count` operand are truncated to five or six bits for 32-bit or 64-bit operands, respectively.

BSWAP

Reverse the order of bytes within an integer value.

Disassembly	Usage
`bswap dst,src` `dbswap dst,src`	`UML_BSWAP(block, dst, src);` `UML_DBSWAP(block, dst, src);`

This instruction can be used to convert between big Endian and little Endian byte order.

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the result will be stored.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
The value to have its byte order reversed.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Set if the result is zero, or cleared otherwise.

sign (S)
Set to the value of the most significant bit of the result (set if the result is a negative signed integer value, or cleared otherwise).

unordered (U)
Undefined.

Simplification rules

- Converted to *MOV*, *AND* or *OR* if the `src` operand is an immediate value.

12.15.10 Floating point arithmetic

FADD

Add two floating point numbers.

Disassembly	Usage
`fsadd dst,src1,src2` `fdadd dst,src1,src2`	`UML_FSADD(block, dst, src1, src2);` `UML_FDADD(block, dst, src1, src2);`

Calculates `dst = src1 + src2`.

Operands

dst (32-bit or 64-bit – memory, floating point register)
The destination where the sum will be stored.

src1 (32-bit or 64-bit – memory, floating point register)
The first addend.

src2 (32-bit or 64-bit – memory, floating point register)
The second addend.

Simplification rules

No simplifications are applied to this instruction.

FSUB

Subtract a floating point number from another floating point number.

Disassembly	Usage
`fssub dst,src1,src2` `fdsub dst,src1,src2`	`UML_FSSUB(block, dst, src1, src2);` `UML_FDSUB(block, dst, src1, src2);`

Calculates `dst = src1 - src2`.

Operands

dst (32-bit or 64-bit – memory, floating point register)
The destination where the difference will be stored.

src1 (32-bit or 64-bit – memory, floating point register)
The minuend (the value to subtract from).

src2 (32-bit or 64-bit – memory, floating point register)
The subtrahend (the value to subtract).

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

No simplifications are applied to this instruction.

FCMP

Compare two floating-point numbers and set the carry, zero and unordered flags.

Disassembly	Usage
`fscmp src1,src2` `fdcmp src1,src2`	`UML_FSCMP(block, src1, src2);` `UML_FDCMP(block, src1, src2);`

Operands

src1 (32-bit or 64-bit – memory, floating point register)
The left-hand side value to compare.

src2 (32-bit or 64-bit – memory, floating point register)
The right-hand side value to compare.

Flags

carry (C)
Set if the value of `src1` is less than the value of `src2`, or cleared otherwise.

overflow (V)
Undefined.

zero (Z)
Set if the values of `src1` and `src2` are equal, or cleared otherwise.

sign (S)
Undefined.

unordered (U)
Set if either `src1` or `src2` is not a number (NaN), or cleared otherwise.

Simplification rules

No simplifications are applied to this instruction.

FMUL

Multiply two floating point numbers.

Disassembly	Usage
`fsmul dst,src1,src2` `fdmul dst,src1,src2`	`UML_FSMUL(block, dst, src1, src2);` `UML_FDMUL(block, dst, src1, src2);`

Calculates `dst = src1 * src2`.

Operands

dst (32-bit or 64-bit – memory, floating point register)
The destination where the product will be stored.

src1 (32-bit or 64-bit – memory, floating point register)
The multiplicand (the value to multiply).

src2 (32-bit or 64-bit – memory, floating point register)
The multiplier (the value to multiply by).

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

No simplifications are applied to this instruction.

FDIV

Divide a floating point number by another floating point number.

Disassembly	Usage
`fsdiv dst,src1,src2` `fddiv dst,src1,src2`	`UML_FSDIV(block, dst, src1, src2);` `UML_FDDIV(block, dst, src1, src2);`

Calculates `dst = src1 / src2`.

Operands

dst (32-bit or 64-bit – memory, floating point register)
The destination where the quotient will be stored.

src1 (32-bit or 64-bit – memory, floating point register)
The dividend (the value to divide).

src2 (32-bit or 64-bit – memory, floating point register)
The divisor (the value to divide by).

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

No simplifications are applied to this instruction.

FNEG

Negate a floating point number.

Disassembly	Usage
`fsneg dst,src` `fdneg dst,src`	`UML_FSNEG(block, dst, src);` `UML_FDNEG(block, dst, src);`

Calculates `dst = -src`.

Operands

dst (32-bit or 64-bit – memory, floating point register)
The destination where the result will be stored.

src (32-bit or 64-bit – memory, floating point register)
The value to be negated.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

No simplifications are applied to this instruction.

FABS

Calculate the absolute value of a floating point number.

Disassembly	Usage
`fsabs dst,src` `fdabs dst,src`	`UML_FSABS(block, dst, src);` `UML_FDABS(block, dst, src);`

Operands

dst (32-bit or 64-bit – memory, floating point register)
The destination where the result will be stored.

src (32-bit or 64-bit – memory, floating point register)
The value to calculate the absolute value of.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

No simplifications are applied to this instruction.

FSQRT

Calculate the square root of a floating point number.

Disassembly	Usage
`fssqrt dst,src` `fdsqrt dst,src`	`UML_FSSQRT(block, dst, src);` `UML_FDSQRT(block, dst, src);`

Operands

dst (32-bit or 64-bit – memory, floating point register)
The destination where the square root will be stored.

src (32-bit or 64-bit – memory, floating point register)
The value to calculate the square root of.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

No simplifications are applied to this instruction.

FRECIP

Calculate an approximate reciprocal value of a floating point number. The algorithm used, precision and nature of inaccuracies in the approximation are unspecified.

Disassembly	Usage
`fsabs dst,src` `fdabs dst,src`	`UML_FSABS(block, dst, src);` `UML_FDABS(block, dst, src);`

Operands

dst (32-bit or 64-bit – memory, floating point register)
The destination where the result will be stored.

src (32-bit or 64-bit – memory, floating point register)
The value to approximate the reciprocal of.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

No simplifications are applied to this instruction.

FRSQRT

Calculate an approximate reciprocal value of the square root of a floating point number. The algorithm used, precision and nature of inaccuracies in the approximation are unspecified.

Disassembly	Usage
`fsrsqrt dst,src` `fdrsqrt dst,src`	`UML_FSRSQRT(block, dst, src);` `UML_FDRSQRT(block, dst, src);`

Operands

dst (32-bit or 64-bit – memory, floating point register)
The destination where the result will be stored.

src (32-bit or 64-bit – memory, floating point register)
The value to approximate the reciprocal of the square root of.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

No simplifications are applied to this instruction.

FRNDS

Round a 64-bit floating point value to 32-bit precision. The current default rounding type set using the *SETFMOD* is used. Note that the instruction size must always be 64 bits for this instruction.

Disassembly	Usage
`fdrnds  dst,src`	`UML_FDRNDS(block, dst, src);`

Operands

dst (64-bit – memory, floating point register)
The destination where the rounded value will be stored.

src (64-bit – memory, floating point register)
The floating point value to round to 32-bit precision.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

No simplifications are applied to this instruction.

12.15.11 Floating point conversion

FTOINT

Convert a floating point number to a signed two's complement integer.

Disassembly	Usage
`fstoint dst,src,size,round` `fdtoint dst,src,size,round`	`UML_FSTOINT(block, dst, src, size, round);` `UML_FDTOINT(block, dst, src, size, round);`

Operands

dst (32-bit or 64-bit – memory, integer register)
The destination where the integer value will be stored. The size/format is controlled by the `size` operand.

src (32-bit or 64-bit – memory, floating point register)
The floating point value to convert to an integer. The instruction size sets the size/format of this operand.

size (access size)
The size of the result. Must be `SIZE_DWORD` (32-bit) or `SIZE_QWORD` (64-bit). Note that this operand controls the size of the `dst` operand while the instruction size sets the size of the `src` operand.

round (rounding type)
The rounding type to use. Must be `ROUND_ROUND` (round to nearest), `ROUND_CEIL` (round toward positive infinity), `ROUND_FLOOR` (round toward negative infinity), `ROUND_TRUNC` (round toward zero) or `ROUND_DEFAULT` (use the current default rounding type set using the *SETFMOD* instruction).

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

No simplifications are applied to this instruction.

FFRINT

Convert a signed two's complement integer to a floating point number.

Disassembly	Usage
`fsfrint dst,src,size` `fdfrint dst,src,size`	`UML_FSFRINT(block, dst, src, size);` `UML_FDFRINT(block, dst, src, size);`

Operands

dst (32-bit or 64-bit – memory, floating point register)
The destination where the floating point value will be stored. The instruction size sets the size/format of this operand.

src (32-bit or 64-bit – memory, integer register, immediate, map variable)
The integer value to convert to a floating point value. The size/format is controlled by the `size` operand.

size (access size)
The size of the input. Must be `SIZE_DWORD` (32-bit) or `SIZE_QWORD` (64-bit). Note that this operand controls the size of the `src` operand while the instruction size sets the size of the `dst` operand.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

- Immediate values for the `src` operand are truncated to the size set using the `size` operand.

FFRFLT

Convert between floating point formats. The current default rounding type set using the *SETFMOD* is used if applicable.

Disassembly	Usage
`fsfrflt dst,src,size` `fdfrflt dst,src,size`	`UML_FSFRFLT(block, dst, src, size);` `UML_FDFRFLT(block, dst, src, size);`

Operands

dst (32-bit or 64-bit – memory, floating point register)
The destination where the converted value will be stored. The instruction size sets the size/format of this operand.

src (32-bit or 64-bit – memory, floating point register)
The floating point value to convert. The size/format is controlled by the `size` operand.

size (access size)
The size of the input. Must be `SIZE_SHORT` (32-bit) or `SIZE_DOUBLE` (64-bit). Note that this operand controls the size of the `src` operand while the instruction size sets the size of the `dst` operand.

Flags

carry (C)
Undefined.

overflow (V)
Undefined.

zero (Z)
Undefined.

sign (S)
Undefined.

unordered (U)
Undefined.

Simplification rules

- Converted to *FMOV* or *NOP* if the `dst` and `src` operands have the same size/format.

12.16 Software 3D Rendering in MAME

- *Background*
- *Concepts*
  - *ObjectType*
  - *Primitives*
  - *Synchronization*
- *The poly_manager class*
  - *Types & Constants*
    - *vertex_t*
    - *extent_t*
    - *render_delegate*
  - *Methods*
    - *poly_manager*
    - *wait*
    - *object_data*
    - *register_poly_array*
    - *render_tile*
    - *render_triangle*
    - *render_triangle_fan*
    - *render_triangle_strip*
    - *render_polygon*
    - *render_extents*
    - *zclip_if_less*
- *Example Renderer*
  - *Types*
  - *Constructor*
  - *swap_buffers*
  - *clear_buffers*
  - *draw_triangle*
  - *draw_triangle_flat*

- *draw_triangle_gouraud*
- *Advanced Topic: the poly_array class*
  - *Methods*
    - *poly_array*
    - *count*
    - *max*
    - *itemsize*
    - *allocated*
    - *byindex*
    - *contiguous*
    - *indexof*
    - *reset*
    - *next*
    - *last*

12.16.1 Background

Beginning in the late 1980s, many arcade games began incorporating hardware-rendered 3D graphics into their video. These 3D graphics are typically rendered from low-level primitives into a frame buffer (usually double- or triple-buffered), then perhaps combined with traditional tilemaps or sprites, before being presented to the player.

When it comes to emulating 3D games, there are two general approaches. The first approach is to leverage modern 3D hardware by mapping the low-level primitives onto modern equivalents. For a cross-platform emulator like MAME, this requires having an API that is flexible enough to describe the primitives and all their associated behaviors with high accuracy. It also requires the emulator to be able to read back from the rendered frame buffer (since many games do this) and combine it with other elements, in a way that is properly synchronized with background rendering.

The alternative approach is to render the low-level primitives directly in software. This has the advantage of being able to achieve pretty much any behavior exhibited by the original hardware, but at the cost of speed. In MAME, since all emulation happens on one thread, this is particularly painful. However, just as with the 3D hardware approach, in theory a software-based approach could be spun off to other threads to handle the work, as long as mechanisms were present to synchronize when necessary, for example, when reading/writing directly to/from the frame buffer.

For the time being, MAME has opted for the second approach, leveraging a templated helper class called **poly_manager** to handle common situations.

12.16.2 Concepts

At its core, **poly_manager** is a mechanism to support multi-threaded rendering of low-level 3D primitives. Callers provide **poly_manager** with a set of *vertices* for a primitive plus a *render callback*. **poly_manager** breaks the primitive into clipped scanline *extents* and distributes the work among a pool of *worker threads*. The render callback is then called on the worker thread for each extent, where game-specific logic can do whatever needs to happen to render the data.

One key responsibility that **poly_manager** takes care of is ensuring order. Given a pool of threads and a number of work items to complete, it is important that—at least within a given scanline—all work is performed serially in order. The basic approach is to assign each extent to a *bucket* based on the Y coordinate. **poly_manager** then ensures that only one worker thread at a time is responsible for processing work in a given bucket.

Vertices in **poly_manager** consist of simple 2D X and Y *coordinates*, plus zero or more additional *iterated parameters*. These iterated parameters can be anything: intensity values for lighting; RGB(A) colors for Gouraud shading; normalized U, V coordinates for texture mapping; 1/Z values for Z buffering; etc. Iterated parameters, regardless of what they represent, are interpolated linearly across the primitive in screen space and provided as part of the extent to the render callback.

ObjectType

When creating a **poly_manager** class, you must provide it a special type that you define, known as **ObjectType**.

Because rendering happens asynchronously on worker threads, the idea is that the **ObjectType** class will hold a snapshot of all the relevant data needed for rendering. This allows the main thread to proceed—potentially modifying some of the relevant state—while rendering happens elsewhere.

In theory, we could allocate a new **ObjectType** class for each primitive rendered; however, that would be rather inefficient. It is quite common to set up the rendering state and then render several primitives using the same state.

For this reason, **poly_manager** maintains an internal array of **ObjectType** objects and keeps a copy of the last **ObjectType** used. Before submitting a new primitive, callers can see if the rendering state has changed. If it has, it can ask **poly_manager** to allocate a new **ObjectType** class and fill it in. When the primitive is submitted for rendering, the most recently allocated **ObjectType** instance is implicitly captured and provided to the render callbacks.

For more complex scenarios, where data might change even more infrequently, there is a **poly_array** template, which can be used to manage data in a similar way. In fact, internally **poly_manager** uses the **poly_array** class to manage its **ObjectType** allocations. More information on the **poly_array** class is provided later.

Primitives

poly_manager supports several different types of primitives:

- The most commonly-used primitive in **poly_manager** is the *triangle*, which has the nice property that iterated parameters have constant deltas across the full surface. Arbitrary-length *triangle fans* and *triangle strips* are also supported.
- In addition to triangles, **poly_manager** also supports *polygons* with an arbitrary number of vertices. The list of vertices is expected to be in either clockwise or anticlockwise order. **poly_manager** will walk the edges to compute deltas across each extent.
- As a special case, **poly_manager** supports a *tile* primitive, which is a simple quad defined by two vertices, a top-left vertex and a bottom-right vertex. Like triangles, tiles have constant iterated parameter deltas across their surface.
- Finally, **poly_manager** supports a fully custom mechanism where the caller provides a list of extents that are more or less fed directly to the worker threads. This is useful if emulating a system that has unusual primitives or requires highly specific behaviors for its edges.

Synchronization

One of the key requirements of providing an asynchronous rendering mechanism is synchronization. Synchronization in **poly_manager** is super simple: just call the `wait()` function.

There are several common reasons for issuing a wait:

- At display time, the pixel data must be copied to the screen. If any primitives were queued which touch the portion of the display that is going to be shown, you need to wait for rendering to be complete before copying. Note that this wait may not be strictly necessary in some situations (for example, a triple-buffered system).
- If the emulated system has a mechanism to read back from the framebuffer after rendering, then a wait must be issued prior to the read in order to ensure that asynchronous rendering is complete.
- If the emulated system modifies any state that is not cached in the **ObjectType** or elsewhere (for example, texture memory), then a wait must be issued to ensure that pending primitives which might consume that state have finished their work.
- If the emulated system can use a previous render target as, say, the texture source for a new primitive, then submitting the second primitive must wait until the first completes. **poly_manager** provides no internal mechanism to help detect this, so it is on the caller to determine when or if this is necessary.

Because the wait operation knows after it is done that all rendering is complete, **poly_manager** also takes this opportunity to reclaim all memory allocated for its internal structures, as well as memory allocated for **ObjectType** structures. Thus it is important that you don't hang onto any **ObjectType** pointers after a wait is called.

12.16.3 The poly_manager class

In most applications, **poly_manager** is not used directly, but rather serves as the base class for a more complete rendering class. The **poly_manager** class itself is a template:

```
template<typename BaseType, class ObjectType, int MaxParams, u8 Flags = 0>
class poly_manager;
```

and the template parameters are:

- **BaseType** is the type used internally for coordinates and iterated parameters, and should generally be either `float` or `double`. In theory, a fixed-point integral type could also be used, though the math logic has not been designed for that, so you may encounter problems.
- **ObjectType** is the user-defined per-object data structure described above. Internally, **poly_manager** will manage a **poly_array** of these, and a pointer to the most-recently allocated one at the time a primitive is submitted will be implicitly passed to the render callback for each corresponding extent.
- **MaxParams** is the maximum number of iterated parameters that may be specified in a vertex. Iterated parameters are generic and treated identically, so the mapping of parameter indices is completely up to the contract between the caller and the render callback. It is permitted for **MaxParams** to be 0.
- **Flags** is zero or more of the following flags:
  - POLY_FLAG_NO_WORK_QUEUE — specify this flag to disable asynchronous rendering; this can be useful for debugging. When this option is enabled, all primitives are queued and then processed in order on the calling thread when `wait()` is called on the **poly_manager** class.
  - POLY_FLAG_NO_CLIPPING — specify this if you want **poly_manager** to skip its internal clipping. Use this if your render callbacks do their own clipping, or if the caller always handles clipping prior to submitting primitives.

Types & Constants

vertex_t

Within the **poly_manager** class, you'll find a **vertex_t** type that describes a single vertex. All primitive drawing methods accept 2 or more of these **vertex_t** objects. The **vertex_t** includes the X and Y coordinates along with an array of iterated parameter values at that vertex:

```
struct vertex_t
{
    vertex_t() { }
    vertex_t(BaseType _x, BaseType _y) { x = _x; y = _y; }

    BaseType x, y;                          // X, Y coordinates
    std::array<BaseType, MaxParams> p;      // iterated parameters
};
```

Note that **vertex_t** itself is defined in terms of the **BaseType** and **MaxParams** template values of the owning **poly_manager** class.

All of **poly_manager**'s primitives operate in screen space, where (0,0) represents the top-left corner of the top-left pixel, and (0.5,0.5) represents the center of that pixel. Left and top pixel values are inclusive, while right and bottom pixel values are exclusive.

Thus, a *tile* rendered from (2,2)-(4,3) will completely cover 2 pixels: (2,2) and (3,2).

When calling a primitive drawing method, the iterated parameter array **p** need not be completely filled out. The number of valid iterated parameter values is specified as a template parameter to the primitive drawing methods, so only that many parameters need to actually be populated in the **vertex_t** structures that are passed in.

extent_t

poly_manager breaks primitives into extents, which are contiguous horizontal spans contained within a single scanline. These extents are then distributed to worker threads, who will call the render callback with information on how to render each extent. The **extent_t** type describes one such extent, providing the bounding X coordinates along with an array of iterated parameter start values and deltas across the span:

```
struct extent_t
{
    struct param_t
    {
        BaseType start;                     // parameter value at start
        BaseType dpdx;                      // dp/dx relative to start
    };
    int16_t startx, stopx;                  // starting (inclusive)/ending (exclusive)␣
↪endpoints
    std::array<param_t, MaxParams> param;   // array of parameter start/deltas
    void *userdata;                         // custom per-span data
};
```

For each iterated parameter, the **start** value contains the value at the left side of the span. The **dpdx** value contains the change of the parameter's value per X coordinate.

There is also a **userdata** field in the **extent_t** structure, which is not normally used, except when performing custom rendering.

render_delegate

When rendering a primitive, in addition to the vertices, you must also provide a **render_delegate** callback of the form:

```
void render(int32_t y, extent_t const &extent, ObjectType const &object, int threadid)
```

This callback is responsible for the actual rendering. It will be called at a later time, likely on a different thread, for each extent. The parameters passed are:

- **y** is the Y coordinate (scanline) of the current extent.
- **extent** is a reference to a **extent_t** structure, described above, which specifies for this extent the start/stop X values along with the start/delta values for each iterated parameter.
- **object** is a reference to the most recently allocated **ObjectType** at the time the primitive was submitted for rendering; in theory it should contain most of not all of the necessary data to perform rendering.
- **threadid** is a unique ID indicating the index of the thread you're running on; this value is useful if you are keeping any kind of statistics and don't want to add contention over shared values. In this situation, you can allocate **WORK_MAX_THREADS** instances of your data and update the instance for the **threadid** you are passed. When you want to display the statistics, the main thread can accumulate and reset the data from all threads when it's safe to do so (e.g., after a wait).

Methods

poly_manager

```
poly_manager(running_machine &machine);
```

The **poly_manager** constructor takes just one parameter, a reference to the **running_machine**. This grants **poly_manager** access to the work queues needed for multithreaded running.

wait

```
void wait(char const *debug_reason = "general");
```

Calling `wait()` stalls the calling thread until all outstanding rendering is complete:

- **debug_reason** is an optional parameter specifying the reason for the wait. It is useful if the compile-time constant **TRACK_POLY_WAITS** is enabled, as it will print a summary of wait times and reasons at the end of execution.

Return value: none.

object_data

```
objectdata_array &object_data();
```

This method just returns a reference to the internally-maintained **poly_array** of the **ObjectType** you specified when creating **poly_manager**. For most applications, the only interesting thing to do with this object is call the `next()` method to allocate a new object to fill out.

Return value: reference to a **poly_array** of **ObjectType**.

register_poly_array

```
void register_poly_array(poly_array_base &array);
```

For advanced applications, you may choose to create your own **poly_array** objects to manage large chunks of infrequently-changed data, such a palettes. After each `wait()`, **poly_manager** resets all the **poly_array** objects it knows about in order to reclaim all outstanding allocated memory. By registering your **poly_array** objects here, you can ensure that your arrays will also be reset after an `wait()` call.

Return value: none.

render_tile

```
template<int ParamCount>
uint32_t render_tile(rectangle const &cliprect, render_delegate callback,
                     vertex_t const &v1, vertex_t const &v2);
```

This method enqueues a single *tile* primitive for rendering:

- **ParamCount** is the number of live values in the iterated parameter array within each **vertex_t** provided; it must be no greater than the **MaxParams** value specified in the **poly_manager** template instantiation.
- **cliprect** is a reference to a clipping rectangle. All pixels and parameter values are clipped to stay within these bounds before being added to the work queues for rendering, unless **POLY_FLAG_NO_CLIPPING** was specified as a flag parameter to **poly_manager**.
- **callback** is the render callback delegate that will be called to render each extent.
- **v1** contains the coordinates and iterated parameters for the top-left corner of the tile.
- **v2** contains the coordinates and iterated parameters for the bottom-right corner of the tile.

Return value: the total number of clipped pixels represented by the enqueued extents.

render_triangle

```
template<int ParamCount>
uint32_t render_triangle(rectangle const &cliprect, render_delegate callback,
                         vertex_t const &v1, vertex_t const &v2, vertex_t const &v3);
```

This method enqueues a single *triangle* primitive for rendering:

- **ParamCount** is the number of live values in the iterated parameter array within each **vertex_t** provided; it must be no greater than the **MaxParams** value specified in the **poly_manager** template instantiation.
- **cliprect** is a reference to a clipping rectangle. All pixels and parameter values are clipped to stay within these bounds before being added to the work queues for rendering, unless **POLY_FLAG_NO_CLIPPING** was specified as a flag parameter to **poly_manager**.
- **callback** is the render callback delegate that will be called to render each extent.
- **v1**, **v2**, **v3** contain the coordinates and iterated parameters for each vertex of the triangle.

Return value: the total number of clipped pixels represented by the enqueued extents.

render_triangle_fan

```
template<int ParamCount>
uint32_t render_triangle_fan(rectangle const &cliprect, render_delegate callback,
                             int numverts, vertex_t const *v);
```

This method enqueues one or more *triangle* primitives for rendering, specified in fan order:

- **ParamCount** is the number of live values in the iterated parameter array within each **vertex_t** provided; it must be no greater than the **MaxParams** value specified in the **poly_manager** template instantiation.
- **cliprect** is a reference to a clipping rectangle. All pixels and parameter values are clipped to stay within these bounds before being added to the work queues for rendering, unless **POLY_FLAG_NO_CLIPPING** was specified as a flag parameter to **poly_manager**.
- **callback** is the render callback delegate that will be called to render each extent.
- **numverts** is the total number of vertices provided; it must be at least 3.
- **v** is a pointer to an array of **vertex_t** objects containing the coordinates and iterated parameters for all the triangles, in fan order. This means that the first vertex is fixed. So if 5 vertices are provided, indicating 3 triangles, the vertices used will be: (0,1,2) (0,2,3) (0,3,4)

Return value: the total number of clipped pixels represented by the enqueued extents.

render_triangle_strip

```
template<int ParamCount>
uint32_t render_triangle_strip(rectangle const &cliprect, render_delegate callback,
                               int numverts, vertex_t const *v);
```

This method enqueues one or more *triangle* primitives for rendering, specified in strip order:

- **ParamCount** is the number of live values in the iterated parameter array within each **vertex_t** provided; it must be no greater than the **MaxParams** value specified in the **poly_manager** template instantiation.
- **cliprect** is a reference to a clipping rectangle. All pixels and parameter values are clipped to stay within these bounds before being added to the work queues for rendering, unless **POLY_FLAG_NO_CLIPPING** was specified as a flag parameter to **poly_manager**.
- **callback** is the render callback delegate that will be called to render each extent.
- **numverts** is the total number of vertices provided; it must be at least 3.
- **v** is a pointer to an array of **vertex_t** objects containing the coordinates and iterated parameters for all the triangles, in strip order. So if 5 vertices are provided, indicating 3 triangles, the vertices used will be: (0,1,2) (1,2,3) (2,3,4)

Return value: the total number of clipped pixels represented by the enqueued extents.

render_polygon

```
template<int NumVerts, int ParamCount>
uint32_t render_polygon(rectangle const &cliprect, render_delegate callback, vertex_t␣
↪const *v);
```

This method enqueues a single *polygon* primitive for rendering:

- **NumVerts** is the number of vertices in the polygon.
- **ParamCount** is the number of live values in the iterated parameter array within each **vertex_t** provided; it must be no greater than the **MaxParams** value specified in the **poly_manager** template instantiation.
- **cliprect** is a reference to a clipping rectangle. All pixels and parameter values are clipped to stay within these bounds before being added to the work queues for rendering, unless **POLY_FLAG_NO_CLIPPING** was specified as a flag parameter to **poly_manager**.
- **callback** is the render callback delegate that will be called to render each extent.
- **v** is a pointer to an array of **vertex_t** objects containing the coordinates and iterated parameters for the polygon. Vertices are assumed to be in either clockwise or anticlockwise order.

Return value: the total number of clipped pixels represented by the enqueued extents.

render_extents

```
template<int ParamCount>
uint32_t render_extents(rectangle const &cliprect, render_delegate callback,
                        int startscanline, int numscanlines, extent_t const *extents);
```

This method enqueues custom extents directly:

- **ParamCount** is the number of live values in the iterated parameter array within each **vertex_t** provided; it must be no greater than the **MaxParams** value specified in the **poly_manager** template instantiation.
- **cliprect** is a reference to a clipping rectangle. All pixels and parameter values are clipped to stay within these bounds before being added to the work queues for rendering, unless **POLY_FLAG_NO_CLIPPING** was specified as a flag parameter to **poly_manager**.
- **callback** is the render callback delegate that will be called to render each extent.
- **startscanline** is the Y coordinate of the first extent provided.
- **numscanlines** is the number of extents provided.
- **extents** is a pointer to an array of **extent_t** objects containing the start/stop X coordinates and iterated parameters. The **userdata** field of the source extents is copied to the target as well (this field is otherwise unused for all other types of rendering).

Return value: the total number of clipped pixels represented by the enqueued extents.

zclip_if_less

```
template<int ParamCount>
int zclip_if_less(int numverts, vertex_t const *v, vertex_t *outv, BaseType clipval);
```

This method is a helper method to clip a polygon against a provided Z value. It assumes that the first iterated parameter in **vertex_t** represents the Z coordinate. If any edge crosses the Z plane represented by **clipval** that edge is clipped.

- **ParamCount** is the number of live values in the iterated parameter array within each **vertex_t** provided; it must be no greater than the **MaxParams** value specified in the **poly_manager** template instantiation.
- **numverts** is the number of vertices in the input array.
- **v** is a pointer to the input array of **vertex_t** objects.
- **outv** is a pointer to the output array of **vertex_t** objects. **v** and **outv** cannot overlap or point to the same memory.
- **clipval** is the value to compare parameter 0 against for clipping.

Return value: the number of output vertices written to **outv**. Note that by design it is possible for this method to produce more vertices than the input array, so callers should ensure there is enough room in the output buffer to accommodate this.

12.16.4 Example Renderer

Here is a complete example of how to create a software 3D renderer using **poly_manager**. Our example renderer will only handle flat and Gouraud-shaded triangles with depth (Z) buffering.

Types

The first thing we need to define is our *externally-visible* vertex format, which is distinct from the internal **vertex_t** that **poly_manager** will define. In theory you could use **vertex_t** directly, but the generic nature of **poly_manager**'s iterated parameters make it awkward:

```
struct example_vertex
{
    float x, y, z;      // X,Y,Z coordinates
    rgb_t color;        // color at this vertex
};
```

Next we define the **ObjectType** needed by **poly_manager**. For our simple case, we define an **example_object_data** struct that consists of pointers to our rendering buffers, plus a couple of fixed values that are consumed in some cases. More complex renderers would typically have many more object-wide parameters defined here:

```
struct example_object_data
{
    bitmap_rgb32 *dest;    // pointer to the rendering bitmap
    bitmap_ind16 *depth;   // pointer to the depth bitmap
    rgb_t color;           // overall color (for clearing and flat shaded case)
    uint16_t depthval;     // fixed depth value (for clearing)
};
```

Now it's time to define our renderer class, which we derive from **poly_manager**. As template parameters we specify `float` as the base type for our data, since that will be enough accuracy for this example, and we also provide our

example_object_data as the **ObjectType** class, plus the maximum number of iterated parameters our renderer will ever need (4 in this case):

```
class example_renderer : public poly_manager<float, example_object_data, 4>
{
public:
    example_renderer(running_machine &machine, uint32_t width, uint32_t height);

    bitmap_rgb32 *swap_buffers();

    void clear_buffers(rgb_t color, uint16_t depthval);
    void draw_triangle(example_vertex const *verts);

private:
    static uint16_t ooz_to_depthval(float ooz);

    void draw_triangle_flat(example_vertex const *verts);
    void draw_triangle_gouraud(example_vertex const *verts);

    void render_clear(int32_t y, extent_t const &extent, example_object_data const &
↪object, int threadid);
    void render_flat(int32_t y, extent_t const &extent, example_object_data const &
↪object, int threadid);
    void render_gouraud(int32_t y, extent_t const &extent, example_object_data const &
↪object, int threadid);

    int m_draw_buffer;
    bitmap_rgb32 m_display[2];
    bitmap_ind16 m_depth;
};
```

Constructor

The constructor for our example renderer just initializes **poly_manager** and allocates the rendering and depth buffers:

```
example_renderer::example_renderer(running_machine &machine, uint32_t width, uint32_t␣
↪height) :
    poly_manager(machine),
    m_draw_buffer(0)
{
    // allocate two display buffers and a depth buffer
    m_display[0].allocate(width, height);
    m_display[1].allocate(width, height);
    m_depth.allocate(width, height);
}
```

swap_buffers

The first interesting method in our renderer is `swap_buffers()`, which returns a pointer to the buffer we've been drawing to, and sets up the other buffer as the new drawing target. The idea is that the display update handler will call this method to get ahold of the bitmap to display to the user:

```
bitmap_rgb32 *example_renderer::swap_buffers()
{
    // wait for any rendering to complete before returning the buffer
    wait("swap_buffers");

    // return the current draw buffer and then switch to the other
    // for future drawing
    bitmap_rgb32 *result = &m_display[m_draw_buffer];
    m_draw_buffer ^= 1;
    return result;
}
```

The most important thing here to note here is the call to **poly_manager**'s `wait()`, which will block the current thread until all rendering is complete. This is important because otherwise the caller may receive a bitmap that is still being drawn to, leading to torn or corrupt visuals.

clear_buffers

One of the most common operations to perform when doing 3D rendering is to initialize or clear the display and depth buffers to a known value. This method below leverages the *tile* primitive to render a rectangle over the screen by passing in (0,0) and (width,height) for the two vertices.

Because the color and depth values to clear the buffer to are constant, they are stored in a freshly-allocated **example_object_data** object, along with a pointer to the buffers in question. The `render_tile()` call is made with a `<0>` suffix indicating that there are no iterated parameters to worry about:

```
void example_renderer::clear_buffers(rgb_t color, uint16_t depthval)
{
    // allocate object data and populate it with information needed
    example_object_data &object = object_data().next();
    object.dest = &m_display[m_draw_buffer];
    object.depth = &m_depth;
    object.color = color;
    object.depthval = depthval;

    // top,left coordinate is always (0,0)
    vertex_t topleft;
    topleft.x = 0;
    topleft.y = 0;

    // bottom,right coordinate is (width,height)
    vertex_t botright;
    botright.x = m_display[0].width();
    botright.y = m_display[0].height();

    // render as a tile with 0 iterated parameters
    render_tile<0>(m_display[0].cliprect(),
```

(continues on next page)

(continued from previous page)

```
                    render_delegate(&example_renderer::render_clear, this),
                    topleft, botright);
}
```

The render callback provided to `render_tile()` is also defined (privately) in our class, and handles a single span. Note how the rendering parameters are extracted from the **example_object_data** struct provided:

```
void example_renderer::render_clear(int32_t y, extent_t const &extent, example_object_
↪data const &object, int threadid)
{
    // get pointers to the start of the depth buffer and destination scanlines
    uint16_t *depth = &object.depth->pix(y);
    uint32_t *dest = &object.dest->pix(y);

    // loop over the full extent and just store the constant values from the object
    for (int x = extent.startx; x < extent.stopx; x++)
    {
        dest[x] = object.color;
        depth[x] = object.depthval;
    }
}
```

Another important point to make is that the X coordinates provided by extent struct are inclusive of startx but exclusive of stopx. Clipping is performed ahead of time so that the render callback can focus on laying down pixels as quickly as possible with minimal overhead.

draw_triangle

Next up, we have our actual triangle rendering function, which will draw a single triangle given an array of three vertices provided in the external **example_vertex** format:

```
void example_renderer::draw_triangle(example_vertex const *verts)
{
    // flat shaded case
    if (verts[0].color == verts[1].color && verts[0].color == verts[2].color)
        draw_triangle_flat(verts);
    else
        draw_triangle_gouraud(verts);
}
```

Because it is simpler and faster to render a flat shaded triangle, the code checks to see if the colors are the same on all three vertices. If they are, we call through to a special flat-shaded case, otherwise we process it as a full Gouraud-shaded triangle.

This is a common technique to optimize rendering performance: identify special cases that reduce the per-pixel work, and route them to separate render callbacks that are optimized for that special case.

draw_triangle_flat

Here's the setup code for rendering a flat-shaded triangle:

```
void example_renderer::draw_triangle_flat(example_vertex const *verts)
{
    // allocate object data and populate it with information needed
    example_object_data &object = object_data().next();
    object.dest = &m_display[m_draw_buffer];
    object.depth = &m_depth;

    // in this case the color is constant and specified in the object data
    object.color = verts[0].color;

    // copy X, Y, and 1/Z into poly_manager vertices
    vertex_t v[3];
    for (int vertnum = 0; vertnum < 3; vertnum++)
    {
        v[vertnum].x = verts[vertnum].x;
        v[vertnum].y = verts[vertnum].y;
        v[vertnum].p[0] = 1.0f / verts[vertnum].z;
    }

    // render the triangle with 1 iterated parameter (1/Z)
    render_triangle<1>(m_display[0].cliprect(),
                        render_delegate(&example_renderer::render_flat, this),
                        v[0], v[1], v[2]);
}
```

First, we put the fixed color into the **example_object_data** directly, and then fill out three **vertex_t** objects with the X and Y coordinates in the usual spot, and 1/Z as our one and only iterated parameter. (We use 1/Z here because iterated parameters are interpolated linearly in screen space. Z is not linear in screen space, but 1/Z is due to perspective correction.)

Our flat-shaded case then calls `render_trangle` specifying <1> iterated parameter to interpolate, and pointing to a special-case flat render callback:

```
void example_renderer::render_flat(int32_t y, extent_t const &extent, example_object_
↪data const &object, int threadid)
{
    // get pointers to the start of the depth buffer and destination scanlines
    uint16_t *depth = &object.depth->pix(y);
    uint32_t *dest = &object.dest->pix(y);

    // get the starting 1/Z value and the delta per X
    float ooz = extent.param[0].start;
    float doozdx = extent.param[0].dpdx;

    // iterate over the extent
    for (int x = extent.startx; x < extent.stopx; x++)
    {
        // convert the 1/Z value into an integral depth value
        uint16_t depthval = ooz_to_depthval(ooz);
```

(continues on next page)

(continued from previous page)

```
        // if closer than the current pixel, copy the color and depth value
        if (depthval < depth[x])
        {
            dest[x] = object.color;
            depth[x] = depthval;
        }

        // regardless, update the 1/Z value for the next pixel
        ooz += doozdx;
    }
}
```

This render callback is a bit more involved than the clearing case.

First, we have an iterated parameter (1/Z) to deal with, whose starting and X-delta values we extract from the extent before the start of the inner loop.

Second, we perform depth buffer testing, using `ooz_to_depthval()` as a helper to transform the floating-point 1/Z value into a 16-bit integer. We compare this value against the current depth buffer value, and only store the pixel/depth value if it's less.

At the end of each iteration, we advance the 1/Z value by the X-delta in preparation for the next pixel.

draw_triangle_gouraud

Finally we get to the code for the full-on Gouraud-shaded case:

```
void example_renderer::draw_triangle_gouraud(example_vertex const *verts)
{
    // allocate object data and populate it with information needed
    example_object_data &object = object_data().next();
    object.dest = &m_display[m_draw_buffer];
    object.depth = &m_depth;

    // copy X, Y, 1/Z, and R,G,B into poly_manager vertices
    vertex_t v[3];
    for (int vertnum = 0; vertnum < 3; vertnum++)
    {
        v[vertnum].x = verts[vertnum].x;
        v[vertnum].y = verts[vertnum].y;
        v[vertnum].p[0] = 1.0f / verts[vertnum].z;
        v[vertnum].p[1] = verts[vertnum].color.r();
        v[vertnum].p[2] = verts[vertnum].color.g();
        v[vertnum].p[3] = verts[vertnum].color.b();
    }

    // render the triangle with 4 iterated parameters (1/Z, R, G, B)
    render_triangle<4>(m_display[0].cliprect(),
                       render_delegate(&example_renderer::render_gouraud, this),
                       v[0], v[1], v[2]);
}
```

Here we have 4 iterated parameters: the 1/Z depth value, plus red, green, and blue, stored as floating point values. We call `render_triangle()` with <4> as the number of iterated parameters to process, and point to the full Gouraud

render callback:

```
void example_renderer::render_gouraud(int32_t y, extent_t const &extent, example_object_
↪data const &object, int threadid)
{
    // get pointers to the start of the depth buffer and destination scanlines
    uint16_t *depth = &object.depth->pix(y);
    uint32_t *dest = &object.dest->pix(y);

    // get the starting 1/Z value and the delta per X
    float ooz = extent.param[0].start;
    float doozdx = extent.param[0].dpdx;

    // get the starting R,G,B values and the delta per X as 8.24 fixed-point values
    uint32_t r = uint32_t(extent.param[1].start * float(1 << 24));
    uint32_t drdx = uint32_t(extent.param[1].dpdx * float(1 << 24));
    uint32_t g = uint32_t(extent.param[2].start * float(1 << 24));
    uint32_t dgdx = uint32_t(extent.param[2].dpdx * float(1 << 24));
    uint32_t b = uint32_t(extent.param[3].start * float(1 << 24));
    uint32_t dbdx = uint32_t(extent.param[3].dpdx * float(1 << 24));

    // iterate over the extent
    for (int x = extent.startx; x < extent.stopx; x++)
    {
        // convert the 1/Z value into an integral depth value
        uint16_t depthval = ooz_to_depthval(ooz);

        // if closer than the current pixel, assemble the color
        if (depthval < depth[x])
        {
            dest[x] = rgb_t(r >> 24, g >> 24, b >> 24);
            depth[x] = depthval;
        }

        // regardless, update the 1/Z and R,G,B values for the next pixel
        ooz += doozdx;
        r += drdx;
        g += dgdx;
        b += dbdx;
    }
}
```

This follows the same pattern as the flat-shaded callback, except we have 4 iterated parameters to step through.

Note that even though the iterated parameters are of `float` type, we convert the color values to fixed-point integers when iterating over them. This saves us doing 3 float-to-int conversions each pixel. The original RGB values were 0-255, so interpolation can only produce values in the 0-255 range. Thus we can use 24 bits of a 32-bit integer as the fraction, which is plenty accurate for this case.

12.16.5 Advanced Topic: the poly_array class

poly_array is a template class that is used to manage a dynamically-sized vector of objects whose lifetime starts at allocation and ends when `reset()` is called. The **poly_manager** class uses several **poly_array** objects internally, including one for allocated **ObjectType** data, one for each primitive rendered, and one for holding all allocated extents.

poly_array has an additional property where after a reset it retains a copy of the most recently allocated object. This ensures that callers can always call `last()` and get a valid object, even immediately after a reset.

The **poly_array** class requires two template parameters:

```
template<class ArrayType, int TrackingCount>
class poly_array;
```

These parameters are:

- **ArrayType** is the type of object you wish to allocate and manage.
- **TrackingCount** is the number of objects you wish to preserve after a reset. Typically this value is either 0 (you don't care to track any objects) or 1 (you only need one object); however, if you are using **poly_array** to manage a shared collection of objects across several independent consumers, it can be higher. See below for an example where this might be handy.

Note that objects allocated by **poly_array** are owned by **poly_array** and will be automatically freed upon exit.

poly_array is optimized for use in high frequency multi-threaded systems. Therefore, one added feature of the class is that it rounds the allocation size of **ArrayType** to the nearest cache line boundary, on the assumption that neighboring entries could be accessed by different cores simultaneously. Keeping each **ArrayType** object in its own cache line ensures no false sharing performance impacts.

Currently, **poly_array** has no mechanism to determine cache line size at runtime, so it presumes that 64 bytes is a typical cache line size, which is true for most x64 and ARM chips as of 2021. This value can be altered by changing the **CACHE_LINE_SHIFT** constant defined at the top of the class.

Objects allocated by **poly_array** are created in 64k chunks. At construction time, one chunk's worth of objects is allocated up front. The chunk size is controlled by the **CHUNK_GRANULARITY** constant defined at the top of the class.

As more objects are allocated, if **poly_array** runs out of space, it will dynamically allocate more. This will produce discontiguous chunks of objects until the next `reset()` call, at which point **poly_array** will reallocate all the objects into a contiguous vector once again.

For the case where **poly_array** is used to manage a shared pool of objects, it can be configured to retain multiple most recently allocated items by using a **TrackingCount** greater than 1. For example, if **poly_array** is managing objects for two texture units, then it can set **TrackingCount** equal to 2, and pass the index of the texture unit in calls to `next()` and `last()`. After a reset, **poly_array** will remember the most recently allocated object for each of the units independently.

Methods

poly_array

```
poly_array();
```

The **poly_array** constructor requires no parameters and simply pre-allocates one chunk of objects in preparation for future allocations.

count

```
u32 count() const;
```

Return value: the number of objects currently allocated.

max

```
u32 max() const;
```

Return value: the maximum number of objects ever allocated at one time.

itemsize

```
size_t itemsize() const;
```

Return value: the size of an object, rounded up to the nearest cache line boundary.

allocated

```
u32 allocated() const;
```

Return value: the number of objects that fit within what's currently been allocated.

byindex

```
ArrayType &byindex(u32 index);
```

Returns a reference to an object in the array by index. Equivalent to [**index**] on a normal array:

- **index** is the index of the item you wish to reference.

Return value: a reference to the object in question. Since a reference is returned, it is your responsibility to ensure that **index** is less than `count()` as there is no mechanism to return an invalid result.

contiguous

```
ArrayType *contiguous(u32 index, u32 count, u32 &chunk);
```

Returns a pointer to the base of a contiguous section of **count** items starting at **index**. Because **poly_array** dynamically resizes, it may not be possible to access all **count** objects contiguously, so the number of actually contiguous items is returned in **chunk**:

- **index** is the index of the first item you wish to access contiguously.
- **count** is the number of items you wish to access contiguously.
- **chunk** is a reference to a variable that will be set to the actual number of contiguous items available starting at **index**. If **chunk** is less than **count**, then the caller should process the **chunk** items returned, then call `contiguous()` again at (**index** + **chunk**) to access the rest.

Return value: a pointer to the first item in the contiguous chunk. No range checking is performed, so it is your responsibility to ensure that **index** + **count** is less than or equal to `count()`.

indexof

```
int indexof(ArrayType &item) const;
```

Returns the index within the array of the given item:

- **item** is a reference to an item in the array.

Return value: the index of the item. It should always be the case that:

```
array.indexof(array.byindex(index)) == index
```

reset

```
void reset();
```

Resets the **poly_array** by semantically deallocating all objects. If previous allocations created a discontiguous array, a fresh vector is allocated at this time so that future allocations up to the same level will remain contiguous.

Note that the **ArrayType** destructor is *not* called on objects as they are deallocated.

Return value: none.

next

```
ArrayType &next(int tracking_index = 0);
```

Allocates a new object and returns a reference to it. If there is not enough space for a new object in the current array, a new discontiguous array is created to hold it:

- **tracking_index** is the tracking index you wish to assign the new item to. In the common case this is 0, but could be non-zero if using a **TrackingCount** greater than 1.

Return value: a reference to the object. Note that the placement new operator is called on this object, so the default **ArrayType** constructor will be invoked here.

last

```
ArrayType &last(int tracking_index = 0) const;
```

Returns a reference to the last object allocated:

- **tracking_index** is the tracking index whose object you want. In the common case this is 0, but could be non-zero if using a **TrackingCount** greater than 1. **poly_array** remembers the most recently allocated object independently for each **tracking_index**.

Return value: a reference to the last allocated object.

12.17 Audio effects

- *1. Generalities*
- *2. audio_effects/aeffects.**
- *3. audio_effects/youreffect.**
- *4. frontend/mame/ui/audioeffects.cpp*
- *5. frontend/mame/ui/audioyoureffect.**

12.17.1 1. Generalities

The audio effects are effects that are applied to the sound between the speaker devices and the actual sound output. In the current implementation the effect chain is fixed (but not the effect parameters), and the parameters are stored in the cfg files. They are honestly not really designed for extensibility yet, if ever.

Adding an effect requires working on four parts:

- audio_effects/aeffects.* for effect object creation and "publishing"
- audio_effects/youreffect.* for the effect implementation
- frontend/mame/ui/audioeffects.cpp to be able to instantiate the effect configuration menu
- frontend/mame/ui/audioyoureffect.* to implement the effect configuration menu

12.17.2 2. audio_effects/aeffects.*

The audio_effect class in the aeffect sources provides three things:

- an enum value to designate the effect type and which must match its position in the chain (iow, the effect chain follows the enum order), in the .h
- the effect name in the audio_effect::effect_names array in the .cpp
- the creation of a correct effect object in audio_effect::create in the .cpp

12.17.3 3. audio_effects/youreffect.*

This is where you implement the effect. It takes the shape of an audio_effect_youreffect class which derives from audio_effect.

The methods to implement are:

```
audio_effect_youreffect(u32 sample_rate, audio_effect *def);

virtual int type() const override;
virtual void config_load(util::xml::data_node const *ef_node) override;
virtual void config_save(util::xml::data_node *ef_node) const override;
virtual void default_changed() override;
virtual u32 history_size() const; // optional
```

The constructor must pass the parameters to `audio_effect` and initialize the effect parameters. `type` must return the enum value for the effect. `config_load` and `config_save` should load or save the effect parameters from/to the cfg file xml tree. `default_changed` is called when the parameters in `m_default` are changed and the parameters may need to be updated. `history_size` allows to tell how many samples should still be available of the previous input frame. Note that this number must not depend on the parameters and only on the sample rate.

An effect has a number of parameters that can come from three sources:

- fixed default value
- equivalent effect object from the default effect chain
- user setting through the UI

The first two are recognized through the value of `m_default` which gets the value of `def` in the constructor. When it's nullptr, the value to use when not set by the user is the fixed one, otherwise it's the one in `m_default`.

At minimum an effect should have a parameter allowing to bypass it.

Managing a parameter uses four methods:

- `type param() const;` returns the current parameter value
- `void set_param(type value);` sets the current parameter value and marks it as set by the user
- `bool isset_param() const;` returns true is the parameter was set by the user
- `void reset_param();` resets the parameter to the default value (from m_default or fixed) and marks it as not set by the user

`config_save` must only save the user-set parameters. `config_load` must retrieve the parameters that are present and mark them as set by the user and reset all the others.

Finally the actual implementation goes into the `apply` method:

```
virtual void apply(const emu::detail::output_buffer_flat<sample_t> &src,␣
↪emu::detail::output_buffer_flat<sample_t> &dest) override;
```

That method takes two buffers with the same number of channels and has to apply the effect to `src` to produce `dest`. The `output_buffer_flat` is non-interleaved with independent per-channel buffers.

To make bypassing easier, the `copy(src, dest)` method of audio_effect allows to copy the samples from `src` to `dest` without changing them.

The effect application part should looks like:

```
u32 samples = src.available_samples();
dest.prepare_space(samples);
u32 channels = src.channels();

// generate channels * samples results and put them in dest

dest.commit(samples);
```

To get pointers to the buffers, one uses:

```
const sample_t *source = src.ptrs(channel, source_index); // source_index must be in [-
↪history_size()..samples-1]
sample_t *destination = dest.ptrw(channel, destination_index); // destination_index must␣
↪be in [0..samples-1]
```

The samples pointed by source and destination are contiguous. The number of channels will not change from one apply call to another, the number of samples will vary though. Also the call happens in a different thread than the main thread and also in a different thread than the parameter setting calls are made from.

12.17.4 4. frontend/mame/ui/audioeffects.cpp

Here it suffices to add a creation of the menu menu_audio_effect_youreffect in menu_audio_effects::handle. The menu effect will pick the effect names from audio_effect (in aeffect.*).

12.17.5 5. frontend/mame/ui/audioyoureffect.*

This is used to implement the configuration menu for the effect. It's a little complicated because it needs to generate the list of parameters and their values, set left or right arrow flags depending on the possible modifications, dim them (FLAG_INVERT) when not set by the user, and manage the arrows and clear keys to change them. Just copy an existing one and change it as needed.

12.18 OSD audio support

12.18.1 Introduction

The audio support in Mame tries to allow the user to freely map between the emulated system audio outputs (called speakers) and the host system audio. A part of it is the OSD support, where a host-specific module ensures the interface between Mame and the host. This is the documentation for that module.

Note: this is currently output-only, but input should follow.

12.18.2 Capabilities

The OSD interface is designed to allow three levels of support, depending on what the API allows and the amount of effort to expend. Those are:

- Level 1: One or more audio targets, only one stream allowed per target (aka exclusive mode)
- Level 2: One or more audio targets, multiple streams per target
- Level 3: One or more audio targets, multiple streams per target, user-visible per-stream-channel volume control

In any case we support having the user use an external interface to change the target of a stream and, in level 3, change the volumes. By support we mean storing the information in the per-game configuration and keeping the internal UI in sync.

12.18.3 Terminology

For this module, we use the terms:

- node: some object we can send audio to. Can be physical, like speakers, or virtual, like an effect system. It should have a unique, user-presentable name for the UI.
- port: a channel of a node, has a name (non-unique, like "front left") and a 3D position
- stream: a connection to a node with allows to send audio to it

12.18.4 Reference documentation

Adding a module

Adding a module is done by adding a cpp file to src/osd/modules/sound which follows this structure,

```
// License/copyright
#include "sound_module.h"
#include "modules/osdmodules.h"

#ifdef MODULE_SUPPORT_KEY

#include "modules/lib/osdobj_common.h"

// [...]
namespace osd {
namespace {

class sound_module_class : public osd_module, public sound_module
{
   sound_module_class() : osd_module(OSD_SOUND_PROVIDER, "module_name"),
                          sound_module()
   // ...
};

}
}
#else
namespace osd { namespace {
  MODULE_NOT_SUPPORTED(sound_module_class, OSD_SOUND_PROVIDER, "module_name")
}}
#endif

MODULE_DEFINITION(SOUND_MODULE_KEY, osd::sound_module_class)
```

In that code, four names must be chosen:

- MODULE_SUPPORT_KEY some #define coming from the genie scripts to tell that this particular module can be compiled (like NO_USE_PIPEWIRE or SDLMAME_MACOSX)
- sound_module_class is the name of the class which makes up the module (like sound_coreaudio)
- module_name is the name to be used in -sound <xxx> to select that particular module (like coreaudio)
- SOUND_MODULE_KEY is a symbol that represents the module internally (like SOUND_COREAUDIO)

The file path needs to be added to scripts/src/osd/modules.lua in osdmodulesbuild() and the module reference to src/osd/modules/lib/osdobj_common.cpp in osd_common_t::register_options with the line:

```
REGISTER_MODULE(m_mod_man, SOUND_MODULE_KEY);
```

This should ensure that the module is reachable through -sound <xxx> on the appropriate hosts.

Interface

The full interface is:

```
virtual bool split_streams_per_source() const override;
virtual bool external_per_channel_volume() const override;

virtual int init(osd_interface &osd, osd_options const &options) override;
virtual void exit() override;

virtual uint32_t get_generation() override;
virtual osd::audio_info get_information() override;
virtual uint32_t stream_sink_open(uint32_t node, std::string name, uint32_t rate)␣
↪override;
virtual uint32_t stream_source_open(uint32_t node, std::string name, uint32_t rate)␣
↪override;
virtual void stream_set_volumes(uint32_t id, const std::vector<float> &db) override;
virtual void stream_close(uint32_t id) override;
virtual void stream_sink_update(uint32_t id, const int16_t *buffer, int samples_this_
↪frame) override;
virtual void stream_source_update(uint32_t id, int16_t *buffer, int samples_this_frame)␣
↪override;
```

The class sound_module provides defaults for minimum capabilities: one stereo target and stream at default sample rate. To support that, only *init*, *exit* and *stream_update* need to be implemented. *init* is called at startup and *exit* when quitting and can do whatever they need to do. *stream_sink_update* will be called on a regular basis with a buffer of sample_this_frame * 2 * int16_t with the audio to play. From this point in the documentation we'll assume more than a single stereo channel is wanted.

Capabilities

Two methods are used by the module to indicate the level of capability of the module:

- split_streams_per_source() should return true when having multiple streams for one target is expected (e.g. Level 2 or 3)
- external_per_channel_volume() should return true when the streams have per-channel volume control that can be externally controlled (e.g. Level 3)

Hardware information and generations

The core runs on the assumption that the host hardware capabilities can change at any time (bluetooth devices coming and going, usb hot-plugging...) and that the module has some way to keep tabs on what is happening, possibly using multi-threading. To keep it lightweight-ish, we use the concept of a *generation* which is a 32-bit number that is incremented by the module every time something changes. The core checks the current generation value at least once every update (once per frame, usually) and if it changed asks for the new state and detects and handles the differences. *generation* should be "eventually stable", e.g. it eventually stops changing when the user stops changing things all the time. A systematic increment every frame would be a bad idea.

```
virtual uint32_t get_generation() override;
```

That method returns the current generation number. It's called at a minimum once per update, which usually means per frame. It should be reasonably lightweight when nothing special happens.

This method must provide all the information about the current state of the host and the module. This state is:

- m_generation: The current generation number
- m_nodes: The vector available nodes (*node_info*)
  - m_name: The name of the node to be used in non-user-visible configurations (can be a uuid or equivalent)
  - m_display_name: The name of the node to be used in the user interfaces (should be readable)
  - m_id: The numeric ID of the node
  - m_rate: The minimum, maximum and preferred sample rate for the node
  - m_port_names: The vector of port names
  - m_port_positions: The vector of 3D position of the ports. Refer to src/emu/speaker.h for the "standard" positions
  - m_sinks: Number of sinks (inputs)
  - m_sources: Number of sources (outputs)
- m_default_sink: ID of the node that is the current "system default" for audio output, 0 if there's no such concept
- m_default_source: same for audio input (currently unused)
- m_streams: The vector of active streams (*stream_info*)
  - m_id: The numeric ID of the stream
  - m_node: The target node of the stream
  - m_volumes: empty if *external_per_channel_volume* is false, current volume value per-channel otherwise

IDs, for nodes and streams, are (independent) 32-bit unsigned non-zero values associated to respectively nodes and streams. IDs should not be reused. A node that goes away then comes back should get a new ID. A stream closing does not allow reuse of its ID.

If a node has both sources and sinks, the sources are *monitors* of the sinks, e.g. they're loopbacks. They should have the same count in such a case.

Nodes must be independent. It must be possible to open streams to two different nodes at the same time. Be careful of multi-api libraries that collide between apis. In addition, with monitoring streams (input on an output), it must be possible to open different streams for input and output. If it's not the case, do not publish the monitoring inputs.

When external control exists, a module should change the value of *stream_info::m_node* when the user changes it, and same for *stream_info::m_volumes*. Generation number should be incremented when this happens, so that the core knows to look for changes.

Volumes are floats in dB, where 0 means 100% and -96 means no sound. audio.h provides osd::db_to_linear and osd::linear_to_db if such a conversion is needed.

Positions have two special values: unknown() means the position of the speaker or microphone is unknown, but it should be used anyway. The used position will be centered. map_on_request_only() means the input or output should not be used on full mappings but only when explicitly requested with a channel mapping.

There is an inherent race condition with this system, because things can change at any point after returning for the method. The idea is that the information returned must be internally consistent (a stream should not point to a node ID that does not exist in the structure, same for default sink) and that any external change from that state should increment the generation number, but that's it. Through the generation system the core will eventually be in sync with reality.

Input and output streams

```
virtual uint32_t stream_sink_open(uint32_t node, std::string name, uint32_t rate)␣
↪override;
virtual uint32_t stream_source_open(uint32_t node, std::string name, uint32_t rate)␣
↪override;
virtual void stream_set_volumes(uint32_t id, const std::vector<float> &db) override;
virtual void stream_close(uint32_t id) override;
virtual void stream_sink_update(uint32_t id, const int16_t *buffer, int samples_this_
↪frame) override;
virtual void stream_source_update(uint32_t id, int16_t *buffer, int samples_this_frame)␣
↪override;
```

Streams are the concept used to send or receive audio from/to the host audio system. A stream is first opened through *stream_sink_open* for speakers and *stream_source_open* for microphones and targets a specific node at a specific sample rate. It is given a name for use by the host sound services for user UI purposes (currently the game name if split_streams_per_source is false, the speaker_device/microphone_device tag if true). The returned ID must be a non-zero, never-used-before for streams value in case of success. Failures, like when the node went away between the *get_information* call and the open one, should be silent and return zero.

stream_set_volumes is used only when *external_per_channel_volume* is true and is used by the core to set the per-channel volume. The call should just be ignored if the stream ID does not exist (or is zero). Do not try to apply volumes in the module if the host API doesn't provide for it, let the core handle it.

stream_close closes a stream, The call should just be ignored if the stream ID does not exist (or is zero).

Opening a stream, closing a stream or changing the volume does not need to touch the generation number.

stream_sink_update is the method used to send data to the node through a given stream. It provides a buffer of *samples_this_frame * node channel count* channel-interleaved int16_t values. The lifetime of the data in the buffer or the buffer pointer itself is undefined after return from the method call. The call should just be ignored if the stream ID does not exist (or is zero).

stream_source_update is the equivalent to retrieve data from a node, writing to the buffer instead of reading from it. The constraints are identical.

When a stream goes away because the target node is lost it should just be removed from the information, and the core will pick up the node departure and close the stream.

Given the assumed raceness of the interface, all the methods should be tolerant of obsolete or zero IDs being used by the core, and that is why ID reuse must be avoided. Also the update methods and the open/close/volume ones may be called at the same time in different threads.

Helper class *abuffer*

```
class abuffer {
public:
    abuffer(uint32_t channels);
    void get(int16_t *data, uint32_t samples);
    void push(const int16_t *data, uint32_t samples);
    uint32_t channels() const;
};
```

The class *abuffer* is a helper provided by *sound_module* to buffer audio input or output. It automatically drops data when there is an overflow and duplicates the last sample on underflow. It must first be initialized with the number of

channels, which can be retrieved with *channels()* if needed. *push* sends *samples* * *channels* 16-bit samples in the buffer. *get* retrieves *samples* * *channels* 16-bit samples from the buffer, on a FIFO basis.

It is not protected against multithreading, but uses no class variables. So just don't read and write from one specific abuffer instance at the same time. The system sound interface mandated locking should be enough to ensure that.

12.19 BGFX vector CRT renderer

- *Purpose and status*
- *Source map*
- *Pipeline overview*
- *Integration and frame ordering*
- *Time and phosphor persistence*
- *CPU instance generation*
  - *Approximate scan timing*
  - *Beam width and resolution scaling*
- *Beam shaders*
  - *Vertex stage*
  - *Intensity-dependent profile*
  - *Pixel-footprint filtering*
  - *Scan-order variation*
- *Bloom*
- *Composite*
- *Runtime controls*
- *Performance characteristics*
  - *Measured 4K performance and VSync*
- *Screenshots*
- *Building and validation*
- *Current limitations and future work*

12.19.1 Purpose and status

The BGFX vector CRT renderer is a GPU pipeline for MAME vector games. It replaces direct, frame-oriented line drawing with a persistent HDR phosphor buffer, instanced Gaussian beam deposition, approximate scan-order timing, bloom, and tone-mapped composition.

The `bgfx_vectorcrt` option is enabled by default; set it to `0` to disable it. The renderer is used only with the BGFX video backend and when the primitive list contains a vector-buffer marker. If its effects, geometry, or RGBA16F render targets cannot be created, BGFX falls back to normal vector line rendering.

12.19.2 Source map

The implementation is split across these locations:

- `src/osd/modules/render/bgfx/vectorrenderer.cpp` and `.h` manage render targets, instance generation, pass scheduling, emulation time, and sliders.
- `src/osd/modules/render/drawbgfx.cpp` and `.h` integrate the renderer, intercept vector primitives, preserve draw ordering, and composite output.
- `src/osd/modules/render/bgfx/shaders/chains/vector-crt/` contains shader source and varying definitions.
- `bgfx/effects/vector-crt/` contains render state, uniforms, samplers, and shader program selection.
- `bgfx/shaders/<backend>/chains/vector-crt/` contains compiled shaders for Direct3D 9, Direct3D 11, OpenGL ES, OpenGL, Metal, and SPIR-V.

12.19.3 Pipeline overview

```
Ordered MAME vector primitives
            |
            v
CPU instance buffer (one record per segment)
            |
            v
Previous RGBA16F accumulation -- exponential decay --+
                                                     |
Instanced Gaussian beam deposition ------------------+
            |
            +----> persistent full-resolution phosphor
            |
            v
Half-resolution box downsample
            |
            v
Two horizontal/vertical Gaussian blur iterations
            |
            v
Phosphor + bloom -> exposure -> tone map -> gamma -> output
```

The persistent accumulation is ping-ponged between two full-resolution RGBA16F targets. Bloom uses two RGBA16F targets rounded up to half the output dimensions:

```
bloom_width  = max(1, (width  + 1) / 2);
bloom_height = max(1, (height + 1) / 2);
```

RGBA16F is required because additive beam contributions must retain HDR energy across frames before tone mapping.

12.19.4 Integration and frame ordering

`renderer_bgfx` creates `bgfx_vector_renderer` when `bgfx_vectorcrt` is enabled. At the start of a frame, `prepare()` scans the primitive list:

- `PRIMFLAG_VECTORBUF` identifies a vector screen and activates the renderer.
- Line primitives with `PRIMFLAG_VECTOR` are collected in display-list order.

These vector lines are omitted from normal BGFX line batching. The completed CRT image is composited where the vector-buffer marker occurs, preserving its order relative to artwork and other primitives.

Each active frame performs the following operations:

1. Decay or clear the persistent phosphor target.
2. Add all vector beam instances to the same target.
3. Downsample the phosphor image into the first bloom target.
4. Apply two iterations of horizontal and vertical bloom blur.
5. Composite persistent phosphor and bloom into the configured BGFX view.

Views use sequential mode because later passes consume textures written by earlier passes.

12.19.5 Time and phosphor persistence

The renderer uses emulated time rather than wall-clock time:

- The first frame assumes 1/60 second.
- A repeated or effectively zero timestamp is ignored, preventing repeated excitation of the same display list while paused.
- Backward time after reset, state load, or rewind clears the accumulation.
- A forward interval is capped at 0.100 second to avoid extreme pass values.

For frame interval `dt` and persistence time constant `tau`, the decay pass applies:

```
decay = exp(-dt / max(tau, 0.001))
I_next = I_previous * decay
```

This is inter-frame phosphor decay. Scan-order attenuation within newly drawn geometry uses a deliberately gentler lifetime described below.

12.19.6 CPU instance generation

Every vector segment becomes one 12-float instance:

```
struct beam_instance
{
    float x0, y0, x1, y1;       // target-pixel endpoints
    float red, green, blue;      // beam colour
    float sigma;                 // Gaussian sigma in target pixels
    float start, duration;       // normalised display-list scan interval
    float intensity;             // primitive alpha
    float unused;
};
```

The instance vectors seen by the vertex shader are:

```
i_data0 = (x0, y0, x1, y1)
i_data1 = (red, green, blue, sigma)
i_data2 = (start, duration, intensity, unused)
```

Approximate scan timing

The implementation does not receive hardware beam timestamps. It approximates constant deflection speed using accumulated segment length:

```
length = max(endpoint_distance, max(primitive_width, 1 pixel))
start = elapsed_length / total_length
duration = length / total_length
```

The width/one-pixel lower bound gives points and degenerate segments a non-zero dwell interval. This preserves display-list order but does not account for blanked moves, hardware slew limits, or vector-generator timing.

Beam width and resolution scaling

The CPU converts MAME line width to Gaussian sigma with an empirical profile calibration:

```
constexpr float BEAM_SIGMA_SCALE = 0.085f;
sigma = primitive.width * m_beam_width * BEAM_SIGMA_SCALE;
```

`0.085` is not a resolution scale or the standard Gaussian full-width at half-maximum conversion. It was calibrated for the desired appearance. If `primitive.width` were interpreted as Gaussian FWHM, the conversion would be `sigma = width / 2.35482`. That would produce a materially different profile and require retuning.

`primitive.width` is already in target pixels. MAME constructs it in `render_target::add_container_primitives()` by multiplying normalised line width by the smaller target-space scale:

```
prim->width = curitem.width()
        * std::min(container_xform.xscale, container_xform.yscale);
```

For the same view and aspect ratio, `primitive.width` doubles from 1080p to 2160p. Beam width therefore scales with the rendered screen item, including layout and letterboxing. Applying another `height / 1080` factor to sigma would double-compensate.

Bloom differs because its radius originates as a UI pixel value referenced to 1080 lines, so it requires an explicit resolution scale.

12.19.7 Beam shaders

Vertex stage

The shared mesh is a six-vertex unit quad. `vs_beam.sc` expands it around each segment in target-pixel space:

1. Compute segment direction, perpendicular normal, and length.
2. Extend both endpoints by `max(6 * sigma, 1 pixel)`.
3. Emit the padded quad through a target-pixel orthographic projection.
4. Pass beam-local `along` and `across` coordinates to the fragment shader.

Six sigma contains the useful Gaussian tails. Longitudinal padding provides pixels for round endpoint caps. Zero-length segments use `(1, 0)` as a stable fallback direction.

Intensity-dependent profile

Intensity is primitive alpha. Its response is compressed before it affects beam shape:

```
intensityResponse = sqrt(clamp(intensity, 0, 1))
coreSigma = baseSigma * mix(1.00, 1.12, intensityResponse)
haloSigma = baseSigma * 3.5 * mix(1.00, 1.25, intensityResponse)
haloStrength = haloControl * mix(0.6, 1.0, intensityResponse)
```

The profile is a Gaussian core plus a wider, weaker halo. Outside the segment endpoints, longitudinal and perpendicular distance are combined for round caps:

```
pastEndpoint = max(-along, along - beamLength, 0)
distanceSquared = across^2 + pastEndpoint^2
```

Pixel-footprint filtering

Very narrow analytic Gaussians alias if sampled only at pixel centres. The fragment shader approximates a pixel as a box filter. For footprint `pixelAcross`, box variance is `pixelAcross^2 / 12` and is added to beam variance:

```
filteredSigma = sqrt(sigma^2 + pixelVariance)
```

The peak is multiplied by `sigma / filteredSigma` so broadening does not create energy. Core and halo are filtered independently. This stabilises subpixel beams without imposing a fixed CPU sigma floor.

Scan-order variation

For a fragment on the physical segment:

```
segmentPosition = clamp(along / beamLength, 0, 1)
arrival = clamp(start + duration * segmentPosition, 0, 1)
age = frameInterval * (1 - arrival)
```

A degenerate segment uses position 0.5. Earlier vectors have greater age at presentation and are attenuated slightly more:

```
scanPersistence = max(phosphorPersistence, frameInterval * 10)
temporal = exp(-age / max(scanPersistence, 0.001))
```

The ten-frame floor prevents short physical persistence from nearly erasing vectors at the beginning of the display list before presentation. It keeps scan variation subtle; inter-frame decay still uses physical persistence.

Final deposited HDR energy is:

```
energy = colour * intensity * radialProfile * temporal * beamEnergyGain
```

The beam effect uses additive blending into the current accumulation target.

12.19.8 Bloom

The full-resolution phosphor image is reduced with a four-sample box filter. Bloom uses a separable nine-tap Gaussian kernel. Each axis samples the centre texel and integer offsets from one through four texels in both directions. The fragment shader calculates weights from the uniform sigma and normalises them for unit DC gain. Two complete horizontal/vertical iterations produce four blur draws per frame.

Bloom radius is referenced to 1080 lines:

```
bloom_scale = clamp(output_height / 1080.0f, 0.25f, 2.0f);
bloom_radius = m_bloom_radius * bloom_scale;
```

This keeps bloom approximately constant as a fraction of screen height through 2160p. The clamp prevents extreme kernels outside that range. The radius is converted from output pixels to bloom-texture texels independently for each axis:

```
sigma_x = bloom_radius * bloom_width  / output_width;
sigma_y = bloom_radius * bloom_height / output_height;

pass_scale   = 1 / sqrt(bloom_passes);
pass_sigma_x = sigma_x * pass_scale;
pass_sigma_y = sigma_y * pass_scale;

horizontal_u_blur = (1 / bloom_width,  0,                pass_sigma_x, 0);
vertical_u_blur   = (0,                1 / bloom_height, pass_sigma_y, 0);
```

`u_blur.xy` is therefore exactly one bloom texel along the active axis, and `u_blur.z` is Gaussian sigma in bloom texels. The fragment shader samples at integer multiples of that texel step. Minification and magnification remain linearly filtered, but the kernel no longer relies on fractional bilinear tap positions. Gaussian variances add across the repeated iterations, so dividing each pass's sigma by the square root of the pass count makes the combined blur match the radius requested by the slider.

There is no brightness threshold. All positive phosphor energy contributes, and `Vector bloom strength` controls bloom during composition.

12.19.9 Composite

The composite shader combines HDR sources and applies exposure:

```
hdr = (phosphor + bloom * bloomStrength) * exposure
```

It computes Rec. 709 luminance and applies exponential tone mapping to luminance rather than independently to RGB:

```
L = dot(hdr, (0.2126, 0.7152, 0.0722))
mappedL = 1 - exp(-L)
mappedRGB = hdr * mappedL / L
```

Scaling RGB by the luminance ratio retains the hue and saturation of bright vectors. The result receives a `1 / 2.2` gamma transform. The shader supports edge vignette through `u_composite.w`, but C++ currently sets it to zero. Additive composition lets the image occupy the vector screen's normal position in a view containing artwork.

12.19.10 Runtime controls

Sliders appear only while a vector screen is present. Construction explicitly invokes each slider callback, so temporary zero-valued member initialisation cannot reach rendering.

Table 1: Vector CRT controls

Control	Range	Default	Meaning
Vector phosphor persistence	1-100 ms	2 ms	Inter-frame exponential time constant
Vector beam width	0.30-4.00x	0.75x	Multiplier on MAME's target-scaled line width
Vector beam intensity	0.10-5.00x	4.00x	HDR energy deposited by each beam
Vector beam halo	0.00-1.00x	0.04x	Strength of the wide Gaussian component
Vector bloom strength	0.00-3.00x	0.20x	Bloom contribution during composition
Vector bloom radius	0.50-2.60 px at 1080p	2.12	Blur radius before resolution scaling
Vector exposure	0.10-4.00x	0.78x	Gain before tone mapping

Core vector controls still affect primitive colour, intensity, and line width before this renderer receives them.

12.19.11 Performance characteristics

CPU work is linear in segment count: collect primitives, measure lengths, and fill transient instance buffers. There is no CPU per-pixel rasterisation.

GPU beam cost is proportional to padded quad area. Bloom has one half- resolution downsample and four half-resolution blur passes. Decay and composition are full-resolution. Memory is dominated by two full-resolution and two half-resolution RGBA16F targets.

Transient instance-buffer exhaustion is handled in batches. Failure to allocate any instances logs a warning and stops beam submission for that frame.

Measured 4K performance and VSync

A Star Wars test at 4K used this development machine:

Table 2: Benchmark host

Component	Specification
System	BESSTAR TECH LIMITED DMAF5 mini PC
CPU	AMD Ryzen 5 3550H, 4 cores and 8 logical processors; 2.10 GHz reported maximum clock
GPU	Integrated AMD Radeon Vega 8 Graphics
GPU driver	31.0.12027.9001, dated 2023-03-30
Memory	16 GB installed; approximately 15.7 GiB visible to Windows
Operating system	Microsoft Windows 10 Home 64-bit, version/build 10.0.19045
Display mode	3840 by 2160 at 60 Hz

Windows reports a 256 MiB adapter-memory allocation for Vega 8, but the integrated GPU also uses shared system memory. This is not total memory available to the renderer.

Table 3: Star Wars 4K benchmark

Mode	Average speed
Vector CRT with normal `waitvsync`	94.85%
Vector CRT with `-nowaitvsync`	100.00%
Vector CRT with `-nowaitvsync -nothrottle`	204.04%

The standard renderer reached approximately 99.8% with normal `waitvsync` on the same machine. The vector CRT renderer can sustain real-time 4K rendering in this test and has roughly twice-real-time raw throughput when synchronisation and throttling are disabled. These results are not portable guarantees.

The approximately 5% difference is related to synchronised presentation rather than inability to render the scene. The 94.85% result does not indicate GPU saturation: disabling VSync restores 100%, and disabling throttling exposes substantial headroom. The likely cause is an interaction between Star Wars' native update rate, MAME throttling, presentation timing, and VSync.

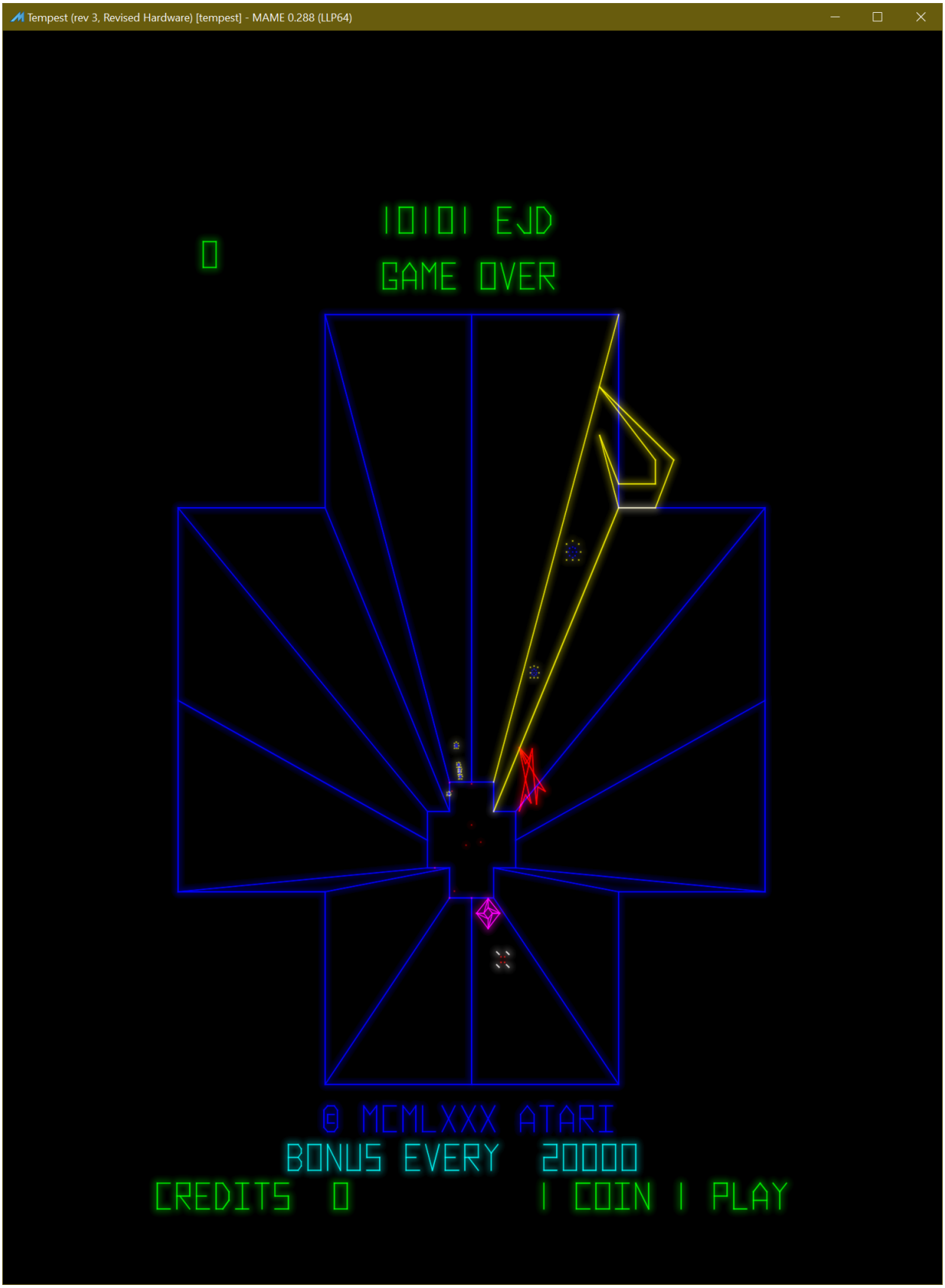
Use `-nowaitvsync -nothrottle` to compare raw throughput. Test normal synchronised operation separately for user-visible frame pacing. Both matter: fewer bloom passes may reduce latency, power, and utilisation without being required for real time, while default-on behaviour must account for users who observe sub-100% speed with normal `waitvsync`.

12.19.12 Screenshots

Asteroids (monochrome vector display)



Tempest (colour vector display)


Tempest (rev 3, Revised Hardware) [tempest] - MAME 0.288 (LLP64)
10101 EJD
0
GAME OVER
© MCMLXXX ATARI
BONUS EVERY 20000
CREDITS 0
1 COIN 1 PLAY

12.19.13 Building and validation

After changing a vector CRT shader, rebuild every supported backend with:

```
make shaders CHAIN=vector-crt
```

Expected binary changes appear below `bgfx/shaders/{dx9,dx11,essl,glsl,metal,spirv}/chains/vector-crt/`.

Useful validation systems include Asteroids, Battlezone, Tempest, and Star Wars. Check at minimum:

- Thin diagonal, horizontal, and vertical vectors for aliasing and equal energy.
- Points and short segments for stable round caps and non-zero scan duration.
- Bright text for excessive core swelling or halo.
- Early and late display-list geometry for subtle scan-order variation.
- Pause, reset, state load, rewind, and large emulation-time steps.
- 1080p and 2160p output for proportional beam width and bloom radius.
- Views with artwork for correct screen scaling and composite order.

12.19.14 Current limitations and future work

Potential improvements include:

- Hardware-derived timestamps, blanked-move timing, and dwell time.
- Independent RGB decay constants and measured phosphor spectra.
- Deflection-amplifier slew limits, endpoint ringing, overshoot, and jitter.
- Position-dependent focus or convergence.
- Measured beam and optical point-spread functions.
- HDR-native output and display-aware calibration.
- Multi-monitor vector timing and resource ownership.

These extensions should preserve the architecture: the CPU uploads ordered metadata while beam rasterisation, persistence, bloom, and display mapping remain GPU operations.

CHAPTER

THIRTEEN

MAME AND SECURITY CONCERNS

MAME is not intended or designed to run in secure sites. It has not been security audited for such types of usage, and has been known in the past to have flaws that could be used for malicious intent if run as administrator or root.

We do not suggest or condone the use of MAME as administrator or root and use as such is done at your own risk.

Bug reports, however, are always welcome.

CHAPTER

FOURTEEN

THE MAME LICENSE

The MAME project as a whole is distributed under the terms of the GNU General Public License, version 2 or later (GPL-2.0+), since it contains code made available under multiple GPL-compatible licenses. A great majority of files (over 90% including core files) are under the 3-Clause BSD License and we would encourage new contributors to distribute files under this license.

Please note that MAME is a registered trademark of Gregory Ember, and permission is required to use the "MAME" name, logo, or wordmark.

> Copyright (c) 1997-2026 MAMEDev and contributors
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Please see the license information for further details.

CHAPTER

FIFTEEN

CONTRIBUTE

The documentation on this site is the handiwork of our many contributors.