|  |  |  |
| --- | --- | --- |
| Bochs User Manual | | |
| [Prev](http://docs.google.com/rom-images.html) | Chapter 4. Setup | [Next](http://docs.google.com/keymap.html) |

4.3. The configuration file bochsrc

Bochs uses a configuration file called bochsrc to know where to look for disk images, how the Bochs emulation layer should work, etc. When you first start up Bochs, it looks around for its configuration file (see [Section 5.2](http://docs.google.com/search-order.html)), and parses it. Here are a few lines from a sample file:

ata0-master: type=disk, path="30M.sample", cylinders=615, heads=6, spt=17  
 boot: disk

The format is very strict, so be sure to put the right number of spaces and use lowercase letters. As you can see, most lines have a keyword telling what is being configured, followed by a colon, followed by a few property=value pairs, separated by commas. For very simple options, sometimes just a single value is needed. The source and binary distributions come with a sample bochsrc, so you can just copy the sample file and edit the settings you need to change.

The syntax used for bochsrc can also be used as command line arguments for Bochs. If you have any spaces in your command line arguments, they should be enclosed in single quotes, for example:

bochs 'boot:floppy' 'floppya: 1\_44=a.img, status=inserted'

For other arguments, see section [Command line arguments](http://docs.google.com/using-bochs.html#COMMANDLINE).

You can use environment variables with the dollar sign prefix in the bochsrc file, for example:

floppya: 1\_44="$IMAGES/bootdisk.img", status=inserted  
 boot: floppy

There are two environment variables with a built-in default value which is set at compile or installation time. $BXSHARE points to the "share" directory which is typically /usr/local/share/bochs on UNIX machines. See the $(sharedir) variable in the Makefile for the exact value. $BXSHARE is used in the config files of the Bochs disk images to locate the directory where the BIOS images and keymaps can be found. If $BXSHARE is not defined, Bochs will supply the default value. Also, $LTDL\_LIBRARY\_PATH points to a list of directories to search in for Bochs plugins. The paths are separated by colons (on Windows: semicolons). A compile-time default is provided if this variable is not defined by the user. On Win32 and MacOSX, the default for the share directory is determined by a platform-specific specific algorithm. On Win32, we use the registry to see what directory Bochs and its support files were installed in. On MacOSX, the share directory is the directory where the application is located.

You can use the **#include** statement in the bochsrc to read the configuration from other files. Now it is possible to put platform or installation defaults in a global config file (e.g. location of rom images). Put this on top of your config file if the global configuration is stored in /etc:

#include /etc/bochsrc

Bochs now treats unknown options as device plugin names. It tries to load this plugin and if successful it tries to call the parser function for this configuration line which is located in the plugin. This mechanism is implemented for the Bochs network, sound and USB devices. If there is a typo in an option name or an obsolete option is used, Bochs will panic and exit with a plugin load failure error message. In that case the failing line in your bochsrc file must be reviewed and fixed.

The section below lists all the supported bochsrc options.

4.3.1. plugin\_ctrl

Example:

plugin\_ctrl: unmapped=0, e1000=1 # unload 'unmapped' and load 'e1000'

Controls the presence of optional device plugins. These plugins are loaded directly with this option and some of them install a config option that is only available when the plugin device is loaded. The value "1" means to load the plugin and "0" will unload it (if loaded before).

These plugins will be loaded by default (if present): 'biosdev', 'extfpuirq', 'gameport', 'iodebug','parallel', 'serial', 'speaker' and 'unmapped'.

These plugins are also supported, but they are usually loaded directly with their bochsrc option: 'e1000', 'es1370', 'ne2k', 'pcidev', 'pcipnic', 'sb16', 'usb\_ohci', 'usb\_uhci', 'usb\_xhci' and 'voodoo'.

4.3.2. config\_interface

The configuration interface is a series of menus or dialog boxes that allows you to edit all the settings that control Bochs' behavior. Depending on the platform there are up to 3 choices of configuration interface: a text mode version called "textconfig" and two graphical versions called "win32config" and "wx". The text mode version uses stdin/stdout and is always compiled in, unless Bochs is compiled for wx only. The choice "win32config" is only available on win32 and it is the default there. The choice "wx" is only available when Bochs is compiled with wxWidgets support, see [Section 3.4.13](http://docs.google.com/compiling.html#COMPILE-WX). If you do not write a config\_interface line, Bochs will choose a default for you (usually textconfig).

**Note:** wxWidgets provides both a configuration interface and a display library. So if you use the "wx" configuration interface, you must also use the "wx" display library, see [display\_library option](http://docs.google.com/bochsrc.html#BOCHSOPT-DISPLAYLIBRARY).

Examples:

config\_interface: textconfig  
 config\_interface: win32config  
 config\_interface: wx

4.3.3. display\_library

The display library is the code that displays the Bochs VGA screen. Bochs has a selection of about 10 different display library implementations for different platforms. If you run configure with multiple --with-\* options, the display\_library option lets you choose which one you want to run with. If you do not use a display\_library line, Bochs will choose a default for you.

**Note:** wxWidgets provides both a configuration interface and a display library. So if you use the "wx" display library, you must also use the "wx" configuration interface, see [config\_interface option](http://docs.google.com/bochsrc.html#BOCHSOPT-CONFIGINTERFACE).

Examples:

display\_library: x  
 display\_library: sdl

Some display libraries now support specific options to control their behaviour. These options are supported by more than one display library:

"gui\_debug" - use GTK debugger gui (sdl, x) / Win32 debugger gui (sdl, sdl2, win32)  
 "hideIPS" - disable IPS output in status bar (rfb, sdl, sdl2, vncsrv, win32, wx, x)  
 "nokeyrepeat" - turn off host keyboard repeat (sdl, sdl2, win32, x)  
 "timeout" - time (in seconds) to wait for client (rfb, vncsrv)

See the examples below for other currently supported options.

display\_library: sdl, options="fullscreen" # startup in fullscreen mode  
 display\_library: sdl2, options="fullscreen" # startup in fullscreen mode

**Table 4-2. display\_library values**

|  |  |
| --- | --- |
| Option | Description |
| x | use X windows interface, cross platform |
| win32 | use native win32 libraries |
| carbon | use Carbon library (for MacOS X) |
| macintosh | use MacOS pre-10 |
| amigaos | use native AmigaOS libraries |
| sdl | use SDL 1.2.x library, cross platform, details in [Section 3.4.11](http://docs.google.com/compiling.html#COMPILE-SDL) |
| sdl2 | use SDL 2.x library, cross platform, details in [Section 3.4.12](http://docs.google.com/compiling.html#COMPILE-SDL2) |
| svga | use SVGALIB library for Linux, allows graphics without X windows |
| term | text only, uses curses/ncurses library, cross platform |
| rfb | provides an interface to AT&T's VNC viewer, cross platform, details in [Section 3.4.9](http://docs.google.com/compiling.html#COMPILE-RFB) |
| vncsrv | use LibVNCServer for extended RFB(VNC) support, details in [Section 3.4.10](http://docs.google.com/compiling.html#COMPILE-VNCSRV) |
| wx | use wxWidgets library, cross platform, details in [Section 3.4.13](http://docs.google.com/compiling.html#COMPILE-WX) |
| nogui | no display at all |

4.3.4. cpu

Example:

cpu: count=2, ips=10000000

This defines the parameters of the cpu inside Bochs:

**model**

Selects CPU configuration to emulate from pre-defined list of all supported configurations. When this option is used and the value is different from 'bx\_generic', the parameters of the [CPUID](http://docs.google.com/bochsrc.html#BOCHSOPT-CPUID) option have no effect anymore. See the [Section 5.4](http://docs.google.com/cpu-models.html) for supported values.

**count**

Set the number of processors:cores per processor:threads per core when Bochs is compiled for SMP emulation. Bochs currently supports up to 14 threads (legacy APIC) or 254 threads (xAPIC or higher) running simultaniosly. If Bochs is compiled without SMP support, it won't accept values different from 1. For more information on SMP see [Section 8.9](http://docs.google.com/smp.html).

**quantum**

Maximum amount of instructions allowed to execute by processor before returning control to another cpu. This option exists only in Bochs binary compiled with SMP support.

**reset\_on\_triple\_fault**

Reset the CPU when triple fault occur (highly recommended) rather than PANIC. Remember that if you are trying to continue after triple fault the simulation will be completely bogus !

**cpuid\_limit\_winnt**

Determine whether to limit maximum CPUID function to 2. This mode is required to workaround WinNT installation and boot issues.

**mwait\_is\_nop**

When this option is enabled MWAIT will not put the CPU into a sleep state. This option exists only if Bochs compiled with --enable-monitor-mwait.

**msrs**

Define path to user CPU Model Specific Registers (MSRs) specification. See example in msrs.def.

**ignore\_bad\_msrs**

Ignore MSR references that Bochs does not understand; print a warning message instead of generating #GP exception. This option is enabled by default but will not be avaiable if configurable MSRs are enabled.

**ips**

Emulated Instructions Per Second. This is the number of IPS that Bochs is capable of running on your machine. You can recompile Bochs with --enable-show-ips option enabled, to find your workstation's capability. Measured IPS value will then be logged into your [log file](http://docs.google.com/bochsrc.html#BOCHSOPT-LOG) or in the status bar (if supported by the gui).

IPS is used to calibrate many time-dependent events within the Bochs simulation. For example, changing IPS affects the frequency of VGA updates, the duration of time before a key starts to autorepeat, and the measurement of BogoMips and other benchmarks. The table below lists some typical IPS settings for different machines[[1]](#2lwamvv).

**Table 4-3. Example IPS Settings**

|  |  |  |  |
| --- | --- | --- | --- |
| Bochs | Speed | Machine/Compiler | Typical IPS |
| 2.4.6 | 3.4Ghz | Intel Core i7 2600 with Win7x64/g++ 4.5.2 | 85 to 95 MIPS |
| 2.3.7 | 3.2Ghz | Intel Core 2 Q9770 with WinXP/g++ 3.4 | 50 to 55 MIPS |
| 2.3.7 | 2.6Ghz | Intel Core 2 Duo with WinXP/g++ 3.4 | 38 to 43 MIPS |
| 2.2.6 | 2.6Ghz | Intel Core 2 Duo with WinXP/g++ 3.4 | 21 to 25 MIPS |
| 2.2.6 | 2.1Ghz | Athlon XP with Linux 2.6/g++ 3.4 | 12 to 15 MIPS |

4.3.5. cpuid

Example:

cpuid: level=6, mmx=1, sep=1, sse=sse4\_2, apic=xapic, aes=1, movbe=1, xsave=1

This defines features and functionality supported by Bochs emulated CPU. These settings are only valid and configurable if the [cpu model](http://docs.google.com/cpu-models.html) is set to the default value 'bx\_generic'.

**level**

Set emulated CPU level information returned by CPUID. Default value is determined by configure option [--enable-cpu-level](http://docs.google.com/compiling.html#CONFIGURE-ENABLE-CPU-LEVEL). Currently supported values are 5 (for Pentium and similar processors) and 6 (for P6 and later processors).

**family**

Set family information returned by CPUID. Default family value determined by configure option [--enable-cpu-level](http://docs.google.com/compiling.html#CONFIGURE-ENABLE-CPU-LEVEL).

**model**

Set model information returned by CPUID. Default model value is 3.

**stepping**

Set stepping information returned by CPUID. Default stepping value is 3.

**vendor\_string**

Set the CPUID vendor string returned by CPUID(0x0). This should be a twelve-character ASCII string.

**brand\_string**

Set the CPUID brand string returned by CPUID(0x80000002 .. 0x80000004]). This should be at most a forty-eight-character ASCII string.

**mmx**

Select MMX instruction set support. This option exists only if Bochs compiled with BX\_CPU\_LEVEL >= 5.

**apic**

Select APIC configuration (LEGACY/XAPIC/XAPIC\_EXT/X2APIC). This option exists only if Bochs compiled with BX\_CPU\_LEVEL >= 5.

**sep**

Select SYSENTER/SYSEXIT instruction set support. This option exists only if Bochs compiled with BX\_CPU\_LEVEL >= 6.

**simd**

Select SIMD instructions support. Any of NONE/SSE/SSE2/SSE3/SSSE3/SSE4\_1/SSE4\_2/AVX/AVX2/AVX512 could be selected. This option exists only if Bochs compiled with BX\_CPU\_LEVEL >= 6. The AVX choises exists only if Bochs compiled with --enable-avx option.

**sse4a**

Select AMD SSE4A instructions support. This option exists only if Bochs compiled with BX\_CPU\_LEVEL >= 6.

**misaligned\_sse**

Select AMD Misaligned SSE mode support. This option exists only if Bochs compiled with BX\_CPU\_LEVEL >= 6.

**aes**

Select AES instruction set support. This option exists only if Bochs compiled with BX\_CPU\_LEVEL >= 6.

**sha**

Select SHA instruction set support. This option exists only if Bochs compiled with BX\_CPU\_LEVEL >= 6.

**movbe**

Select MOVBE Intel(R) Atom instruction support. This option exists only if Bochs compiled with BX\_CPU\_LEVEL >= 6.

**adx**

Select ADCX/ADOX instructions support. This option exists only if Bochs compiled with BX\_CPU\_LEVEL >= 6.

**xsave**

Select XSAVE extensions support. This option exists only if Bochs compiled with BX\_CPU\_LEVEL >= 6.

**xsaveopt**

Select XSAVEOPT instruction support. This option exists only if Bochs compiled with BX\_CPU\_LEVEL >= 6.

**avx\_f16c**

Select AVX float16 convert instructions support. This option exists only if Bochs compiled with --enable-avx option.

**avx\_fma**

Select AVX fused multiply add (FMA) instructions support. This option exists only if Bochs compiled with --enable-avx option.

**bmi**

Select BMI1/BMI2 instructions support. This option exists only if Bochs compiled with --enable-avx option.

**fma4**

Select AMD four operand FMA instructions support. This option exists only if Bochs compiled with --enable-avx option.

**xop**

Select AMD XOP instructions support. This option exists only if Bochs compiled with --enable-avx option.

**tbm**

Select AMD TBM instructions support. This option exists only if Bochs compiled with --enable-avx option.

**x86\_64**

Enable x86-64 and long mode support. This option exists only if Bochs compiled with x86-64 support.

**1g\_pages**

Enable 1G page size support in long mode. This option exists only if Bochs compiled with x86-64 support.

**pcid**

Enable Process-Context Identifiers (PCID) support in long mode. This option exists only if Bochs compiled with x86-64 support.

**smep**

Enable Supervisor Mode Execution Protection (SMEP) support. This option exists only if Bochs compiled with BX\_CPU\_LEVEL >= 6.

**smap**

Enable Supervisor Mode Access Prevention (SMAP) support. This option exists only if Bochs compiled with BX\_CPU\_LEVEL >= 6.

**mwait**

Select MONITOR/MWAIT instructions support. This option exists only if Bochs compiled with --enable-monitor-mwait.

**vmx**

Select VMX extensions emulation support. This option exists only if Bochs compiled with --enable-vmx option.

**svm**

Select AMD SVM (Secure Virtual Machine) extensions emulation support. This option exists only if Bochs compiled with --enable-svm option.

4.3.6. memory

Examples:

memory: guest=512, host=256

Set the amount of physical memory you want to emulate.

**guest**

Set amount of guest physical memory to emulate. The default is 32MB, the maximum amount limited only by physical address space limitations.

**host**

Set amount of host memory you want to allocate for guest RAM emulation. It is possible to allocate less memory than you want to emulate in guest system. This will fake guest to see the non-existing memory. Once guest system touches new memory block it will be dynamically taken from the memory pool. You will be warned (by FATAL PANIC) in case guest already used all allocated host memory and wants more.

**Note:** Due to limitations in the host OS, Bochs fails to allocate more than 1024MB on most 32-bit systems. In order to overcome this problem configure and build Bochs with --enable-large-ramfile option.

4.3.7. megs

Examples:

megs: 32  
 megs: 128

This option sets the 'guest' and 'host' memory parameters to the same value. In all other cases the 'memory' option should be used instead.

4.3.8. romimage

Examples:

romimage: file=bios/BIOS-bochs-latest, address=0xfffe0000  
 romimage: file=$BXSHARE/BIOS-bochs-legacy, address=0xffff0000  
 romimage: file=mybios.bin, address=0xfff80000  
 romimage: file=mybios.bin

The ROM BIOS controls what the PC does when it first powers on. Normally, you can use a precompiled BIOS in the source or binary distribution called BIOS-bochs-latest. The default ROM BIOS is usually loaded starting at address 0xfffe0000, and it is exactly 128k long. The legacy version of the Bochs BIOS is usually loaded starting at address 0xffff0000, and it is exactly 64k long. You can use the environment variable $BXSHARE to specify the location of the BIOS. The usage of external large BIOS images (up to 512k) at memory top is now supported, but we still recommend to use the BIOS distributed with Bochs. The start address is optional, since it can be calculated from image size.

4.3.9. vgaromimage

Examples:

vgaromimage: file=bios/VGABIOS-elpin-2.40  
 vgaromimage: file=$BXSHARE/VGABIOS-lgpl-latest  
 vgaromimage: file=$BXSHARE/VGABIOS-lgpl-latest-cirrus

This tells Bochs what VGA ROM BIOS to load (at 0xC0000).

A VGA BIOS from Elpin Systems, Inc. as well as a free LGPL'd VGA BIOS are provided in the source and binary distributions.

**Note:**  Please check with the [vga option](http://docs.google.com/bochsrc.html#BOCHSOPT-VGA) to decide what VGA BIOS to use.

4.3.10. optromimage1, optromimage2, optromimage3 or optromimage4

Example:

optromimage1: file=optionalrom.bin, address=0xd0000

This enables Bochs to load up to 4 optional ROM images.

Be sure to use a read-only area, typically between C8000 and EFFFF. These optional ROM images should not overwrite the rombios (located at F0000-FFFFF) and the videobios (located at C0000-C7FFF).

Those ROM images will be initialized by the BIOS if they contain the right signature (0x55AA).

It can also be a convenient way to upload some arbitrary code/data in the simulation, that can be retrieved by the boot loader

4.3.11. vga

Examples:

vga: extension=cirrus, update\_freq=10, realtime=1  
 vga: extension=vbe

This defines parameters related to the VGA display

The 'extension' option can be used to specify the VGA display extension. With the value 'none' you can use standard VGA with no extension. Other supported values are 'vbe' for Bochs VBE (needs VGABIOS-lgpl-latest as VGA BIOS, see [vgaromimage option](http://docs.google.com/bochsrc.html#BOCHSOPT-VGAROMIMAGE)) and 'cirrus' for Cirrus SVGA support (needs VGABIOS-lgpl-latest-cirrus as VGA BIOS).

The VGA update frequency specifies the number of display updates per second. This parameter can be changed at runtime. The default value is 5.

The 'realtime' option specifies the operation mode of the VGA update timer. If set to 1, the VGA timer is based on realtime, otherwise it is based on the ips setting. If the host is slow (low ips, update\_freq) and the guest uses HLT appropriately, setting this to 0 and "clock: sync=none" may improve the responsiveness of the guest GUI when the guest is otherwise idle. The default value is 1.

4.3.12. voodoo

Example:

voodoo: enabled=1, model=voodoo1

This defines the Voodoo Graphics emulation (experimental). Currently supported models are 'voodoo1' and 'voodoo2'. The Voodoo2 support is not yet complete.

4.3.13. keyboard

Examples:

keyboard: type=mf, serial\_delay=200, paste\_delay=100000  
 keyboard: keymap=gui/keymaps/x11-pc-de.map  
 keyboard: user\_shortcut=ctrl-alt-del

This defines parameters related to the emulated keyboard.

**type**

Type of keyboard return by a "identify keyboard" command to the keyboard controller. It must be one of "xt", "at" or "mf". Defaults to "mf". It should be ok for almost everybody. A known exception is french macs, that do have a "at"-like keyboard.

**serial\_delay**

Approximate time in microseconds that it takes one character to be transferred from the keyboard to controller over the serial path.

**paste\_delay**

Approximate time in microseconds between attempts to paste characters to the keyboard controller. This leaves time for the guest os to deal with the flow of characters. The ideal setting depends on how your operating system processes characters. The default of 100000 usec (.1 seconds) was chosen because it works consistently in Windows.

If your OS is losing characters during a paste, increase the paste delay until it stops losing characters.

**keymap**

This enables a remap of a physical localized keyboard to a virtualized us keyboard, as the PC architecture expects.

Keyboard mapping is available for the display libraries x, sdl (Linux port) and wx (GTK port). For SDL you have to use keymaps designed for SDL, the wxWidgets GUI uses the keymaps for X11.

**user\_shortcut**

This defines the keyboard shortcut to be sent when you press the "user" button in the [headerbar](http://docs.google.com/textconfig.html#HEADERBAR). The shortcut string is a combination of maximum 3 key names (listed below) separated with a '-' character.

Valid key names:

"alt", "bksl", "bksp", "ctrl", "del", "down", "end", "enter", "esc", "f1", ... "f12", "home", "ins", "left", "menu", "minus", "pgdwn", "pgup", "plus", "power", "print", "right", "scrlck", "shift", "space", "tab", "up" and "win".

4.3.14. mouse

Examples:

mouse: enabled=1  
 mouse: type=imps2, enabled=1  
 mouse: type=serial, enabled=1  
 mouse: enabled=0, toggle=ctrl+f10

This defines parameters for the emulated mouse type, the initial status of the mouse capture and the runtime method to toggle it.

**type**

With the mouse type option you can select the type of mouse to emulate. The default value is 'ps2'. The other choices are 'imps2' (wheel mouse on PS/2), 'serial', 'serial\_wheel', 'serial\_msys' (one com port requires setting 'mode=mouse', see [com option](http://docs.google.com/bochsrc.html#BOCHSOPT-COM)) and 'bus' (if present). To connect a mouse to an USB port, see the [usb\_uhci](http://docs.google.com/bochsrc.html#BOCHSOPT-USB-UHCI), 'usb\_ohci 'or 'usb\_xhci' option (requires PCI and USB support).

**enabled**

The Bochs gui creates mouse "events" unless the 'enabled' option is set to 0. The hardware emulation itself is not disabled by this. Unless you have a particular reason for enabling the mouse by default, it is recommended that you leave it off. You can also toggle the mouse usage at runtime (see [headerbar](http://docs.google.com/textconfig.html#HEADERBAR) and the 'toggle' option below).

**toggle**

The default method to toggle the mouse capture at runtime is to press the CTRL key and the middle mouse button ('ctrl+mbutton'). This option allows to change the method to 'ctrl+f10' (like DOSBox) or 'ctrl+alt' (like QEMU) or 'f12' (replaces win32 'legacyF12' option).

4.3.15. pci

Examples:

pci: enabled=1, chipset=i440fx # default if compiled with PCI support  
 pci: enabled=1, chipset=i440fx, slot1=pcivga, slot2=ne2k

This option controls the presence of a PCI chipset in Bochs. Currently it only supports the i430FX and i440FX chipsets. You can also specify the devices connected to PCI slots. Up to 5 slots are available. For these combined PCI/ISA devices assigning to slot is mandatory if you want to emulate the PCI model: cirrus, ne2k and pcivga. These PCI-only devices are also supported, but they are auto-assigned if you don't use the slot configuration: e1000, es1370, pcidev, pcipnic, usb\_ohci and usb\_xhci.

4.3.16. clock

This defines the parameters of the clock inside Bochs:

**sync**

This defines the method how to synchronize the Bochs internal time with realtime. With the value 'none' the Bochs time relies on the IPS value and no host time synchronization is used. The 'slowdown' method sacrifices performance to preserve reproducibility while allowing host time correlation. The 'realtime' method sacrifices reproducibility to preserve performance and host-time correlation. It is possible to enable both synchronization methods.

**rtc\_sync**

If this option is enabled together with the realtime synchronization, the RTC runs at realtime speed. This feature is disabled by default.

**time0**

Specifies the start (boot) time of the virtual machine. Use a time value as returned by the time(2) system call or a string as returned by the ctime(3) system call. If no time0 value is set or if time0 equal to 1 (special case) or if time0 equal 'local', the simulation will be started at the current local host time. If time0 equal to 2 (special case) or if time0 equal 'utc', the simulation will be started at the current utc time.

Syntax:  
 clock: sync=[none|slowdown|realtime|both], time0=[timeValue|local|utc]  
  
Examples:  
 clock: sync=none, time0=local # Now (localtime)  
 clock: sync=slowdown, time0=315529200 # Tue Jan 1 00:00:00 1980  
 clock: sync=none, time0="Mon Jan 1 00:00:00 1990" # 631148400  
 clock: sync=realtime, time0=938581955 # Wed Sep 29 07:12:35 1999  
 clock: sync=realtime, time0="Sat Jan 1 00:00:00 2000" # 946681200  
 clock: sync=none, time0=1 # Now (localtime)  
 clock: sync=none, time0=utc # Now (utc/gmt)  
  
Default value are sync=none, rtc\_sync=0, time0=local

4.3.17. cmosimage

Example:

cmosimage: file=cmos.img, rtc\_init=time0

This defines image file that can be loaded into the CMOS RAM at startup. The rtc\_init parameter controls whether initialize the RTC with values stored in the image. By default the time0 argument given to the [clock option](http://docs.google.com/bochsrc.html#BOCHSOPT-CLOCK) is used. With 'rtc\_init=image' the image is the source for the initial time.

4.3.18. private\_colormap

Example:

private\_colormap: enabled=1

Requests that the GUI creates and uses its own non-shared colormap. This colormap will be used when in the Bochs window. If not enabled, a shared colormap scheme may be used. Once again, enabled=1 turns on this feature and 0 turns it off.

4.3.19. floppya/floppyb

Examples:

2.88M 3.5" media:  
 floppya: 2\_88=a:, status=inserted  
1.44M 3.5" media (write protected):  
 floppya: 1\_44=floppya.img, status=inserted, write\_protected=1  
1.2M 5.25" media:  
 floppyb: 1\_2=/dev/fd0, status=inserted  
720K 3.5" media:  
 floppya: 720k=/usr/local/bochs/images/win95.img, status=inserted  
auto-detect floppy media type:  
 floppya: image=floppy.img, status=inserted  
use directory as VFAT media:  
 floppya: 1\_44=vvfat:path, status=inserted  
1.44M 3.5" floppy drive, no media:  
 floppya: type=1\_44

Floppya is the first drive, and floppyb is the second drive. If you're booting from a floppy, floppya should point to a bootable disk. To read from a disk image, write the name of the image file. In many operating systems Bochs can read directly from a raw floppy drive. For raw disk access, use the device name (Unix systems) or the drive letter and a colon (Windows systems).

Following floppy media types are supported: 2\_88, 1\_44, 1\_2, 720k, 360k, 320k, 180k, 160k, as well as "image" to let Bochs auto-detect the type of floppy media (does only work with images, not with raw floppy drives). In that case the size must match one of the supported types.

You can set the initial status of the media to ejected or inserted. Usually you will want to use inserted.

The parameter 'type' can be used to enable the floppy drive without media and status specified. Usually the drive type is set up based on the media type.

The optional parameter 'write\_protected' can be used to control the media write protect switch. By default it is turned off.

4.3.20. ata0, ata1, ata2, ata3

Examples:

ata0: enabled=1, ioaddr1=0x1f0, ioaddr2=0x3f0, irq=14  
ata1: enabled=1, ioaddr1=0x170, ioaddr2=0x370, irq=15  
ata2: enabled=1, ioaddr1=0x1e8, ioaddr2=0x3e0, irq=11  
ata3: enabled=1, ioaddr1=0x168, ioaddr2=0x360, irq=9

These options enables up to 4 ata channels. For each channel the two base io addresses and the irq must be specified. ata0 and ata1 are enabled by default, with the values shown above.

4.3.21. ata0-master, ata0-slave, ata1-\*, ata2-\*, ata3-\*

Examples:

ata0-master: type=disk, path=10M.img, mode=flat, cylinders=306, heads=4, spt=17, translation=none  
ata1-master: type=disk, path=2GB.cow, mode=vmware3, cylinders=5242, heads=16, spt=50, translation=echs  
ata1-slave: type=disk, path=3GB.img, mode=sparse, cylinders=6541, heads=16, spt=63, translation=auto  
ata2-master: type=disk, path=7GB.img, mode=undoable, cylinders=14563, heads=16, spt=63, translation=lba  
ata2-slave: type=cdrom, path=iso.sample, status=inserted

This defines the type and characteristics of all attached ata devices:

**Table 4-4. ata devices configuration options**

|  |  |  |
| --- | --- | --- |
| Option | Comments | Possible values |
| type | type of attached device | [disk | cdrom] |
| path | path of the image |  |
| mode | image type, only valid for disks | [flat | concat | external | dll | sparse | vmware3 | vmware4 | undoable | growing | volatile | vpc | vbox | vvfat ] |
| cylinders | only valid for disks |  |
| heads | only valid for disks |  |
| spt | only valid for disks |  |
| status | only valid for cdroms | [inserted | ejected] |
| biosdetect | type of biosdetection | [none | auto], only for disks on ata0 [cmos] |
| translation | type of translation done by the BIOS (legacy int13), only for disks | [none | lba | large | rechs | auto] |
| model | string returned by identify device ATA command |  |
| journal | optional filename of the redolog for undoable, volatile and vvfat disks |  |

You have to tell the type of the attached device. For Bochs 2.0 or later, it can be disk or cdrom.

You have to point the "path" at a hard disk image file, cdrom iso file, or physical cdrom device. To create a hard disk image, try running **bximage** (see [Section 8.2](http://docs.google.com/diskimagehowto.html)). It will help you choose the size and then suggest a line that works with it.

In Unix it is possible to use a raw device as a Bochs hard disk, but *we don't recommend it* for safety reasons. In Windows, there is no easy way.

Disk geometry autodetection works with images created by bximage if CHS is set to 0/0/0 (cylinders are calculated using heads=16 and spt=63). For other hard disk images and modes the cylinders, heads, and spt are mandatory. In all cases the disk size reported from the image must be exactly C\*H\*S\*512. Flat hard disk images from other projects might store additional information at the end of the file that makes this check fail. Only in this case it is safe to select "continue" when Bochs panics.

The disk translation scheme (implemented in legacy int13 BIOS functions, and used by older operating systems like MS-DOS), can be defined as:

* none : no translation, for disks up to 528MB (1032192 sectors)
* large : a standard bitshift algorithm, for disks up to 4.2GB (8257536 sectors)
* rechs : a revised bitshift algorithm, using a 15 heads fake physical geometry, for disks up to 7.9GB (15482880 sectors). (don't use this unless you understand what you're doing)
* lba : a standard lba-assisted algorithm, for disks up to 8.4GB (16450560 sectors)
* auto : autoselection of best translation scheme. (it should be changed if system does not boot)

Please see [Section 8.16.2](http://docs.google.com/bios-tips.html#BIOS-DISK-TRANSLATION) for a discussion on translation scheme.

The mode option defines how the disk image is handled. Disks can be defined as:

* flat : one file flat layout
* concat : multiple files layout
* external : developer's specific, through a C++ class
* dll : developer's specific, through a DLL
* sparse : stackable, commitable, rollbackable
* vmware3 : vmware version 3 disk support
* vmware4 : vmware version 4 disk support (aka VMDK)
* undoable : read-only base file with commitable redolog
* growing : growing file
* volatile : read-only base file with volatile redolog
* vpc: fixed / dynamic size VirtualPC image
* vbox: fixed / dynamic size Oracle(tm) VM VirtualBox image (VDI version 1.1)
* vvfat: local directory appears as VFAT disk (with volatile redolog / optional commit)

Please see [Section 8.20](http://docs.google.com/harddisk-modes.html) for a discussion on disk modes.

Default values are:

mode=flat, biosdetect=auto, translation=auto, model="Generic 1234"

The biosdetect option has currently no effect on the BIOS.

**Note:**  Make sure the proper [ata option](http://docs.google.com/bochsrc.html#BOCHSOPT-ATA) is enabled when using a device on that ata channel.

4.3.22. boot

Examples:

boot: floppy  
 boot: cdrom, disk  
 boot: network, disk  
 boot: cdrom, floppy, disk

This defines the boot sequence. You can specify up to 3 boot drives, which can be 'floppy', 'disk', 'cdrom' or 'network' (boot ROM). Legacy 'a' and 'c' are also supported.

4.3.23. floppy\_bootsig\_check

Example:

floppy\_bootsig\_check: disabled=1

This disables the 0xaa55 signature check on boot floppies The check is enabled by default.

4.3.24. log

Examples:

log: bochsout.txt  
 log: -  
 log: /dev/tty (Unix only)  
 log: /dev/null (Unix only)  
 log: nul (win32 only)

Give the path of the log file you'd like Bochs debug and misc. verbiage to be to be written to. If you don't use this option or set the filename to '-' the output is written to the console. If you really don't want it, make it "/dev/null" (Unix) or "nul" (win32). :^(

4.3.25. logprefix

Examples:

logprefix: %t-%e-@%i-%d  
 logprefix: %i%e%d

This handles the format of the string prepended to each log line. You may use those special tokens :

%t : 11 decimal digits timer tick  
 %i : 8 hexadecimal digits of current cpu eip (ignored in SMP configuration)  
 %e : 1 character event type ('i'nfo, 'd'ebug, 'p'anic, 'e'rror)  
 %d : 5 characters string of the device, between brackets

Default is %t%e%d

4.3.26. debug/info/error/panic

Examples:

debug: action=ignore, pci=report  
 info: action=report  
 error: action=report  
 panic: action=ask

During simulation, Bochs encounters certain events that the user might want to know about. These events are divided into four levels of importance: debug, info, error, and panic. Debug messages are usually only useful when writing Bochs code or when trying to locate a problem. There may be thousands of debug messages per second, so be careful before turning them on. Info messages tell about interesting events that don't happen that frequently. Bochs produces an "error" message when it finds a condition that really shouldn't happen, but doesn't endanger the simulation. An example of an error might be if the emulated software produces an illegal disk command. Panic messages mean that Bochs cannot simulate correctly and should probably shut down. A panic can be a configuration problem (like a misspelled bochsrc line) or an emulation problem (like an unsupported video mode).

The debug, info, error, and panic lines in the bochsrc control what Bochs will do when it encounters each type of event. The allowed actions are: fatal (terminate bochs), ask (ask the user what to do), report (print information to the console or log file), or ignore (do nothing). The recommended settings are listed in the sample above.

It is also possible to specify the 'action' to do for each Bochs facility separately (e.g. crash on panics from everything except the cdrom, and only report those). See the [log function module table](http://docs.google.com/howto.html#LOGOPTS-BY-DEVICE) for valid module names.

**Tip:** The safest action for panics is "fatal" or "ask". If you are getting lots of panics and get tired of telling it to continue each time, you can try action=report instead. If you allow Bochs to continue after a panic, don't be surprised if you get strange behavior or crashes after a panic occurs. Please report panic messages to the [bochs-developers mailing list](http://docs.google.com/mailinglist.html#BOCHS-DEVELOPERS) unless it is just a configuration problem like "could not find hard drive image."

4.3.27. debugger\_log

Examples:

debugger\_log: debugger.out  
 debugger\_log: /dev/null (Unix only)  
 debugger\_log: -

Give the path of the log file you'd like Bochs to log debugger output. If you really don't want it, make it '/dev/null', or '-'.

4.3.28. com[1-4]

Examples:

com1: enabled=1, mode=null  
 com1: enabled=1, mode=mouse  
 com1: enabled=1, mode=term, dev=/dev/ttyp9  
 com2: enabled=1, mode=file, dev=serial.out  
 com3: enabled=1, mode=raw, dev=com1  
 com3: enabled=1, mode=socket-client, dev=localhost:8888  
 com3: enabled=1, mode=socket-server, dev=localhost:8888  
 com4: enabled=1, mode=pipe-client, dev=\\.\pipe\mypipe  
 com4: enabled=1, mode=pipe-server, dev=\\.\pipe\mypipe

This defines a serial port (UART type 16550A).

When using the mode 'term', you can specify a device to use as com1. This can be a real serial line, or a pty. To use a pty (under X/Unix), create two windows (xterms, usually). One of them will run Bochs, and the other will act as com1. Find out the tty of the com1 window using the `tty' command, and use that as the `dev' parameter. Then do `sleep 1000000' in the com1 window to keep the shell from messing with things, and run Bochs in the other window. Serial I/O to com1 (port 0x3f8) will all go to the other window.

When using socket\* and pipe\* (win32 only) modes Bochs becomes either socket/named pipe client or server. In client mode it connects to an already running server (if connection fails Bochs treats com port as not connected). In server mode it opens socket/named pipe and waits until a client application connects to it before starting simulation. This mode is useful for remote debugging (e.g. with gdb's "target remote host:port" command or windbg's command line option -k com:pipe,port=\\.\pipe\pipename). Socket modes use simple TCP communication, pipe modes use duplex byte mode pipes.

Other serial modes are 'null' (no input/output), 'file' (output to a file specified as the 'dev' parameter), 'raw' (use the real serial port - under construction for win32), 'mouse' (standard serial mouse - requires [mouse option](http://docs.google.com/bochsrc.html#BOCHSOPT-MOUSE) setting 'type=serial' or 'type=serial\_wheel').

4.3.29. parport[1-2]

Examples:

parport1: enabled=1, file="parport.out"  
 parport2: enabled=1, file="/dev/lp0"  
 parport1: enabled=0

This defines a parallel (printer) port. When turned on and an output file is defined, the emulated printer port sends characters printed by the guest OS into the output file. On some platforms, a device filename can be used to send the data to the real parallel port (e.g. "/dev/lp0" on Linux, "lpt1" on win32 platforms).

4.3.30. sound

Example for one driver (uses platform-default):

sound: driver=default, waveout=/dev/dsp

Example for different drivers:

sound: waveoutdrv=sdl, waveindrv=alsa, midioutdrv=dummy

This defines the lowlevel sound driver(s) for the wave (PCM) input / output and the MIDI output feature and (if necessary) the devices to be used. It can have several of the following properties. All properties are in the format sound: property=value.

* waveoutdrv: This defines the driver to be used for the waveout feature. Possible values are 'file' (all wave data sent to file), 'dummy' (no output) and the platform-dependant drivers 'alsa', 'oss', 'osx', 'sdl' and 'win'.
* waveout: This defines the device to be used for wave output (if necessary) or the output file for the 'file' driver.
* waveindrv: This defines the driver to be used for the wavein feature. Possible values are 'dummy' (recording silence) and platform-dependent drivers 'alsa', 'oss' and 'win'.
* wavein: This defines the device to be used for wave output (if necessary).
* midioutdrv: This defines the driver to be used for the MIDI output feature. Possible values are 'file' (all MIDI data sent to file), 'dummy' (no output) and platform-dependent drivers 'alsa', 'oss', 'osx' and 'win'.
* midiout: This defines the device to be used for MIDI output (if necessary).
* driver: This defines the driver to be used for all sound features with one property. Possible values are 'default' (platform default) and all other choices described above. Overriding one or more settings with the specific driver parameter is possible.

See [Section 5.6](http://docs.google.com/using-sound.html) for more information.

4.3.31. speaker

Example:

speaker: enabled=1, mode=sound

This defines the PC speaker output mode. In the 'sound' mode the beep is generated by the square wave generator which is a part of the lowlevel sound support. The 'system' mode is only available on Linux and Windows. On Linux /dev/console is used for output and on Windows the Beep() function. The 'gui' mode forwards the beep to the related gui methods (currently only used by the Carbon gui).

4.3.32. sb16

Example:

sb16: midimode=2, midifile=output.mid, wavemode=3, wavefile=output.wav  
 loglevel=2, log=sb16.log, dmatimer=600000

**Note:** The example is wrapped onto several lines for formatting reasons, but it should all be on one line in the actual bochsrc file.

This defines the Sound Blaster 16 emulation, see the [developer documentation](http://docs.google.com/development/sb16-emulation-basics.html) for more information. It can have several of the following properties. All properties are in the usual "property=value" format.

* enabled: This optional property controls the presence of the SB16 emulation. The emulation is turned on unless this property is used and set to 0.
* midimode: This parameter specifies what to do with the MIDI output.  
   0 = no output  
   1 = output to device specified with the sound option (system dependent)  
   2 = MIDI or raw data output to file (depends on file name extension)  
   3 = dual output (mode 1 and 2 at the same time)
* midifile: This is the file where the midi output is stored (midimode 2 or 3).
* wavemode: This parameter specifies what to do with the PCM output.  
   0 = no output  
   1 = output to device specified with the sound option (system dependent)  
   2 = VOC, WAV or raw data output to file (depends on file name extension)  
   3 = dual output (mode 1 and 2 at the same time)
* wavefile: This is the file where the wave output is stored (wavemode 2 or 3).
* log: The file to write the sb16 emulator messages to.
* loglevel:  
   0 = No log.  
   1 = Resource changes, midi program and bank changes.  
   2 = Severe errors.  
   3 = All errors.  
   4 = All errors plus all port accesses.  
   5 = All errors and port accesses plus a lot of extra information.  
   It is possible to change the loglevel at runtime.
* dmatimer: Microseconds per second for a DMA cycle. Make it smaller to fix non-continuous sound. 750000 is usually a good value. This needs a reasonably correct setting for the **ips** parameter of the [cpu option](http://docs.google.com/bochsrc.html#BOCHSOPT-CPU-IPS). It is possible to adjust the dmatimer value at runtime.

4.3.33. es1370

Examples:

es1370: enabled=1, wavemode=1 # use 'sound' parameters  
 es1370: enabled=1, wavemode=2, wavefile=output.voc # send output to file

This defines the ES1370 sound emulation (recording and playback - except DAC1+DAC2 output at the same time). The parameter 'enabled' controls the presence of the device. The wave and MIDI output can be sent to device, file or both using the parameters 'wavemode', 'wavefile', 'midimode' and 'midifile'. See the description of these parameters at the SB16 directive.

4.3.34. ne2k

The ne2k line configures an emulated NE2000-compatible Ethernet adapter, which allows the guest machine to communicate on the network. To disable the NE2000 just comment out the ne2k line.

Examples:

ne2k: ioaddr=0x300, irq=9, mac=b0:c4:20:00:00:00, ethmod=fbsd, ethdev=xl0  
ne2k: ioaddr=0x300, irq=9, mac=b0:c4:20:00:00:00, ethmod=fbsd, ethdev=en0 #macosx  
ne2k: ioaddr=0x300, irq=9, mac=b0:c4:20:00:00:00, ethmod=linux, ethdev=eth0  
ne2k: ioaddr=0x300, irq=9, mac=b0:c4:20:00:00:01, ethmod=win32, ethdev=*MYCARD*  
ne2k: ioaddr=0x300, irq=9, mac=b0:c4:20:00:00:01, ethmod=vde, ethdev="/tmp/vde.ctl"  
ne2k: ioaddr=0x300, irq=9, mac=b0:c4:20:00:00:01, ethmod=vnet, ethdev="c:/temp"  
ne2k: ioaddr=0x300, irq=9, mac=fe:fd:00:00:00:01, ethmod=tap, ethdev=tap0  
ne2k: ioaddr=0x300, irq=9, mac=fe:fd:00:00:00:01, ethmod=tuntap, ethdev=/dev/net/tun0, script=./tunconfig  
ne2k: mac=b0:c4:20:00:00:01, ethmod=slirp, script=slirp.conf, bootrom=ne2k\_pci.rom  
  
IOADDR, IRQ: You probably won't need to change ioaddr and irq, unless there  
are IRQ conflicts. These parameters are ignored if the NE2000 is assigned to  
a PCI slot.  
  
MAC: The MAC address MUST NOT match the address of any machine on the net.  
Also, the first byte must be an even number (bit 0 set means a multicast  
address), and you cannot use ff:ff:ff:ff:ff:ff because that's the broadcast  
address. For the ethertap module, you must use fe:fd:00:00:00:01. There may  
be other restrictions too. To be safe, just use the b0:c4... address.  
  
ETHMOD: The ethmod value defines which low level OS specific module to be  
used to access physical ethernet interface. You can also specify a network  
simulator or a module with no input/output ("null"). See the table below for  
currently supported values.  
  
ETHDEV: The ethdev value is the name of the network interface on your host  
platform. On UNIX machines, you can get the name by running ifconfig. On  
Windows machines, you must run niclist to get the name of the ethdev.  
Niclist source code is in misc/niclist.c and it is included in Windows  
binary releases.  
  
SCRIPT: The script value is optional, and is the name of a script that  
is executed after bochs initialize the network interface. You can use  
this script to configure this network interface, or enable masquerading.  
This is mainly useful for the tun/tap devices that only exist during  
Bochs execution. The network interface name is supplied to the script  
as first parameter.  
  
BOOTROM: The bootrom value is optional, and is the name of the ROM image  
to load. Note that this feature is only implemented for the PCI version of  
the NE2000.

The following table shows the available ethernet modules with description, whether the "ethdev" and "script" parameters are used or not and the Bochs version where this module was added.

**Table 4-5. Ethernet modules**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module | Description | ethdev | script | Bochs version |
| fbsd | FreeBSD / OpenBSD packetmover. | Yes | No | 1.0 |
| linux | Linux packetmover - 'root' privileges required, no connection to the host machine. | Yes | No | 1.3 |
| null | Null packetmover. All packets are discarded, but logged to a few files. | No | No | 1.0 |
| tap | TAP packetmover. | Yes | Yes | 1.4 |
| tuntap | TUN/TAP packetmover - see  [Configuring and using a tuntap network interface](http://docs.google.com/config-tuntap.html). | Yes | Yes | 2.0 |
| vde | Virtual Distributed Ethernet packetmover. | Yes | Yes | 2.2 |
| vnet | ARP, ping (ICMP-echo), DHCP and read/write TFTP simulation. The virtual host uses 192.168.10.1. DHCP assigns 192.168.10.2 to the guest. The TFTP server uses the 'ethdev' value for the root directory and doesn't overwrite files. | Yes, for TFTP | No | 2.2 |
| slirp | Built-in Slirp support with DHCP / TFTP servers. Adds user mode networking to Bochs - see [Using the 'slirp' networking module](http://docs.google.com/using-slirp.html). The 'script' parameter can be used to set up an alternative IP configuration or additional features. The TFTP server uses the 'ethdev' value for the root directory and doesn't overwrite files. | Yes, for TFTP | Yes, for Slirp config | 2.6.5 |
| win32 | Win32 packetmover - WinPCap driver required. | Yes | No | 1.3 |

4.3.35. pcipnic

Example:

pcipnic: enabled=1, mac=b0:c4:20:00:00:00, ethmod=vnet

To support the Bochs/Etherboot pseudo-NIC, Bochs must be compiled with the --enable-pnic configure option. It accepts the same syntax (for mac, ethmod, ethdev, script, bootrom) and supports the same networking modules as the NE2000 adapter.

4.3.36. e1000

Example:

e1000: enabled=1, mac=52:54:00:12:34:56, ethmod=slirp, script=slirp.conf

To support the Intel(R) 82540EM Gigabit Ethernet adapter, Bochs must be compiled with the --enable-e1000 configure option. It accepts the same syntax (for mac, ethmod, ethdev, script, bootrom) and supports the same networking modules as the NE2000 adapter.

4.3.37. usb\_uhci

Examples:

usb\_uhci: enabled=1, port1=mouse, port2=disk:usbstick.img  
 usb\_uhci: enabled=1, port1=hub:7, port2=disk:growing:usbdisk.img  
 usb\_uhci: enabled=1, port2=disk:undoable:usbdisk.img, options2=journal:redo.log  
 usb\_uhci: enabled=1, port2=disk:vvfat:vvfat, options2=speed:full  
 usb\_uhci: enabled=1, port1=printer:printdata.bin, port2=cdrom:image.iso

This option controls the presence of the USB root hub which is a part of the i440FX PCI chipset.

With the port*X* option you can connect devices to the hub (currently supported: 'mouse', 'tablet', 'keypad', 'disk', 'cdrom', 'hub' and 'printer').

If you connect the mouse or tablet to one of the ports, Bochs forwards the mouse movement data to the USB device instead of the selected mouse type. When connecting the keypad to one of the ports, Bochs forwards the input of the numeric keypad to the USB device instead of the PS/2 keyboard.

To connect a 'flat' mode image as an USB hardisk you can use the 'disk' device with the path to the image separated with a colon. To use other disk image modes similar to ATA disks the syntax 'disk:mode:filename' must be used (see above).

To emulate an USB cdrom you can use the 'cdrom' device name and the path to an ISO image or raw device name also separated with a colon. An option to insert/eject media is available in the runtime configuration.

The device name 'hub' connects an external hub with max. 8 ports (default: 4) to the root hub. To specify the number of ports you have to add the value separated with a colon. Connecting devices to the external hub ports is only available in the runtime configuration.

The device 'printer' emulates the HP Deskjet 920C printer. The PCL data is sent to a file specified in bochsrc.txt. The current code appends the PCL code to the file if the file already existed. It would probably be nice to overwrite the file instead, asking user first.

The options*X* parameter can be used to assign specific options to the device connected to the corresponding USB port. Currently this feature is used to set the speed reported by device ('low', 'full', 'high' or 'super'). The availabe speed choices depend on both HC and device. For the USB 'disk' device the optionsX parameter can be used to specify an alternative redolog file (journal) of some image modes. For 'vvfat' mode USB disks the optionsX parameter can be used to specify the disk size (range 128M ... 128G). If the size is not specified, it defaults to 504M.

**Note:** PCI support must be enabled to use USB UHCI.

4.3.38. usb\_ohci

Example:

usb\_ohci: enabled=1, port1=printer:printdata.bin

This option controls the presence of the USB OHCI host controller with a 2-port hub. The portX parameter accepts the same device types with the same syntax as the UHCI controller (see the [usb\_uhci option](http://docs.google.com/bochsrc.html#BOCHSOPT-USB-UHCI)). The optionsX parameter is also available on OHCI.

4.3.39. usb\_xhci

Example:

usb\_xhci: enabled=1

This option controls the presence of the experimental USB xHCI host controller with a 4-port hub. The portX parameter accepts the same device types with the same syntax as the UHCI controller (see the [usb\_uhci option](http://docs.google.com/bochsrc.html#BOCHSOPT-USB-UHCI)). The optionsX parameter is also available on xHCI. NOTE: port 1 and 2 are USB3 and only support super-speed devices, but port 3 and 4 are USB2 and support speed settings low, full and high.

4.3.40. pcidev

Example:

pcidev: vendor=0xbabe, device=0x2bad

Enables the mapping of a host PCI hardware device within the virtual PCI subsystem of the Bochs x86 emulator. The arguments vendor and device should contain the PCI vendor ID respectively the PCI device ID of the host PCI device you want to map within Bochs.

**Note:** The PCI device mapping is still in a very early stage of development and thus it is very experimental. This feature requires Linux as a host operating system.

Besides the pcidev config line you will need to load a pcidev kernel module within your Linux host OS. This kernel module is located in the bochs/host/linux/pcidev/ directory.

4.3.41. gdbstub

Example:

gdbstub: enabled=1, port=1234, text\_base=0, data\_base=0, bss\_base=0

Default:

gdbstub: enabled=0

This enables the GDB stub. See [Section 8.14](http://docs.google.com/debugging-with-gdb.html).

4.3.42. magic\_break

Example:

magic\_break: enabled=1

This enables the "magic breakpoint" feature when using the debugger. The useless cpu instruction XCHG BX, BX causes Bochs to enter the debugger mode. This might be useful for software development.

4.3.43. debug\_symbols

Example:

debug\_symbols: file=mysymbols.sym  
 debug\_symbols: file=mysymbols.sym, offset=0x1000

This loads symbols from the specified file for use in Bochs' internal debugger. Symbols are loaded into global context. This is equivalent to issuing ldsym debugger command at start up.

4.3.44. port\_e9\_hack

Example:

port\_e9\_hack: enabled=1

The 0xE9 port doesn't exists in normal ISA architecture. However, we define a convention here, to display on the console of the system running Bochs anything that is written to it. The idea is to provide debug output very early when writing BIOS or OS code for example, without having to bother with setting up a serial port or etc. Reading from port 0xE9 will will return 0xe9 to let you know if the feature is available. Leave this 0 unless you have a reason to use it.

4.3.45. user\_plugin

Example:

user\_plugin: name=testdev

Load user-defined plugin. This option is available only if Bochs is compiled with plugin support. Maximum 8 different plugins are supported. See the example in the Bochs sources how to write a plugin device.

### Notes

|  |  |
| --- | --- |
| [[1]](http://docs.google.com/bochsrc.html#AEN1723) | IPS measurements depend on OS and compiler configuration in addition to host processor clock speed. |

|  |  |  |
| --- | --- | --- |
| [Prev](http://docs.google.com/rom-images.html) | [Home](http://docs.google.com/index.html) | [Next](http://docs.google.com/keymap.html) |
| ROM images | [Up](http://docs.google.com/setup.html) | How to write your own keymap table |