Building Mini vMac

Skip down to Platform Index

How to build the Mini vMac program from the source code.

First download the source code archive from the download page, a file with the name “minivmac-3.x.x.src.zip”. Extract from this zip file a disk image (named “minivmac-3.x.x.src.dsk”).

Now launch Mini vMac (version 3.0.0 or later), booting from a disk image containing a system folder. (The source code disk image doesn't contain a system folder.) (See the Start page for information about getting started with Mini vMac.)

Mount the source code disk image in Mini vMac. At the top level of this disk is an application named "MnvM_b34". Launch this application. A text editing window will open in which to type in the desired options.

note: The "MnvM_b34" application can also be run on a real 680x0 Macintosh, or in a different Macintosh 680x0 emulator besides Mini vMac. (To access the build system files, you need to mount its disk image. See the Disk Image page.)

Only the “Target” option is required. Type in a line such as:

```
-t imch
```

If this option is used by itself, the build system will generate the files needed to compile the standard version of Mini vMac for Macintosh OS X on Intel using Apple's Xcode 2.4.1 development environment. Options for some other targets and development environments are listed below.

You may also type in other options listed on the Mini vMac Options page. Each option can be on a separate line, or can instead be separated by a spaces. (The new line, space, and tab characters are equivalent.)

The Develop page has more options useful to developers and maintainers.

The build system ignores text contained in brackets (between and including the symbols ‘{’ and ‘}’), so you don't need to type these comments.

When you are done specifying options, choose the ‘Go’ command from the File menu (of the emulated Macintosh). Clicking on the

progress indicator area at the bottom of the window is equivalent to choosing the ‘Go’ command. The build system will generate an archive containing all the files needed to compile Mini vMac with the chosen options.
The build system can handle multiple sets of options at once, separated by ";".
Another feature for automation is importing files, such as by using the “Import” command in the File menu. This is equivalent to copying the contents of the file, and pasting it into the build system window after removing any existing text (such as by the ‘Select All’ and ‘Clear’ commands), and then choosing the ‘Go’ command. Another way of importing files is dropping their icons onto the build system window or application icon. Multiple files can be dropped, and they will be all be processed. (Though if there is an error, that error is reported, and all remaining files are forgotten.) If the build system application was not yet running when icons are dropped on it, then the application automatically quits after processing all the files. Other ways of generating kAEOpenDocuments apple events, besides dropping files on the application icon, should also work, such as AppleScript. You can also double click on files in the Finder that have the build system as their creator. The “Export” command in the File menu creates such files.
When the build system is run Mini vMac 3.0.0 or later, it will automatically export this archive to the real computer. If the build system is run on a different emulator, or a real 680x0 Macintosh, this archive will be left in the folder “output”, in the folder containing the “MnvM_b34” application. If the folder "System Folder:Preferences:Gryphel:Build:output" exists, output is directed there instead. (This is useful if you keep the Mini vMac source disk image locked, to help ensure consistent compile results.)
Extract this archive on your real computer, and the resulting folder has all the files needed to compile Mini vMac with your development environment. More specific instructions for various targets and development environments follow:

... Platform Index ...

Macintosh OS X - PowerPC
Xcode
Metrowerks CodeWarrior
Microsoft Windows
Microsoft Visual C++
lcc-win32 system
Bloodshed Dev-C++
Metrowerks CodeWarrior
Macintosh OS 9
MPW
Metrowerks CodeWarrior
Linux
Linux command line
Solaris
Gnu C compiler

Macintosh OS X - Intel

Building with Xcode 2.4.1:
(For other versions of Xcode, use the “-ev” option.)
Use this option in the build system:

```
-t imch
```

Extract the archive and in the resulting folder open “minivmac.xcodeproj”. Choose the “Build” command from the “Build” menu. When it finishes there should now be a program called “minivmac” in the same folder.

Macintosh OS X - PowerPC

Building with Xcode 2.4.1:
(For other versions of Xcode, use the “-ev” option.)
Use this option in the build system:

```
-t mach
```

Extract the archive and in the resulting folder open “minivmac.xcodeproj”. Choose the “Build” command from the “Build” menu. When it finishes there should now be a program called “minivmac” in the same folder.

Building with Metrowerks CodeWarrior:
Mini vMac was last tested with version “8.3”. Use this option in the build system:

```
-t mach -e mw8
```

Launch the CodeWarrior IDE, and choose the “Import Project ...” command from the “File” menu. select the file “minivmac.mcp.xml”. It then asks where you wish to put the project file. You must put it in the same folder as “minivmac.mcp.xml” or it won’t work. Unfortunately, the dialog doesn’t default to this folder. You can name the file “minivmac.mcp”, but the exact name doesn’t matter. You can now choose the “Make” command from the “Project” menu, and when it finishes there should now be a program called “minivmac” in the same folder.

Microsoft Windows

Building with Microsoft Visual C++ 2005 Express Edition:
(For other versions of Microsoft Visual C++, use the “-ev” option.)
Use this option in the build system:

```
-t wx86
```

Extract the archive and in the resulting folder open the file “minivmac.sln”. Choose the “Build Solution” command from the “Build” menu. When it finishes there should now be a program called “minivmac.exe” in the same folder.

Building with the lcc-win32 system:
Use this option in the build system:

```
-t wx86 -e lcc
```

Launch lcc-win32 and choose the “Import...” command from the “Project” menu. Select the file “minivmac.prj”. Choose the “Make” command from the “Compiler” menu. When it finishes there should now be a program called “minivmac.exe” in the same folder.

Building with Bloodshed Dev-C++:
Mini vMac was last tested with version “4.9.8.0”.
Use this option in the build system:

```
-t wx86 -e dvc
```

Launch Bloodshed Dev-C++ and choose the “Open Project or File...” command from the “File” menu. Select the file “minivmac.dev”. Choose the “Compile” command from the “Execute” menu. When it finishes there should now be a program called “minivmac.exe” in the same folder.

Building with Metrowerks CodeWarrior:
Mini vMac was last tested with version “8.3”.
Use this option in the build system:

```
-t wx86 -e mw8
```

Launch the CodeWarrior IDE, and choose the “Import Project ...” command from the “File” menu. select the file “minivmac.mcp.xml”. It then asks where you wish to put the project file. You must put it in the same folder as “minivmac.mcp.xml” or it won’t work. You can name the file “minivmac.mcp”, but the exact name doesn’t matter. You can now choose the “Make” command from the “Project” menu, and when it finishes there should now be a program called “minivmac” in the same folder.

Macintosh OS 9

Building with the Macintosh Programmers Workshop:

Mini vMac was tested with the most recent version, with the updated components from 12/04/02. No further MPW updates are expected.

Use this option in the build system:

```
-t mppc
```

Set the current directory to the extracted folder. (One way to do this is the “Set Directory...” command in the “Directory” menu.) Type “make” in the Worksheet window in a line by itself, and then press the “Enter” key to execute this command. This creates a list of commands. Select all of these commands (one way to do this is to use the “undo” command twice) and then press the “Enter” key to execute them. When the cursor stops spinning, there should now be a program called “minivmac” in the same folder.

Building with Metrowerks CodeWarrior:

Mini vMac was last tested with version “8.3”.

Use this option in the build system:

```
-t mppc -e mw8
```

Launch the CodeWarrior IDE, and choose the “Import Project ...” command from the “File” menu. select the file “minivmac.mcp.xml”. It then asks where you wish to put the project file. You must put it in the same folder as “minivmac.mcp.xml” or it won’t work. Unfortunately, the dialog doesn’t default to this folder. You can name the file “minivmac.mcp”, but the exact name doesn’t matter. You can now choose the “Make” command from the “Project” menu, and when it finishes there should now be a program called “minivmac” in the same folder.

Linux

Building with the Linux command line:

Mini vMac for Intel 32 bit Linux is compiled with the Gnu tools in Red Hat Linux 7 (running in Microsoft Virtual PC). Mini vMac for x86-64 Linux is compiled with the Gnu tools in Ubuntu 10.04 LTS . Mini vMac has also been compiled on a number of other Linux distributions without change. It probably should work with most Linux distributions, and some other Unix and similar operating systems, perhaps with some adjustments to the Makefile and the configuration files.
Use one of these options in the build system:

```
-t lx86 { Linux - Intel (32 bit) }
-t lx64 { Linux - x86-64 }
-t lppc { Linux - PowerPC }
```

Open a terminal window and change the current directory to the extracted folder. (Such as by using the “cd” command.) Then type “make” (and press return). When it finishes, there should now be a program called “minivmac” in the same folder.

Solaris

Building with the Gnu C compiler:
For SPARC use this option in the build system:

```
-t slrs
```

For Intel use this option in the build system:

```
-t sl86
```

(There currently isn't any difference in the source generated for SPARC and Intel except for the version name.)
Open a terminal window and change the current directory to the extracted folder. (Such as by using the “cd” command.) Then type “make” (and press return). When it finishes, there should now be a program called “minivmac” in the same folder.
:

Mini vMac Options

Skip down to Options Index
The main Download page provides the standard variation of Mini vMac. But much of the power of Mini vMac comes from the many other possible variations.

Mini vMac has no preference settings that persist across quitting and relaunching. This helps to keep the program small and fast, and the source code simple. But Mini vMac does have options that can be chosen at compile time. You can compile your own variations with precisely the combinations of options you want, following the instructions on the Build page.
Or, I can compile custom variations for you. See:
Mini vMac Variations Service . . . try it out!
Mini vMac does not create preference files or change registry settings or make any other changes to your computer. So there is no problem in having multiple variations of Mini vMac installed.

... Options Index ...

Sound Volume
Mininum Extensions
Mouse Emulation Accuracy
Grab Keys in Full Screen
32 bit drawing
Alternate Happy Mac Icon

Check ROM Checksum
Disable Rom Check
Disable Ram Test
LocalTalk
Icon Master

Target

What kind of computer do you want to run Mini vMac on? Choose one of these lines:

```
-t imch { Macintosh OS X - Intel }
-t mach { Macintosh OS X - PowerPC }
-t mppc { Macintosh OS 9 and earlier - PowerPC }
-t m68k { Macintosh - 680x0 }
-t wx86 { Microsoft Windows - Intel }
-t wx64 { Microsoft Windows - x86-64 }
-t lx86 { Linux - Intel (32 bit) }
-t lx64 { Linux - x86-64 }
-t lppc { Linux - PowerPC }
-t larm { Linux - ARM }
-t lspr { Linux - SPARC }
-t fbsd { FreeBSD on x86-32 }
-t fb64 { FreeBSD on x86-64 }
-t obsd { OpenBSD on x86-32 }
-t ob64 { OpenBSD on x86-64 }
-t nbsd { NetBSD on x86-32 }
-t nb64 { NetBSD on x86-64 }
-t dbsd { Dragonfly BSD on x86-32 }
-t db64 { Dragonfly BSD on x86-64 }
-t oind { OpenIndiana on x86-32 }
-t oi64 { OpenIndiana on x86-64 }
-t wcar { Pocket PC - ARM }
-t minx { Minix 3.2 }
```

The x86-64 versions are currently slower, for lack of assembly language tweaking, and should not be used if the x86-32 versions will work.

Language

By default, the user interface of Mini vMac is in English. Other languages can be chosen with one of these lines:

```
-lang eng { (default) English }
-lang fre { French }
-lang ita { Italian }
-lang ger { German }
-lang dut { Dutch }
-lang spa { Spanish }
-lang pol { Polish }
-lang ptb { Brazilian Portuguese }
```

If you would be interested in translating the user interface into some other language, see the Localization page for more details.

Model

By default, Mini vMac emulates a Macintosh Plus. But it also can be compiled to emulate a few other computers, with one of these lines:

```
-m 128K { Macintosh 128K }
-m 512Ke { Macintosh 512Ke }
-m Plus { (default) Macintosh Plus }
-m SE { Macintosh SE }
-m Classic { Macintosh Classic }
-m SEFDHD { Macintosh SE FDHD }
-m II { Macintosh II * }
```

* The Macintosh II emulation is incomplete, and will remain incomplete for all 3.3.x releases. This is not a matter of bugs - some critical pieces of the hardware are just not emulated at all, and any software that tries to use them will crash. Bug reports about what doesn't work are of no help to me.

The Macintosh II emulation is not available on a Macintosh 680x0 target (“-t m68k”), because the compiler used doesn’t support 64 bit integers currently needed for FPU emulation.

Since a Macintosh II can be hard to find, the Macintosh II emulator will accept the ROM from Macintosh IIx. The Macintosh IIx ROM appears to work with Macintosh II hardware. The Macintosh IIcx, the Macintosh II FDHD, and the Macintosh SE/30 all have the same ROM as the Macintosh IIx.

Screen Size

You can choose the emulated screen size with lines such as:

```
-hres 640 { horizontal resolution }
-vres 512 { vertical resolution }
```

Mini vMac requires that the horizontal resolution be a multiple of 32. For the Macintosh Plus, and other Macintosh computers without Color Quickdraw, this is a hack implemented with numerous ROM patches. It emulates a computer that never existed, so there will definitely be compatibility issues with some software. Also keep in mind that most games that will work on the Macintosh Plus are designed for 512x342, and don't benefit from a larger screen. You can also set the emulated screen smaller than 512x342, which could be useful on portable devices, but that will really cause compatibility issues.

For the Macintosh II emulation, these options specify the resolution of the external monitor, and shouldn't cause compatibility problems, at least if you choose values that were common on real monitors of the era. On the other hand, the Macintosh II emulation is unfinished, and not really usable yet.

Some example screen sizes:

common old Macintosh screen sizes

```
-hres 512 -vres 384
-hres 640 -vres 480
-hres 800 -vres 600
-hres 1024 -vres 768
```

common current screen sizes, but divided by two, so can use Magnify in full screen

```
-hres 512 -vres 384 { half 1024 x 768 }
-hres 640 -vres 400 { half 1280 x 800 }
-hres 640 -vres 512 { half 1280 x 1024 }
-hres 704 -vres 450 { half 1440 x 900, width
constrained }
-hres 832 -vres 525 { half 1680 x 1050, width
constrained }
-hres 960 -vres 540 { half 1920 x 1080 }
-hres 960 -vres 600 { half 1920 x 1200 }
```

Mini vMac allows up to 4 Megabytes of Video RAM in the Macintosh II emulation. Anything over 1 Megabyte requires a hack, since each NuBus slot only gets 1 Megabyte of the address space in 24 bit mode. Space from adjacent NuBus slots is given to the emulated video card.

Screen Depth

You can choose the emulated screen color depth with lines such as:

```
-depth 0 { black and white }
-depth 1 { 2 bit color (4 colors) }
-depth 2 { 4 bit color (16 colors) }
-depth 3 { 8 bit color (256 colors) (default for Mac II
emulation) }
-depth 4 { 16 bit color (thousands) }
-depth 5 { 32 bit color (millions) }
```

These options only work for Macintosh models that support Color Quickdraw, which currently means Mac II emulation only.
Color is currently only implemented for Macintosh OS X and Windows, and experimentally in the X versions.
Color depth is a compile time option, instead of run time option, to help keep Mini vMac simple and small. However, regardless of the chosen color depth, Black and White is also available, and can be selected from the "Monitors" control panel. (In fact, you may not see color until selecting it from the "Monitors" control panel.)

Full Screen Mode

```
-fullscreen 0 { (default) start with Full Screen Mode
off }
-fullscreen 1 { start with Full Screen Mode on }
-var-fullscreen 0 { Full Screen Mode is constant }
-var-fullscreen 1 { (default) Full Screen Mode is
variable }
```

"-var-fullscreen 0" combines with -fullscreen like so:

```
-var-fullscreen 0 -fullscreen 0 { Never run in Full
Screen Mode }
-var-fullscreen 0 -fullscreen 1 { Always run in Full
Screen Mode }
```

In both cases, the "F" control mode command disappears.

Magnify

```
-magnify 0 { (default) start with Magnify Mode off }
-magnify 1 { start with Magnify Mode on }
```

The magnification factor can be changed:

```
-mf 1 { disable magnification }
-mf 2 { (default) 2x }
-mf 3 { 3x }
-mf 4 { 4x }
...
```

Disabling magnification with "-mf 1" removes the Control-M command.

Sound
The Macintosh, Windows, Linux, FreeBSD, and NetBSD versions have sound emulation on by default. The Dragonfly BSD and OpenIndiana version can be compiled with sound, but this hasn't been tested. The OpenBSD version can be compiled with sound, but it doesn't work. (Other versions don't implement sound.)

```
-sound 1 { Emulate sound }
-sound 0 { No sound emulation }
```

One reason to disable sound is to avoid the Macintosh start up beep. Another reason is to get a few percent better performance.

Sound Sample Size

```
-sss 3 { (default) 8 bit sound }
-sss 4 { 16 bit sound }
```

A Macintosh Plus outputs 8 bit sound (256 possible levels), which is then modulated by the sound volume setting (8 possible levels), and also by the square wave generator. By default Mini vMac currently outputs 8 bit sound, but there is an option to output 16 bit sound, which allows more accurate output when the sound volume setting is less than maximum, and when the square wave generator is used.

Number of disk images
By default, Mini vMac can mount up to 6 disk images (but for the Mac 128K/512K emulation the default is 2). This is because the replacement disk driver is trying to match the size of a data structure used by the real disk driver (on the Macintosh 128K/512K this structure is smaller). You can at compile time choose to support more disk images, up to 32, which makes this data structure larger, at some slight decrease in authenticity, and some slight memory and time overhead. Use a line such as:

```
-drives 1
-drives 2
-drives 3
...
-drives 6 { (default) }
...
-drives 16
...
-drives 32
```

According to Apple’s Technical Note FL530, some versions of the System Software will not work correctly with more than 20 mounted

volumes. The Standard File dialogs will corrupt the stack and may crash.
One reason for using the “-drives” option is that the installer programs for some Macintosh applications don't cope very well with multiple floppy drives, and insist on constantly ejecting the boot or destination disks rather than the install disks it has finished with. Some installers with this problem work much better if all installation disks can be mounted at once before starting.

Checksum disk images

```
-sony-sum 1
```

With the above line, Mini vMac will update the checksum in a Disk Copy 4.2 disk image when it is unmounted. This prevents other programs that deal with such images from complaining about an invalid checksum. (I didn't include this by default, because it makes Mini vMac slightly bigger and slower.)

Disk images file tags

```
-sony-tag 1
```

With the above line, Mini vMac tries to support file tags, for disk image formats that support them. There are an additional 12 bytes for each 512 byte block on a 400K or 800K floppy disk, containing some additional information that was supposed to aid in recovering damaged disks, but was never actually used much. The Disk Copy 4.2 disk image format can support these tags. (The more usual raw format, such as found in Blanks, does not.)
For full support of Disk Copy 4.2 format, enable both checksums and tags:

```
-sony-sum 1 -sony-tag 1
```

Disk Copy 4.2 format

```
-sony-dc42 0
```

The above line completely disables support for disk images in disk copy 4.2 format. This could be useful when trying to compile the smallest and simplest version of Mini vMac possible for some specific purpose. It should not be used when compiling a version of Mini vMac for general distribution, because a primary goal of Mini vMac is that disk images that work with any past version of the program should also work with the current and any future version (at least when default compile options are used).

Command Control Swap

You can swap the emulated Command and Control keys with:
`-ccs { Command Control Swap }`
This could be useful for people who are used to Windows, so that you can use Control-C to copy instead of Command-C (or actually Alt-C, as Alt is located where the Command key is on a Macintosh keyboard).
It can also be useful for people used to Macintosh who have remapped the keys on their Windows machine to swap Control and Alt. Using '-ccs' will swap them back for Mini vMac.
Linux is the same as Windows in this way, as far as I have seen.
The '-ccs' option does not affect F1 and F2, but should affect any other keys that get mapped to Control and Command.

Alternate Keyboard Mode

`-akm { enable Alternate Keyboard Mode }`
Some commonly used keys, such as the arrow keys, are a bit of a stretch to reach, and more importantly, their positions can be different on different keyboards. The Alternate Keyboard Mode provides an alternate way to type some of these keys.
The program starts with the mode on. Pressing the 'm' key leaves the mode. The intent is that you only leave the mode temporarily to type text. (I believe this is a similar idea to how the vi text editor works.)
The alternate keyboard mode is entered by pressing the ';' key. Pressing ';' has no effect when in the alternate keyboard mode, so it can be used at any time to bring the program to a known state.
In the Alternate Keyboard Mode, the letter keys are remapped as follows (non letter keys are unchanged):
j - Left arrow
l - Right arrow
i - Up arrow
k - Down arrow
s - Shift
d - Option
f - Command
z - F1 (often means Undo)
x - F2 (often means Cut)
c - F3 (often means Copy)
v - F4 (often means Paste)

e - BackSpace
r - Return
t - Tab
g - Enter
y - Escape
a - SemiColon
u - [
o -]
b - BackSlash
h - Equal
n - Minus
q - Grave

There is a visual indication when the mode is off, intended to be easy to see in peripheral vision, without covering up where text is normally typed.

Holding down the command, option, or shift keys will temporarily turn off the Alternate Keyboard Mode. So if you have the mode on, commands like Command-S will still work, and anything except lowercase letters can be typed without leaving the mode.

International Keyboard fix

The Virtual-Key Codes of Microsoft Windows, that are independent of differences in keyboard hardware, turn out not to be independent of the choice of Keyboard Layout. Changing the Keyboard Layout to something other than "US" may scramble the Virtual-Key Codes, strangely enough. Mini vMac will check the current Keyboard Layout, and attempt to unscramble the codes, so that the Keyboard Layout chosen in Macintosh operating system running within Mini vMac will work properly. You can disable this fix with:

```
-ikb 0 { International Keyboard fix off }
```

One reason for disabling it is because this is fairly large amount of code and US only users don't need it. But also it can be disabled because I'm not sure I really understand this. Why did no one complain about this issue in a decade?

This option is only for Microsoft Windows.

Speed

You can use one of these lines to set initial speed:

```
-speed z { 1x }
```

```
-speed 1 { 2x }
-speed 2 { 4x }
-speed 3 { (default) 8x }
-speed 4 { 16x }
-speed 5 { 32x }
-speed a { All out }
```

You can use one of these lines to set initial value of the Run in Background option:

```
-bg 0 { (default) start with Run in Background off }
-bg 1 { start with Run in Background on }
```

You can use one of these lines to set initial value of the AutoSlow option:

```
-as 0 { start with AutoSlow disabled }
-as 1 { (default) start with AutoSlow enabled }
```

For Macintosh II emulation, AutoSlow is disabled by default. AutoSlow may need some further tuning to work well with Mac II emulation.

Timing Accuracy

Mini vMac estimates the number of clock cycles used by each instruction excuted. There are three levels of accuracy.

```
-ta 0 { least accurate timing }
-ta 1 { (default) }
-ta 2 { most accurate timing }
```

In "-ta 0" all instructions are assumed to take the same number of cycles. This closely matches Mini vMac 3.1.3 and earlier.

For "-ta 1", Mini vMac assigns an average number of cycles for each of the 65536 primary opcodes. This table is generated using timings from Motorola documentation. When a range of timings are possible for a primary opcode, an average was just guessed. In some future version of Mini vMac, this table should be tested and calibrated by comparing to real hardware.

For "-ta 2", Mini vMac supplements the cycles table by computing more accurate estimates for certain instructions depending on the current data. It is still not completely accurate. Completely accurate timing would be exceedingly difficult. For example, the CPU and video output conflict for accesses to RAM, and that would seem very complex to model.

Currently 68000 timings are used even in the 68020 emulation. More accurate timing for 68020 should be added in a future version. Truly accurate timing for 68020 would be much more difficult than for the 68000 because of pipelining and caching, probably to the point of being unfeasible for Mini vMac. But more accurate averages should be possible.

Emulated CPU

Most computers emulated by Mini vMac have a 68000 processor. But you can force Mini vMac to emulate a 68020 processor with:

```
-em-cpu 2 { 68020 }
```

Memory

By default, Mini vMac emulates a Macintosh Plus with 4M of memory. But it also can be compiled to emulate other memory sizes, depending on model:

```
Macintosh 128K and Macintosh 512Ke :
-mem 128K { 128 Kilobytes }
-mem 512K { 512 Kilobytes }

Macintosh Plus :
-mem 128K { 128 Kilobytes }
-mem 512K { 512 Kilobytes }
-mem 1M { 1 Megabyte }
-mem 2M { 2 Megabytes }
-mem 2.5M { 2.5 Megabytes }
-mem 4M { (default) 4 Megabytes }

Macintosh SE, Classic, and SE FDHD :
-mem 512K { 512 Kilobytes }
-mem 1M { 1 Megabyte }
-mem 2M { 2 Megabytes }
-mem 2.5M { 2.5 Megabytes }
-mem 4M { (default) 4 Megabytes }

Macintosh II :
-mem 1M { 1 Megabyte }
-mem 2M { 2 Megabytes }
-mem 4M { 4 Megabytes }
-mem 5M { 5 Megabytes }
-mem 8M { (default) 8 Megabytes }
```

The build system checks that the memory size you specify is supported by the Macintosh model you have chosen to emulate.

Caret Blink Time

```
-cbt 3 { Fast }
-cbt 8 { Medium }
-cbt 15 { Slow }
```

The “Rate of Insertion Point Blinking” is stored in the Parameter RAM. The default is 3 (Fast), except for Macintosh II emulation where it is 8 (Medium). Must be between 1 and 15.

Double Click Time

```
-dct 5 { Fast }
-dct 8 { Medium }
-dct 12 { Slow }
```

The “Double Click Time” (the maximum time between mouse button clicks which will be treated as a double click) is stored in the Parameter RAM. The default is 5 (Fast), except for Macintosh II emulation where it is 8 (Medium). Must be between 1 and 15.

Menu Blinks

```
-mnb 0 { None }
-mnb 1 { once }
-mnb 2 { twice }
-mnb 3 { (default) thrice }
```

The number of times the selected menu item blinks when when the mouse button is released, which is stored in the Parameter RAM.

Keyboard Repeat Threshold

```
-kyt 0 { Off }
-kyt 10 { Long }
-kyt 6 { (default) }
-kyt 4
-kyt 3 { Short }
```

The delay after a key is held down until it begins to automatically repeat, which is stored in the Parameter RAM. Must be between 0 and 15.

Keyboard Repeat Rate

```
-kyr 0 { Slow }
-kyr 6
-kyr 4
-kyr 3 { (default) }
-kyr 1 { Fast }
```

The rate at which a key automatically repeats when it is held down, which is stored in the Parameter RAM. Must be between 0 and 15.

Sound Volume

```
-svl 0 { Minimum }
```

```
...
-svl 7 { Maximum }
```

The Sound Volume is stored in the Parameter RAM. The default is 7 (Maximum) when sound is enabled, otherwise it is 0.

Mininum Extensions

```
-min-extn { Mininum Extensions }
```

This option turns off all but the minimum Mini vMac extensions, such as importing and exporting the clipboard, and creating new disk images and files, leaving just what is needed for the replacement disk driver to operate.

This makes the program smaller, and perhaps reduces potential security concerns. So it could be worth doing for when the extensions aren't needed, such as for most games.

Mouse Emulation Accuracy

```
-emm 0 { less accurate emulation in Full Screen Mode }
```

This option disables the more accurate mouse emulation normally used in Full Screen Mode, which looks at the motion of the real mouse rather than its absolute position. This allows Mini vMac to work somewhat better on tablet computers without a mouse. But such computers are still not really supported.

Grab Keys in Full Screen

```
-gkf 0 { don't grab keys in Full Screen Mode }
```

Normally, when in Full Screen Mode, Mini vMac will try to “grab” the keyboard, preventing the operating system from intercepting keys. So in the Windows version, the ‘windows’ key can be used as an ‘option’ key, instead of popping up the “Start” menu. And in the OS X version, Command-Tab won't switch away from Mini vMac. This is also implemented in the X version.

This option disables grabbing the keyboard, allowing the operating system to intercept keys when Mini vMac is in Full Screen Mode.

32 bit drawing

```
-ci 0 { do not use 32 bit drawing }
```

This option is only for Linux and other X versions, when color depth is zero. Passing single bit per pixel images to the operating system is suspected to be unreliable, so Mini vMac now always passes 32 bit images. This option forces it to use single bit images, which may be more efficient, when you know that it works (i.e. there weren't problems in Mini vMac 3.4 and earlier).

Alternate Happy Mac Icon

```
-ahm none { (default) }
-ahm aside
-ahm cheese
-ahm evil
-ahm horror
-ahm lady_mac
-ahm moustache
-ahm nerdy
-ahm pirate
-ahm sleepy
-ahm sly
-ahm sunglasses
-ahm surprise
-ahm tongue
-ahm yuck
-ahm zombie
```

Patch the ROM to replace the “Happy Mac” icon displayed on boot when a disk is inserted, with one of the images designed by Steve Chamberlin for his Mac ROM-inator (used with permission).

ROM Size

```
-rsz 16 { 64K }
-rsz 17 { 128K }
-rsz 18 { 256K }
-rsz 19 { 512K }
-rsz 20 { 1024K }
```

The build system will normally select the correct ROM Size for the Macintosh model you have chosen to emulate. But you can override this, such as to use a ROM image for Steve Chamberlin’s Mac ROM-inator. If you use this option, you will also need “-chr 0” as described below. (You may also want “-drc 0 -drt 0 -speed z -ta 2 -sony-sum 1 -sony-tag 1” to test the image with maximum accuracy.)

Check ROM Checksum

The first 4 bytes of a Macintosh ROM contain a checksum of the remaining bytes, and there is code in the ROM to check the checksum on boot. Mini vMac patches the ROM image, so it disables this checking code. Mini vMac does the check itself before patching the ROM. It also checks that the 4 byte checksum matches one of the known ROM images for the model you have chosen to emulate. If you want to have Mini vMac use a ROM image that has been

modified you can disable these checks with:
`-chr 0`

Disable Rom Check

There is code in the Macintosh ROM to checksum the ROM at boot. Since Mini vMac patches the ROM, it also patches out this check. If you are using a ROM image with Mini vMac that is already patched (such as for Steve Chamberlin's Mac ROM-inator), this check may already be patched out. In that case Mini vMac doesn't need to, and probably shouldn't, to avoid interference in case a different method of patching out is used.
`-drc 0 { Don't Disable Rom Check}`

Disable Ram Test

There is code in the Macintosh ROM to test proper operation of RAM at boot. Mini vMac normally patches the ROM to disable this test, to speed up booting. For greater realism, you can leave this test in.
`-drt 0 { Don't Disable Ram Test}`

LocalTalk

`-lt { LocalTalk emulation }`

This enables Mike Fort's LocalTalk emulation. There are currently some limitations. It is only implemented for OS X. It requires running the command "sudo chmod ugo+rw /dev/bpf*" to allow Mini vMac (and everyone else) access to all network traffic. The "-lt" option also causes Mini vMac to run in the background by default, because Mini vMac can't be a proper LocalTalk node if it isn't running. And you need to manually turn on AppleTalk in the chooser - I can set the PRAM flags to boot with AppleTalk already on, but it doesn't work properly.

Icon Master

`-im 1 { Icon Master }`

When compiled with the "Icon Master" setting turned on, Mini vMac will try to take ownership of Disk Image files and ROM Image files. It is better to have at most one copy of Mini vMac compiled with '-im 1' on a single computer. It may be preferred to have no copies of Mini vMac compiled with '-im 1', to avoid possible conflicts with other programs.
In Mac OS X, Windows, and Mac OS 9 and earlier, double clicking on a Disk Image icon will open it with a copy of Mini vMac compiled with '-im 1'. Also, Disk Image and ROM Image files will be given

Mini vMac icons.
In Mac OS 9, and early versions of OS X, Disk Image files should have the creator type set to “MnvM” for double clicking to work (see “SetFType”). Also, the file type should be “MvIm” to see the Mini vMac Disk Image icon. ROM Image files should have the creator type “MnvM” and the file type “ROM!” to see the Mini vMac ROM Image icon.
In Windows and later versions of Mac OS X, Disk Images files should have a name that ends in “.dsk” for double clicking to work and to see the Mini vMac Disk Image icon. ROM Images files should have a name that ends in “.ROM” to see the Mini vMac Disk Image icon.
In Windows, a copy of Mini vMac compiled with '-im 1' will install itself in the registry when it is launched. So it will not take ownership of Disk Image and ROM Image files until after the first launch. (And the Mini vMac document icons may not appear until the computer is rebooted.) Mini vMac can only install itself in the registry if it is launched from an administrator account.
This setting currently has no effect in the Linux and other X versions.