|  |  |  |
| --- | --- | --- |
| Bochs User Manual | | |
| [Prev](http://docs.google.com/install-binary.html) | Chapter 3. Installation | [Next](http://docs.google.com/setup.html) |

3.4. Compiling Bochs

3.4.1. Standard Compile

Bochs is written in C++, so you need a C++ compiler on your system. Most platforms have been tested with GNU gcc/g++, but other compilers are known to work too. By now, you should have unpacked your source TAR file or checked out Bochs from SVN. If not, you can return to [Downloading Bochs](http://docs.google.com/installation.html#DOWNLOADING) for details. The top level directory of the source code will be referred to as $BOCHS. ($BOCHS contains the files bochs.h and main.cc and subdirectories cpu and bios.)

The standard compile process has three basic steps: **configure**, **make**, and **make install**. Each step is described in a separate section below. The standard compile process is used on all Unix machines, MacOS X, and Cygwin (win32). There are separate instructions for [compiling for Win32 with Microsoft VC++](http://docs.google.com/compiling.html#COMPILING-WIN32).

3.4.1.1. Configure

There is a script called **configure** which tests your machine, C/C++ compiler and libraries to discover what settings should work on your system. If you run **configure** with no arguments after it, defaults will be used for all settings. To change the settings, you can run **configure** with options that override the defaults. You can get a list of valid configure options by typing **configure --help**. One useful configure option is --prefix=*directory*, which sets the directory in which Bochs will be installed. All the possible configure options are documented in a [later section](http://docs.google.com/compiling.html#CONFIG-OPTS).

Among other things, the configure script tries to detect your platform and which compile options to use. If you want to control this, set these environment variables before running configure: CC, CXX, CFLAGS, CXXFLAGS. Here is an example that sets the environment variables, using bash/ksh[[1]](#49x2ik5) syntax:

export CC=egcs  
 export CXX="$CC"  
 export CFLAGS="-Wall -O2 -m486 -fomit-frame-pointer -pipe"  
 export CXXFLAGS="$CFLAGS"

Once the configure script knows what options are selected, it creates a Makefile in every source code directory, and creates $BOCHS/config.h with all the option values written as preprocessor #defines. Now the sources are ready to compile.

3.4.1.1.1. Configure Shortcut Scripts

In the Bochs source directory, you will see a series of scripts called .conf.*platform*. These scripts run the **configure** script for you, with a set of options that are appropriate for that platform. It is not necessary to use the shortcut scripts; they are simply there to show you an example that the developers have used. Some of these scripts have been used to build official binary packages.

**Tip:** If a shortcut script is "almost right" for you, just edit it and then run it! If you run a shortcut script, you don't need to run configure manually.

Run a shortcut script using Bourne shell, like this:

sh .conf.win32-vcpp

These .conf.*platform* have been tested in recent Bochs versions:

.conf.linux  
 .conf.sparc  
 .conf.macos  
 .conf.macosx  
 .conf.win32-vcpp  
 .conf.win32-cygwin

3.4.1.2. Make

The **make** command compiles Bochs. Make is a program used by many software projects that reads the Makefile in each source directory and follows the instructions that it finds there. A Makefile tells which files depend on which other files, what commands to use to compile and link the code, and more. After you have finished the configure step, just type **make** to build the source code.

The reason that make is so popular is that it is smart about when to compile and when not to compile. If you run make once, it compiles every file. But when you run it again, it checks to see if any source files have been modified; if not, there's nothing to do! For example, the Makefile says that main.o depends on main.cc. Knowing this, it will only compile main.cc if it is newer than main.o.

Of course, make can only do the right thing if the Makefile lists all the dependencies correctly, so human error can sometimes lead make astray. If make refuses to build something that you think it should, or you are getting strange compile errors, try doing **make all-clean** and then **make** again. All-clean means to clean up the compiled files in every subdirectory, while **make clean** means to clean up just the current directory[[2]](#2p2csry). However, it's important to note that **make all-clean** leaves the configuration intact. You do not have to run **configure** again.

If you're really in the mood for cleaning, **make dist-clean** erases all the configuration information too. In theory, after a dist-clean your directory should look much like when you first untarred it or checked it out. There's usually some extra stuff lying around, but the Makefile tries at least to erase any files that it created.

3.4.1.3. Make Install

Once the program has been built, the next step is typically to run **make install** to copy the executables, documentation, and other required files into a public place so that all users can use it. By default the files are copied to some directories in /usr/local. The following tables shows the directories and their contents.

**Table 3-5. Installed files**

|  |  |
| --- | --- |
| bin | binary executables (bochs, bximage) |
| lib/bochs/plugins | plugins (if present) |
| man/man1 | manpages for installed binaries |
| man/man5 | manpage for the config file (bochsrc) |
| share/bochs | BIOS images, VGABIOS images, keymaps |
| share/doc/bochs | HTML docs, license, readme, changes, bochsrc sample |

3.4.1.4. Installing the DLX Linux demo

To download and install the DLX Linux demo distributed with Bochs binary release packages, use these two **make** commands:

make unpack\_dlx  
 make install\_dlx

The package will be downloaded from the Bochs website and installed at same location as the files of the Bochs base system.

3.4.2. Configure Options

This section describes the configure options for Bochs. Perhaps the most important option is --help, since it gives you a list of all the other options. The configure script will detect your platform and choose the default GUI for your platform. If the default choice is not what you want, use the --with-\* options to override the default. The options in the first table tell which GUI library is the default for each platform. Starting in version 2.0, you can use multiple --with-\* options at once to compile with multiple display libraries, and then choose between them at runtime with the [display\_library option](http://docs.google.com/bochsrc.html#BOCHSOPT-DISPLAYLIBRARY) in the configuration file. Or, you can let the configure script detect which libraries are on your system and use them all, by configuring with --with-all-libs.

**Note:** The concept of platform detection and default GUIs was added in Bochs 1.4. In Bochs 1.3 and before, the X11 GUI was always the default.

**Table 3-6. Defaults by Platform**

|  |  |  |
| --- | --- | --- |
| Platform | Default GUI | Extra compile flags |
| win32, Cygwin or MinGW/MSYS | --with-win32 | If using nmake method, compile using cl /nologo /MT /W3 /EHs-c- /DNDEBUG /DWIN32 /D\_WINDOWS /D\_CRT\_SECURE\_NO\_WARNINGS. If using Visual C++ workspace, see the workspace file for compile settings. See [Compiling on Win32 with Microsoft VC++](http://docs.google.com/compiling.html#COMPILING-WIN32) for instructions. |
| MacOS X or Darwin | --with-carbon | -fpascal-strings -fno-common -arch ppc -Wno-four-char-constants -Wno-unknown-pragmas -Dmacintosh |
| MacOS 9 or before | --with-macos | none |
| AmigaOS | --with-amigaos | none |
| any other platform | --with-x11 | none |

**Table 3-7. Configure Options to Select the Display Library (optional)**

|  |  |
| --- | --- |
| Option | Comments |
| --with-x11 | Use X windows user interface. On many operating systems, Bochs will use X windows by default. |
| --with-win32 | Use the native Win32 GUI. This is the default on win32 platforms. |
| --with-carbon | Compile for MacOS X with the Carbon GUI. See the .conf.macosx file for the correct MacOS X compile options. |
| --with-amigaos | Compile for Amiga MorphOS. This code is written by Nicholai Benalal. |
| --with-rfb | Enable support for the RFB protocol to talk to AT&T's [VNC Viewer](http://www.realvnc.com/). Refer to [Section 3.4.9](http://docs.google.com/compiling.html#COMPILE-RFB) for details. |
| --with-vncsrv | Enable support for an extended RFB(VNC) GUI using the LibVNCServer library. Refer to [Section 3.4.10](http://docs.google.com/compiling.html#COMPILE-VNCSRV) for details. |
| --with-sdl | Enable support for the SDL 1.2.x GUI interface; see [Section 3.4.11](http://docs.google.com/compiling.html#COMPILE-SDL). |
| --with-sdl2 | Enable support for the SDL 2.x GUI interface; see [Section 3.4.12](http://docs.google.com/compiling.html#COMPILE-SDL2). |
| --with-term | Use text-only gui with curses library. Almost certainly won't work right with the debugger or the textconfig interface. |
| --with-macos | Use Macintosh/CodeWarrior environment. This is for running configure on a platform which supports running configure, so that you may then transfer the configured code over to the real compile environment. |
| --with-wx | Enable support for wxWidgets configuration and display interface; see [Section 3.4.13](http://docs.google.com/compiling.html#COMPILE-WX). |
| --with-svga | Use SVGALIB library for Linux. This allows a full-screen text and graphics display without X windows. The SVGALIB port was written by Igor Popik. |
| --with-nogui | No native GUI; just use blank stubs. This is if you don't care about having video output, but are just running tests. |
| --with-all-libs | Automatically detect which libraries are installed on your system and enable them. This option is still experimental; it might enable libraries that are not usable and cause the compile to fail. If you have trouble, just list the --with-\* options for the specific display libraries that you want. |

The remaining options can generally be used with any GUI. For each option such as --enable-cdrom, you can also write --disable-cdrom to explicitly turn it off.

**Table 3-8. Configure Options**

|  |  |  |
| --- | --- | --- |
| Option | Default | Comments |
| --enable-cpu-level={3,4,5,6} | 6 | Select which CPU level to emulate. Choices are 3,4,5,6 which mean to target 386, 486, Pentium, or Pentium Pro and later emulation. |
| --enable-smp | no | Compile in support for SMP simulation. This allows you to boot Linux and maybe other OSes in SMP mode, and bochs will simulate all the different CPUs and communication between them. Do not expect this option to speed up your simulation! On the contrary, it has to spend extra time simulating the different CPUs (even if they're mostly idle) and the communication between them. Use it to try out an SMP OS if you don't have an SMP machine, or to debug SMP OS drivers. Refer to [Section 8.9](http://docs.google.com/smp.html) for more details on SMP in Bochs. |
| --enable-fpu | yes | If you want to compile Bochs to make use of the FPU emulator written by Stanislav Shwartsman, use this option. |
| --enable-3dnow | no | 3DNow! instruction set support (incomplete) |
| --enable-x86-64 | no | Add support for x86-64 instruction set, written by Peter Tattam and Stanislav Shwartsman. |
| --enable-vmx | no | Intel Virtualization Extensions (VMX) support |
| --enable-svm | no | AMD SVM (Secure Virtual Machine) extensions emulation support |
| --enable-avx | no | AVX instructions support |
| --enable-x86-debugger | no | X86 debugger support. If the software you run in bochs needs to use the x86 hardware debugging facilities such as DR0..DR7, instruction and data breakpoints etc., then you should use this option. Otherwise don't use it, as it will slow down the emulation. |
| --enable-monitor-mwait | no | MONITOR/MWAIT instructions support (still experimental) |
| --enable-alignment-check | yes if cpu level >= 4 | support for alignment check in the CPU and #AC exception |
| --enable-misaligned-sse | no | support for AMD's misaligned SSE extension |
| --enable-configurable-msrs | no | support for user configurations of emulated MSR registers (see example in msrs.def) |
| --enable-long-phy-address | no | support for guest physical address larger than 32 bit |
| --enable-a20-pin | yes | support for the A20 pin |
| --enable-large-ramfile | yes | support for guest memory larger than the host supports |
| --enable-cdrom | yes | Enable use of a real CDROM/DVD drive. The cdrom emulation and the portable ISO image file support are always present. You can use this option to compile in support for accessing the media in your workstation's cdrom drive. The supported platforms are Linux, Solaris, FreeBSD, OpenBSD, MacOS X and Windows. For other platforms, a small amount of code specific to your platform must be written. |
| --enable-sb16 | no | Enable Sound Blaster emulation. The available lowlevel sound interfaces are autodetected. See section [Using Sound](http://docs.google.com/using-sound.html) for supported platforms and more info. This option also enables the standard PC gameport which is a part of the SB16. If you don't want to use it, you might use --disable-gameport. |
| --enable-es1370 | no | Enable ES1370 sound emulation. Just like the SB16 option, the available lowlevel sound interfaces are autodetected and the gameport is turned on. |
| --enable-gameport | no | Enables the standard PC gameport. This option is only necessary if you want to have a gameport, but no SB16 or ES1370 (see above). The connection to a real joystick is currently supported on Linux and win32 only. |
| --enable-idle-hack | no | Enables Roland Mainz's experimental idle code, which is intended to keep Bochs from always using 100% of CPU time. When Bochs is waiting around for an interrupt, the idle code uses a select() loop and some X11 tricks to wait until user input arrives. This is designed to keep Bochs responsive while still yielding to other processes when Bochs is idle. It only works with X11 or term GUI. |
| --enable-ne2000 | no | Enable NE2000 network card support. This requires a low-level component to be written for each OS. The NE2000 option is only supported on FreeBSD, OpenBSD, Linux, and Windows 9x/NT/2K/XP. When enabled and configured, the NE2000 device model can talk to any computer on the network EXCEPT FOR the local host. Exception: Under most circumstances, Bochs can talk to the local host on Windows, and in the ethertap interface for Linux. |
| --enable-pnic | no | Enable PCI pseudo NIC (network card) support. |
| --enable-e1000 | no | Enable Intel(R) 82540EM Gigabit Ethernet adapter support. |
| --enable-clgd54xx | no | Enable Cirrus Logic GD54xx (CL-GD5430 ISA or CL-GD5446 PCI) video card support. For more information see [Section 8.19](http://docs.google.com/cirrus-notes.html). |
| --enable-voodoo | no | Enable experimental 3dfx Voodoo Graphics emulation. Currently only the Voodoo1 model is known to work (Voodoo2 is under construction). |
| --enable-plugins | no | Plugins are shared libraries that can be loaded on demand. Example: the serial device is implemented as a plugin. In Unix, the serial plugin is called libbx\_serial.so. When Bochs reads its configuration file, if the serial device is enabled it loads libbx\_serial.so. See the Features section for supported platforms. |
| --enable-repeat-speedups | no | enable support repeated I/O and memory copy speedups |
| --enable-fast-function-calls | no | enable support for fast function calls (gcc on x86 only) |
| --enable-handlers-chaining | no | enable support for handlers chaining optimization |
| --enable-all-optimizations | no | Turn on the enables for all speed optimizations that the developers believe are safe to use: --enable-repeat-speedups, --enable-fast-function-calls, --enable-handlers-chaining. |
| --enable-debugger | no | Compile in support for Bochs internal command-line debugger. This has nothing to do with x86 hardware debug support. It is a more powerful and non-intrusive native debugger. Enabling this will of course slow down the emulation. You only need this option if you know you need it. After you have run **./configure**, you may want to edit config.h to customize the debugger further; see [Section 8.13](http://docs.google.com/internal-debugger.html) for more information. |
| --enable-debugger-gui | yes if debugger is on | Enable support for the gui frontend of the Bochs debugger. This feature is supported on Windows hosts and on hosts with GTK2 installed. |
| --enable-readline | detected by configure | Compile the debugger with the GNU readline library, which gives command line editing and history. |
| --enable-gdb-stub | no | Enable support for the remote GDB stub. See [Using Bochs and the remote GDB stub](http://docs.google.com/debugging-with-gdb.html). NOTE: This feature cannot be used if the Bochs debugger is also enabled. |
| --enable-disasm | yes | Compile in support for built-in disassembler. Bochs has a built-in disassembler, which is useful if you either run the built-in debugger (--enable-debugger), or want disassembly of the current instruction when there is a panic in bochs. |
| --enable-iodebug | yes if debugger is on | Dave Poirier has written an experimental interface to the debugger using I/O ports, so that software running in the guest OS can access features of the debugger. You only want this option if you are developing guest OS code for use in Bochs. In other words, most people don't. Also, it should only be used with --enable-debugger. See the  [developer documentation](http://bochs.sourceforge.net/doc/docbook/development/iodebug.html) for details. |
| --enable-pci | yes | Enable limited i440FX PCI support. This is still incomplete, but usable. |
| --enable-pcidev | no | Enable PCI host device mapping support. This requires --enable-pci to be set as well as Linux 2.4, 2.6 or 3.x as host. |
| --enable-usb | no | Enable i440FX PCI USB support (UHCI). The host controller with 2-port root hub and 6 USB device types are available. |
| --enable-usb-ohci | no | Enable USB OHCI support. The host controller with 2-port root hub is available (USB device types: same as UHCI). |
| --enable-usb-xhci | no | Enable experimental USB xHCI support. The host controller with 4-port root hub is available (USB device types: same as UHCI). |
| --enable-raw-serial | no | Enable support for the serial port emulation to access to the host's serial port. This feature is partly implemented for Windows hosts only. |
| --enable-docbook | detected | Build the docbook documentation in doc/docbook. The configure script will enable this option automatically if you have a program called docbook2html installed. |
| --enable-instrumentation=directory | no | Compile in support for instrumentation. This allows you to collect instrumentation data from bochs as it executes code. You have to create your own instrumentation library and define the instrumentation macros (hooks in bochs) to either call your library functions or not, depending upon whether you want to collect each piece of data. |
| --enable-xpm | yes | Enables the check for the XPM library. This option is only valid if the x display library is enabled (--with-x11). |
| --enable-show-ips | yes | Enables logging of measured IPS, see the **ips** parameter of the [cpu option](http://docs.google.com/bochsrc.html#BOCHSOPT-CPU-IPS). |
| --enable-logging | yes | By default, Bochs generates logfile output that can be configured with bochsrc options and at runtime. Use --disable-logging to turn off this feature completely. |
| --enable-assert-checks | yes if debugger is on | BX\_ASSERT event causes a panic on assertion failure if enabled (logging feature must be enabled, too). |
| --enable-cpp | no | Use .cpp as C++ suffix. Renames all the .cc files to .cpp for use with compilers which want that, like older MS C++ compilers. Don't use this option unless you know you need it. |

3.4.3. Transcript of Successful Compilation

This example shows the output of a compilation and installation on Linux.

user$ **ls -l bochs-2.5.tar.gz**  
 -rw-r--r-- 1 user users 4117999 27. Nov 17:14 bochs-2.5.tar.gz  
 user$ **gunzip -c bochs-2.5.tar.gz | tar -xvf -**  
 bochs-2.5/  
 bochs-2.5/.conf.sparc  
 bochs-2.5/build/  
 bochs-2.5/build/macosx/  
 bochs-2.5/build/macosx/pbdevelopment.plist  
 bochs-2.5/build/macosx/make-dmg.sh  
 .  
 .  
 .  
 bochs-2.5/memory/memory.cc  
 bochs-2.5/memory/misc\_mem.cc  
 bochs-2.5/memory/Makefile.in  
 user$ **cd bochs-2.5**  
 user$ **./configure --enable-cpu-level=6**  
 checking build system type... x86\_64-unknown-linux-gnu  
 checking host system type... x86\_64-unknown-linux-gnu  
 checking target system type... x86\_64-unknown-linux-gnu  
 checking if you are configuring for another platform... no  
 checking for standard CFLAGS on this platform...  
 checking for gcc... gcc  
 .  
 .  
 .  
 config.status: creating build/win32/nsis/bochs.nsi  
 config.status: creating host/linux/pcidev/Makefile  
 config.status: creating config.h  
 config.status: creating ltdlconf.h  
 user$ **make**  
 cd iodev && \  
 make libiodev.a  
 make[1]: Entering directory `/home/volker/Archiv/test/bochs-2.5/iodev'  
 g++ -c -I.. -I./.. -I../instrument/stubs -I./../instrument/stubs -g -O2 -D\_FILE\_OFFSET\_BITS=64 -D\_LARGE\_FILES devices.cc -o devices.o  
 g++ -c -I.. -I./.. -I../instrument/stubs -I./../instrument/stubs -g -O2 -D\_FILE\_OFFSET\_BITS=64 -D\_LARGE\_FILES virt\_timer.cc -o virt\_timer.o  
 g++ -c -I.. -I./.. -I../instrument/stubs -I./../instrument/stubs -g -O2 -D\_FILE\_OFFSET\_BITS=64 -D\_LARGE\_FILES slowdown\_timer.cc -o slowdown\_timer.o  
 g++ -c -I.. -I./.. -I../instrument/stubs -I./../instrument/stubs -g -O2 -D\_FILE\_OFFSET\_BITS=64 -D\_LARGE\_FILES pic.cc -o pic.o  
 .  
 .  
 .  
 echo done  
 done  
 /bin/sh ./libtool --mode=link g++ -o bochs -g -O2 -D\_FILE\_OFFSET\_BITS=64 -D\_LARGE\_FILES -DBX\_SHARE\_PATH='"/usr/local/share/bochs"' -export-dynamic logio.o main.o config.o load32bitOShack.o pc\_system.o osdep.o plugin.o crc.o \  
 iodev/libiodev.a cpu/libcpu.a cpu/cpudb/libcpudb.a \  
 memory/libmemory.a gui/libgui.a \  
 disasm/libdisasm.a \  
 fpu/libfpu.a \  
 -lSM -lICE -lX11 -lXpm -lXrandr \  
 \  
 \  
 \  
 \  
 -lm  
 mkdir .libs  
 g++ -o bochs -g -O2 -D\_FILE\_OFFSET\_BITS=64 -D\_LARGE\_FILES -DBX\_SHARE\_PATH=\"/usr/local/share/bochs\" logio.o main.o config.o load32bitOShack.o pc\_system.o osdep.o plugin.o crc.o -Wl,--export-dynamic iodev/libiodev.a cpu/libcpu.a cpu/cpudb/libcpudb.a memory/libmemory.a gui/libgui.a disasm/libdisasm.a fpu/libfpu.a -lSM -lICE -lX11 -lXpm -lXrandr -lm  
 gcc -c -I. -I./. -Iinstrument/stubs -I./instrument/stubs -g -O2 -D\_FILE\_OFFSET\_BITS=64 -D\_LARGE\_FILES ./misc/bximage.c -o misc/bximage.o  
 /bin/sh ./libtool --mode=link g++ -o bximage -g -O2 -D\_FILE\_OFFSET\_BITS=64 -D\_LARGE\_FILES misc/bximage.o  
 g++ -o bximage -g -O2 -D\_FILE\_OFFSET\_BITS=64 -D\_LARGE\_FILES misc/bximage.o   
 gcc -c -I. -I./. -Iinstrument/stubs -I./instrument/stubs -g -O2 -D\_FILE\_OFFSET\_BITS=64 -D\_LARGE\_FILES ./misc/bxcommit.c -o misc/bxcommit.o  
 /bin/sh ./libtool --mode=link g++ -o bxcommit -g -O2 -D\_FILE\_OFFSET\_BITS=64 -D\_LARGE\_FILES misc/bxcommit.o  
 g++ -o bxcommit -g -O2 -D\_FILE\_OFFSET\_BITS=64 -D\_LARGE\_FILES misc/bxcommit.o   
 user$ **su**  
 root# **make install**  
 cd iodev && \  
 make libiodev.a  
 make[1]: Entering directory `/home/volker/Archiv/test/bochs-2.5/iodev'  
 .  
 .  
 .  
 for i in CHANGES COPYING README TODO; do if test -f $i; then install -m 644 $i /usr/local/share/doc/bochs; else install -m 644 ./$i /usr/local/share/doc/bochs; fi; done  
 rm -f /usr/local/share/doc/bochs/README  
 cat ./build/linux/README.linux-binary ./README > /usr/local/share/doc/bochs/README  
 install -m 644 ./.bochsrc /usr/local/share/doc/bochs/bochsrc-sample.txt  
 root# **exit**  
 user$ \_

3.4.4. Compiling on Win32 with Microsoft VC++

The standard compile uses the configure script, but the Windows platform cannot run the configure script natively. The current solution to this problem is that the Bochs configure script must be run on a different platform that does support shell scripts, with options that cause it to configure for a Win32 platform instead of the native one. Many people have access to a UNIX machine that could run the configure script, or you can use Cygwin or MinGW/MSYS to run the configure script. [[3]](#147n2zr).

Download the Bochs sources on a machine that can run shell scripts. Edit the configure shortcut script .conf.win32-vcpp if you want to adjust the configure options. Then type these commands in the Bochs source directory:

sh .conf.win32-vcpp  
 make win32\_snap

These commands will run the configure step, produce VC++ makefiles and workspace files, and pack it all into a .zip file in the directory above the source directory [[4]](#3o7alnk). The .zip file is all ready to transfer to the target Windows machine to be unzipped and compiled. Or, if you run the sh/make steps in Cygwin, you are already on the target machine so you don't need the .zip file.

When you have the Win32 sources transferred to a Windows machine with VC++, find the workspace file called bochs.sln in the folder "vs2013" and load it in VC++. Choose **Project:Set Active Project** and be sure that "bochs" is selected. Then choose **Build:Build bochs.exe**. This will build all the required libraries (iodev, cpu, etc.) and the auxiliary programs bximage.exe and niclist.exe.

Using workspaces is easy and intuitive, but there is one caveat. The workspaces come directly out of a ZIP file in build/win32/vs2013ex-workspace.zip, and they are not controlled by the configure script. When you compile with certain configure options (e.g. --with-sdl) you need to link with additional libraries. For now you must add them to the VC++ workspace by hand. In version 2.0, we have improved the situation considerably by adding #if...#endif around every optional file and including nearly every Bochs source file in the workspace. This solves the problem of having to manually add source files to the workspace when you turn on configure options such as --enable-debugger. The problem of adding link libraries remains unresolved.

**Tip:** To compile with the Bochs debugger enabled, add --enable-debugger to the configure line in .conf.win32-vcpp before running it. No modifications to the workspace are necessary.

An alternative way to compile is to run nmake.exe in an MS-DOS window. Instead of using the workspace files, nmake uses the Bochs makefiles that are generated by configure. The nmake method is currently used to build the release binaries.

The **make install** doesn't work with nmake yet. Currently it must be run inside of Cygwin or MinGW/MSYS and requires the environment variable *INSTDIR* to be set.

3.4.5. Compiling on Win32 with Cygwin or MinGW/MSYS

Cygwin is a free Unix-like environment for Windows written by Steve Chamberlain and now maintained by RedHat, Inc. You can download it from [www.cygwin.com](http://www.cygwin.com). Because Cygwin supports the configure script and uses GNU gcc/g++, you can use the standard compile process. The configure script should automatically detect Cygwin and add "-mno-cygwin -DWIN32" to the compiler options. You should get a working Bochs if you just type:

configure  
 make

Optionally, you can use the configure shortcut script for Cygwin, .conf.win32-cygwin, instead of running configure directly. If this script is close to what you need, just edit the script and then run it. To use the configure shortcut script and compile in Cygwin, the commands are

sh .conf.win32-cygwin  
 make

To find out the options which are known to work in Cygwin, open the file .conf.win32-cygwin in any text editor/viewer and have a look at the end of that file.

When using gcc 4.7 or newer you need to add the switch *-mno-ms-bitfields* to the CFLAGS, to make sure that hdimage and network structures are packed as expected.

The command *make install* installs the Bochs files in the directory structure of your build environment. To install Bochs into any desired folder you need to use the *install\_win32* target. It requires the environment variable *INSTDIR* to be set.

3.4.6. Compiling on MacOS 9 with CodeWarrior

It is possible that this hasn't been tried since 1999. In theory, you run **sh .conf.macos** on a Unix box to build the makefiles and headers, copy the whole thing over to a Mac, and then use CodeWarrior to compile. Since it hasn't been tested in so long, it is quite likely that some work is needed to bring the Mac port up to date.

If you are interested and have the required MacOS development tools, please let us know by contacting the [bochs-developers mailing list](http://docs.google.com/mailinglist.html#BOCHS-DEVELOPERS). Someone requests a MacOS port almost once a month, but none of the developers know how to help them.

3.4.7. Compiling on MacOS X

The port to MacOS X with Carbon API by Emmanuel Mailliard is quite new, and the configure and makefile support was added by Bryce Denney. You will need the compiler and libraries from the development tools CD. Bochs should configure and compile with the Carbon GUI if you simply type:

configure  
 make

Optionally, you can use the configure shortcut script for MacOS X, .conf.macosx, instead of running configure directly. If this script is close to what you need, just edit the script and then run it. To use the configure shortcut script and compile, the commands are

sh .conf.macosx  
 make

MacOS X has a special format for an application bundle, which looks like a directory that contains the required resource files and binaries. The Makefile currently creates this application bundle "by hand" using mkdir and copy, which is surely the wrong way to do it. Bryce doesn't know the official way to create an application from a Makefile, so this hack will remain until a real Mac developer helps to clean it up.

On MacOS X the default GUI is the Carbon interface, but you can also try other Bochs GUIs. Use --with-x11 for X windows, --with-rfb for VNC/RFB, --with-sdl for SDL or --with-sdl2 for SDL2.

3.4.8. Compiling on Amiga/MorphOS

Nicholai Benalal created this port to MorphOS running on Amiga. It should compile with:

configure  
 make

If the platform is not detected properly, you might need to use --enable-amigaos as a configure option. Optionally, you can use the configure shortcut script, .conf.amigaos.

3.4.9. Compiling with the RFB interface

The RFB code was written by Don Becker <[x-odus@iname.com](mailto:x-odus@iname.com)>, who has a Bochs-RFB web page on his site, <http://www.psyon.org/bochs-rfb/>.

This interface allows you to view the Bochs display with AT&T's [VNC Viewer](http://www.realvnc.com/). The protocol used between a VNC server and a VNC viewer is called RFB. Because the RFB code in Bochs is written with portable network socket and POSIX thread code, it can be compiled on many platforms and has been tested in Linux and Win32. No additional libraries are required. To try it, type:

configure --with-rfb  
 make

RFB currently uses this setup:

* port range 5900 to 5949 (using the first one available)
* no authentification
* by default 30 seconds waiting for client
* 8 bpp (BGR233) supported only
* if client doesn't support resize: desktop size 720x480 (for text mode and standard VGA)
* if resize supported: maximum resolution 1280x1024

With the display library option "timeout" the default value of 30 seconds can be changed. With a value of 0 it is possible to start the simulation without a client connected.

3.4.10. Compiling with the VNCSRV interface

The [LibVNCServer](http://libvnc.github.io/) is a cross-platform library that allows to implement VNC server functionality easily. On platforms supported by LibVNCServer Bochs can be compiled with support for it this way:

configure --with-vncsrv  
 make

Unlike the RFB GUI this new implementation is not limited to 8 bpp and it is possible to connect a Bochs session with a web browser.

3.4.11. Compiling with the SDL interface

Dave Poirier has written an SDL interface for Bochs. Simple DirectMedia Layer, or SDL, is a cross-platform multimedia library distributed from [libsdl.org](http://libsdl.org/). SDL is available for many platforms including Win32, Linux, BSD, IRIX, MacOS, MacOS X, and AmigaOS.

To compile Bochs with SDL, you must first install the SDL library from [libsdl.org](http://libsdl.org/). You can either get the source code and compile it yourself, or install the development libraries for your platform (already compiled). Then, go into the Bochs directory and type:

configure --with-sdl  
 make

If you are on FreeBSD and have SDL installed using the ports collection, Bochs won't be able to find the library automatically, as the SDL config script is called sdl11-config in that case (even for version 1.2). The easiest way to make Bochs find it, is to create a symlink to that script called sdl-config inside a directory which is in the path. For example:

ln -s /usr/local/bin/sdl11-config ~/bin/sdl-config

To compile in Microsoft VS2013Ex, you have to configure on a different system (see [Compiling on Win32 with Microsoft VC++](http://docs.google.com/compiling.html#COMPILING-WIN32)). Before running the configure script, you have to add --with-sdl to the shortcut script. If you have already configured, you can set **BX\_WITH\_SDL** to 1 in config.h.

Then you have to set up the project for SDL this way:

- add source file sdl.cc to the module **gui**  
- add path to the SDL include files to the modules **bochs** and **gui**  
- add lbraries SDL.lib and SDLmain.lib to the module bochs  
- change the runtime library for all modules to **Multithreaded-DLL (/MD)**

Compiling with MSVC **nmake** is not supported yet.

3.4.12. Compiling with the SDL version 2 interface

The SDL library version 2 is now also supported by Bochs. For some basic information about SDL, see [Section 3.4.11](http://docs.google.com/compiling.html#COMPILE-SDL). The configure option to enable SDL2 support is --with-sdl2 and the configuration script is called **sdl2-config**. Note that the Bochs guis for SDL version 1.2.x and 2.x are mutually exclusive. The legacy SDL gui support will be removed someday. When using the gui library autodetection (--with-all-libs) the configure script probes for SDL version 2 first.

3.4.13. Compiling with the wxWidgets interface

wxWidgets is a cross-platform C++ user interface library which you can download for free at [wxwidgets.org](http://wxwidgets.org/). The wxWidgets port of Bochs provides both a graphical configuration interface for editing bochsrc options (see [config\_interface option](http://docs.google.com/bochsrc.html#BOCHSOPT-CONFIGINTERFACE)) and a display (see [display\_library option](http://docs.google.com/bochsrc.html#BOCHSOPT-DISPLAYLIBRARY)). It was written by Bryce Denney, Don Becker, Dave Poirier, and Volker Ruppert.

configure --with-wx  
 make

If you want Bochs to use a wxWidgets installation not in your path (but installed somewhere else), you need to set the WX\_CONFIG environment variable to the proper wx-config script, before running configure (example for csh):

setenv WX\_CONFIG '/home/compile/wx/bin/wx-config'

3.4.14. Building an RPM on Linux

RPM stands for "RedHat Package Manager." An RPM is a compressed file containing files to be installed on your system. Bochs has a special shell script called make-rpm that helps to build an RPM of Bochs. Start with a clean source directory. Edit .conf.linux first if you want to adjust the configure options. Then, type:

./build/redhat/make-rpm | tee /tmp/build.txt

When this command completes, you should have a source RPM and a binary RPM of Bochs in the current directory. The **tee** part of the command (optional) saves a transcript of the build process into /tmp/build.txt, which is very useful if anything goes wrong. Instructions for installing an RPM are [here](http://docs.google.com/install-binary.html#LINUXRPM).

**Note:** In the past, you had to build rpms as root, but as of version 2.0 you can build them as a normal user.

3.4.15. Compile Problems

3.4.15.1. The configure script failed

The **configure** script can fail for various reasons. If the error message on the shell doesn't give enough information, it is recommended to review the output file config.log to find out what exactly happened. The following issues can be fixed by modifying the configure options used:

* deprecated option (option from previous Bochs version not needed / supported anymore)
* mutually exclusive option (e.g. --enable-debugger and --enable-gdb-stub)
* dependent option (e.g. USB support requires PCI support)
* incomplete option (e.g. --enable-cpu-level requires an argument)
* development package for library not installed (e.g. for --with-wx)

If the configure issue cannot be fixed by option changes and / or installing development packages / tools it should be reported in the [bochs-developers mailing list](http://docs.google.com/mailinglist.html#BOCHS-DEVELOPERS) or the [SF bug tracker](http://docs.google.com/feedback-support.html#SF-TRACKERS) for the Bochs project.

3.4.15.2. Make / compilation failure

The **make** utility itself can only fail if the specified target is not supported by the environment or simply doesn't exist. In most cases the programs called from **make** are causing the failure (e.g. compiler / linker).

In some cases it might be useful to rebuild Bochs completely by calling **make dist-clean** and running the **configure** script once again. These steps are recommended if the development sources from SVN are used and one of the build system files (e.g. Makefile or config.h.in) has been changed.

Sometimes it is possible to isolate the failing piece of code by disabling one or more configure options. For example, if the compilation fails somewhere in the cpu/avx\* files, it might help to configure without --enable-avx. The same could be done with other Bochs facilities controlled by configure options.

3.4.15.3. Runtime issues with self-compiled Bochs

After a successful compilation the self-compiled Bochs can fail if at least one library is not installed properly. This can happen if the library consists of a development and a runtime package and the second one is missing. Another problem can appear on build environments like Cygwin and MinGW/MSYS. Applications built there are usually designed to be run inside of the environment. When starting them from the Windows shell, it may cause errors like "missing cygwin1.dll" or similar. In that case, the missing DLL must be copied from e.g. the Cygwin folder to the location of the compiled EXE file.

### Notes

|  |  |
| --- | --- |
| [[1]](http://docs.google.com/compiling.html#AEN897) | The syntax for bash and ksh is given. In csh and variants, use the syntax **setenv *VARIABLE* *value*** to change environment variables. Do not use an equal sign for csh! |
| [[2]](http://docs.google.com/compiling.html#AEN935) | This is different from the terminology of some other projects, and it may cause confusion. Sometimes "clean" implies that all subdirectories are affected. |
| [[3]](http://docs.google.com/compiling.html#AEN1316) | Because Bochs depends so much on the configure script, if you are doing much win32 Bochs development, you should consider downloading Cygwin or MinGW/MSYS so that you can run the configure step natively. |
| [[4]](http://docs.google.com/compiling.html#AEN1321) | If the source directory is /home/joe/bochs-win32, the resulting .zip file is in /home/joe/bochs-win32-msvc-src.zip. |

|  |  |  |
| --- | --- | --- |
| [Prev](http://docs.google.com/install-binary.html) | [Home](http://docs.google.com/index.html) | [Next](http://docs.google.com/setup.html) |
| Installing a Binary | [Up](http://docs.google.com/installation.html) | Setup |