Magic Maintainer’s Manual #1: Hints for System Maintainers

John Ousterhout
Walter Scott

Computer Science Division
Electrical Engineering and Computer Sciences
University of California
Berkeley, CA 94720

(Updated by others, too.)

This tutorial corresponds to Magic version 6.

Tutorials to read first:

All of them.

Commands covered in this tutorial:

:*profile, :*runstats, :*seeflags, :*watch

Macros covered in this tutorial:

None.

1. Introduction

This document provides some information to help would-be Magic maintainers learn about the system. It is not at all complete, and like most infrequently-used documentation, will probably become less and less correct over time as the system evolves but this tutorial doesn’t. So, take what you read here with a grain of salt. We believe that everything in this tutorial was up-to-date as of the 1990 DECWRL/Livermore Magic release. Before doing anything to the internals of Magic, you should read at least the first, and perhaps all four, of the papers on Magic that appeared together in the *1984 Design Automation Conference*. In addition, the following portions of magic have their own papers:

extractor	*1985 Design Automation Conference*, page 286.
channel router	*1985 Chapel Hill Conference on VLSI*, page 145.

irouter and mzrouter	*1988 Design Automation Conference*, page 672.
resistance extractor	*1987 Design Automation Conference*, page 570.

2. Installing Magic

If you've received Magic on the 1990 DECWRL/Livermore Magic tape, then it shouldn't take much work to get it running. You should first pick a location for Magic's directory tree. Normally ~cad is chosen, but you might want to pick some other location to start. If you do pick another location, set your shell environment variable **CAD_HOME** to that location and mentally translate the ~cad references in this document to the location you chose. After reading the tape in, there will be a DECstation 3100 binary version of Magic in ~cad/bin and a set of library subdirectories in ~cad/lib/magic. If this isn't so, then at the very least you'll need to get a Magic system library set up in ~cad/lib/magic/sys: this directory contains information like technology files and colormaps and Magic can't run at all without it.

If you're running on a DECstation 3100 you shouldn't need to do anything besides what's mentioned above. Just run X11 and then run Magic. If you are running on some other workstation, you'll need to read the next section on how to make a new binary. The rest of this section concerns serial-line displays, so if you are using any sort of workstation with a built-in display there is no need to read the next couple of paragraphs.

If you're running on a mainframe with a serial-line color display, you'll probably need to do some additional setup. If the display is an AED512 or similar display, it will be attached to the mainframe via an RS232 port. Magic needs to be able to read from this port, and there are two ways to do this. The first is simply to have no login process for that port and have your system administrator change the protection to allow all processes to read from the port and write to it. The second way is to have users log in on the display and run a process that changes the protection of the display. There is a program called Sleeper that we distribute with Magic; if it's run from an AED port it will set everything up so Magic can use the port. Sleeper is clumsy to use, so we recommend that you use the first solution (no login process).

When you're running on mainframes, Magic will need to know which color display port to use from each terminal port. Users can type this information as command-line switches but it's clumsy. To simplify things, Magic checks the file ~**cad/lib/displays** when it starts up. The displays file tells which color display port to use for which text terminal port and also tells what kind of display is attached. Once this file is set up, users can run Magic without worrying about the system configuration. See the manual page for *displays* (5).

One last note: if you're running on an AED display, you'll need to set communication switches 3-4-5 to up-down-up.

3. Source Directory Structure

There are approximately 45 source subdirectories in Magic. Most of these consist of modules of source code for the system, for example **database**, **main**, and **utils**. See Section 4 of this document for brief descriptions of what's in each source directory. Besides the source code, the other subdirectories are:

doc Contains sources for all the documentation, including *man* pages, tutorials, and maintenance manuals. Subdirectories of **doc**, e.g. **doc/scmos**, contain the technology manuals. The Makefile in each directory can be used to run off the documentation. The tutorials, maintenance manuals, and technology manuals all use the Berkeley Grn/Ditroff package, which means that you can’t run them off without Grn/Ditroff unless you change the sources.

include Contains installed (i.e. ‘‘safe’’) versions of all the header files (*.h) from all the modules.

lib Contains installed (i.e. ‘‘safe’’) versions of each of the compiled and linked modules (*.o).

installed Most sites don’t use this directory. If you want it to be used, you can modify the script ~**cad/src/magic/:instclean**. If used, it contains one subdirectory for each of the source code directories. Each subdirectory contains ‘‘safe’’ versions of the source files for that module. These files correspond to the installed **.o** files in **lib**.

magic This directory is where the modules of Magic are combined together to form an executable version of the system.

contributed This directory contains Magic source code or programs that were contributed by other people. None of it has been tested by any member of the Magic team.

cadlib This is a symbolic link to the directory where Magic stores cell libraries and official installed versions of technology files and color maps. Normally, **cadlib** is a symbolic link to ~**cad/lib/magic**.

Magic is a relatively large system: there are around 575 source files, 250,000 lines of C code. In order to make all of this manageable, we’ve organized the sources in a two-level structure. Each module has its own subdirectory, and you can make changes to the module and recompile it by working within that subdirectory. In addition to the information in the subdirectory, there is an ‘‘installed’’ version of each module, which consists of the files in the **lib**, **include**, and **installed** subdirectories. The installed version of each module is supposed to be stable and reliable. At Berkeley, when a module is changed it is tested carefully without re-installing it, and is only re-installed when it is in good condition. Note that ‘‘installed’’ doesn’t mean that Magic users see the module; it only means that other Magic maintainers will see it. Each of the makefiles invokes the script **:instclean** to do the module installation. This optionally will copy files to the installed directory.

By keeping modules separate, it’s possible for several maintainers to work at once as long as they are modifying different source subdirectories. Each maintainer works with the uninstalled version of a module, and links that with the installed versions of all other modules. Thus, for example, one maintainer can modify **database/DBcell.c** and another can modify **dbwind/DBWundo.c** at the same time.

4. Making Magic

The top-level Makefile (~cad/src/magic/Makefile) provides many options. Before using the Makefile, be sure to set your CAD_HOME shell environment variable to the location of your top-level cad directory (if it is not the standard ~cad).

The most useful Makefile options are:

make config Configure the Magic system for a particular type of display or operating system. This just runs the :config shell script to set up a couple of files. The curious may examine the script directly. If your configuration isn't handled by this script, then you can use it simply as a guide as to what to do. Much of the configuration is done with compilation flags. See a later section of this manual for a full listing of them.

make magic Make a version of Magic. All sub-modules are remade, if needed, and then the final magic binary is produced.

make everything Same as "make magic". Both options make auxilary programs like ext2sim.

make force Force recompilation. Like a "make everything", except that object files are first removed to force recompilation.

make clean Delete files that can be remade, such as binaries.

make install Install the Magic binaries in ~cad (or $CAD_HOME if you have that set).

Putting together a runnable Magic system proceeds in two steps after a source file has been modified. First, the source file is compiled, and all the files in its module are linked together into a single file *xyz***.o**, where *xyz* is the name of the module. Then all of the modules are linked together to form an executable version of Magic. The command **make** in each source directory will compile and link the module locally; **make install** will compile and link it, and also install it in the **include**, **lib**, and **installed** directories. All makefiles are set up to use the compiler flags found in ~**cad/src/magic/misc/DFLAGS** and ~**cad/src/magic/misc/CFLAGS**. A list of flags appears later in this manual.

The command **make** in the subdirectory **magic** will produce a runnable version of Magic in that directory, using the installed versions of all modules. To work with the uninstalled version of a module, create another subdirectory identical to **magic**, and modify the Makefile so that it uses uninstalled versions of the relevant modules. For example, the Magic team uses subdirectories **hamachitest**, **mayotest**, **mhatest**, **oustertest**, and **wsstest** that we use to test new versions of modules before installing them. If you want to remake the entire system, type ‘‘make magic’’ in the top-level directory (~**cad/src/magic**).

One last thing -- there are some customizations you may want to make to Magic. If magic core dumps, it sends mail to somebody. That somebody is set in the MAIL_COMMAND definition in the file misc/niceabort.c. You may want to change it. Also in the same file is CRASHDIR which says where core dumps should be placed. Paths used by magic are located in misc/paths.h. You shouldn't need to change them, though, because the CAD_HOME shell environment variable controls what Magic uses for the location of ~cad.

5. Summary of Magic Modules

This section contains brief summaries of what is in each of the Magic source subdirectories.

calma Contains code to read and write Calma Stream-format files. It uses many of the procedures in the **cif** module.

cif Contains code to process the CIF sections of technology files, and to generate CIF files from Magic.

cmwind Contains code to implement special windows for editing color maps.

commands The procedures in this module contain the top-level command routines for layout commands (commands that are valid in all windows are handled in the **windows** module). These routines generally just parse the commands, check for errors, and call other routines to carry out the actions.

database This is the largest and most important Magic module. It implements the hierarchical corner-stitched database, and reads and writes Magic files.

dbwind Provides display functions specific to layout windows, including managing the box, redisplaying layout, and displaying highlights and feedback.

debug There's not much in this module, just a few routines used for debugging purposes.

drc This module contains the incremental design-rule checker. It contains code to read the **drc** sections of technology files, record areas to be rechecked, and recheck those areas in a hierarchical fashion.

ext2sim The **ext2sim** directory isn't part of Magic itself. It's a self-contained program that flattens the hierarchical **.ext** files generated by Magic's extractor into a single file in **.sim** format. See the manual page **ext2sim (1)**.

ext2spice This is another self-contained program. It converts **.ext** files into single file in spice format. See the manual page **ext2spice (1)**.

extflat Contains code that is used by the **extract** module and the **ext2...** programs. The module produces a library that is linked in with the above programs.

extract Contains code to read the **extract** sections of technology files, and to generate hierarchical circuit descriptions (**.ext** files) from Magic layouts.

fsleeper	Like **ext2sim**, this directory is a self-contained program that allows a graphics terminal attached to one machine to be used with Magic running on a different machine. See the manual page **fsleeper (1)**.
garouter	Contains the gate array router from Lawrence Livermore National Labs.
gcr	Contains the channel router, which is an extension of Rivest's greedy router that can handle switchboxes and obstacles in the channels.
graphics	This is the lowest-level graphics module. It contains driver routines for X11 as well as less-used drivers for the AED family of displays and for Sun Windows. The code here does basic clipping and drawing. If you want to make Magic run on a new kind of display, this is the only module that should have to change.
grouter	The files in this module implement the global router, which computes the sequence of channels that each net is to pass through.
irouter	Contains the interactive router written by Michael Arnold at Lawrence Livermore National Labs. This router allows the user to route nets interactively, using special hint layers to control the routing.
macros	Implements simple keyboard macros.
magicusage	Like **ext2sim**, this is also a self-contained program. It searches through a layout to find all the files that are used in it. See **magicusage** (1).
main	This module contains the main program for Magic, which parses command-line parameters, initializes the world, and then transfers control to **textio**.
misc	A few small things that didn't belong anyplace else.
mpack	Contains routines that implement the Tpack tile-packing interface using the Magic database. (not supported)
mzrouter	Contains maze routing routines that are used by the irouter and garouter modules.
net2ir	Contains a program to convert a netlist into irouter commands.
netlist	Netlist manipulation routines.
netmenu	Implements netlists and the special netlist-editing windows.
parser	Contains the code that parses command lines into arguments.
plot	The internals of the **:plot** command.
plow	This module contains the code to support the **:plow** and **:straighten** commands.

prleak Also not part of Magic itself. Prleak is a self-contained program intended for use in debugging Magic's memory allocator. It analyzes a trace of mallocs/frees to look for memory leaks. See the manual page **prleak (8)** for information on what the program does.

resis Resis is a module that does better resistance extraction via the :extresis command. Courtesy of Don Stark of Stanford.

router Contains the top-level routing code, including procedures to read the router sections of technology files, chop free space up into channels, analyze obstacles, and paint back the results produced by the channel router.

select This module contains files that manage the selection. The routines here provide facilities for making a selection, enumerating what's in the selection, and manipulating the selection in several ways, such as moving it or copying it.

signals Handles signals such as the interrupt key and control-Z.

sim Provides an interactive interface to the simulator rsim. Courtesy of Mike Chow of Stanford.

tech This module contains the top-level technology file reading code, and the current technology files. The code does little except to read technology file lines, parse them into arguments, and pass them off to clients in other modules (such as **drc** or **database**).

textio The top-level command interpreter. This module grabs commands from the keyboard or mouse and sends them to the window module for processing. Also provides routines for message and error printout, and to manage the prompt on the screen.

tiles Implements basic corner-stitched tile planes. This module was separated from **database** in order to allow other clients to use tile planes without using the other database facilities too.

undo The **undo** module provides the overall framework for undo and redo operations, in that it stores lists of actions. However, all the specific actions are managed by clients such as **database** or **netmenu**.

utils This module implements a whole bunch of utility procedures, including a geometry package for dealing with rectangles and points and transformations, a heap package, a hash table package, a stack package, a revised memory allocator, and lots of other stuff.

windows This is the overall window manager. It keeps track of windows and calls clients (like **dbwind** and **cmwind**) to process window-specific operations such as redisplaying or processing commands. Commands that are valid in all windows, such as resizing or moving windows, are implemented here.

wiring The files in this directory implement the **:wire** command. There are routines to select wiring material, add wire legs, and place contacts.

6. Portability Issues

Magic runs on a variety of machines. Running "make config" in the top-level source directory sets the compiletime options. If you are porting Magic, you should modify the configuration section at the end of file "misc/magic.h" to suit your machine, by testing compiler flags. No changes should be made that would hamper Magic's operation on other machines.

7. Compilation Switches

Over the years Magic has acquired a number of compilation switches. While it's undesirable to have so many, it seems unavoidable since people use Magic on such a wide variety of machines. The file ~cad/src/magic/misc/DFLAGS should contain the compile switches that you wish to use at your site. All makefiles for Magic reference the common DFLAGS file. The switches in this release are shown below.

These flags are normally setup by running the "make config" script in ~cad/src/magic. Some of them are turned on in "magic.h" when a particular machine configuration is detected.

7.0.1. Machine/OS Compiletime Options

The following switches should be defined automatically by the compiler.

vax
: For VAX machines.

mips
: For mips processors, such as the DECstation 3100.

MIPSEL
: For little-endian mips processors, such as the DECstation 3100.

MIPSEB
: For big-endian mips processors.

pyramid
: For Pyramid machines.

sun
: For Sun machines.

mc68000
: For machines which have a version of the 68000 as the processor.

sparc
: Sparc-based machines.

lint
: Used to bypass things that lint complains about. Don't turn this on. Lint turns it on itself.

If needed, you should put the following switches in the DFLAGS file.

macII
: For the MacII.

SUNVIEW
: Used when including Magic's SunView graphics drivers.

SUN120
: For the Sun120 machine.

BSD4_2
: Used in the utils module to patch around a broken version of flsbuf() that is needed in the VAX version of Unix 4.2 BSD systems. This is rarely needed, since almost all version of Unix now have this bug fixed.

FASYNC
: Hack for some versions of Sun2 software (old stuff).

NO_VARARGS
: Hack for machines without a VARARGS package.

SYSV
: For Unix System V.

Flags defined, if needed, in "magic.h" based on other flags.

BIG_ENDIAN
: Indicates big endian byte ordering is being used.

LITTLE_ENDIAN
: Indicates little endian byte ordering is being used.

NEED_MONCNTL
: Hack for machines without a moncontrol procedure.

NEED_VFPRINTF
: Hack for machines without a vfprintf procedure.

SIG_RETURNS_INT
: Defined in magic.h for systems that expect a signal handler to return an integer rather than a void.

7.0.2. Graphics Driver Compiletime Options

X10
: Used in the graphics module for the X10 driver.

X11
: Used in the graphics module for the X11 driver.

OLD_R2_FONTS
: Used when X11 is release 2. Magic normally assumes release 3.

AED
: Used in the graphics module when compiling for AED displays.

GTCO
: Used in the graphics module when using a GTCO bitpad with an AED display.

7.0.3. Compiletime Options for Module Inclusion

NO_CALMA
: Flag to eliminate the calma module, to reduce the size of Magic.

NO_CIF
: Flag to eliminate the cif module, to reduce the size of Magic.

NO_EXT
: Flag to eliminate the ext module, to reduce the size of Magic.

NO_PLOT
: Flag to eliminate the plot module, to reduce the size of Magic.

NO_ROUTE
: Flag to eliminate the router modules, to reduce the size of Magic.

NO_SIM
: Flag to eliminate the sim module, to reduce the size of Magic.

7.0.4. Debugging Compiletime Options

CELLDEBUG
: Debugging flag for the database module.

COUNTWIDTHCALLS
: Debugging flag for the plow module.

DEBUGWIDTH
: Debugging flag for the plow module.

DRCRULESHISTO
: Debugging/tuning flag for the drc module.

FREEDEBUG
: Memory allocation debugging flag.

MALLOCMEASURE
: Memory allocation debugging flag.

MALLOCTRACE
: Memory allocation debugging flag.

NOMACROS
: Memory allocation debugging flag.

PAINTDEBUG
: Debugging flag for the database painting routines.

PARANOID
: Flag to enable consistency checking. With a system the complexity of Magic, you should always leave this flag turned on.

8. Technology and Other Support Files

Besides the source code files, there are a number of other files that must be managed by Magic maintainers, including color maps, technology files, and other stuff. Below is a listing of those files and where they are located.

8.1. Technology Files

See ''Magic Maintainer's Manual #2: The Technology File'' for information on the contents of technology files. The sources for technology files are contained in the subdirectory **tech**, in files like **scmos.tech** and **nmos.tech**. The technology files that Magic actually uses at runtime are kept in the directory **cadlib/sys**; **make install** in **tech** will copy the sources to **cadlib/sys**. Technology file formats have evolved rapidly during Magic's life, so we use version numbers to allow multiple formats of technology files to exist at once. The installed versions of technology files have names like **nmos.tech26**, where **26** is a version number. The current version is defined in the Makefile for **tech**, and should be incremented if you ever change the format of technology files; if you install a new format without changing the version number, pre-existing versions of Magic won't be able to read the files. After incrementing the version number, you'll also have to re-make the **tech** module since the version number is referenced by the code that reads the files.

8.2. Display Styles

The display style file sources are contained in the source directory **graphics**. See ''Magic Maintainer's Manual #3: The Display Style and Glyph Files'' and the manual page *dstyle* (5) for a description of their contents. **Make install** in **graphics** will copy the files to **cadlib/sys**, which is where Magic looks for them when it executes.

8.3. Glyph Files

Glyph files are described in Maintainer's Manual #3 and the manual page *glyphs* (5); they define patterns that appear in the cursor. The sources for glyph files appear in two places: some of them are in **graphics**, in files like **color.glyphs**, and some others are defined in **windows/window*XX*.glyphs**. When you **make install** in those directories, the glyphs are copied to **cadlib/sys**, which is where Magic looks for them when it executes.

8.4. Color Maps

The color map sources are also contained in the source directory **graphics**. Color maps have names like **mos.7bit.std.cmap**, where **mos** is the name of the technology style to which the color map applies, **7bit** is the display style, and **std** is a type of monitor. If monitors have radically different phosphors, they may require different color maps to achieve the same affects. Right now we only support the **std** kind of monitor. When Magic executes, it looks for color maps in **cadlib/sys**; **make install** in **graphics** will copy them there. Although color map files are textual, you shouldn't edit them by hand; use Magic's color map editing window instead.

9. New Display Drivers

The most common kind of change that will be made to Magic is probably to adapt it for new kinds of color displays. Each display driver contains a standard collection of procedures to perform basic functions such as placing text, drawing filled rectangles, or changing the shape of the cursor. A table (defined in **graphics/grMain.c**) holds the addresses of the routines for the current display driver. At initialization time this table is filled in with the addresses of the routines for the particular display being used. All graphics calls pass through the table.

If you have a display other than an AED or an X11 workstation, the first thing you should do is check the directory **contributed**. Each of the subdirectories in **contributed** contains additional code that was contributed by some site outside of Berkeley. Each subdirectory should contain a file named **ReadMe** (or something similar), which explains how to install and use the code. If you have any troubles with one of the these drivers, you'll have to contact the people that contributed the driver; we here at Berkeley don't know anything about them. If the contributors wish to be contacted, they will identify themselves in the **ReadMe** file.

If you need to build a new display driver, we recommend starting with the routines for either the AED (all the files in **graphics** with names like **grAed1.c**), or the Sun (names like **grSunW1.c**). For stand-alone displays, the AED routines are probably the easiest to work from; for integrated workstations with pre-existing window packages, the Sun routines may be easiest. Copy the files into a new set for your display, change the names of the routines, and modify them to perform the equivalent functions on your display. Write an initialization routine like **aedSetDisplay**, and add information to the display type tables in **graphics/grMain.c**. At this point you should be all set. There shouldn't be any need to modify anything outside of the graphics module.

10. Debugging and Wizard Commands

Magic seems to work fine under the latest version of dbx, such as the dbx found on the DECstation 3100. The Makefiles are set up to compile all files with the **-g** switch, which creates debugging information in *dbx*'s format.

Because of the size of Magic and the way Unix handles debugging symbols, it's slow to compile a complete version of Magic with debugging information for everything, and the executable file ends up being enormous. To solve this problem the Makefiles are set up to strip off debugging information before installing. Thus, you have to link with uninstalled versions to get debugging information. In most cases, debugging information is only needed for a few modules at a time, namely the modules you're currently modifying. The database module is set up to install with debugging symbols, since it seems to be involved in almost all debugging.

If you try to use older versions of *dbx*, you'll discover that Magic has too many procedures for the default table sizes; *dbx* runs out of space and dies. The solution is either to recompile *dbx* with larger tables or throw away pieces of Magic to reduce the number of procedures (we recommend the first alternative).

There are a number of commands that we implemented in Magic to assist in debugging. These commands are called *wizard commands*, and aren't visible to normal Magic users. They all start with ‘‘*****’’. To get terse online help for the wizard commands, type **:help wizard** to Magic. The wizard commands aren't documented very well. Some of the more useful ones are:

***watch** *plane*

This causes Magic to display on the screen the corner-stitched tile structure for one of the planes of the edit cell. For example, ***watch subcell** will display the structure of the subcell tile plane, including the address of the record for each tile and the values of its corner stitches. Without this command it would have been virtually impossible to debug the database module.

***profile on|off**

If you're using the Unix profiling tools to figure out where the cycles are going, this command can be used to turn profiling off for everything except the particular operation you want to measure. This command doesn't work on many systems, because the operating system doesn't support selective enabling and disabling of profiling.

***runstats**

This command prints out the CPU time usage since the last invocation of this command, and also the total since starting Magic.

***seeflags** *flag*

If you're working on the router, this command allows you to see the various channel router flags by displaying them as feedback areas. The cursor should first be placed over the channel whose flags you want to see.