Magic Maintainer's Manual #1: Installation and Development

John Ousterhout
Walter Scott

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

Tim Edwards
Johns Hopkins University Applied Physics Laboratory
Laurel, MD 20723

This tutorial corresponds to Magic version 7.

Tutorials to read first:

All of them.

Commands introduced in this tutorial:

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

Macros introduced in this tutorial:

(None)

1 Introduction

This document provides some information to help system administrators and would-be Magic maintainers learn about the system. 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 Compiling and Installing Magic

If you've downloaded Magic via FTP, 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, meaning that "cad" is a username on the system with a home directory typically something like /home/cad/ or /usr/local/cad/, but you might want to pick some other location to start, particularly if you do not have root privilege to create or write into the ˜cad directory. If you choose a different 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.

The download file comes in tarred, gzipped format. Follow the standard procedure to uncompress and expand:

```
tar xzf magic-7.1.tar.gz
cd magic-7.1
```

Followed by

```
make config
```

The first prompt asks for selection of the graphics interface(s). Magic is designed to link its generic graphics calls to specific driver calls at runtime, so any combination of choices is possible. Choose more than one option with a space-separated list of option numbers at the prompt. The choices are as follows:

1. **X11** for all versions of X11 (currently X11R6 is standard). Several other magic options make use of X11 calls, such as the extended macro package, which uses the X server's key symbol lookup to allow macro definitions on function and keypad keys, so the X11 package is preferred.

2. **OpenGL** for systems having OpenGL capability under X11. Generally, this applies to SGI hardware, and Linux systems with accelerated 3D video hardware implementing the OpenGL API, and an OpenGL-capable accelerated X server. This option is *not* recommended for non-hardware-accelerated (i.e., software-implemented) OpenGL or compatible (e.g., Mesa) servers, because the interface makes heavy use of color blending, implemented in software by dreadfully slow flood fills.

3. **SunView** for Sun Workstations. This option is rather out of date, because although Solaris still supports SunView, it also supports the superior X11 protocol.

4. **X10** is legacy code for support of X10, the precursor of X11.

5. **AED graphics terminals** is legacy code for support of ancient serial-line graphics terminals. They were great machines, once long ago (see Appendix A).

The next prompt asks for the target operating system. Most modern UNIX types are supported, and any others usually fall under the category of BSD- or SYSV-compatible. At worst, an unsupported system may require tweaking the compiler flags, which is best done directly to the file `misc/CFLAGS` after running "make config". Magic can be compiled for only one system at a time. Select one of the following options:

1. **Linux**
2. **NetBSD 1.x**
3. **FreeBSD 2.x**
4. **OSF/1** for 64-bit systems such as the Digital Alpha AXP
5. **Solaris 2.x**
6. **SunOS 4.x** for pre-Solaris Suns.
7. **SGI IRIX** for Silicon Graphics systems before SGI moved to the Linux/68000 platform
8. **OS/2 Warp** IBM's much-too-late attempt to overthrow the Bill Gates empire
9. **BSD Unix systems** for Ultrix and various Berkeley BSD 4.3-based systems.
10. **SYSV Unix systems** for HPUX, Apple's defunct A/UX, and various other System V-based systems.

The third prompt asks for machine architecture for any machines requiring special compile flags. They are the following:

1. **Intel 80x86-based workstations** for Intel and AMD platforms (Linux, NetBSD, FreeBSD, OS/2 Warp)
2. **HP 68000-based workstations**
3. **HP/PA-based workstations**
4. **MIPS workstation** (RISCos4.0; not DECStations)
5. **An Apple MacII** (A/UX)
6. **None of the above** for everything else (Suns, SGIs, DECStations, DEC Alpha)

The final set of prompts selects various optional modules. On modern systems with vast amounts of memory and disk space, the best choice is to use all of them. Systems with low memory overhead ($<$ 128MB) may want to avoid SCM (the scheme interpreter).

1. **CALMA**—module which enables reading and writing GDS-II (otherwise known as CALMA or "streams") format files.
2. **CIF**—module which enables reading and writing Caltech Intermediate Format (CIF) files. Magic only knows how to write CIF and GDS-II, so at least one of these two ought to be selected.
3. **PLOT**—module for graphics output. Supports PostScript, direct pixel output, versatec, and gremlin formats.
4. **READLINE**—module incorporating the GNU "readline" package (version 4.1) into the magic command-line interface. Readline implements command-line history and editing.
5. **ROUTE**—module which supports various routing tools (standard router, interactive router, maze router, channel router, gate-array router, and global router).
6. **SCM**—module implementing the "scheme" command-line interpreter, a lisp-like programming language for creating new commands and procedures.
7. **SIM**—the interactive interface to Stanford irsim (rsim) digital switch simulator.
8. **.magic**—Choice of using either the old or the new style of system startup (.magic) file. The old style retains compatibility; the new file makes use of interactive macro capability, and, if compiled under X11, extended macro capability for function, cursor, and keypad keys.

After configuration, compile and install using

```
make force
make install
```

The remaining sections of this manual deal with technical issues related to Magic source code and its development.

3 Source Directory Structure

There are 49 source subdirectories in Magic. Most of these consist of modules of source code for the system, for example **database**, **main**, and **utils**. See Section 5 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 LaTeX, which means that you will need the LaTeX package to recompile the manuals from source. Documentation is also available online in HTML format.

- **include**
  Contains copies of all the header files (*.h) from all the modules.

- **lib**
  Contains copies of each of the compiled and linked modules (*.o and *.a).

- **magic**
  In addition to the source main() routine, this directory is where the modules of Magic are linked to form the executable version of the system.

Magic is a relatively large system: there are around 575 source files, and 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. The CVS method of software project version management has been implemented to make it possible for several maintainers to work in parallel. The CVS repository for magic is kept at host **csl.cornell.edu** in directory **/ufs/repository**. Participation in Magic development requires a remote CVS username and password on server **csl**. Logging in simply requires the execution of the following CVS command:

cvs -d :pserver:*cvslogin*@csl.cornell.edu:/ufs/repository login

To download the latest release of magic, use

cvs -d :pserver:*cvslogin*@csl.cornell.edu:/ufs/repository checkout magic

This will create a directory called "**magic**". It contains the entire magic distribution. Once in the **magic/** directory, the **-d** option to CVS is no longer required. See the **cvs (1)** manual page for details. The critical CVS commands are "cvs add" to introduce new files to the repository, "cvs delete" to remove them, "cvs update" to merge in any new changes found in the repository, and "cvs commit" to send local changes back to the repository. Simultaneous changes to the same file are merged merged by heuristic; conflicts are flagged, to be resolved by hand on a case by case basis.

There are two mailing lists associated with Magic development:

1. `magic-hackers@csl.cornell.edu` is for general news and discussions about the development process.

2. `magic-dev@csl.cornell.edu` is for developers only and provides feedback on any CVS changes made in the repository.

4 Compiling and Installing

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 Section **??** 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** and **ext2spice**.

- **make force**
  Force recompilation. Like a "make everything", except that object files are first removed to force complete recompilation from scratch.

- **make clean**
  Delete files that can be remade, such as binaries, object, and library files.

- **make install**
  Install the Magic binaries in **~cad/bin** (or **${CAD_HOME}/bin** 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** and **lib** 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 in Section **??**.

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**).

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.

- **ext2dlys**
  The **ext2dlys** directory isn't part of Magic itself. It's a self-contained program that uses the hierarchical **.ext** files generated by Magic's extractor and an optional netlist file designating net pinouts (for purposes of counting I/O loads), and produces a wire-delay file. Also compiles an executable **sim2dlys** for delay calculations from **.sim** files. These programs are no longer compiled and the module has been commented out in the top-level **Makefile**.

- **ext2sim**
  This is another self-contained program. 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)**.

- **extcheck**
  Yet another independent program. This one checks the **.ext** files for global node connectivity and summarizes the number of FETs, nodes, etc. See the manual page **extcheck (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 and OpenGL as well as legacy drivers for the AED family of display terminals, Sun Windows, and X10. 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.

- **lisp**
  This module contains code which, if the SCHEME option is chosen at compile time, implements the lisp-like "scheme" interpreter. Scheme enables magic commands to be executed in a programming language framework, so complex functions can be defined.

- **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. Code to write PostScript, raw pixel, versatec, and gremlin formats.

- **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.

- **readline**
  "readline" is an independent library of routines implementing command-line history and editing. Version 7.1 of magic uses GNU readline-4.1.

- **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.

7.1 Machine/OS Compiletime Options

The following switches should be defined automatically by the **:config** script upon selection of the target hardware and OS.

- mips
  For mips processors, such as the DECstation.
- MIPSEL
  For little-endian mips processors, such as the DECstation 3100.
- MIPSEB
  For big-endian mips processors.
- sun
  For Sun machines.
- mc68000
  For machines which have a version of the 68000 as the processor.
- sparc
  Sparc-based machines.
- i386
  For Intel x86-based machines.
- linux
  For Linux systems.

- vax
  For VAX machines (legacy).

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

If needed, you can put the following switches in the DFLAGS file:

- macII
  For the Apple Mac-II (running A/UX) (legacy).

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

- SUN120
  For the Sun120 machine (legacy).

- 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 (legacy).

- FASYNC
  Hack for some versions of Sun2 software (legacy).

- 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.

7.2 Graphics Driver Compile-time Options

- X11
  Used in the graphics module for the X11 driver.
- OpenGL
  Used in the graphics module for the OpenGL/GLX driver.
- XLIB
  Used for all graphics modules based on an X server (currently, that means X11 and OpenGL).
- X10
  Used in the graphics module for the X10 driver (legacy).
- AED
  Used in the graphics module when compiling for AED displays (legacy).
- GTCO
  Used in the graphics module when using a GTCO bitpad with an AED display (legacy).

7.3 Compile-time 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 (legacy; not among configuration choices).
- 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.
- NO_SCHEME_INTERPRETER
  Flag to eliminate the "scheme" command-line interpreter.
- USE_READLINE
  Flag to include the GNU "readline" package.
- OLD_DOT_MAGIC
  Flag indicating use of the original (backwardly compatible) system startup "**.magic**" file.

- LLNL
  Flag to incorporate Lawrence Livermore extensions, including an area router, new channel router, and stretch graphs (experimental; not among configuration choices).

7.4 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 (set automatically by the **:config** script).

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 **${CAD_HOME}/lib/magic/sys**;**make install** in **tech** will copy the sources to **${CAD_HOME}/lib/magic/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.tech27**, where **27** 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 **${CAD_HOME}/lib/magic/sys**, which is where Magic looksfor 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 **${CAD_HOME}/lib/magic/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 **${CAD_HOME}/lib/magic/sys**;**make install** in **graphics** will copy them there. Although color map files are textual, editing by hand is undesirable; 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 the ones currently defined (X11, OpenGL/GLX, SunView, and the legacy X10 and AED drivers), and you want to build a new display driver, we recommend starting with the routines for the X11 (all the files in **graphics** named **grX11su*n*.c**), or the Sun (named **grSunW*n*.c**). 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 **x11suSetDisplay**, 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.

The significant difference between the X11 driver and the Sun driver depends on the nature of the server: The X11 server polls for new events, and the typical "main loop" of an X11 application is a call to **XtMainLoop()** which never exits. This presents a problem for Magic, which is interrupt driven, a choice made due to the large amount of internal processing (e.g., interactive DRC checking) as compared to the small amount of user input (e.g., keystrokes and mouse buttons). Magic has its own blocking loop, which is a call to **select**), a routine which continuously polls I/O devices for interrupts. Because X11 (at least up to and including the R6 version) is not thread-safe, these two loops cannot be threads. The only remaining possibility is to make the X11 main loop run as a separate process (in the case of X11, called "X11Helper"), capture relevant X protocol messages for the Magic window, and transmit the information to the Magic process through an I/O pipe (the use of which triggers the I/O interrupt and breaks the **select()** loop). This method has certain drawbacks, the main one being that if Magic crashes for any reason, the helper process remains hanging until killed by hand or by a system reboot.

The Sun interface makes graphics calls as requested, which makes implementation for Magic much simpler than X11, but consequently prevents remote display operation and efficient multitasking, the major reasons for X11's success.

Note that any graphics interfaces based on *thread-safe* graphics systems (e.g., Windows NT, and hopefully X11R7, if it ever happens) can reduce the graphics interface complexity by running the X11 event handler as a detached thread.

10 Debugging and Wizard Commands

Magic works fine under the latest versions of dbx and GNU gdb. The Makefiles are set up to compile all files with the **-g** switch, which creates debugging information.

In the past, memory and speed limitations made it useful to include a flag, found in script **:instmodule**, to strip the debugging symbols from the object files when linking the Magic executable. Modern systems are fast and have copious amounts of both system memory and disk space, so this option is set to "NO" in the script.

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.

A Serial-Line Graphics Displays

This section remains for information regarding the serial-line graphics driver. Although totally outdated, these are the simplest graphics driver routines, and at least for now are worth keeping around for reference, particularly when attempting to implement a new graphics driver.

Serial-line displays require 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** distributed with Magic versions 6.5.1 and earlier; 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.