NAME

wish − Simple windowing shell

SYNOPSIS

wish ?−**encoding** *name*? ?*fileName arg arg ...*?

OPTIONS

−**encoding** *name*	Specifies the encoding of the text stored in *fileName*. This option is only recognized prior to the *fileName* argument.
−**colormap** *new*	Specifies that the window should have a new private colormap instead of using the default colormap for the screen.
−**display** *display*	Display (and screen) on which to display window.
−**geometry** *geometry*	Initial geometry to use for window. If this option is specified, its value is stored in the **geometry** global variable of the application's Tcl interpreter.
−**name** *name*	Use *name* as the title to be displayed in the window, and as the name of the interpreter for **send** commands.
−**sync**	Execute all X server commands synchronously, so that errors are reported immediately. This will result in much slower execution, but it is useful for debugging.
−**use** *id*	Specifies that the main window for the application is to be embedded in the window whose identifier is *id*, instead of being created as an independent toplevel window. *Id* must be specified in the same way as the value for the −**use** option for toplevel widgets (i.e. it has a form like that returned by the **winfo id** command). Note that on some platforms this will only work correctly if *id* refers to a Tk **frame** or **toplevel** that has its −**container** option enabled.
−**visual** *visual*	Specifies the visual to use for the window. *Visual* may have any of the forms supported by the **Tk_GetVisual** procedure.
− −	Pass all remaining arguments through to the script's **argv** variable without interpreting them. This provides a mechanism for passing arguments such as −**name** to a script instead of having **wish** interpret them.

DESCRIPTION

Wish is a simple program consisting of the Tcl command language, the Tk toolkit, and a main program that reads commands from standard input or from a file. It creates a main window and then processes Tcl commands. If**wish** is in voked with arguments, then the first few arguments, ?−**encoding** *name*? ?*fileName*?, specify the name of a script file, and, optionally, the encoding of the text data stored in that script file. A value for *fileName* is recognized if the appropriate argument does not start with "−".

If there are no arguments, or the arguments do not specify a *fileName*, then wish reads Tcl commands interactively from standard input. It will continue processing commands until all windows have been deleted or until end-of-file is reached on standard input. If there exists a file "**.wishrc**" in the home directory of the user, **wish** evaluates the file as a Tcl script just before reading the first command from standard input.

If arguments to **wish** do specify a *fileName*, then *fileName* is treated as the name of a script file. **Wish** will evaluate the script in *fileName* (which presumably creates a user interface), then it will respond to events until all windows have been deleted. Commands will not be read from standard input. There is no automatic evaluation of "**.wishrc**" when the name of a script file is presented on the**wish** command line, b ut the script file can always **source** it if desired.

Note that on Windows, the **wish***version***.exe** program varies from the **tclsh***version***.exe** program in an additional important way: it does not connect to a standard Windows console and is instead a windowed program. Because of this, it additionally provides access to its own **console** command.

OPTION PROCESSING

Wish automatically processes all of the command-line options described in the **OPTIONS** summary above. Any other command-line arguments besides these are passed through to the application using the **argc** and **argv** variables described later.

APPLICATION NAME AND CLASS

The name of the application, which is used for purposes such as **send** commands, is taken from the **−name** option, if it is specified; otherwise it is taken from *fileName*, if it is specified, or from the command name by which **wish** was invoked. In the last two cases, if the name contains a "/" character, then only the characters after the last slash are used as the application name.

The class of the application, which is used for purposes such as specifying options with a **RESOURCE_MANAGER** property or .Xdefaults file, is the same as its name except that the first letter is capitalized.

VARIABLES

Wish sets the following Tcl variables:

argc Contains a count of the number of *arg* arguments (0 if none), not including the options described above.

argv Contains a Tcl list whose elements are the *arg* arguments that follow a **−−** option or do not match any of the options described in **OPTIONS** above, in order, or an empty string if there are no such arguments.

argv0 Contains *fileName* if it was specified. Otherwise, contains the name by which **wish** was invoked.

geometry If the **−geometry** option is specified, **wish** copies its value into this variable. If the variable still exists after *fileName* has been evaluated, **wish** uses the value of the variable in a **wm geometry** command to set the main window's geometry.

tcl_interactive Contains 1 if **wish** is reading commands interactively (*fileName* was not specified and standard input is a terminal-like device), 0 otherwise.

SCRIPT FILES

If you create a Tcl script in a file whose first line is

#!/usr/local/bin/wish

then you can invoke the script file directly from your shell if you mark it as executable. This assumes that **wish** has been installed in the default location in /usr/local/bin; if it is installed somewhere else then you will have to modify the above line to match. Many UNIX systems do not allow the **#!** line to exceed about 30 characters in length, so be sure that the **wish** executable can be accessed with a short file name.

An even better approach is to start your script files with the following three lines:

```
#!/bin/sh
# the next line restarts using wish \
exec wish "$0" ${1+"$@"}
```

This approach has three advantages over the approach in the previous paragraph. First, the location of the **wish** binary does not have to be hard-wired into the script: it can be anywhere in your shell search path. Second, it gets around the 30-character file name limit in the previous approach. Third, this approach will work even if **wish** is itself a shell script (this is done on some systems in order to handle multiple architectures or operating systems: the **wish** script selects one of several binaries to run). The three lines cause both **sh** and **wish** to process the script, but the **exec** is only executed by **sh**. **sh** processes the script first; it treats the second line as a comment and executes the third line. The **exec** statement cause the shell to stop processing and instead to start up **wish** to reprocess the entire script. When **wish** starts up, it treats all three lines as comments, since the backslash at the end of the second line causes the third line to be treated as part of the comment on the second line.

The end of a script file may be marked either by the physical end of the medium, or by the character, "\032"

("\u001a", control-Z). If this character is present in the file, the **wish** application will read text up to but not including the character. An application that requires this character in the file may encode it as "\032", "\x1a", or "\u001a"; or may generate it by use of commands such as **format** or **binary**.

PROMPTS

When **wish** is invoked interactively it normally prompts for each command with "**%** ". You can change the prompt by setting the variables **tcl_prompt1** and **tcl_prompt2**. If variable **tcl_prompt1** exists then it must consist of a Tcl script to output a prompt; instead of outputting a prompt **wish** will evaluate the script in **tcl_prompt1**. The variable **tcl_prompt2** is used in a similar way when a newline is typed but the current command is not yet complete; if **tcl_prompt2** is not set then no prompt is output for incomplete commands.

SEE ALSO

tclsh(1), toplevel(n), Tk_Main(3), Tk_MainLoop(3), Tk_MainWindow(3)

KEYWORDS

application, argument, interpreter, prompt, script file, shell, toolkit, toplevel