MAME Documentation

Release 0.227

MAMEdev Team

Apr 11, 2021

CONTENTS

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 on GitHub.

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?). Over time, MAME (originally stood for Multiple Arcade Machine Emulator) absorbed the sister-project MESS (Multi Emulator Super System), so MAME now documents a wide variety of (mostly vintage) computers, video game consoles and calculators, in addition to the arcade video games that were its initial focus.

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 games. 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 game code are included in the executable.

1.2 II. Cost

MAME is free. Its source code is free. The project as whole is distributed under the GNU General Public License, version 2 or later (GPL-2.0+), but most of code (including core functionality) is also available under the 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 cannot be distributed without the explicit permission of the copyright holder(s). They are not "abandonware", nor has any of the software supported by MAME passed out of copyright.

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 was derived.

1.5 V. Official Contact Information

For questions regarding the MAME license, trademark, or other usage, go to http://www.mamedev.org/

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 and compiling MAME.

3.1 An Introduction to MAME

MAME, formerly was 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 machines, while MESS covered everything else. They are now merged into the one MAME.

MAME is mostly programmed in C with some core components in C++. MAME can currently emulate over 32000 individual systems from the last 5 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

ProjectMESS 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 mac.c 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 to 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 follow the instructions at the contact page or post to the MAME Forums at http://forum.mamedev.org/

3.4 Supported OS

The current source code can be directly compiled under all the main OSes: Microsoft Windows (both with DirectX/BGFX native support or with SDL support), Linux, FreeBSD, and Mac OS X. Also, both 32-bit and 64-bit are supported, but be aware 64-bit often shows significant speed increases.

3.5 System Requirements

MAME is written in fairly generic C/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 binaries are compiled and designed to run on a standard Windows-based system. The minimum requirements are:

- Intel Core series CPU or equivalent, at least 2.0 GHz
- 32-bit OS (Vista SP1 or later on Windows, 10.9 or later on Mac)
- 4 GB RAM
- DirectX 9.0c for Windows
- A Direct3D, or OpenGL capable graphics card
- Any DirectSound capable sound card/onboard audio

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 the games to full screen. To make use of this, you should have a modern Direct3D 8-capable video card with at least 16MB of video RAM.

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 dropping 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.

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

3.6 Installing MAME

3.6.1 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?

3.6.2 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 *All Platforms*.

3.7 Compiling MAME

3.7.1 All Platforms

- To compile MAME, you need a C++17 compiler and runtime library. We support building with GCC version 7.2 or later and clang version 6 or later. MAME should run with GNU libstdc++ version 7.2 or later.
- Whenever you are changing build parameters, (such as switching between a SDL-based build and a native Windows renderer one, or adding tools to the compile list) you need to run a **make REGENIE=1** to allow the settings to be regenerated. Failure to do this will cause you very 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 statement, like **make REGENIE=1 TOOLS=1**
- You can do driver specific builds by using *SOURCES=<driver>* in your make statement. For instance, building Pac-Man by itself would be **make SOURCES=src/mame/drivers/pacman.cpp REGENIE=1** including the necessary *REGENIE* for rebuilding the settings.
- Speeding up the compilation can be done by using more cores from your CPU. This is done with the **-j** parameter. *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 may increase compilation time. The optimal number depends on many factors, including number of CPU cores, available RAM, disk and filesystem performance, and memory bandwidh.* For instance, **make -j5** is a good starting point on a system with a quad-core CPU.
- 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.

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

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

```
make SOURCES=src/mame/drivers/pacman.cpp TOOLS=1 REGENIE=1 -j5
```

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

```
make -j3
```

3.7.2 Microsoft Windows

MAME for Windows is built using the MSYS2 environment. You will need Windows 7 or later and a reasonably up-to-date MSYS2 installation. We strongly recommend building MAME on a 64-bit system. Instructions may need to be adjusted for 32-bit systems.

- A pre-packaged MSYS2 installation including the prerequisites for building MAME can be downloaded from the MAME Build Tools page.
- After initial installation, you can update the MSYS2 environment using the **pacman** (Arch package manage) command.
- 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 2.0.3 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 5.

Using a standard MSYS2 installation

You may also build MAME using a standard MSYS2 installation and adding the 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.
- Download the latest version of the `mame-essentials` package from the MAME package repository and install it using the **pacman** command.
- Add the `mame` repository to `/etc/pacman.conf` using `/etc/pacman.d/mirrorlist.mame` for locations.
- Install packages necessary to build MAME. At the very least, you'll need `bash`, `git`, `make`.
- For 64-bit builds you'll need `mingw-w64-x86_64-gcc` and `mingw-w64-x86_64-python`.
- For 32-bit builds you'll need `mingw-w64-i686-gcc` and `mingw-w64-i686-python`.
- For debugging you may want to install `gdb`.
- To link using the LLVM linker (generally much faster than the GNU linker), you'll need `mingw-w64-x86_64-lld` and `mingw-w64-x86_64-libc++` for 64-bit builds, or `mingw-w64-i686-lld` and `mingw-w64-i686-libc++` for 32-bit builds.
- To build against the portable SDL interfaces, you'll need `mingw-w64-x86_64-SDL2` and `mingw-w64-x86_64-SDL2_ttf` for 64-bit builds, or `mingw-w64-i686-SDL2` and `mingw-w64-i686-SDL2_ttf` for 32-bit builds.
- To build the Qt debugger, you'll need `mingw-w64-x86_64-qt5` for 64-bit builds, or `mingw-w64-i686-qt5` for 32-bit builds.
- To build the HTML user/developer documentation, you'll need `mingw-w64-x86_64-librsvg`, `mingw-w64-x86_64-python-sphinx`, `mingw-w64-x86_64-python-sphinx_rtd_theme` and `mingw-w64-x86_64-python-sphinxcontrib-svg2pdfconverter` for a 64-bit MinGW environment (or alternatively `mingw-w64-i686-librsvg`, `mingw-w64-i686-python-sphinx`, `mingw-w64-i686-python-sphinx_rtd_theme` and `mingw-w64-x86_64-python-sphinxcontrib-svg2pdfconverter` a 32-bit MinGW environment).
- 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`.
- For 64-bit builds, open **MSYS2 MinGW 64-bit** from the start menu, and set up the environment variables `MINGW64` to `/mingw64` and `MINGW32` to an empty string (e.g. using the command **export MINGW64=/mingw64 MINGW32=** in the Bash shell).
- For 32-bit builds, open **MSYS2 MinGW 32-bit** from the start menu, and set up the environment variables `MINGW32` to `/mingw32` and `MINGW64` to an empty string (e.g. using the command **export MINGW32=/mingw32 MINGW64=** in the Bash shell).

For example you could use these commands to ensure you have the packages you need to compile MAME, omitting the ones for configurations you don't plan to build for or combining multiple **pacman** commands to install more packages at once:

```
pacman -Syu
pacman -S curl git make
pacman -S mingw-w64-x86_64-gcc mingw-w64-x86_64-libc++ mingw-w64-x86_64-lld mingw-w64-
↪x86_64-python
pacman -S mingw-w64-x86_64-SDL2 mingw-w64-x86_64-SDL2_ttf
pacman -S mingw-w64-x86_64-qt5
pacman -S mingw-w64-i686-gcc mingw-w64-i686-libc++ mingw-w64-i686-lld mingw-w64-i686-
↪python
pacman -S mingw-w64-i686-SDL2 mingw-w64-i686-SDL2_ttf
pacman -S mingw-w64-i686-qt5
```

You could use these commands to install the current version of the mame-essentials package and add the MAME package repository to your pacman configuration:

```
curl -O "https://repo.mamedev.org/x86_64/mame-essentials-1.0.6-1-x86_64.pkg.tar.xz"
pacman -U mame-essentials-1.0.6-1-x86_64.pkg.tar.xz
echo -e '\n[mame]\nInclude = /etc/pacman.d/mirrorlist.mame' >> /etc/pacman.conf
```

Building with Microsoft Visual Studio

- You can generate Visual Studio 2017 projects using **make vs2017**. The solution and project files will be created in `build/projects/windows/mame/vs2017` by default (the name of the `build` folder can be changed using the `BUILDDIR` option). This will always regenerate the settings, so **REGENIE=1** is *not* needed.
- Adding **MSBUILD=1** to the make options will build 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.
- MAME can only be compiled with the Visual Studio 15.7.6 tools. Bugs in newer versions of the Microsoft Visual C/C++ compiler prevent it from compiling MAME.
- 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 `/ming64/bin` and/or `/mingw32/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` 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 may have issues debugging MinGW programs like MAME. You may get better results by installing the MinGW version of gdb and running it from a Microsoft command prompt window to debug MAME.

GNU make supports both POSIX-style shells (e.g. bash) and the Microsoft cmd.exe shell. One issue to be aware of when using the cmd.exe shell is that the `copy` command doesn't provide a useful exit status, so file copy tasks can fail silently.

It is not possible to cross-compile a 32-bit version of MAME using 64-bit MinGW tools on Windows, the 32-bit MinGW tools must be used. This causes issues due to the size of MAME. It is not possible to link a full 32-bit MAME build including the SDL OS-dependent layer and the Qt debugger. GNU ld and lld will both run out of memory, leaving an output file that doesn't work. It's also impossible to make a 32-bit build with full local variable symbols. GCC may run out of memory, and certain source files may exceed the limit of 32,768 sections imposed by the PE/COFF object file format.

3.7.3 Fedora Linux

You'll need a few prerequisites from your Linux distribution. Make sure you get SDL2 2.0.4 or later as earlier versions are buggy:

```
sudo dnf install gcc gcc-c++ SDL2-devel SDL2_ttf-devel libXi-devel libXinerama-devel␣
↪qt5-qtbase-devel qt5-qttools expat-devel fontconfig-devel alsa-lib-devel
```

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 SPHINXBUILD=sphinx-build-3 html
```

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

You'll need a few prerequisites from your Linux distribution. Make sure you get SDL2 2.0.4 or later as earlier versions are buggy:

```
sudo apt-get install git build-essential python libsdl2-dev libsdl2-ttf-dev␣
↪libfontconfig-dev qt5-default
```

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*).

3.7.5 Arch Linux

You'll need a few prerequisites from your distro:

```
sudo pacman -S base-devel git sdl2 gconf sdl2_ttf gcc qt5
```

Compilation is exactly as described above in All Platforms.

3.7.6 Apple macOS

You'll need a few prerequisites to get started. Make sure you're on OS X 10.14 Mojave or later for Intel Macs or macOS 11.0 Big Sur for Apple Silicon. You will need SDL2 2.0.4 or later for Intel or SDL2 2.0.14 on Apple Silicon.

- 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 SDL2 installed.

- Go to this site and download the *Mac OS X* .dmg file
- If the .dmg doesn't auto-open, open it
- Click "Macintosh HD" (your Mac's hard disk) in the Locations sidebar 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.

Lastly 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.

3.7.7 Emscripten Javascript and HTML

First, download and install Emscripten 1.37.29 or later by following the instructions at the official site

Once Emscripten has been installed, 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/drivers/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/drivers/apple2e.cpp,src/mame/machine/
↪applefdc.cpp
```

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

Emscripten supports compiling to WebAssembly with a JavaScript loader instead of all-JavaScript, and in later versions this is actually the default. To force WebAssembly on or off, add **WEBASSEMBLY=1** or **WEBASSEMBLY=0** to the make command line.

Other make parameters can also be used, e.g. **-j** for parallel 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.

3.7.8 Compiling the Documentation

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

On Debian/Ubuntu flavors of Linux, you'll need python3-sphinx/python-sphinx and the python3-pip/python-pip packages, depending on whether you're using Python 3 or Python 2:

```
sudo apt-get install python3-sphinx python3-pip
```

or:

```
sudo apt-get install python-sphinx python-pip
```

You'll then need to install the SVG handler:

```
pip3 install sphinxcontrib-svg2pdfconverter
```

or:

```
pip install sphinxcontrib-svg2pdfconverter
```

depending on whether you're using Python 3 or Python 2.

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

```
sudo apt-get install 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* containing the output).

3.7.9 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.)

PYTHON_EXECUTABLE Set the Python interpreter command. You need Python 2.7 or Python 3 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.

Optional features

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

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

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.

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_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_BUNDLED_LIB_SDL2 Set to **1** to prefer the version of SDL provided with the MAME source over the system installation. (This is enabled by default for Visual Studio and Android builds. For other configurations, the system installation of SDL is preferred.)

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.

3.7.10 Known Issues

Issues with specific compiler versions

- GCC 7 for 32-bit x86 targets produces spurious out-of-bounds access warnings. Adding **NOWERROR=1** to your build options works around this by not treating warnings as errors.

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 `_FORTIY_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.

3.7.11 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).

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++17 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.

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 doubleclicking 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 Macintosh OS X 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 softlists, 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 softlists, 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 softlist-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 softlist in order to work in this way.

If you are loading an arcade board or other non-softlist 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 shortname 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 you 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 *Frontends* available for MAME.

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

mame.exe -help

tells what MAME is the basic structure of MAME launching options, i.e. 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 Commandline Options* section of this manual.

mame.exe -showconfig

gives you a (quite long) list of available configuration options for MAME. These configuration 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 Commandline 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, hence 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 basically 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 configure a bit more your setup. In this case, keep in mind the order in which options are read; see *Order of Config Loading* for details.

4.2 Default Keyboard Controls

4.2.1 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.

4.2.2 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. Switches to the next UI panel.

~ (tilde key) Toggles the On-Screen Display.

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

When the on-screen display is visible, you can use the following keys to control it:

- **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.
- **Enter** - reset parameter value to its default.
- **Control+Left** - decrease the value by 10x.
- **Shift+Left** - decrease the value by 0.1x.
- **Alt+Left** - decrease the value by the smallest amount.
- **Control+Right** - increase the value by 10x.
- **Shift+Right** - increase the value by 0.1x.
- **Alt+Right** - increase the value by the smallest amount.
- **End** - temporarily hide the On Screen Display.
- **Home** - bring the On Screen Display back after hiding it.

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 Highlight first UI menu option.

End Select last UI menu option.

Left Shift+Tab Select previous UI panel.

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.

[UI Previous Group

] UI Next Group

P Pauses the emulated machine.

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

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

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 display. 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.
- **Page Up/Page Down** - scroll up/down one page at a time.
- **Home/End** - move to top/bottom of list.
- **-/+** - increase/decrease the 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.
- **Page Up/Page Down** - scroll up/down one page at a time.
- **Home/End** - move to top/bottom of list.
- **Left/Right** - change color displayed.
- **R** - rotate tiles 90 degrees clockwise.

- **-/+** - increase/decrease the number of tiles per row.
- **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.
- **-/+** - increase/decrease the zoom factor.
- **Enter** - switch to palette/colortable mode.

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

Left Ctrl+F5 Toggle Filter. (*SDL MAME only*)

Left Alt+Left Ctrl+F5 Toggle HLSL Post-Processing. (*Windows non-SDL MAME only*)

F6 Toggle cheat mode. (if started with "-cheat")

Left Ctrl+F6 Decrease Prescaling.

Left Ctrl+F7 Increase Prescaling.

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 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.

F8 Decrease frame skipping on the fly.

F9 Increase frame skipping on the fly.

F10 Toggle speed throttling.

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 Fast forward. While held, runs game with throttling disabled and with the maximum frameskip. (*Windows non-SDL MAME only*)

Page Down Fast forward. While held, runs game with throttling disabled and with the maximum frameskip. (*SDL MAME only*)

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 Exits emulator. Cancel current UI option.

4.2.3 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.

- ***(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

4.2.4 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

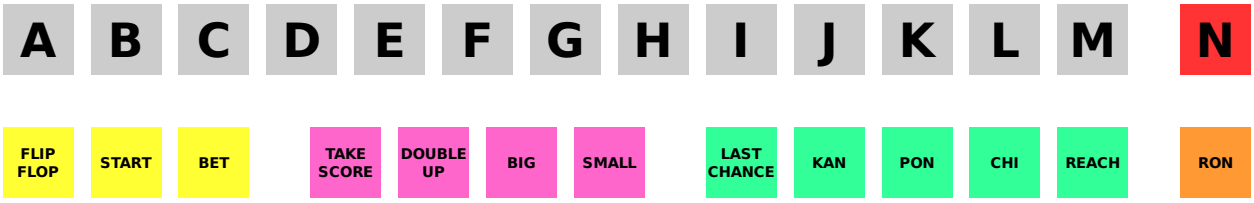
0 ***(on numeric keypad)*** Player 4 Button 1

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

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

4.2.5 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

Semicolon Player 1 Taiwanese Mahjong P

Q Player 1 Taiwanese Mahjong Q

4.2.6 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

4 ***(not numeric keypad)*** Take Score

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

3 ***(not numeric keypad)*** 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.

D Half Gamble

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

A High

S Low

O Door

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

4.2.7 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 (this Machine)" section of MAME's configuration 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.

4.2.8 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 (this Machine)" section of MAME's configuration menu for details for the machine you are currently using.

4.3 MAME Menus

If you started MAME without any command line parameters, you'll be shown the game selection menu immediately. While the keys listed above will let you navigate the menus, you can also use a mouse.

[todo: This needs SERIOUS expansion. Waiting on answer to a few questions..]

4.4 Frontends

There are a number of third party tools for MAME to make system and software selection simpler. These tools are called "Frontends", and there are far too many to list conclusively here. Some are free, some are commercial-- caveat emptor. Some older frontends predate the merging of MAME and MESS and do not support the additional console, handheld, etc functionality that MAME inherited from MESS.

This following list is not an endorsement of any of these frontends by the MAME team, but simply showing a number of commonly used free frontends as a good starting point to begin from.

QMC2 (multiple platforms)
Download: http://qmc2.batcom-it.net/

IV/Play (Microsoft Windows)
Download: http://www.mameui.info/

EmuLoader (Microsoft Windows)
Download: http://emuloader.mameworld.info/

The MAME team will not provide support for issues with frontends. For support, we suggest contacting the frontend author or trying any of the popular MAME-friendly forums on the internet.

4.5 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.

4.5.1 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.

4.5.2 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.

4.5.3 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.

4.5.4 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.

4.5.5 ROMs and CHDs

ROM chip data tends to be relatively small and gets loaded to system memory outright. Some games also used additional storage mediums such as hard drives, CD-ROMs, DVDs, and Laserdiscs. Those storage mediums are, for multiple technical reasons, not well-suited to being stored the same way as ROM data and won't fit completely in memory in some cases.

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

4.6 Common Issues and Questions (FAQ)

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

4.6.1 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 mashing the button. The only ways you could get credit feeding 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'll not trigger this.

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

In many cases, updates to various subsystems such as HLSL, BGFX, or Lua plugins come as updates to the external shader files as well as to the core MAME code. Unfortunately, builds that come from third parties may come as just a main MAME executable or with outdated external files, which can break the coupling between these external files and MAME core code. Despite repeated attempts at contacting some of these third parties to warn them, they persist in distributing broken MAME updates.

As we have no control over how third parties distribute these, all we really can do is disclaim the use of sites like EmuCR and say that we cannot provide support for packages we didn't build. Compile your own MAME or use one of the official packages provided by us.

You may also run into this problem if you have not replaced the contents of the HLSL and BGFX folders with the latest files when updating your MAME install with a new official build.

4.6.3 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.

4.6.4 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.

4.6.5 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.

4.6.6 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).

4.6.7 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*.

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

While this rule isn't always true, there is typically a method to how sets are arranged. The usual priority is to go with the **World** set if it's available, **US** if no World English set exists, and **Japanese** or other origin region if no World or US English set.

Exceptions arise where the US or World sets have significant censorship/changes from the original version. For instance, Gals Panic (set **galsnew**) uses the US version as parent because it has additional features compared to the world export version (set **galsnewa**). These features are optional censorship, an additional control layout option (stick with no button use), and English-language voice clips.

Another exception comes for games where it was licensed to a third party for export release. Pac-Man, for instance, was published by Midway in the US though it was created by Namco. As a result, the parent set is the Japanese **puckman** set, which retains the Namco copyright.

Lastly, a developer adding a new set can choose to use whatever naming and parent scheme they wish and are not restricted to the above rules. Most follow these guidelines, however.

4.6.9 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-commerical 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.

4.6.10 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 & white issue.

4.6.11 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.

4.6.12 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.

4.6.13 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 his system, he is not allowed to legally include any ROMs with his system. If he has purchased a license to the ROMs in your name from a distributor or vendor with legitimate licenses, then he is okay to include them 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.

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

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

Even worse, a lot of these folks like to claim that they are helping the project. In fact, 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. You are only helping criminals make a profit through selling software they have no right to sell. **Anybody using the MAME name and/or logo to sell such products is also in violation of the MAME trademark.**

4.6.15 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 violating the copyright on computer programs, which is what you are doing if you make copies of ROMs.

4.6.16 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 put ROMs up for download on their sites, in order to justify the fact that they were breaking the law. There is nothing like this in any copyright law.

4.6.17 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.

4.6.18 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.

4.6.19 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

4.6.20 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 do raw reads of the joystick(s), mouse/mice, and keyboard(s). This works with most devices and provides the most stable results. However, some devices need special drivers to translate their output and these drivers may not work with raw input.

One thing you can try is setting the keyboardprovider, mouseprovider, or joystickprovider setting (depending on which kind of device your input device acts as) from rawinput to one of the other options such as dinput or win32. See *OSD-related Options* for details on supported providers.

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

MAME added support in version 0.23 for the use of a soundcard's onboard OPL2 (Yamaha YM3812 chip) instead of emulating the OPL2. This feature was never supported on the official Windows version of MAME and was dropped entirely as of MAME 0.60, because the OPL2 emulation in MAME had become advanced enough to be a better solution in almost all cases as of that time; now it's superior to using a sound card's YM3812 in all cases, especially as modern cards lack a YM3812.

Unofficial builds of MAME may have supported it for varying amounts of time as well.

4.6.22 What happened to the MAME support for autofire?

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

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

- Start MAME with no system selected.
- Choose *Plugins* at the bottom (just above Exit)
- Turn Autofire on.

The setting will be automatically saved for future use.

Once you're running a system of your choice, bring up the MAME menu (using the **Tab** key by default), select *Plugin Options* and then select *Autofire*. From there, depending on whether you have an existing autofire button set up or not, it will either show the existing entry/entries or it will ask you to select the input for the autofire.

Typically you'll be choosing *P1 Button 1* for systems like Galaga, Alcon, and the like. The *Hotkey* is the button you press for the autofire effect. This can be any keyboard key or joystick button you wish. As of 0.216, mouse buttons are not yet supported.

On frames and *Off frames* are how long to leave the button pressed and released in number of frames. Some systems do not read the inputs fast enough for 1 and 1 to be usable. You may need to try 2 and 2 (e.g. Alcon) or other combinations. Try fine-tuning these to your taste.

The autofire configuration for that system will be saved in a `systemname.cfg` file (e.g. `alcon.cfg`) inside the Autofire folder for future use. Each system will have its own configuration.

Note that if you set the autofire button to an input button that's also defined in MAME's inputs for the running system, you may get unexpected results-- Using Gradius as an example:

> If you set button 1 on your controller to fire, then set autofire to button 1 as well, holding the button down to shoot will not trigger the autofire because the button never gets released from you holding the non-autofire button 1. This will also happen if you set a different button as autofire (say, button 3 in this case), and hold button 1 down while holding button 3 down.
>
> If you set button 3 on your controller to autofire and set button 3 to be 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 suggested you choose a button for autofire that is not in use for anything else in the current system.

4.6.23 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 COMMANDLINE USAGE AND OS-SPECIFIC CONFIGURATION

5.1 Universal Commandline Options

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

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 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
```

(continues on next page)

(continued from previous page)

```
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)
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-a07.171           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)
                                  ...
                                  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
```

(continues on next page)

(continued from previous page)

```
                                    paddles          Apple II paddles
```

-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
---------------- ---------------------------- ----------------------------
↪---
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>
```

(continues on next page)

(continued from previous page)

```
                        </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
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
```

-uifontprovider

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

Table 1: Supported UI font providers per-platform

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

Example:

```
mame ajax -uifontprovider dwrite
```

-keyboardprovider

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

Table 2: Supported keyboard input providers per-platform

Microsoft Windows	auto[2].	rawinput	dinput	win32	none	sdl[3].
SDL (macOS and Linux)	auto[4].				none	sdl
Linux	auto[4].				none	sdl

[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.

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

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

Table 3: Supported mouse input providers per-platform

Microsoft Windows	auto[5].	rawinput	dinput	win32	none	sdl[6].
SDL (macOS and Linux)	auto[7].				none	sdl
Linux	auto[7].				none	sdl

Example:

```
mame indy_4610 -mouseprovider win32
```

-lightgunprovider

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

Table 4: Supported light gun input providers per-platform

Microsoft Windows	auto[8].	rawinput	win32	none		
macOS	auto[9].			none		
Linux	auto[10].			none		x11

Example:

```
mame lethalen -lightgunprovider x11
```

-joystickprovider

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

Table 5: Supported joystick input providers per-platform

Microsoft Windows	auto[11].	winhybrid	dinput	xinput	none	sdl
SDL	auto[12].				none	sdl

[2] `auto` on Windows will try `rawinput` with fallback to `dinput`.
[3] SDL support on Windows requires that you compile MAME with the support in. By default SDL is not included in Windows builds of MAME.
[4] `auto` on SDL will default to `sdl`.
[5] On Windows, `auto` will try `rawinput` with fallback to `dinput`.
[6] SDL support on Windows requires that you compile MAME with the support in. By default SDL is not included in Windows builds of MAME.
[7] `auto` on SDL will default to `sdl`.
[8] On Windows, auto will try `rawinput` with fallback to `win32`, or `none` if it doesn't find any.
[9] On non-Linux SDL, `auto` will default to `none`.
[10] On SDL/Linux, `auto` will default to `x11`, or `none` if it doesn't find any.
[11] On Windows, auto will default to `dinput`.
[12] On SDL, auto will default to `sdl`.

Tip: Note that Microsoft XBox 360 and XBox One controllers connected to Windows will work best with `winhybrid` or `xinput`. The `winhybrid` option supports a mix of DirectInput and XInput controllers at the same time.

Example:

```
mame mk2 -joystickprovider winhybrid
```

Tip: On Windows, winhybrid is likely to give the best experience by supporting both XInput and DirectInput controllers.

5.1.8 OSD CLI Options

-listmidi

Create a list of 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

Create a list of 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)
- `source/`<*driver*>`.ini` (based on the source filename of the driver)
- `vertical.ini` (for systems with vertical monitor orientation)
- `horizont.ini` (for systems with horizontal monitor orientation)
- `arcade.ini` (for systems in source added with `GAME()` macro)
- `console.ini` (for systems in source added with `CONS()` macro)
- `computer.ini` (for systems in source added with `COMP()` macro)
- `othersys.ini` (for systems in source added with `SYST()` macro)
- `vector.ini` (for vector systems only)
- <*parent*>`.ini` (for clones only, may be called recursively)
- <*systemname*>`.ini`

(See *Order of Config 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*> <*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 default input configuration files. Multiple paths can be specified by separating them with semicolons.

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 `sofware` (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.

-record_timecode

Tells MAME to create a timecode file. It contains a line with elapsed times on each press of timecode shortcut key (default is **F12**). This option works only when recording mode is enabled (**-record** option). The timecode file is saved in the `inp` folder.

By default, no timecode file is saved.

Example:

```
mame pacman -record worldrecord -record_timecode
```

-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 -wavewrite 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 seperator 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 called `final.png` to the system's snapshot directory.

Example:

```
mame pacman -seconds_to_run 60
```

-[no]throttle

Enable or disable thottling 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
```

-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 present for doing CPU benchmarks 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.

Using `d3d` tells MAME to use Direct3D for rendering. This produces the better quality output than `gdi` and enables additional rendering options. It is recommended if you have a semi-recent (2002+) video card or onboard Intel video of the HD3000 line or better.

On other platforms (including SDL on Windows):

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 will work on any platform.

Defaults:

The default on Windows is `d3d`.

The default for Mac OS X is `opengl` because OS X 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 flexability.

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

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

Specifies override artwork for external artwork and internal driver layout.

Example:

```
mame galaga -override_artwork puckman
```

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 Debugging 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 devs, 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

-gl_glsl

Enable OpenGL GLSL, if available.

The default is OFF.

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 systcm 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
```

-gl_glsl_vid_attr

Enable OpenGL GLSL handling of brightness and contrast. Better RGB system performance.

Default is `on`.

Example:

```
mame pacman -gl_glsl -gl_glsl_vid_attr off
```

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 startup volume. It can later be changed with the user interface (see Keys section). The volume is an attenuation in dB: e.g., "**-volume -12**" will start with -12dB attenuation.

The default is `0`.

Example:

```
mame pacman -volume -30
```

-sound <*dsound* | *coreaudio* | *sdl* | *xaudio2* | *portaudio* | *none*>

Specifies which sound subsystem to use. Selecting `none` disables sound output altogether (sound hardware is still emulated).

On Windows and Linux, *portaudio* is likely to give the lowest possible latency, while Mac users will find *coreaudio* provides the best results.

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 `dsound` on Windows. On Mac, `coreaudio` is the default. On all other platforms, `sdl` is the default.

Example:

```
mame pacman -sound portaudio
```

Table 6: Supported sound subsystems per-platform

Microsoft Windows	dsound	xaudio2	portaudio		sdl[13].	none
macOS			portaudio	coreaudio	sdl	none
Linux and others			portaudio		sdl	none

-audio_latency <*value*>

The exact behavior depends on the selected audio output module. Smaller values provide less audio delay while requiring better system performance. Higher values increase audio delay but may help avoid buffer under-runs and audio interruptions.

The default is `1`.

For PortAudio, see the section on *-pa_latency*.
XAudio2 calculates audio_latency as 10ms steps.
DSound calculates audio_latency as 10ms steps.
CoreAudio calculates audio_latency as 25ms steps.
SDL calculates audio_latency as Xms steps.

Example:

```
mame galaga -audio_latency 1
```

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*>

Enables support for special controllers. Configuration files are loaded from the ctrlrpath. They are in the same format as the .cfg files that are saved, but only control configuration data is read from the file.

The default is `NULL` (no controller 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.

The default is OFF (**-nomouse**).

Example:

[13] While SDL is not a supported option on official builds for Windows, you can compile MAME with SDL support on Windows.

```
mame centiped -mouse
```

-[no]joystick / -[no]joy

Controls whether or not MAME makes use of joystick/gamepad controllers.

When this is enabled, MAME will ask the system about which controllers are connected.

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 map to the mouse, so using **-lightgun** and **-mouse** together may produce strange results.

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
```

-[no]offscreen_reload / -[no]reload

Controls whether or not MAME treats a second button input from a lightgun as a reload signal. In this case, MAME will report the gun's position as (0,MAX) with the trigger held, which is equivalent to an offscreen reload.

This is only needed for games that required you to shoot offscreen to reload, and then only if your gun does not support off screen reloads.

The default is OFF (**-nooffscreen_reload**).

Example:

```
mame lethalen -offscreen_reload
```

-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 `0.3`.

Example:

```
mame sinistar -joystick_deadzone 0.45
```

-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
```

-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 compatability 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 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 or has to support it.
- The selected keyboard *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/Korean)

Example:

```
mame coco2 -natural
```

-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 controls autoenabling the mouse, joystick, or lightgun depending on the presence of a particular class of analog control for a particular system. For example, if you specify the option **-paddle mouse**, then any game that has a paddle control will automatically enable mouse controls just as if you had explicitly specified **-mouse**.

Example:

```
mame sbrkout -paddle_device mouse
```

Tip: Note that these controls override the values of **-[no]mouse**, **-[no]joystick**, etc.

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, the debugger is entered by pressing the tilde (**~**) key during emulation. It is also entered immediately at startup.

The default is OFF (**-nodebug**).

Example:

```
mame indy_4610 -debug
```

-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 -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_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 `15122`.

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 "`15122`".

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 core if available for maximum speed.

The default is ON (**-drc**).

Example:

```
mame ironfort -nodrc
```

-drc_use_c

Force DRC to use the C code backend.

The default is OFF (**-nodrc_use_c**).

Example:

```
mame ironfort -drc_use_c
```

-drc_log_uml

Write DRC UML disassembly log.

The default is OFF (**-nodrc_log_uml**).

Example:

```
mame ironfort -drc_log_uml
```

-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 **-listxml** output will list all of the possible BIOS names for a system.

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 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 name of a font file to use for the UI font. If this font cannot be found or cannot be loaded, the system will fall back to its built-in UI font. On some platforms *fontname* can be a system font name instead of a BDF font file.

The default is `default` (use the OSD-determined default font).

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
```

-ramsize *[n]*

Allows you to change the default RAM size (if supported by driver).

Example:

```
mame coco -ramsize 16K
```

-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
```

-ui_mouse

Displays a mouse cursor when using the built-in UI for MAME.

The default is (**-noui_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
```

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 issue a 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
```

5.1.32 PortAudio Options

-pa_api *API*

Choose which API that PortAudio should use to talk to your sound hardware. You can use **-verbose** to see which APIs are available.

The default is `none`.

Example 1:

```
mame -sound portaudio -verbose
Attempting load of mame.ini
...
PortAudio: API MME has 20 devices
PortAudio: MME: " - Input"
PortAudio: MME: "Microphone (3- USB Camera-B4.09"
PortAudio: MME: "Line (AVerMedia Live Gamer HD 2"
PortAudio: MME: "Digital Audio Interface (AVerMe"
PortAudio: MME: "Headset Microphone (Razer Krake"
...
PortAudio: MME: " - Output"
PortAudio: MME: "Headset Earphone (Razer Kraken "
PortAudio: MME: "Digital Audio (S/PDIF) (High De"
PortAudio: MME: "NX-EDG27 (NVIDIA High Definitio"
...
PortAudio: API Windows DirectSound has 20 devices
PortAudio: Windows DirectSound: "Primary Sound Capture Driver"
PortAudio: Windows DirectSound: "Headset Microphone (Razer Kraken 7.1 V2)
↪"
PortAudio: Windows DirectSound: "Primary Sound Driver" (default)
PortAudio: Windows DirectSound: "Headset Earphone (Razer Kraken 7.1 V2)"
PortAudio: Windows DirectSound: "Digital Audio (S/PDIF) (High Definition␣
↪Audio Device)"
PortAudio: Windows DirectSound: "NX-EDG27 (NVIDIA High Definition Audio)"
...
PortAudio: API Windows WASAPI has 18 devices
PortAudio: Windows WASAPI: "Headset Earphone (Razer Kraken 7.1 V2)"
PortAudio: Windows WASAPI: "Digital Audio (S/PDIF) (High Definition␣
↪Audio Device)"
PortAudio: Windows WASAPI: "NX-EDG27 (NVIDIA High Definition Audio)"
PortAudio: Windows WASAPI: "Headset Microphone (Razer Kraken 7.1 V2)"
...
PortAudio: API Windows WDM-KS has 22 devices
PortAudio: Windows WDM-KS: "Output (NVIDIA High Definition Audio)"
PortAudio: Windows WDM-KS: "SPDIF Out (HD Audio SPDIF out)"
PortAudio: Windows WDM-KS: "Headset Microphone (Razer Kraken 7.1 V2)"
PortAudio: Windows WDM-KS: "Headset Earphone (Razer Kraken 7.1 V2)"
PortAudio: Windows WDM-KS: "Microphone (VDVAD Wave)"
PortAudio: Windows WDM-KS: "Speakers (VDVAD Wave)"
...
PortAudio: Sample rate is 48000 Hz, device output latency is 218.67 ms
PortAudio: Allowed additional buffering latency is 18.00 ms/864 frames
```

Example 2:

```
mame suprmrio -sound portaudio -pa_api "Windows WASAPI"
```

-pa_device *device*

Choose which sound device to output through. This would typically be one of the outputs on your sound card or a USB headset.

The default is `none`.

Example:

```
mame suprmrio -sound portaudio -pa_api "Windows WASAPI" -pa_device "NX-
↪EDG27 (NVIDIA High Definition Audio)"
```

-pa_latency *latency*

Choose the buffer size for PortAudio output; this is specified in seconds. Lower numbers have less latency but may increase stutter in the sound. Decimal places are supported. Try starting from 0.20 and decrease or increase until you find the best number your hardware and OS are capable of handling.

The default is `0`.

Example:

```
mame suprmrio -sound portaudio -pa_api "Windows WASAPI" -pa_device "NX-
↪EDG27 (NVIDIA High Definition Audio)" -pa_latency 0.20
```

5.2 Windows-Specific Commandline 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. 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. 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. 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. 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. 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. 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 connected at the same time. This option requires -lightgun. This option is a hack for supporting certain older dual lightgun setups. If you have multiple lightguns connected, you will probably just need to enable -mouse and configure each lightgun independently. The default is *OFF* (**-nodual_lightgun**).

5.3 SDL-Specific Commandline Options

This section contains configuration options that are specific to any build supported by SDL (including Windows where compiled as SDL instead of native).

5.3.1 Performance Options

-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 Filename. Default is `keymap.dat`.

5.3.5 SDL Joystick Mapping

-joy_idx1 <*name*>
-joy_idx2 <*name*>
...
-joy_idx8 <*name*>

Name of joystick mapped to a given joystick slot, default is *auto*.

-sixaxis

Use special handling for PS3 SixAxis controllers. Default is OFF (**-nosixaxis**)

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 Commandline Index

This is a complete index of all commandline options and commands for MAME, suitable for quickly finding a given command.

5.4.1 Universal Commandline Options

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

Core Commands

help
validate

Configuration Commands

createconfig
showconfig
showusage

Frontend Commands

listxml
listfull
listsource
listclones
listbrothers
listcrc
listroms
listsamples
verifyroms
verifysamples
romident
listdevices
listslots
listmedia
listsoftware
verifysoftware
getsoftlist
verifysoftlist

OSD-related Options

uimodekey
uifontprovider
keyboardprovider
mouseprovider
lightgunprovider
joystickprovider

OSD CLI Options

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
exit_after_playback
record
record_timecode
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
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

Core Screen Options

brightness
contrast
gamma
pause_brightness
effect

Core Vector Options

beam_width_min
beam_width_max
beam_intensity_weight
flicker

Core Video OpenGL Debugging Options

[no]gl_forcepow2texture
[no]gl_notexturerect
[no]gl_vbo
[no]gl_pbo

Core Video OpenGL GLSL Options

gl_glsl
gl_glsl_filter
glsl_shader_mame[0-9]
glsl_shader_screen[0-9]
gl_glsl_vid_attr

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
[no]offscreen_reload
joystick_map
joystick_deadzone
joystick_saturation
natural

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
debugscript
[no]update_in_pause
watchdog
debugger_font
debugger_font_size

Core Communication Options

comm_localhost
comm_localport
comm_remotehost
comm_remoteport
[no]comm_framesync

Core Misc Options

[no]drc
drc_use_c
drc_log_uml
drc_log_native
bios
[no]cheat
[no]skip_gameinfo
uifont
ui

ramsize
confirm_quit
ui_mouse
language
[no]nvram_save

5.4.2 Scripting Options

autoboot_command
autoboot_delay
autoboot_script
[no]console
[no]plugins
plugin
noplugin

5.4.3 HTTP Server Options

http
http_port
http_root

5.4.4 Windows-Specific Commandline Options

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

5.4.5 SDL-Specific Commandline Options

This section contains configuration options that are specific to any build supported by SDL (including Windows where compiled as SDL instead of native).

SDL Performance Options

sdlvideofps

SDL Video Options

[no]centerh
[no]centerv

SDL Video Soft-Specific Options

scalemode

SDL Keyboard Mapping

keymap
keymap_file

SDL Joystick Mapping

joyidx
sixaxis

SDL Low-level Driver Options

videodriver
audiodriver
gl_lib

CHAPTER

SIX

ADVANCED CONFIGURATION

6.1 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 configs.

6.1.1 Order of Config 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. System type INI file (`arcade.ini`, `console.ini`, `computer.ini`, or `othersys.ini`). Both Pac-Man and Street Fighter Alpha are arcade games, so `arcade.ini` will be loaded here, while Atari 2600 will load `console.ini` as it is a home game console.
6. 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.
7. 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.

8. 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.
9. 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.
10. System INI file. Using the previous example, The Last Soldier will load `lastsold.ini`.

6.1.2 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. `arcade.ini` (system type)
  6. `raster.ini` (monitor type)
  7. `source/jack.ini` (driver source file)
  8. (no BIOS set)
  9. `zzyzzyxx.ini` (parent system)
  10. `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. `arcade.ini` (system type)
  6. `raster.ini` (monitor type)
  7. `source/cps2.ini` (driver source file)
  8. (no BIOS set)
  9. (no parent system)
  10. `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. `arcade.ini` (system type)
  6. `raster.ini` (monitor type)
  7. `source/stv.ini` (driver source file)

8. `stvbios.ini` (BIOS set)
9. `smleague.ini` (parent system)
10. `finlarch.ini` (system)

Remember command line parameters take precedence over all else!

6.1.3 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.

[todo: more practical examples]

6.2 MAME Path Handling

MAME has a specific order it uses when checking for user files such as ROM sets and cheat files.

6.2.1 Order of Path Loading

Let's use an example of the cheat file for AfterBurner 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.]

6.3 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.

6.3.1 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.

6.4 BGFX Effects for (nearly) Everyone

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.

6.4.1 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.

6.4.2 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` 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 referenced in *Order of Config 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 config files, you'll want to copy the BGFX settings from `mame.ini` to one of the other config 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.

6.4.3 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.

bgfx_backend

Selects a rendering backend for BGFX to use. Possible choices include `d3d9`, `d3d11`, `d3d12`, `opengl`, `metal`, and `` `vulkan ``. The default is `**auto**`, which will let MAME choose the best selection for you.

`d3d9` -- Direct3D 9.0 Renderer (Requires Windows XP or higher)
`d3d11` -- Direct3D 11.0 Renderer (Requires Windows Vista with D3D11 update or Windows 7 or higher)
`d3d12` -- Direct3D 12.0 Renderer (Requires Windows 10)
`opengl` -- OpenGL Renderer (Requires OpenGL drivers, may work better on some poorly designed video cards, supported on Linux/Mac OS X)
`metal` -- Metal Apple Graphics API (Requires OS X 10.11 El Capitan or newer)
`vulkan` -- Vulkan Renderer

bgfx_debug

Enables BGFX debugging features. Most users will not need to use this.

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 filterered output
`unfiltered` -- nearest neighbor unfiltered output
`hlsl` -- HLSL display simulation through shaders

We make a distinction between emulated device screens (which we'll call a **screen**) and physical displays (which we'll call a **window**, set by `-numscreens`) here. We use colons (:) to seperate windows, and commas (,) to seperate screens. Commas always go on the outside of the chain (see House Mannequin example)

On a combination of a 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 multiple entries (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 mulitple 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 mahjongg 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,unfiltered:hlsl,unfiltered,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**`.

6.4.4 Tweaking BGFX HLSL Settings inside MAME

Warning: Currently BGFX HLSL settings are not saved or loaded from any configuration files. This is expected to change in the future.

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 changable on a per-screen basis.

6.5 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.

6.5.1 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.

6.5.2 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.

6.5.3 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.

6.5.4 Configuration Editing

As referenced in *Order of Config 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.

6.5.5 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 qubic 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

6.5.6 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

6.6 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.

6.6.1 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.

6.6.2 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.

6.6.3 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.

6.6.4 Configuration Editing

As referenced in *Order of Config 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.

6.6.5 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.

6.7 Stable Controller IDs

By default, the mapping between devices and controller IDs is not stable. For instance, a gamepad controller may be assigned to "Joy 1" initially, but after a reboot, it may get re-assigned to "Joy 3".

The reason is that MAME enumerates attached devices and assigns controller IDs based on the enumeration order. Factors that can cause controller IDs to change include plugging / unplugging USB devices, changing ports / hubs and even system reboots.

It is quite cumbersome to ensure that controller IDs are always correct.

That's where the "mapdevice" configuration setting comes into the picture. This setting allows you to map a device id to a controller ID, ensuring that the specified device always maps to the same controller ID in MAME.

6.7.1 Usage of mapdevice

The "mapdevice" xml element is specified under the input xml element in the controller configuration file. It requires two attributes, "device" and "controller". NOTE: This setting only take effect when added to the **ctrlr** config file.

The "device" attribute specifies the id of the device to match. It may also be a substring of the id. To see the list of available devices, enable verbose output and available devices will then be listed to the console at startup (more on this below).

The "controller" attribute specifies the MAME controller ID. It is made up of a controller class (i.e. `JOYCODE`, `GUNCODE`, `MOUSECODE`) and controller index. For example: `JOYCODE_1`.

6.7.2 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 entries map player 1 and 2 lightguns to Gun 1 and Gun 2, respectively. We use a substring of the full raw device names to match each devices. Note that, since this is XML, we needed to escape the `&` using `&`.

The last two mapdevices entries map player 1 and player 2 gamepad controllers to Joy 1 and Joy 2, respectively. In this case, these are XInput devices.

6.7.3 Listing Available Devices

How did we obtain the device id's in the above example? Easy!

Run MAME with -v parameter to enable verbose output. It will then list available devices include the corresponding "device id" to the console.

Here an example:

Input: Adding Gun #0:
Input: Adding Gun #1:
Input: Adding Gun #2: HID-compliant mouse (**device id: \?HID#VID_045E&PID_0053#7&18297dcb&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd}**)
Input: Adding Gun #3: HID-compliant mouse (**device id: \?HID#IrDeviceV2&Col08#2&2818a073&0&0007#{378de44c-56ef-11d1-bc8c-00a0c91405dd}**)
Input: Adding Gun #4: HID-compliant mouse (**device id: \?HID#VID_D209&PID_1602&MI_02#8&389ab7f3&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd}**)
Input: Adding Gun #5: HID-compliant mouse (**device id: \?HID#VID_D209&PID_1601&MI_02#9&375eebb1&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd}**)
Input: Adding Gun #6: 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 Joy #0: ATRAK Device #1 (**device id: ATRAK Device #1**)
Skipping DirectInput for XInput compatible joystick Controller (XBOX 360 For Windows).
Input: Adding Joy #1: ATRAK Device #2 (**device id: ATRAK Device #2**)
Input: Adding Joy #2: XInput Player 1 (**device id: XInput Player 1**)
Input: Adding Joy #3: XInput Player 2 (**device id: XInput Player 2**)

Furthermore, when devices are mapped using mapdevice, you'll see that in the verbose logging too, such as:

Input: Remapped Gun #0: HID-compliant mouse (device id: \?HID#VID_D209&PID_1601&MI_02#9&375eebb1&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd})
Input: Remapped Gun #1: HID-compliant mouse (device id: \?HID#VID_D209&PID_1602&MI_02#8&389ab7f3&0&0000#{378de44c-56ef-11d1-bc8c-00a0c91405dd})
Input: Remapped Joy #0: XInput Player 1 (device id: XInput Player 1)
Input: Remapped Joy #1: XInput Player 2 (device id: XInput Player 2)

6.8 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.

6.8.1 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.

6.8.2 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*"
        Driver "libinput"
        Option "Floating" "yes"
        Option "AutoServerLayout" "no"
EndSection
```

6.8.3 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"

These next lines then tell MAME to keep lightguns consistent across sessions.

- offscreen_reload 1

Lastly, as most lightgun games require offscreen reloading and we're using a device that actually can point away from the screen, we enable that functionality.

CHAPTER

SEVEN

MAME DEBUGGER

This section covers the built-in MAME debugger, which is invoked with the -debug switch on the command line.

7.1 General Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

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 -- outputs a brief history of visited opcodes (**to fix: help missing for this command**)
trackpc -- visually track visited opcodes [boolean to turn on and off, for the given CPU, clear]
trackmem -- record which PC writes to each memory address [boolean to turn on and off, clear]
pcatmem -- query which PC wrote to a given memory address for the current CPU
rewind -- go back in time by loading the most recent rewind state
statesave -- save a state file for the current driver
stateload -- load a state file for the current driver
snap -- save a screen snapshot.
source -- reads commands from <filename> and executes them one by one
quit -- exits MAME and the debugger

7.1.1 do

do <expression>

The do command simply evaluates the given <expression>. This is typically used to set or modify variables.

Examples:

do pc = 0

Sets the register 'pc' to 0.

Back to *General Debugger Commands*

7.1.2 symlist

symlist [<cpu>]

Lists registered symbols. If <cpu> is not specified, then symbols in the global symbol table are displayed; otherwise, the symbols for <cpu>'s specific CPU are displayed. Symbols are listed alphabetically. Read-only symbols are flagged with an asterisk.

Examples:

symlist

Displays the global symbol table.

symlist 2

Displays the symbols specific to CPU #2.

Back to *General Debugger Commands*

7.1.3 softreset

softreset

Executes a soft reset.

Examples:

softreset

Executes a soft reset.

Back to *General Debugger Commands*

7.1.4 hardreset

hardreset

Executes a hard reset.

Examples:

hardreset

Executes a hard reset.

Back to *General Debugger Commands*

7.1.5 print

print <item>[,...]

The print command prints the results of one or more expressions to the debugger console as hexadecimal values.

Examples:

print pc

Prints the value of 'pc' to 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*

7.1.6 printf

printf <format>[,<item>[,...]]

The printf command performs a C-style printf to the debugger console. Only a very limited set of formatting options are available:

%[0][<n>]d -- prints <item> as a decimal value with optional digit count and zero-fill
%[0][<n>]x -- prints <item> as a hexadecimal value with optional digit count and zero-fill

All remaining formatting options are ignored. Use %% together to output a % character. Multiple lines can be printed by embedding a \n in the text.

Examples:

printf "PC=%04X",pc

Prints PC=<pcval> where <pcval> is displayed in hexadecimal with 4 digits with 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*

7.1.7 logerror

logerror <format>[,<item>[,...]]

The logerror command performs a C-style printf to the error log. Only a very limited set of formatting options are available:

%[0][<n>]d -- logs <item> as a decimal value with optional digit count and zero-fill
%[0][<n>]x -- logs <item> as a hexadecimal value with optional digit count and zero-fill

All remaining formatting options are ignored. Use %% together to output a % character. Multiple lines can be printed by embedding a \n in the text.

Examples:

logerror "PC=%04X",pc

Logs PC=<pcval> where <pcval> is displayed in hexadecimal with 4 digits with 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*

7.1.8 tracelog

tracelog <format>[,<item>[,...]]

The tracelog command performs a C-style printf and routes the output to the currently open trace file (see the 'trace' command for details). If no file is currently open, tracelog does nothing. Only a very limited set of formatting options are available. See the *printf* help for details.

Examples:

tracelog "PC=%04X",pc

Outputs PC=<pcval> where <pcval> is displayed in hexadecimal with 4 digits with zero-fill.

printf "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.

Back to *General Debugger Commands*

7.1.9 tracesym

tracesym <item>[,...]

The tracesym command prints the specified symbols and routes the output to the currently open trace file (see the 'trace' command for details). If no file is currently open, tracesym does nothing.

Examples:

tracelog pc

Outputs PC=<pcval> where <pcval> is displayed in the default format.

Back to *General Debugger Commands*

7.1.10 trackpc

trackpc [<bool>,<cpu>,<bool>]

The trackpc command displays which program counters have already been visited in all disassembler windows. The first boolean argument toggles the process on and off. The second argument is a CPU selector; if no CPU is specified, the current CPU is automatically selected. The third argument is a boolean denoting if the existing data should be cleared or not.

Examples:

trackpc 1

Begin tracking the current CPU's pc.

trackpc 1, 0, 1

Continue tracking pc on CPU 0, but clear existing track info.

Back to *General Debugger Commands*

7.1.11 trackmem

trackmem [<bool>,<cpu>,<bool>]

The trackmem command logs the PC at each time a memory address is written to. The first boolean argument toggles the process on and off. The second argument is a CPU selector; if no CPU is specified, the current CPU is automatically selected. The third argument is a boolean denoting if the existing data should be cleared or not. Please refer to the pcatmem command for information on how to retrieve this data. Also, right clicking in a memory window will display the logged PC for the given address.

Examples:

trackmem

Begin tracking the current CPU's pc.

trackmem 1, 0, 1

Continue tracking memory writes on CPU 0, but clear existing track info.

Back to *General Debugger Commands*

7.1.12 pcatmem

pcatmem(p/d/i) <address>[,<cpu>]

pcatmemp <address>[,<cpu>] -- query which PC wrote to a given program memory address for the current CPU
pcatmemd <address>[,<cpu>] -- query which PC wrote to a given data memory address for the current CPU
pcatmemi <address>[,<cpu>] -- query which PC wrote to a given I/O memory address for the current CPU (you can also query this info by right clicking in a memory window)

The pcatmem command returns which PC wrote to a given memory address for the current CPU. The first argument is the requested address. The second argument is a CPU selector; if no CPU is specified, the current CPU is automatically selected. Right clicking in a memory window will also display the logged PC for the given address.

Examples:

pcatmem 400000

Print which PC wrote this CPU's memory location 0x400000.

Back to *General Debugger Commands*

7.1.13 rewind

rewind[rw]

The rewind command loads the most recent RAM-based state. Rewind states, when enabled, are saved when "step", "over", or "out" command gets executed, storing the machine state as of the moment before actually stepping. 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" or "reverse-next". 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.

Back to *General Debugger Commands*

7.1.14 statesave

statesave[ss] <filename>

The statesave command creates a save state at this exact moment in time. The given state file gets written to the standard state directory (sta), and gets .sta added to it - no file extension necessary. All output for this command is currently echoed into the running machine window.

Examples:

statesave foo

Writes file 'foo.sta' in the default state save directory.

Back to *General Debugger Commands*

7.1.15 stateload

stateload[sl] <filename>

The stateload command retrieves a save state from disk. The given state file gets read from the standard state directory (sta), and gets .sta added to it - no file extension necessary. All output for this command is currently echoed into the running machine window. Previous memory and PC tracking statistics are cleared.

Examples:

stateload foo

Reads file 'foo.sta' from the default state save directory.

Back to *General Debugger Commands*

7.1.16 snap

snap [[<filename>], <scrnum>]

The snap command takes a snapshot of the current video display and saves it to the configured snapshot directory. If <filename> is specified explicitly, a single screenshot for <scrnum> is saved under the requested filename. If <filename> is omitted, all screens are saved using the same default rules as the "save snapshot" key in MAME proper.

Examples:

snap

Takes a snapshot of the current video screen and saves to the next non-conflicting filename in the configured snapshot directory.

snap shinobi

Takes a snapshot of the current video screen and saves it as 'shinobi.png' in the configured snapshot directory.

Back to *General Debugger Commands*

7.1.17 source

source <filename>

The source command reads in a set of debugger commands from a file and executes them one by one, similar to a batch file.

Examples:

source break_and_trace.cmd

Reads in debugger commands from break_and_trace.cmd and executes them.

Back to *General Debugger Commands*

7.1.18 quit

quit

The quit command exits MAME immediately.

Back to *General Debugger Commands*

7.2 Memory Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

dasm -- disassemble to the given file
find -- search program memory, data memory, or I/O memory for data
dump -- dump program memory, data memory, or I/O memory as text
save -- save binary program, data, or I/O memory to the given file

load -- load binary program memory, data memory, or I/O memory from the given file
map -- map logical program, data, or I/O address to physical address and bank

7.2.1 dasm

dasm <filename>,<address>,<length>[,<opcodes>[,<cpu>]]

The dasm command disassembles program memory to the file specified in the <filename> parameter. <address> indicates the address of the start of disassembly, and <length> indicates how much memory to disassemble. The range <address> through <address>+<length>-1 inclusive will be output to the file. By default, the raw opcode data is output with each line. The optional <opcodes> parameter can be used to enable (1) or disable (0) this feature. Finally, you can disassemble code from another CPU by specifying the <cpu> parameter.

Examples:

dasm venture.asm,0,10000

Disassembles addresses 0-ffff in the current CPU, including raw opcode data, to the file 'venture.asm'.

dasm harddriv.asm,3000,1000,0,2

Disassembles addresses 3000-3fff from CPU #2, with no raw opcode data, to the file 'harddriv.asm'.

Back to *Memory Debugger Commands*

7.2.2 find

f[ind][{d|i}] <address>,<length>[,<data>[,...]]

The **find/findd/findi** commands search through memory for the specified sequence of data. 'find' will search program space memory, while 'findd' will search data space memory and 'findi' will search I/O space memory. <address> indicates the address to begin searching, and <length> indicates how much memory to search. <data> can either be a quoted string or a numeric value or expression or the wildcard character '?'. Strings by default imply a byte-sized search; non-string data is searched by default in the native word size of the CPU. To override the search size for non-strings, you can prefix the value with b. to force byte- sized search, w. for word-sized search, d. for dword-sized, and q. for qword-sized. Overrides are remembered, so if you want to search for a series of words, you need only to prefix the first value with a w. Note also that you can intermix sizes in order 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 current CPU for the string "HIGH SCORE" followed by a 0 byte.

```
findd 3000,1000,w.abcd,4567
```

Searches the data memory address range 3000-3fff 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*

7.2.3 dump

dump[{d|i}] <filename>,<address>,<length>[,<size>[,<ascii>[,<cpu>]]]

The **dump/dumpd/dumpi** commands dump memory to the text file specified in the <filename> parameter.
'dump' will dump program space memory, while 'dumpd' will dump data space memory and 'dumpi' will dump I/O space memory.
<address> indicates the address of the start of dumping, and <length> indicates how much memory to dump. The range <address> through <address>+<length>-1 inclusive will be output to the file.
By default, the data will be output in byte format, unless the underlying address space is word/dword/qword-only. You can override this by specifying the <size> parameter, which can be used to group the data in 1, 2, 4 or 8-byte chunks.
The optional <ascii> parameter can be used to enable (1) or disable (0) the output of ASCII characters to the right of each line; by default, this is enabled.
Finally, you can dump memory from another CPU by specifying the <cpu> parameter.

Examples:

```
dump venture.dmp,0,10000
```

Dumps addresses 0-ffff in the current CPU in 1-byte chunks, including ASCII data, to the file 'venture.dmp'.

```
dumpd harddriv.dmp,3000,1000,4,0,3
```

Dumps data memory addresses 3000-3fff from CPU #3 in 4-byte chunks, with no ASCII data, to the file 'harddriv.dmp'.

Back to *Memory Debugger Commands*

7.2.4 save

save[{d|i}] <filename>,<address>,<length>[,<cpu>]

The **save/saved/savei** commands save raw memory to the binary file specified in the <filename> parameter.
'save' will save program space memory, while 'saved' will save data space memory and 'savei' will save I/O space memory.
<address> indicates the address of the start of saving, and <length> indicates how much memory to save. The range <address> through <address>+<length>-1 inclusive will be output to the file.
You can also save memory from another CPU by specifying the <cpu> parameter.

Examples:

save venture.bin,0,10000

Saves addresses 0-ffff in the current CPU to the binary file 'venture.bin'.

saved harddriv.bin,3000,1000,3

Saves data memory addresses 3000-3fff from CPU #3 to the binary file 'harddriv.bin'.

Back to *Memory Debugger Commands*

7.2.5 load

load[{d|i}] <filename>,<address>[,<length>,<cpu>]

The **load/loadd/loadi** commands load raw memory from the binary file specified in the <filename> parameter.
'load' will load program space memory, while 'loadd' will load data space memory and 'loadi' will load I/O space memory.
<address> indicates the address of the start of saving, and <length> indicates how much memory to load. The range <address> through <address>+<length>-1 inclusive will be read in from the file.
If you specify <length> = 0 or a length greater than the total length of the file it will load the entire contents of the file and no more.
You can also load memory from another CPU by specifying the <cpu> parameter.

NOTE: This will only actually write memory that is possible to overwrite in the Memory Window

Examples:

load venture.bin,0,10000

Loads addresses 0-ffff in the current CPU from the binary file 'venture.bin'.

loadd harddriv.bin,3000,1000,3

Loads data memory addresses 3000-3fff from CPU #3 from the binary file 'harddriv.bin'.

Back to *Memory Debugger Commands*

7.2.6 map

map[{d|i}] <address>

The map/mapd/mapi commands map a logical address in memory to the correct physical address, as well as specifying the bank.
'map' will map program space memory, while 'mapd' will map data space memory and 'mapi' will map I/O space memory.

Example:

map 152d0

Gives physical address and bank for logical address 152d0 in program memory

Back to *Memory Debugger Commands*

7.3 Execution Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

step -- single steps for <count> instructions (F11)
over -- single steps over <count> instructions (F10)
out -- single steps until the current subroutine/exception handler is exited (Shift-F11)
go -- resumes execution, sets temp breakpoint at <address> (F5)
gint -- resumes execution, setting temp breakpoint if <irqline> is taken (F7)
gtime -- resumes execution until the given delay has elapsed
gvblank -- resumes execution, setting temp breakpoint on the next VBLANK (F8)
next -- executes until the next CPU switch (F6)
focus -- focuses debugger only on <cpu>
ignore -- stops debugging on <cpu>
observe -- resumes debugging on <cpu>
trace -- trace the given CPU to a file (defaults to active CPU)
traceover -- trace the given CPU to a file, but skip subroutines (defaults to active CPU)
traceflush -- flushes all open trace files.

7.3.1 step

s[tep] [<count>=1]

The step command single steps one or more instructions in the currently executing CPU. By default, step executes one instruction each time it is issued. You can also tell step to step multiple instructions by including the optional <count> parameter.

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*

7.3.2 over

o[ver] [<count>=1]

The over command single steps "over" one or more instructions in 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 call completes. By default, over executes one instruction each time it is issued. You can also tell step to step multiple instructions by including the optional <count> parameter.

Note that the step over functionality may not be implemented on 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*

7.3.3 out

out

The out command single steps until it encounters a return from subroutine or return from exception instruction. Note that because it detects return from exception conditions, if you attempt to step out of a subroutine and an interrupt/exception occurs before you hit the end, then you may stop prematurely at the end of the exception handler.

Note that the step out functionality may not be implemented on all CPU types. If it is not implemented, then 'out' will behave exactly like 'step'.

Examples:

out

Steps until the current subroutine or exception handler returns.

Back to *Execution Debugger Commands*

7.3.4 go

g[o] [<address>]

The go command resumes execution of the current code. Control will not be returned to the debugger until a breakpoint or watchpoint is hit, or until you manually break in using the assigned key. The go command takes an optional <address> parameter which is a temporary unconditional breakpoint that is set before executing, and automatically removed when hit.

Examples:

g

Resume execution until the next break/watchpoint or until a manual break.

g 1234

Resume execution, stopping at address 1234 unless something else stops us first.

Back to *Execution Debugger Commands*

7.3.5 gvblank

gv[blank]

The gvblank command resumes execution of the current code. Control will not be returned to the debugger until a breakpoint or watchpoint is hit, or until the next VBLANK occurs in the emulator.

Examples:

gv

Resume execution until the next break/watchpoint or until the next VBLANK.

Back to *Execution Debugger Commands*

7.3.6 gint

gi[nt] [<irqline>]

The gint command resumes execution of the current code. Control will not be returned to the debugger until a breakpoint or watchpoint is hit, or until an IRQ is asserted and acknowledged on the current CPU. You can specify <irqline> if you wish to stop execution only on a particular IRQ line being asserted and acknowledged. If <irqline> is omitted, then any IRQ line will stop execution.

Examples:

gi

Resume execution until the next break/watchpoint or until any IRQ is asserted and acknowledged on the current CPU.

gint 4

Resume execution until the next break/watchpoint or until IRQ line 4 is asserted and acknowledged on the current CPU.

Back to *Execution Debugger Commands*

7.3.7 gtime

gt[ime] <milliseconds>

The gtime command resumes execution of the current code. Control will not be returned to the debugger until a specified delay has elapsed. The delay is in milliseconds.

Example:

gtime #10000

Resume execution for ten seconds

Back to *Execution Debugger Commands*

7.3.8 next

n[ext]

The next command resumes execution and continues executing until the next time a different CPU is scheduled. Note that if you have used 'ignore' to ignore certain CPUs, you will not stop until a non-'ignore'd CPU is scheduled.

Back to *Execution Debugger Commands*

7.3.9 focus

focus <cpu>

Sets the debugger focus exclusively to the given <cpu>. This is equivalent to specifying 'ignore' on all other CPUs.

Example:

focus 1

Focus exclusively CPU #1 while ignoring all other CPUs when using the debugger.

Back to *Execution Debugger Commands*

7.3.10 ignore

ignore [<cpu>[,<cpu>[,...]]]

Ignores the specified <cpu> in the debugger. This means that you won't ever see execution on that CPU, nor will you be able to set breakpoints on that CPU. To undo this change use the 'observe' command. You can specify multiple <cpu>s in a single command. Note also that you are not permitted to ignore all CPUs; at least one must be active at all times.

Examples:

ignore 1

Ignore CPU #1 when using the debugger.

ignore 2,3,4

Ignore CPU #2, #3 and #4 when using the debugger.

ignore

List the CPUs that are currently ignored.

Back to *Execution Debugger Commands*

7.3.11 observe

observe [<cpu>[,<cpu>[,...]]]

Re-enables interaction with the specified <cpu> in the debugger. This command undoes the effects of the 'ignore' command. You can specify multiple <cpu>s in a single command.

Examples:

observe 1

Stop ignoring CPU #1 when using the debugger.

observe 2,3,4

Stop ignoring CPU #2, #3 and #4 when using the debugger.

observe

List the CPUs that are currently observed.

Back to *Execution Debugger Commands*

7.3.12 trace

trace {<filename>|OFF}[,<cpu>[,[noloop|logerror][,<action>]]]

Starts or stops tracing of the execution of the specified <cpu>. If <cpu> is omitted, the currently active CPU is specified.

When enabling tracing, specify the filename in the <filename> parameter. To disable tracing, substitute the keyword 'off' for <filename>.

<detectloops> should be either true or false.

If 'noloop' is omitted, the trace will have loops detected and condensed to a single line. If 'noloop' is specified, the trace will contain every opcode as it is executed.

If 'logerror' is specified, logerror output will augment the trace. If you wish to log additional information on each trace, you can append an <action> parameter which is a command that is executed before each trace is logged. Generally, this is used to include a 'tracelog' command. 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 trace command itself.

Examples:

trace joust.tr

Begin tracing the currently active CPU, logging output to joust.tr.

trace dribling.tr,0

Begin tracing the execution of CPU #0, logging output to dribling.tr.

trace starswep.tr,0,noloop

Begin tracing the execution of CPU #0, logging output to starswep.tr, with loop detection disabled.

trace starswep.tr,0,logerror

Begin tracing the execution of CPU #0, logging output (along with logerror output) to starswep.tr.

trace starswep.tr,0,logerror|noloop

Begin tracing the execution of CPU #0, logging output (along with logerror output) to starswep.tr, with loop detection

disabled.

trace >>pigskin.tr

Begin tracing the currently active CPU, appending log output to pigskin.tr.

trace off,0

Turn off tracing on CPU #0.

trace asteroid.tr,0,,{tracelog "A=%02X ",a}

Begin tracing the execution of CPU #0, logging output to asteroid.tr. Before each line, output A=<aval> to the tracelog.

Back to *Execution Debugger Commands*

7.3.13 traceover

traceover {<filename>|OFF}[,<cpu>[,<detectloops>[,<action>]]]

Starts or stops tracing of the execution of the specified <cpu>.

When tracing reaches a subroutine or call, tracing will skip over the subroutine. The same algorithm is used as is used in the step over command. This means that traceover will not work properly when calls are recursive or the return address is not immediately following the call instruction.

<detectloops> should be either true or false. If <detectloops> is *true or omitted*, the trace will have loops detected and condensed to a single line. If it is false, the trace will contain every opcode as it is executed.
If <cpu> is omitted, the currently active CPU is specified.
When enabling tracing, specify the filename in the <filename> parameter.
To disable tracing, substitute the keyword 'off' for <filename>.
If you wish to log additional information on each trace, you can append an <action> parameter which is a command that is executed before each trace is logged. Generally, this is used to include a 'tracelog' command. 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 trace command itself.

Examples:

traceover joust.tr

Begin tracing the currently active CPU, logging output to joust.tr.

traceover dribling.tr,0

Begin tracing the execution of CPU #0, logging output to dribling.tr.

```
traceover starswep.tr,0,false
```

Begin tracing the execution of CPU #0, logging output to starswep.tr, with loop detection disabled.

```
traceover off,0
```

Turn off tracing on CPU #0.

```
traceover asteroid.tr,0,true,{tracelog "A=%02X ",a}
```

Begin tracing the execution of CPU #0, logging output to asteroid.tr. Before each line, output A=<aval> to the tracelog.

Back to *Execution Debugger Commands*

7.3.14 traceflush

traceflush

Flushes all open trace files.

Back to *Execution Debugger Commands*

7.4 Breakpoint Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

bpset -- sets breakpoint at <address>
bpclear -- clears a given breakpoint or all if no <bpnum> specified
bpdisable -- disables a given breakpoint or all if no <bpnum> specified
bpenable -- enables a given breakpoint or all if no <bpnum> specified
bplist -- lists all the breakpoints

7.4.1 bpset

bp[set] <address>[,<condition>[,<action>]]

Sets a new execution breakpoint at the specified <address>.
The optional <condition> parameter lets you specify an expression that will be evaluated each time the breakpoint is hit. If the result of the expression is true (non-zero), the breakpoint will actually halt execution; otherwise, execution will continue with no notification.
The optional <action> parameter provides a command that is executed whenever the breakpoint 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 bpset command itself. Each breakpoint that is set is assigned an index which can be used in other breakpoint commands to reference this breakpoint.

Examples:

bp 1234

Set a breakpoint that will halt execution whenever the PC is equal to 1234.

bp 23456,a0 == 0 && a1 == 0

Set a breakpoint that will halt execution whenever the PC is equal to 23456 AND the expression (a0 == 0 && a1 == 0) is true.

bp 3456,1,{printf "A0=%08X\n",a0; g}

Set a breakpoint that will halt execution whenever the PC is equal to 3456. When this happens, print A0=<a0val> and continue executing.

bp 45678,a0==100,{a0 = ff; g}

Set a breakpoint 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 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 16 times.

Back to *Breakpoint Debugger Commands*

7.4.2 bpclear

bpclear [<bpnum>]

The bpclear command clears a breakpoint. If <bpnum> is specified, only the requested breakpoint is cleared, otherwise all breakpoints are cleared.

Examples:

bpclear 3

Clear breakpoint index 3.

bpclear

Clear all breakpoints.

Back to *Breakpoint Debugger Commands*

7.4.3 bpdisable

bpdisable [<bpnum>]

The bpdisable command disables a breakpoint. If <bpnum> is specified, only the requested breakpoint is disabled, otherwise all breakpoints are disabled. Note that disabling a breakpoint does not delete it, it just temporarily marks the breakpoint as inactive.

Examples:

bpdisable 3

Disable breakpoint index 3.

bpdisable

Disable all breakpoints.

Back to *Breakpoint Debugger Commands*

7.4.4 bpenable

bpenable [<bpnum>]

The bpenable command enables a breakpoint. If <bpnum> is specified, only the requested breakpoint is enabled, otherwise all breakpoints are enabled.

Examples:

bpenable 3

Enable breakpoint index 3.

bpenable

Enable all breakpoints.

Back to *Breakpoint Debugger Commands*

7.4.5 bplist

bplist

The bplist command lists all the current breakpoints, along with their index and any conditions or actions attached to them.

Back to *Breakpoint Debugger Commands*

7.5 Watchpoint Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

wpset -- sets program, data, or I/O space watchpoint
wpclear -- clears a given watchpoint or all if no <wpnum> specified
wpdisable -- disables a given watchpoint or all if no <wpnum> specified
wpenable -- enables a given watchpoint or all if no <wpnum> specified
wplist -- lists all the watchpoints

7.5.1 wpset

wp[{d|i}][set] <address>,<length>,<type>[,<condition>[,<action>]]

Sets a new watchpoint starting at the specified <address> and extending for <length>. The inclusive range of the watchpoint is <address> through <address> + <length> - 1.

The 'wpset' command sets a watchpoint on program memory; the 'wpdset' command sets a watchpoint on data memory; and the 'wpiset' sets a watchpoint on I/O memory.

The <type> parameter specifies which sort of accesses to trap on. It can be one of three values: 'r' for a read watchpoint 'w' for a write watchpoint, and 'rw' for a read/write watchpoint.

The optional <condition> parameter lets you specify an expression that will be evaluated each time the watchpoint is hit. If the result of the expression is true (non-zero), the watchpoint will actually halt execution; otherwise, execution will continue with no notification.

The optional <action> parameter provides a command that is executed whenever the watchpoint 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 wpset command itself.

Each watchpoint that is set is assigned an index which can be used in other watchpoint commands to reference this watchpoint.

In order to help <condition> expressions, two variables are available. 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.

Examples:

```
wp 1234,6,rw
```

Set a watchpoint that will halt execution whenever a read or write occurs in the address range 1234-1239 inclusive.

```
wp 23456,a,w,wpdata == 1
```

Set a watchpoint that will halt execution whenever a write occurs in the address range 23456-2345f AND the data written is equal to 1.

```
wp 3456,20,r,1,{printf "Read @ %08X\n",wpaddr; g}
```

Set a watchpoint that will halt execution whenever a read occurs in the address range 3456-3475. When this happens, print Read @ <wpaddr> and continue executing.

```
temp0 = 0; wp 45678,1,w,wpdata==f0,{temp0++; g}
```

Set a watchpoint that will halt execution whenever a write occurs to the address 45678 AND the value being written is equal to f0. When that happens, increment the variable temp0 and resume execution.

Back to *Watchpoint Debugger Commands*

7.5.2 wpclear

wpclear [<wpnum>]

The wpclear command clears a watchpoint. If <wpnum> is specified, only the requested watchpoint is cleared, otherwise all watchpoints are cleared.

Examples:

wpclear 3

Clear watchpoint index 3.

wpclear

Clear all watchpoints.

Back to *Watchpoint Debugger Commands*

7.5.3 wpdisable

wpdisable [<wpnum>]

The wpdisable command disables a watchpoint. If <wpnum> is specified, only the requested watchpoint is disabled, otherwise all watchpoints are disabled. Note that disabling a watchpoint does not delete it, it just temporarily marks the watchpoint as inactive.

Examples:

wpdisable 3

Disable watchpoint index 3.

wpdisable

Disable all watchpoints.

Back to *Watchpoint Debugger Commands*

7.5.4 wpenable

wpenable [<wpnum>]

The wpenable command enables a watchpoint. If <wpnum> is specified, only the requested watchpoint is enabled, otherwise all watchpoints are enabled.

Examples:

wpenable 3

Enable watchpoint index 3.

wpenable

Enable all watchpoints.

Back to *Watchpoint Debugger Commands*

7.5.5 wplist

wplist

The wplist command lists all the current watchpoints, along with their index and any conditions or actions attached to them.

Back to *Watchpoint Debugger Commands*

7.6 Registerpoints Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

rpset -- sets a registerpoint to trigger on <condition>
rpclear -- clears a given registerpoint or all if no <rpnum> specified
rpdisable -- disabled a given registerpoint or all if no <rpnum> specified
rpenable -- enables a given registerpoint or all if no <rpnum> specified
rplist -- lists all the registerpoints

7.6.1 rpset

rp[set] {<condition>}[,<action>]]

Sets a new registerpoint which will be triggered when <condition> is met. The condition must be specified between curly braces to prevent the condition from being evaluated as an assignment.
The optional <action> parameter provides a command that is executed whenever the registerpoint is hit. 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 rpset command itself.
Each registerpoint that is set is assigned an index which can be used in other registerpoint commands to reference this registerpoint.

Examples:

rp {PC==0150}

Set a registerpoint that will halt execution whenever the PC register equals 0x150.

temp0=0; rp {PC==0150},{temp0++; g}

Set a registerpoint that will increment the variable temp0 whenever the PC register equals 0x0150.

rp {temp0==5}

Set a registerpoint that will halt execution whenever the temp0 variable equals 5.

Back to *Registerpoints Debugger Commands*

7.6.2 rpclear

rpclear [<rpnum>]

The rpclear command clears a registerpoint. If <rpnum> is specified, only the requested registerpoint is cleared, otherwise all registerpoints are cleared.

Examples:

rpclear 3

Clear registerpoint index 3.

rpclear

Clear all registerpoints.

Back to *Registerpoints Debugger Commands*

7.6.3 rpdisable

rpdisable [<rpnum>]

The rpdisable command disables a registerpoint. If <rpnum> is specified, only the requested registerpoint is disabled, otherwise all registerpoints are disabled. Note that disabling a registerpoint does not delete it, it just temporarily marks the registerpoint as inactive.

Examples:

rpdisable 3

Disable registerpoint index 3.

rpdisable

Disable all registerpoints.

Back to *Registerpoints Debugger Commands*

7.6.4 rpenable

rpenable [<rpnum>]

The rpenable command enables a registerpoint. If <rpnum> is specified, only the requested registerpoint is enabled, otherwise all registerpoints are enabled.

Examples:

rpenable 3

Enable registerpoint index 3.

rpenable

Enable all registerpoints.

Back to *Registerpoints Debugger Commands*

7.6.5 rplist

rplist

The rplist command lists all the current registerpoints, along with their index and any actions attached to them.

Back to *Registerpoints Debugger Commands*

7.7 Code Annotation Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

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 a file
comlist -- print currently available comments from file
commit -- gives a bulk comadd then comsave command

7.7.1 comadd

comadd[//] <address>,<comment>

Adds a string <comment> to the disassembled code at <address>. The shortcut for this command is simply '//'

Examples:

comadd 0, hello world.

Adds the comment 'hello world.' to the code at address 0x0

// 10, undocumented opcode!

Adds the comment 'undocumented opcode!' to the code at address 0x10

7.7.2 comdelete

comdelete

Deletes the comment at the specified memory offset. The comment which is deleted is in the currently active memory bank.

Examples:

comdelete 10

Deletes the comment at code address 0x10 (using the current memory bank settings)

7.7.3 comsave

comsave

Saves the working comments to the driver's XML comment file.

Examples:

comsave

Saves the comments to the driver's comment file

7.7.4 comlist

comlist

Prints the currently available comment file in human readable form in debugger output window.

Examples:

comlist

Shows currently available comments.

7.7.5 commit

commit[/*] <address>,<comment>

Adds a string <comment> to the disassembled code at <address> then saves to file. Basically same as comadd + comsave via a single line.
The shortcut for this command is simply '/*'

Examples:

commit 0, hello world.

Adds the comment 'hello world.' to the code at address 0x0

/* 10, undocumented opcode!

Adds the comment 'undocumented opcode!' to the code at address 0x10

7.8 Cheat Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

cheatinit -- initialize the cheat search to the selected memory area
cheatrange -- add to the cheat search the selected memory area
cheatnext -- continue cheat search comparing with the last value
cheatnextf -- continue cheat search comparing with the first value
cheatlist -- show the list of cheat search matches or save them to <filename>
cheatundo -- undo the last cheat search (state only)

7.8.1 cheatinit

cheatinit [<sign><width><swap>,[<address>,<length>[,<cpu>]]]

The cheatinit command initializes the cheat search to the selected memory area.

If no parameter is specified the cheat search is initialized to all changeable memory of the main CPU.

<sign> can be s(signed) or u(unsigned)
<width> can be b(8 bit), w(16 bit), d(32 bit) or q(64 bit)
<swap> append s for swapped search

Examples:

cheatinit ub,0x1000,0x10

Initialize the cheat search from 0x1000 to 0x1010 of the first CPU.

cheatinit sw,0x2000,0x1000,1

Initialize the cheat search with width of 2 byte in signed mode from 0x2000 to 0x3000 of the second CPU.

cheatinit uds,0x0000,0x1000

Initialize the cheat search with width of 4 byte swapped from 0x0000 to 0x1000.

Back to *Cheat Debugger Commands*

7.8.2 cheatrange

cheatrange <address>,<length>

The cheatrange command adds the selected memory area to the cheat search.

Before using cheatrange it is necessary to initialize the cheat search with cheatinit.

Examples:

cheatrange 0x1000,0x10

Add the bytes from 0x1000 to 0x1010 to the cheat search.

Back to *Cheat Debugger Commands*

7.8.3 cheatnext

cheatnext <condition>[,<comparisonvalue>]

The cheatnext command will make comparisons with the last search matches.

Possible <condition>:

all

No <comparisonvalue> needed.

Use to update the last value without changing the current matches.

equal [eq]

Without <comparisonvalue> search for all bytes that are equal to the last search.
With <comparisonvalue> search for all bytes that are equal to the <comparisonvalue>.

notequal [ne]

Without <comparisonvalue> search for all bytes that are not equal to the last search.
With <comparisonvalue> search for all bytes that are not equal to the <comparisonvalue>.

decrease [de, +]

Without <comparisonvalue> search for all bytes that have decreased since the last search.
With <comparisonvalue> search for all bytes that have decreased by the <comparisonvalue> since the last search.

increase [in, -]

Without <comparisonvalue> search for all bytes that have increased since the last search.
With <comparisonvalue> search for all bytes that have increased by the <comparisonvalue> since the last search.

decreaseorequal [deeq]

No <comparisonvalue> needed.

Search for all bytes that have decreased or have same value since the last search.

increaseorequal [ineq]

No <comparisonvalue> needed.

Search for all bytes that have decreased or have same value since the last search.

smallerof [lt]

Without <comparisonvalue> this condition is invalid
With <comparisonvalue> search for all bytes that are smaller than the <comparisonvalue>.

greaterof [gt]

Without <comparisonvalue> this condition is invalid
With <comparisonvalue> search for all bytes that are larger than the <comparisonvalue>.

changedby [ch, ~]

Without <comparisonvalue> this condition is invalid

With <comparisonvalue> search for all bytes that have changed by the <comparisonvalue> since the last search.

Examples:

cheatnext increase

Search for all bytes that have increased since the last search.

cheatnext decrease, 1

Search for all bytes that have decreased by 1 since the last search.

Back to *Cheat Debugger Commands*

7.8.4 cheatnextf

cheatnextf <condition>[,<comparisonvalue>]

The cheatnextf command will make comparisons with the initial search.

Possible <condition>:

all

No <comparisonvalue> needed.

Use to update the last value without changing the current matches.

equal [eq]

Without <comparisonvalue> search for all bytes that are equal to the initial search.
With <comparisonvalue> search for all bytes that are equal to the <comparisonvalue>.

notequal [ne]

Without <comparisonvalue> search for all bytes that are not equal to the initial search.
With <comparisonvalue> search for all bytes that are not equal to the <comparisonvalue>.

decrease [de, +]

Without <comparisonvalue> search for all bytes that have decreased since the initial search.
With <comparisonvalue> search for all bytes that have decreased by the <comparisonvalue> since the initial search.

increase [in, -]

Without <comparisonvalue> search for all bytes that have increased since the initial search.

With <comparisonvalue> search for all bytes that have increased by the <comparisonvalue> since the initial search.

decreaseorequal [deeq]

No <comparisonvalue> needed.

Search for all bytes that have decreased or have same value since the initial search.

increaseorequal [ineq]

No <comparisonvalue> needed.

Search for all bytes that have decreased or have same value since the initial search.

smallerof [lt]

Without <comparisonvalue> this condition is invalid.
With <comparisonvalue> search for all bytes that are smaller than the <comparisonvalue>.

greaterof [gt]

Without <comparisonvalue> this condition is invalid.
With <comparisonvalue> search for all bytes that are larger than the <comparisonvalue>.

changedby [ch, ~]

Without <comparisonvalue> this condition is invalid
With <comparisonvalue> search for all bytes that have changed by the <comparisonvalue> since the initial search.

Examples:

cheatnextf increase

Search for all bytes that have increased since the initial search.

cheatnextf decrease, 1

Search for all bytes that have decreased by 1 since the initial search.

Back to *Cheat Debugger Commands*

7.8.5 cheatlist

cheatlist [<filename>]

Without <filename> show the list of matches in the debug console.
With <filename> save the list of matches in basic XML format to <filename>.

Examples:

cheatlist

Show the current matches in the debug console.

cheatlist cheat.txt

Save the current matches in XML format to cheat.txt.

Back to *Cheat Debugger Commands*

7.8.6 cheatundo

cheatundo

Undo the results of the last search.

The undo command has no effect on the last value.

Examples:

cheatundo

Undo the last search (state only).

Back to *Cheat Debugger Commands*

7.9 Image Debugger Commands

You can also type **help <command>** for further details on each command in the MAME Debugger interface.

images -- lists all image devices and mounted files
mount -- mounts file to named device
unmount -- unmounts file from named device

7.9.1 images

images

Used to display list of available image devices.

Examples:

images

Show list of devices and mounted files for current driver.

7.9.2 mount

mount <device>,<filename>

Mount <filename> to image <device>.

<filename> can be softlist item or full path to file.

Examples:

mount cart,aladdin

Mounts softlist item aladdin on cart device.

7.9.3 unmount

unmount <device>

Unmounts file from image <device>.

Examples:

unmount cart

Unmounts any file mounted on device named cart.

7.10 Debugger Expressions Guide

Expressions can be used anywhere a numeric parameter is expected. The syntax for expressions is very close to standard C-style syntax with full operator ordering and parentheses. There are a few operators missing (notably the trinary ? : operator), and a few new ones (memory accessors). The table below lists all the operators in their order, highest precedence operators first.

( ) : standard parentheses
++ -- : postfix increment/decrement
++ -- ~ ! - + b@ w@ d@ q@ : prefix inc/dec, binary NOT, logical NOT, unary +/-, memory access
* / % : multiply, divide, modulus
+ - : add, subtract
<< >> : shift left/right
< <= > >= : less than, less than or equal, greater than, greater than or equal
== != : equal, not equal
& : binary AND
^ : binary XOR
| : binary OR
&& : logical AND
|| : logical OR
= *= /= %= += -= <<= >>= &= |= ^= : assignment
, : separate terms, function parameters

7.10.1 Numbers

Numbers are prefixed according to their bases:

- Hexadecimal (base-16) numbers are prefixed with `$` or `0x`.
- Decimal (base-10) numbers are prefixed with `#`.
- Octal (base-8) numbers are prefixed with `0o`.
- Binary (base-2) numbers are prefixed 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 9 decimal.
- `0b123` is invalid.

7.10.2 Differences from C Behaviors

- First, all math is performed on full 64-bit unsigned values, so things like **a < 0** won't work as expected.
- Second, the logical operators **&&** and || do not have short-circuit properties -- both halves are always evaluated.
- Finally, the new memory operators work like this:
  - **b!<addr>** refers to the byte at <addr> but does *NOT* suppress side effects such as reading a mailbox clearing the pending flag, or reading a FIFO removing an item.
  - **b@<addr>** refers to the byte at <addr> while suppressing side effects.
  - Similarly, **w@** and **w!** refer to a *word* in memory, **d@** and **d!** refer to a *dword* in memory, and **q@** and **q!** refer to a *qword* in memory.

    The memory operators can be used as both lvalues and rvalues, so you can write **b@100 = ff** to store a byte in memory. By default these operators read from the program memory space, but you can override that by prefixing them with a 'd' or an 'i'.

    As such, **dw@300** refers to data memory word at address 300 and **id@400** refers to an I/O memory dword at address 400.

CHAPTER

EIGHT

MAME EXTERNAL TOOLS

This section covers various extra tools that come with your MAME distribution (e.g. *imgtool*)

8.1 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.

8.2 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.

8.3 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

8.4 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

8.5 Imgtool Format Info

8.5.1 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

8.5.2 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

8.5.3 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

8.5.4 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

8.5.5 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

8.5.6 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

8.5.7 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

8.5.8 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

8.5.9 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

8.5.10 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

8.5.11 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

8.5.12 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

8.5.13 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

8.5.14 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

8.5.15 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

8.5.16 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

8.5.17 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

8.5.18 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

8.5.19 Concept floppy disk image - (*concept*)

Driver specific options for module 'concept':

No image specific file options

No image specific creation options

8.5.20 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

8.5.21 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

8.5.22 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

8.5.23 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

8.5.24 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

8.5.25 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

8.5.26 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

8.5.27 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

8.5.28 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

8.5.29 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

8.5.30 Cybiko Xtreme File System - (*cybikoxt*)

Driver specific options for module 'cybikoxt':

No image specific file options

No image specific creation options

8.5.31 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

8.5.32 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

8.5.33 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

8.5.34 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

8.5.35 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

8.5.36 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

8.5.37 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

8.5.38 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

8.5.39 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

8.5.40 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

8.5.41 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

8.5.42 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

8.5.43 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

8.5.44 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

8.5.45 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

8.5.46 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

8.5.47 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

8.5.48 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

8.5.49 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

8.5.50 Formatted Disk Image (FAT format) - (*fdi_fat*)

Driver specific options for module 'fdi_fat':

No image specific file options

No image specific creation options

8.5.51 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

8.5.52 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

8.5.53 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

8.5.54 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

8.5.55 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

8.5.56 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

8.5.57 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

8.5.58 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

8.5.59 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

8.5.60 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

8.5.61 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

8.5.62 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

8.5.63 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

8.5.64 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

8.5.65 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

8.5.66 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

8.5.67 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

8.5.68 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

8.5.69 TI99 Diskette (PC99 FM format) - (*pc99fm*)

Driver specific options for module 'pc99fm':

No image specific file options

No image specific creation options

8.5.70 TI99 Diskette (PC99 MFM format) - (*pc99mfm*)

Driver specific options for module 'pc99mfm':

No image specific file options

No image specific creation options

8.5.71 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

8.5.72 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

8.5.73 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

8.5.74 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

8.5.75 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

8.5.76 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

8.5.77 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

8.5.78 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

8.5.79 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

8.5.80 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

8.5.81 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

8.5.82 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

8.5.83 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

8.5.84 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

8.5.85 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

8.5.86 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]

8.5.87 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

8.5.88 TI99 Harddisk - (*ti99hd*)

Driver specific options for module 'ti99hd':

No image specific file options

No image specific creation options

8.5.89 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

8.5.90 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]

8.6 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.

8.7 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

8.8 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

8.9 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.

8.10 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

8.11 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_RWTS18

Apple II RWTS18-type image

File extensions: rti

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

8.12 Other tools included with MAME

8.12.1 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.

8.13 Developer-focused tools included with MAME

8.13.1 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.

8.13.2 nltool

Discrete component conversion tool. Most users will not need to work with this.

8.13.3 nlwav

Discrete component conversion and testing tool. Most users will not need to work with this.

8.13.4 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. Most users will not need to work with this.

8.13.5 ldresample

This tool recompresses video data for laserdisc and VHS dumps. Most users will not need to work with this.

8.13.6 ldverify

This tool is used for comparing laserdisc or VHS CHD images with the source AVI. Most users will not need to work with this.

8.13.7 unidasm

Universal disassembler for many of the architectures supported in MAME. Most users will not need to work with this.

CHAPTER

NINE

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.

9.1 MAME Naming Conventions

9.1.1 Introduction

To promote consistency and readability in MAME source code, we have some naming conventions for various elements.

9.1.2 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`

9.1.3 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").

9.1.4 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 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.

9.1.5 C++ 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, withe 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).

9.2 MAME Layout Files

9.2.1 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.

9.2.2 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.

9.2.3 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. The exception is that, for historical reasons, views are processed last. This means views see the final values of all parameters at the end of the `mamelayout` element, and may refer to elements and groups that appear after them.

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 one or more *components*, but an element is treated as as 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 and reels) 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 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 using the UI font in the specified colour. 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, and 2 (two) means right-aligned. If the `align` attribute is absent, the text will be centred.

dotmatrix Draws an eight-pixel horizontal segment of a dot matrix display, using circular pixels in the specified colour. The bits of the element's state determine which pixels are lit, with the least significant bit corresponding to the leftmost pixel. Unlit pixels are drawn at low intensity (0x20/0xff).

dotmatrix5dot Draws a five-pixel horizontal segment of a dot matrix display, using circular pixels in the specified colour. The bits of the element's state determine which pixels are lit, with the least significant bit corresponding to the leftmost pixel. Unlit pixels are drawn at low intensity (0x20/0xff).

dotmatrixdot Draws a single element of a dot matrix display as a circular pixels in the specified colour. The least significant bit of the element's state determines whether the pixel is lit. An unlit pixel is drawn at low intensity (0x20/0xff).

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).

led8seg_gts1 Draws an eight-segment digital fluorescent display of the type used in Gottlieb System 1 pinball machines (actually a Futaba part). Compared to standard seven-segment displays, these displays have no decimal point, the horizontal middle bar is broken in the centre, and there is a broken vertical middle bar controlled by the bit that would control the decimal point in a standard seven-segment display. 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 using the system font in the specified colour. 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 2 (two) means right-aligned; if absent, the text will be centred.

reel Used for drawing slot machine reels. Supported attributes include `symbollist`, `stateoffset`, `numsymbolsvisible`, `reelreversed`, and `beltreel`.

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" />
```

(continues on next page)

(continued from previous page)

```
        <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" />
    </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. For historical reasons, `view` elements are processed *after* all other child elements of the top-level `mamelayout` element. This means a view can reference elements and groups that appear after it in the file, and parameters from the enclosing scope will have their final values from the end of the `mamelayout` element.

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.

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" />
```

(continues on next page)

(continued from previous page)

```
    <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>
```

(continues on next page)

(continued from previous page)

```
    </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 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" />
            <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.

9.2.4 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, simple counter and reel 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 mouse 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:

```
<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 mouse input, MAME treats all layout elements as being rectangular, and only activates the first clickable item whose area includes the location of the mouse pointer.

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. Withing 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 value (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>
```

(continues on next page)

(continued from previous page)

```
    <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" />
    </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>
```

9.2.5 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.

9.2.6 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 screens attached to the system”.
- 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.

9.2.7 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

9.2.8 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.

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.

unkeinv.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.

9.3 MAME Layout Scripting

9.3.1 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
```

If 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.

9.3.2 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 an introduction to Lua scripting in MAME, see *Scripting MAME via Lua*; for detailed descriptions of MAME's Lua classes, see *MAME Lua Class Reference*.

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" />
        <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 -->
```

(continues on next page)

(continued from previous page)

```
    <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")
                    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(
```

(continues on next page)

(continued from previous page)

```
                                        ((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.
- 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 outlined of a square -->
        <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 -->
```

(continues on next page)

(continued from previous page)

```
    <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 horizotnally, 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 horiztonal␣
↪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" />
        </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"]
```

(continues on next page)

(continued from previous page)

```
                    -- 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
                                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
```

(continues on next page)

(continued from previous page)

```
                                        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(
                            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.
- 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.)

9.3.3 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.render_bounds` and `emu.render_color` functions for creating bounds and colour objects.
- `emu.print_error`, `emu.print_info` and `emu.print_debug` functions for diagnostic output.
- Standard Lua `pairs`, `ipairs`, `table.insert` and `table.remove` functions for manipulating tables and other containers.
- Standard Lua `print` function for text output to the console.
- Standard Lua `string.format` function for string formatting.

9.3.4 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.

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).

9.4 Object Finders

9.4.1 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.

9.4.2 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.

9.4.3 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"),
    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.

9.4.4 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);
    }

    sbus_device(machine_config const &mconfig, char const *tag, device_t *owner, u32␣
↪clock) :
        device_t(mconfig, type, 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);
```

9.4.5 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) :
        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),
        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 of which is 8 KiB organised as 4,096 big-Endian 16-bit words.

9.4.6 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_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,
        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) :
```

(continues on next page)

(continued from previous page)

```
    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),
    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();
}
```

9.4.7 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.

9.4.8 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 must be manually resolved, they are not automatically resolved.
- Output finders cannot have their target changed after construction.
- Output finders are array-like, and support an arbitrary number of dimensions.
- Output names are global, the base device has no influence. (This will change in the future.)

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.

You must call `resolve` on each output finder before it can be used. This should be done at start time for the output values to be included in save states:

```
void mmd2_state::machine_start()
{
    m_digits.resolve();
    m_p.resolve();
    m_led_halt.resolve();
    m_led_hold.resolve();

    save_item(NAME(m_digit));
}
```

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;
}
```

9.5 The device_memory_interface

9.5.1 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.

9.5.2 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.

9.5.3 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;

    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());
}
```

9.5.4 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.

9.5.5 5. MMU support for disassembler

```
bool translate(int spacenum, int intention, offs_t &address);
```

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 (`TRANSLATE_(READ\|WRITE\|FETCH)(\|_USER\|_DEBUG)`) and address is an in/out parameter holding the address to translate on entry and the translated version on return. Should return `true` if the translation went correctly, or `false` if the address is unmapped.

Note that for some historical reason, the device itself must override the virtual method `memory_translate` with the same signature.

9.6 The device_rom_interface

9.6.1 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.

9.6.2 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.

9.6.3 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.

9.6.4 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.

9.6.5 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.

9.7 The device_disasm_interface and the disassemblers

9.7.1 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.

9.7.2 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.

9.7.3 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.

9.7.4 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.

9.7.5 5. Missing stuff

There currently is no way for the debugger GUI to add per-core configuration. It is needed for in particular the s2650 and the 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).

9.8 Emulated system memory and address spaces management

9.8.1 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.

9.8.2 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.

9.8.3 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;
    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.

```
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. Alternatively `set_base` sets the base for entry 0 and selects it.

`set_entry` allows to dynamically and efficiently select the current active entry, `entry()` gets that selection back, 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 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.

```
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_bank {
    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.

9.8.4 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();
```

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 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 a 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).

4.5 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".

9.8.5 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 that 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)
space.install_read_handler(addrstart, addrend, addrmask, addrmirror, addrselect, read_
↪delegate, unitmask, cswidth)
space.install_write_handler(addrstart, addrend, write_delegate, unitmask, cswidth)
space.install_write_handler(addrstart, addrend, addrmask, addrmirror, addrselect,␣
↪write_delegate, unitmask, cswidth)
space.install_readwrite_handler(addrstart, addrend, read_delegate, write_delegate,␣
↪unitmask, cswidth)
space.install_readwrite_handler(addrstart, addrend, addrmask, addrmirror, addrselect,␣
↪read_delegate, write_delegate, unitmask, cswidth)
```

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` and `cswidth` 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_writeonly(addrstart, addrend, void *pointer)
space.install_writeonly(addrstart, addrend, addrmirror, void *pointer)
space.install_ram(addrstart, addrend, void *pointer)
space.install_ram(addrstart, addrend, addrmirror, void *pointer)
```

Installs a memory block in an address space, with or without mirror. `_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_write_bank(addrstart, addrend, memory_bank *bank)
space.install_write_bank(addrstart, addrend, addrmirror, memory_bank *bank)
space.install_readwrite_bank(addrstart, addrend, memory_bank *bank)
space.install_readwrite_bank(addrstart, addrend, addrmirror, 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_write_port(addrstart, addrend, const char *wtag)
space.install_write_port(addrstart, addrend, addrmirror, 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)
```

Install ports by name for reading, writing or both.

5.6 Dropped accesses

```
space.nop_read(addrstart, addrend, addrmirror)
space.nop_write(addrstart, addrend, addrmirror)
space.nop_readwrite(addrstart, addrend, addrmirror)
```

Drops the accesses for a given range with an optional mirror.

5.7 Unmapped accesses

```
space.unmap_read(addrstart, addrend, addrmirror)
space.unmap_write(addrstart, addrend, addrmirror)
space.unmap_readwrite(addrstart, addrend, addrmirror)
```

Unmaps the accesses (e.g. logs the access as unmapped) for a given range with an optional mirror.

5.8 Device map installation

```
space.install_device(addrstart, addrend, device, map, unitmask, cswidth)
```

Install a device address with an address map in a space. The `unitmask` and `cswidth` arguments are optional.

5.9 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.

9.9 The new floppy subsystem

9.9.1 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

9.9.2 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 tengentially to the track, e.g in the direction 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" 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.

[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.

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 floppys, 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 endocing 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 (+/- 2% is not rare) and local perturbations (air turbulence, surface distance...) make the instant 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 build 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 to correct the phase on every '1' cell, making the synchronization work if the rotational speed is not too off. Subsequent generations of controllers used a Phase-Locked Loop (PLL) which vary both phase and window duration to adapt better to wrong rotational speeds, with usually a tolerance of +/- 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.

9.9.3 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.

9.10 The new SCSI subsystem

9.10.1 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.

9.10.2 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.

9.10.3 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.

9.10.4 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.

9.10.5 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 overriden 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).

9.10.6 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.

9.10.7 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.

9.11 Scripting MAME via Lua

9.11.1 Introduction

It is now possible to externally drive MAME via Lua scripts. This feature initially appeared in version 0.148, when a minimal Lua engine was implemented. Today, the Lua interface is rich enough to let you inspect and manipulate devices' state, access CPU registers, read and write memory, and draw a custom HUD on screen.

Internally, MAME makes extensive use of Sol3 to implement this feature. The idea is to transparently expose as many of the useful internals as possible.

Finally, a warning: 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 of the objects support runtime you can introspection.

9.11.2 Features

The API is not yet complete, but this is a partial list of capabilities currently available to Lua scripts:

- session information (app version, current emulated system, ROM details)
- 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)

Many of the classes are documented on the *Lua class reference* page.

9.11.3 Usage

MAME supports external scripting via Lua (>= 5.3) scripts, either entered at the interactive console or loaded as a file. To reach the console, enable the console plugin (e.g. run MAME with `-plugin console`) and you will be greeted with a `[MAME]>` prompt where you can enter Lua script interactively.

To load a whole script at once, store it in a plain text file and pass it using `-autoboot_script`. Please note that script loading may be delayed (a few seconds by default), but you can override the default with the `-autoboot_delay` option.

To control the execution of your code, you can use a loop-based or event-based approach. The former is not encouraged as it is resource-intensive and makes control flow unnecessarily complex. Instead, we suggest to register custom hooks to be invoked on specific events (e.g. at each frame rendering).

9.11.4 Walkthrough

Let's first run MAME in a terminal to reach the Lua console:

```
$ mame -console YOUR_ROM
        /|  /|     /|      /|  /|     ________
       / | / |    / |     / | / |    /       /
      /  |/  |   /  |    /  |/  |   /  _____/
     /        | /   |   /        | /  /_
    /         |/    | /         |/  __/
   /  /|  /|     /| |/  /|  /|     /_____
  /  / | / |    / |     / | / |          /
 / _/  |/  /   /  |___/  |/  /________/
          /   /
         / _/

mame 0.227
Copyright (C) Nicola Salmoria and the MAME team

Lua 5.3
Copyright (C) Lua.org, PUC-Rio

[MAME]>
```

At this point, your game is probably running in demo 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 back error messages.

You can check at runtime which version of MAME you are running, with:

```
[MAME]> print(emu.app_name() .. " " .. emu.app_version())
mame 0.227
```

We now start exploring screen related methods. First, let's enumerate available screens:

```
[MAME]> for tag, screen in pairs(manager.machine.screens) do print(tag) end
:screen
```

`manager.machine` is the *running machine* object for your 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 is tagged as `:screen`, and we can further inspect it:

```
[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
```

We have several methods to draw a HUD on the screen composed of lines, boxes and text:

```
[MAME]> -- we define a HUD-drawing function, and then call it
[MAME]> function draw_hud()
[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)
```

(continues on next page)

(continued from previous page)

```
[MAME]>> s:draw_line(20, 20, 80, 80, 0xff00ffff) -- (x0, y0, x1, y1, line-color)
[MAME]>> end
[MAME]> draw_hud()
```

This will draw some useless art on the screen. However, when resuming the game, your HUD needs to be refreshed otherwise it will just disappear. In order to do this, you have to register your hook to be called on every frame repaint:

```
[MAME]> emu.register_frame_done(draw_hud, "frame")
```

All colors are specified in ARGB format (eight bits per channel), while screen origin (0,0) normally corresponds to the top-left corner.

Similarly to screens, you can inspect all the devices attached to a machine:

```
[MAME]> for tag, device in pairs(manager.machine.devices) do print(tag) end
:audiocpu
:maincpu
:saveram
:screen
:palette
[...]
```

On some of them, you can also inspect and manipulate memory and state:

```
[MAME]> cpu = manager.machine.devices[":maincpu"]
[MAME]> -- enumerate, read and write state registers
[MAME]> for k, v in pairs(cpu.state) do print(k) end
D5
SP
A4
A3
D0
PC
[...]
[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
[MAME]> mem = cpu.spaces["program"]
[MAME]> print(mem:read_i8(0xc000))
41
```

9.12 MAME Lua Class Reference

9.12.1 Introduction

Various aspects of MAME can be controlled using Lua scripting. Many key classes are exposed as Lua objects.

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 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 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.

9.12.2 Core classes

Many of MAME's core classes used to implement an emulation session are available to Lua scripts.

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.

Properties

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 emulation 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.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.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.attenuation (read/write) The output volume attenuation in decibels. Should generally be a negative integer 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.

Methods

output:set_value(name, val) Sets the specified output value. The value must be an integer. The output will be created if it does not already exist.

output:set_indexed_value(prefix, index, val) Appends the index (formatted as a decimal integer) to the prefix and sets the value of the corresponding output. The value must be an integer. The output will be created if it does not already exist.

output:get_value(name) Returns the value of the specified output, or zero if it doesn't exist.

output:get_indexed_value(prefix, index) Appends the index (formatted as a decimal integer) to the prefix and returns the value of the corresponding output, or zero if it doesn't exist.

output:name_to_id(name) Gets the per-session unique integer ID for the specified output, or zero if it doesn't exist.

output:id_to_name(id) Gets the name for the output with the specified per-session unique ID, or `nil` if it doesn't exist. This method has O(n) complexity, so avoid calling it when performance is important.

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.

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.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.type (read-only) A string providing a system type. Will be one of `"arcade"`, `"console"`, `"computer"` or `"other"`. This is for informational purposes only, and may not be supported in the future.

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.clickable_artwork (read-only) A Boolean indicating whether the system requires clickable artwork features 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.

9.12.3 Devices

Several device classes and device mix-ins classes are exposed to Lua. Devices can be looked up by tag or enumerated.

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.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.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.

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.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.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: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.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.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.

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 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 screen pixels, with the origin at (0, 0). Note that 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(x1, y1, x2, y2, bottom, [color]) Draws a line from (x1, y1) to (x2, y2).

Coordinates are floating-point numbers in units of screen pixels, with the origin at (0, 0). Note that 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 screen pixels, with the origin at (0, 0). Note that 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 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 foreground colour is not provided, the UI text colour is used; if the background colour is not provided, the UI background colour is used.

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_attoseconds (read-only) The screen's configured refresh interval in attoseconds (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.

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.

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 `"pass"` or `"fail"`.

image:load_software(name) Loads a media image described in a software list. Returns `"pass"` or `"fail"`.

image:unload() Unloads the mounted image.

image:create(filename) Creates and mounts a media image file with the specified name. Returns `"pass"` or `"fail"`.

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.

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*.

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*.

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.

9.12.4 Memory system

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

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 represent's 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.

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`.

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_i{8,16,32,64}(offs) Reads a signed integer value of the size in bits from the specified offset in the memory region.

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.

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.

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.

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.

9.12.5 Input system

Allows scripts to get input from the user, and access I/O ports in the emulated system.

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 port 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_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 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. 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 configuration.

ioport:token_to_input_type(string) Returns the input type and player number for the specified input type token.

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.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*.

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: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 settings. 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 is used to configure general input settings. 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. This is gets general input settings. 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.sensitivity (read-only) The sensitivity or gain for analog 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.optional (read-only) A Boolean indicating whether the field is optional and not required to use the emulated system.

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.rotated (read-only) A Boolean indicating whether the field corresponds to a control that is rotated relative its standard orientation.

field.analog_reverse (read-only) A Boolean indicating whether the field corresponds to an analog control that 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).

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 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 Boolen 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 inptu 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.

input:axis_sequence_poller() Returns an *input sequence poller* for obtaining an input sequence for configuring a digital input.

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 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.

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:find_mouse() Returns host system mouse pointer X position, Y position, button state, and the *render target* it falls in. The position is in host pixels, where zero is at the top/left. The button state is a Boolean indicating whether the primary mouse button is pressed.

If the mouse pointer is not over one of MAME's windows, this may return the position and render target from when the mouse pointer was most recently over one of MAME's windows. The render target may be `nil` if the mouse pointer is not over one of MAME's windows.

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.

9.12.6 Render system

The render system is responsible for drawing what you see in MAME's windows, including emulated screens, artwork, and UI elements.

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 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).

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.

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.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.screens[tag].container Gets the render container used to draw a given screen.

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 container user settings for a given 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.

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.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 not all views in the XML file may 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 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.

Instantiation

Layout scripts generally

manager.machine.render.targets[index].current_view Gets the currently selected view for a given render target.

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.

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 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.

Layout view item

Wraps MAME's `layout_view::item` class, representing an item in a view. An item is drawn as a rectangular textured surface. The texture is supplied by an emulated screen or a layout element.

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.

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.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.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.

9.12.7 Debugger

Some of MAME's core debugging features can be controlled from Lua script. The debugger must be enabled to use the debugging features (usually by passing `-debug` on the command line).

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-only) 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-only) 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.

9.13 The new 6502 family implementation

9.13.1 Introduction

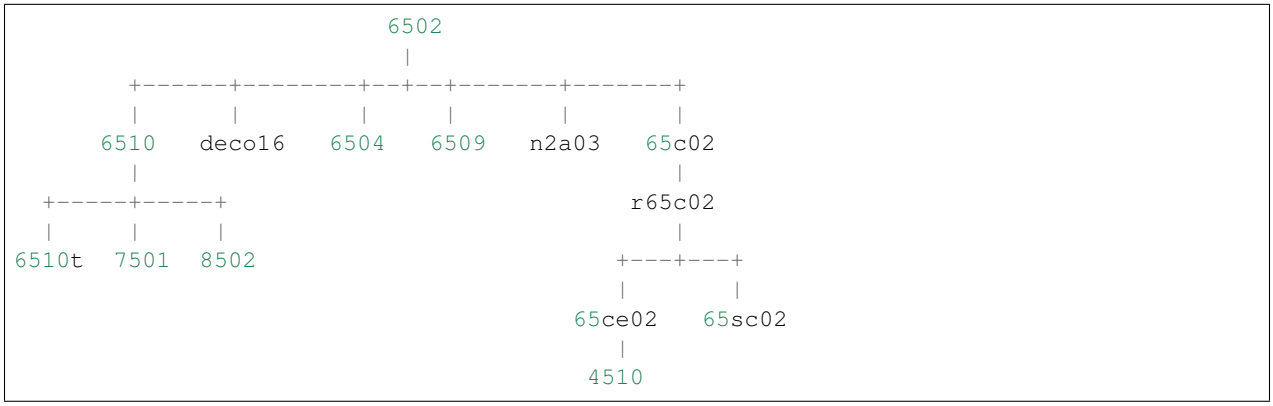
The new 6502 family implementation has been created to reach sub-instruction accuracy in observable behaviour. It is designed with 3 goals in mind:

- 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.

9.13.2 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 offshots (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 n2a03 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.

9.13.3 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, 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.

9.13.4 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 f.i. m65c02. The .inc file will provide the definitions.

9.13.5 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.

9.13.6 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.

9.13.7 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()**.

9.13.8 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.

9.13.9 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.

9.13.10 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-bits 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).

9.13.11 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 subsytem in the n2a03
- Add decent hookups for the Apple 3 madness

CHAPTER

TEN

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 administator 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

ELEVEN

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 BSD-3-Clause 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) 1997-2021 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 LICENSE.md for further details.

CHAPTER

TWELVE

CONTRIBUTE

The documentation on this site is the handiwork of our many contributors.