Magic Tcl Tutorial #1: Introduction

R. Timothy Edwards

Space Department
Johns Hopkins University
Applied Physics Laboratory
Laurel, MD 20723

This tutorial corresponds to Tcl-based Magic version 7.2

1 What is Tcl-based Magic, and Why?

In Magic version 7.0, Rajit Manohar incorporated a SCHEME interpreter into the Magic source, noting the limitation of magic to handle definitions and variables, conditionals, and block structures. By embedding an interpreter into the code, the interpreter's functions are made available on the magic command line, making magic extensible. The SCHEME interpreter and various extensions incorporated into loadable scripts are outlined in the tutorials `tutscm1.ps` through `tutscm4.ps`.

While making Magic considerably more flexible, the embedded SCHEME interpreter had some notable drawbacks. The primary one is that the SCHEME language is syntactically different from Magic's command-line syntax. Also, the interpreter is largely disconnected from the code, and does not affect or extend the graphics or handle results from magic commands.

Beginning in Magic version 7.2, Magic has been recast into a framework called *ScriptEDA*, in which existing applications become **extensions** of an interpreter rather than having an interpreter embedded in the application. The main advantage of extending over embedding is that the application becomes a module of the interpreter language, which does not preclude the use of additional, unrelated modules in the same interpretive environment. For example, in Tcl-based Magic, graphics are handled by Tk (the primary graphics package for use with Tcl), and applications such as IRSIM (the digital simulator) can be run as if they were an extension of magic itself. Commands for Tcl, Tk, IRSIM, BLT, and any other Tcl-based package can be mixed on the magic command line.

While *ScriptEDA* suggests the use of the **SWIG** package to give applications the ability to be compiled as extensions of any interpreter (Tcl, Python, SCHEME, Perl, etc.), there are specific advantages to targeting Tcl. Foremost, the syntax of Tcl is virtually 100% compatible with that of Magic. This is not coincidentally because both Magic and Tcl were written by John Ousterhout! Many ideas from the development of Magic were incorporated into the overall concept and design of the Tcl interpreter language.

Largely due to the syntactical compatibility, Tcl-based magic is completely backwardly-compatible with the non-interpreter version of magic. Either can be selected at compile-time, in addition to compiling with embedded SCHEME, which has been retained in full as an option. A few minor issues, such as the appearance of the cursor when magic is used without a Tk console window, are addressed below.

Magic extensions under Tcl are considerable, and explanations of the features have been split into several tutorial files, as listed in Table 1.

Magic Tcl Tutorial #1:	Introduction
Magic Tcl Tutorial #2:	The GUI Wrapper
Magic Tcl Tutorial #3:	Extraction and Netlisting
Magic Tcl Tutorial #4:	Simulation with IRSIM
Magic Tcl Tutorial #5:	Writing Tcl Scripts for Magic

Table 1: The Magic Tcl tutorials and other documentation.

2 Features of Tcl-based Magic

In summary, the features of Tcl-based Magic (corresponding to Magic version 7.2, revision 31) are as follows:

1. The command name **magic** itself is a script, not a compiled executable. The script launches Tcl and gives it the name of a Tcl script to evaluate (`magic.tcl`). The Tcl script loads Magic as an extension of Tcl and performs all other operations necessary to start up the application.

2. Command-line arguments passed to Magic have been changed. Some older, unused arguments have been removed. Several arguments have been added, as follows:

   (a) `-noconsole` Normally, under Tcl/Tk, Magic starts by launching a Tk-based console window (`tkcon.tcl`) which is the window that accepts magic commands. Previous versions of Magic accepted commands from the calling terminal. The former style of running commands from the calling terminal can be selected by choosing this argument at runtime. Note, however, that due to fundamental differences in the underlying input routines between Tcl and magic, the terminal-based command entry does not exactly match the original behavior of magic. In particular, the background DRC function does not change the prompt. In addition, "Xterm" consoles must select option "Allow SendEvents" for keystrokes to be echoed from the layout window into the terminal. For security reasons, the application cannot change this option on the terminal.

   (b) `-wrapper` To enforce backward-compatibility, magic appears as it would without the Tcl interpreter (apart from the text entry console) when launched with the same arguments. However, most users will want to make use of the extensions and convenience functions provided by the "wrapper" GUI. These functions are detailed in a separate tutorial (see Tutorial #2).

3. Magic-related programs “ext2spice” and “ext2sim” have been recast as magic commands instead of standalone executables. This makes a good deal of sense, as both programs make heavy use of the magic internal database. Most values required to produce netlists were passed through the intermediary file format `.ext`. However, not all necessary values were included in the `.ext` file format specification, and some of these missing values were hard-coded into the programs where they have since become mismatched to the magic database. This has been corrected so that all netlist output corresponds to the technology file used by a layout.

   Both commands “ext2spice” and “ext2sim” allow all of the command-line arguments previously accepted by the standalone programs. Some of the more common functions, however, have been duplicated as command options in the usual format of magic commands. For instance, one may use the magic command:

   ext2sim help

   to get more information on `ext2sim` command-line options.

4. All Tcl interpreter procedures may be freely mixed with magic commands on the command line. For instance, a user can use Tcl to compute arithmetic expressions on the command line:

   copy e [expr{276 * 5 + 4}]

5. A number of magic commands pass values back to the interpreter. For instance, the command

   box values

   returns the lower left- and upper right-hand coordinates of the cursor box in magic internal units. This return value can be incorporated into an expression, such as the one below which moves the box to the left by the value of its own width:

   set bbox [box values]
   set bwidth expr {[lindex $bbox 2] - [lindex $bbox 0]}]
   move e $bwidth

6. Magic prompts for text have been recast as Tk dialog boxes. For instance, the command **writeall** will pop up a dialog box with five buttons, one for each of the available choices (write, flush, skip, abort, and auto).

7. IRSIM is no longer available as a “mode” reached by switching tools by command or the space-bar macro. Instead, IRSIM (version 9.6) can be compiled as a Tcl extension in the same manner as Magic, at compile time. When this is done, IRSIM is invoked simply by typing

   irsim

on the Magic command line. IRSIM is loaded as a Tcl package, and IRSIM and Magic commands may be freely mixed on the Magic command line:

assert [getnode]

IRSIM does not require the name of a file to start. Without arguments, it will assume the currently loaded cell is to be simulated, and it will generate the `.sim` file if it does not already exist.

8. In addition to IRSIM, programs **xcircuit** and **netgen** can be compiled as Tcl extensions and provide cabability for schematic capture and layout-vs.-schematic, respectively.

9. Tcl-based Magic makes use of the Tcl variable space. For instance, it keeps track of the installation directory through the variable `CAD_HOME`, which mirrors the value of the shell environment variable of the same name. In addition, it makes use of variables `VDD` and `GND` in the technology file to aid in the extraction of substrate- and well-connected nodes on transistors.

10. Magic input files, such as the `.magic` startup file, are sourced as Tcl scripts, and so may themselves contain a mixture of Tcl and magic commands.

3 Compilation and Installation

Magic is selected for compilation as a Tcl extention when running the

make config

script. The first question asks whether magic should be compiled as a Tcl extension, with the SCHEME interpreter embedded, or as the original version with no interpreter. Subsequent to choosing Tcl as the interpreter and answering the remaining configuration questions, Tcl should be compiled and installed using the commands

make tcl

and

make install-tcl

Note that if magic is to be compiled under different interpreters, it is necessary to perform

make clean

between the compilations, so that all references to the last compiled version are deleted.

4 Dual-Source Input and Backward Compatibility

From its inception, Magic has used an unusual but very effective interface in which commands may be passed to the program from two different sources, the layout window, and the calling terminal. Keystrokes in the layout window are handled by the graphics package. These are interpreted as *macros*. The keystroke values are expanded into magic command-line commands and executed by the command-line command dispatcher routine. Two macros, '`.`' and '`:`' are reserved: The period expands to the last command executed from the command line (*not* from a macro expansion), and the colon initiates a redirection of keystrokes from the layout window into the calling terminal. Magic users quickly get used to the combination of keystrokes, mouse functions, and command-line commands.

In the Tcl-based version of Magic, the command-line dispatching is given over entirely to Tcl, and the dispatching of keystroke events in the layout window is given over entirely to Tk. Unfortunately, in relinquishing these duties, Magic loses some of the effectiveness of its dual-source input model. One aspect of this is that Tcl is a line-based interpreter, and does not recognize anything on the command line until the return key has been pressed and the entire line is passed to the command dispatcher routine. Without any understanding of character-based input, it is difficult to impossible to directly edit the command line from outside the calling terminal, because it is the terminal, and not Tcl, which interprets keystrokes on a character-by-character basis.

The way around this problem is to use a *console*, which is an application that runs in the manner usually expected by Tk, in that it is a GUI-driven application. The interpreter is split into *master* and *slave* interpreters (see the Tcl documentation on the **interp** command), with the master interpreter running the console application and the slave interpreter running the application. The master interpreter then has character-based control over the command line, and the Magic dual-source input model can be implemented exactly as originally designed. The background DRC function can change the command line cursor from '*%*' to '*]*' to indicate that the DRC is in progress, and user input can be redirected from the layout window to the console with the '**:**' keystroke.

In addition to these functions, the console makes use of Tcl's ability to rename commands to recast the basic Tcl output function '**puts**' in such a way that output to `stdout` and `stdin` can be handled differently. In the **TkCon** console, output to `stdout` is printed in blue, while output to `stderr` is printed in red. Magic makes use of Tcl's output procedures so that returned values and information are printed in blue, while warnings and error messages are printed in red. The console also implements command-line history and cut-and-paste methods. The console command-line history replaces the embedded *readline* implementation in magic.

The **TkCon** console is a placeholder for what is intended to be a "project manager" console, with functions more appropriate to the Electronic Design Automation suite of Tcl-based tools. In general, the menu functions which are displayed on the TkCon console are not of particular interest to the Magic user, with the exception of the **History** menu, which can be used to re-enter previously executed commands, and the **Prefs** menu, which includes highlighting options and a very useful calculator mode.

5 Tk Graphics Methods

Because the graphics under Tcl are managed by the Tk package, the only graphics options which can be compiled are the X11 and the OpenGL options. There are numerous differences between these graphics interfaces as they exist under Tcl and under the non-Tcl-based version. The two primary differences are the way windows are generated and the way commands are sent to specific windows. In Tcl-based magic, a layout window does not have to be a top-level application window. by using the command syntax

openwindow *cellname tk_pathname*

An unmapped window can be generated, corresponding to the Tk window hierarchy specified by *tk_pathname* (see the Tk documentation for the specifics of Tk window path name syntax). This window is then mapped and managed by Tk commands. Using this method, a magic window can be embedded inside a "wrapper" application, an example of which has been done with the GUI wrapper invoked with the "`-w`" command-line argument. Extensions of magic window commands have been added so that the wrapper can control the window frame, including the scrollbars and title.

Whenever a window is created, Magic creates a Tcl/Tk command with the same name as the window (this is conventional practice with Tk widgets). Magic and Tcl commands can be passed as arguments to the window command. Such commands are executed relative to the specific window. This applies to all of magic's window-based commands, including instructions such as **move**, **load**, and so forth. Commands which apply to all windows will automatically be sent to all windows. These commands are used primarily by the wrapper GUI, but are also called (transparently to the end user) whenever a command is executed from a layout window via any macro, including the '**:**' command-line entry. In addition, however, they may be called from the command line to perform an action in a specific window. By default (in the absence of a wrapper GUI), magic's windows are named *.magic1*, *.magic2*, and so forth, in order of appearance, and these names are reflected in the title bar of the window. So it is equivalent to do

:move s 10

from layout window '*.magic2*' (where the colon indicates the key macro for command-line entry), and

.magic2 move s 10

typed in the console.

6 Tutorial Examples

Example sessions of running magic in various modes are presented below, along with examples of methods specific to each mode. In the examples below, prompts are shown to indicate the context of each command. The # sign indicates the shell prompt, *(gdb)* indicates the GNU debugger prompt, and % indicates the Tcl (i.e., Magic) prompt.

Example 1: Standard Magic Execution

Run Tcl-based magic in its most basic form by doing the following:

 # magic -noconsole tut2a

Magic looks generally like its traditional form, except that the command-line prompt is the Tcl '%' prompt. It should be possible to write commands from either the terminal or using the colon keystroke, and it is possible to partially type a command after the colon keystroke and finish the command inside the terminal, but not vice versa. Enabling the colon keystroke may require setting "Allow SendEvents" mode on the calling terminal.

Example 2: Console-based Magic Execution

Run the TkCon console-based magic by doing the following:

 # magic tut2a

The layout window will still look like the usual, basic form. However, the calling terminal will be suspended (unless the application is backgrounded; however, if backgrounding the application, be aware that any output sent to the terminal will hang the application until it is foregrounded) and the Tcl prompt will appear in a new window, which is the **console**. The console may have a slightly different appearance depending on the graphics mode used. For instance, magic has historically had difficulties running in 8-bit (PseudoColor) graphics mode, because it installs its own colormap. Because it does not share the colormap with the calling terminal, the calling terminal gets repainted in random colors from magic's colormap when the cursor is in the layout window. In unlucky setups, text and background may be unreadable.

In the TkCon console setup, the console is mapped prior to determining the graphics mode required, so it also does not share the colormap. However, it is possible to query Magic's colormap to find the location of specific colors, and repaint the text and background of the console accordingly. Thus, the Magic console can be used when the display is in 8-bit PseudoColor mode, without extreme color remappings in the console which make it potentially impossible to read the console when the cursor is in a layout window.

If compiled with both OpenGL and X11 graphics capability, magic will start in X11 mode by default. The OpenGL interface can only be enabled at startup by specifying:

 # magic -d OGL tut4x

The OpenGL interface, in addition to having more solid, vibrant colors, has an additional feature which draws three-dimensional views of a layout. This is discussed in Tutorial #??.

Example 4: The Magic Wrapper GUI

The magic GUI interface is invoked by starting magic with the **-w** option:

 # magic -w tut2b

The immediately noticeable differences are the layer toolbar on the side, and the menu and redesigned title bar on the top. Experimenting with some mouse clicks, the user will note that the magic coordinates of the cursor box are displayed on the right-hand side of the titlebar.

The toolbar contains one example of each layer defined in the magic technology file. Position a box on the screen, then put the cursor over a layer icon and press the middle mouse button. This paints the layer into the box. You will also notice that the name of the layer is printed in the title bar while the cursor is over the layer icon.

The icons have other responses, too. The first and third mouse buttons respectively show and hide the layer in the layout. This works with labels, subcell boundaries, and error paint (the top three layer icons) as well as with regular paintable layers. In addition to mouse button responses, the buttons invoke various commands in reponse to keystrokes. Key 'p' paints the layer; key 'e' erases it. Key 's' selects the layer in the box while key 'S' unselects it.

The menubar has three items, **File**, **Cell**, and **Tech**. Button **File** pops up a menu with options to read and write layout, read and write CIF or GDS output, open a new layout window or close the existing one, or flush the current edit cell. Buttons **Cell** and **Tech** reveal transient windows showing information about the cell hierarchy and the technology file, respectively. The cell hierarchy view is only available if the Tcl/Tk package **BLT** has been compiled and installed on the system. If so, it gives a tree view of the cell hierarchy and allows specific cells to be loaded, edited, and expanded, or viewed to fit the window. This can be expecially useful for querying the cell hierarchy of GDS files, which do not declare top-level cells like CIF files do.

The technology manager window reports the current technology file, its version and description, and the current CIF input and output styles, the current extraction style, and the current value of lambda in microns. The technology file and the CIF input and output styles and extraction style are also buttons which can be used to select a new style or technology from those currently available. Clicking on the current extract style, for instance, gives a list of the styles which have been specified in the current technology file. Clicking on one of the entries in the list makes it the new current extract style.

Cell hierarchy and technology manager windows should be closed by clicking on the "Close" button at the bottom, not by invoking any titlebar functions (which will probably cause the whole application to exit).

Example 5: Batch-mode Magic

Unlike previous versions of magic, it is not necessary to have a layout window present to run magic. Magic may be invoked in "batch mode" by the following command-line:

\# **magic -nowindow tut2b**

Note however, that most magic commands expect a window to be present to execute the function. The main use for batch mode is for wrapper-type applications to delay opening a layout window until one is requested. For example:

\# **magic -nowindow tut2b**
% **toplevel .myframe**
% **openwindow tut2b .myframe.mylayout**
% **pack .myframe.mylayout**

> % **.myframe.mylayout box 0 0 12 12**
> % **.myframe.mylayout select area poly**
> % **wm withdraw .myframe**
> % **wm deiconify .myframe**
> % **.myframe.mylayout closewindow**

Note that this is the basic setup of the standard GUI wrapper, albeit with much greater sophistication. The standard GUI wrapper is generated entirely by script, which can be found in the library directory **${CAD_HOME}/lib/magic/tcl/wrapper.tcl**.

Example 6: Tcl-Magic under the Debugger

When running under Tcl, Magic cannot be debugged in the usual manner of executing, for instance, "**gdb magic**", because the main executable is actually the program "**wish**". To run Magic with debugging capability, it is necessary to do the steps below. In the following, it is assumed that Magic has been installed in the default location `CAD_HOME=/usr/local/`, and that the GNU debugger `gdb` is the debugger of choice.

> \# **gdb wish**
> *(gdb)* **run**
> % **source /usr/local/lib/magic/tcl/magic.tcl**
> .
> .
> .
> % *(type Control-C in the terminal window)*
> *(gdb)* **break TxTclDispatch**
> *(gdb)* **cont**
> % **paint m1**

```
Breakpoint 1, TxTclDispath (clientData=0x0, argc=2,
     argv=0xbffff400)
     at txCommands.c:1146
1146         DRCBreak();
```
(gdb)

Command-line arguments can be passed to magic through the Tcl variables `argc` and `argv`. The integer value `argc` must match the number of entries passed in the `argv` list. For example, do the following:

> \# **gdb wish**
> *(gdb)* **run**
> % **set argc 4**
> % **set argv {-w -d OGL tut1 }**
> % **source /usr/local/lib/magic/tcl/magic.tcl**