NAME

console − Control the console on systems without a real console

SYNOPSIS

console *subcommand* ?*arg ...*?

DESCRIPTION

The console window is a replacement for a real console to allow input and output on the standard I/O channels on platforms that do not have a real console. It is implemented as a separate interpreter with the Tk toolkit loaded, and control over this interpreter is given through the **console** command. The behaviour of the console window is defined mainly through the contents of the *console.tcl* file in the Tk library. Except for TkAqua, this command is not available when Tk is loaded into a tclsh interpreter with "**package require Tk**", as a conventional terminal is expected to be present in that case. In TkAqua, this command is disabled when there is a startup script and stdin is **/dev/null** (as is the case e.g. when a bundled application embedding Tk is started by the macOS Launcher). To enable the command in that case, define the environment variable **TK_CONSOLE**. This can be done by modifying the Info.plist file by adding the LSEnvironment key to the main dict and setting its value to be a dict with the key **TK_CONSOLE**.

console eval *script*

Evaluate the *script* argument as a Tcl script in the console interpreter. The normal interpreter is accessed through the **consoleinterp** command in the console interpreter.

console hide

Hide the console window from view. Precisely equivalent to withdrawing the **.** window in the console interpreter.

console show

Display the console window. Precisely equivalent to deiconifying the **.** window in the console interpreter.

console title ?*string*?

Query or modify the title of the console window. If*string* is not specified, queries the title of the console window, and sets the title of the console window to *string* otherwise. Precisely equivalent to using the **wm title** command in the console interpreter.

ACCESS TO THE MAIN INTERPRETER

The **consoleinterp** command in the console interpreter allows scripts to be evaluated in the main interpreter. It supports two subcommands: **eval** and **record**.

consoleinterp eval *script*

Evaluates *script* as a Tcl script at the global level in the main interpreter.

consoleinterp record *script*

Records and evaluates *script* as a Tcl script at the global level in the main interpreter as if *script* had been typed in at the console.

ADDITIONAL TRAP CALLS

There are several additional commands in the console interpreter that are called in response to activity in the main interpreter. *These are documented here for completeness only; they form part of the internal implementation of the console and are likely to change or be modified without warning.*

Output to the console from the main interpreter via the stdout and stderr channels is handled by invoking the **tk::ConsoleOutput** command in the console interpreter with two arguments. The first argument is the name of the channel being written to, and the second argument is the string being written to the channel (after encoding and end-of-line translation processing has been performed.)

When the **.** window of the main interpreter is destroyed, the **tk::ConsoleExit** command in the console interpreter is called (assuming the console interpreter has not already been deleted itself, that is.)

DEFAULT BINDINGS

The default script creates a console window (implemented using a text widget) that has the following behaviour:

[1] Pressing the tab key inserts a TAB character (as defined by the Tcl \t escape.)

[2] Pressing the return key causes the current line (if complete by the rules of **info complete**) to be passed to the main interpreter for evaluation.

[3] Pressing the delete key deletes the selected text (if any text is selected) or the character to the right of the cursor (if not at the end of the line.)

[4] Pressing the backspace key deletes the selected text (if any text is selected) or the character to the left of the cursor (of not at the start of the line.)

[5] Pressing either Control+A or the home key causes the cursor to go to the start of the line (but after the prompt, if a prompt is present on the line.)

[6] Pressing either Control+E or the end key causes the cursor to go to the end of the line.

[7] Pressing either Control+P or the up key causes the previous entry in the command history to be selected.

[8] Pressing either Control+N or the down key causes the next entry in the command history to be selected.

[9] Pressing either Control+B or the left key causes the cursor to move one character backward as long as the cursor is not at the prompt.

[10] Pressing either Control+F or the right key causes the cursor to move one character forward.

[11] Pressing F9 rebuilds the console window by destroying all its children and reloading the Tcl script that defined the console's behaviour.

Most other behaviour is the same as a conventional text widget except for the way that the <<*Cut*>> event is handled identically to the <<*Copy*>> event.

EXAMPLE

Not all platforms have the **console** command, so debugging code often has the following code fragment in it so output produced by **puts** can be seen while during development:

```
catch {console show}
```

SEE ALSO

destroy(n), fconfigure(n), history(n), interp(n), puts(n), text(n), wm(n)

KEYWORDS

console, interpreter, window, interactive, output channels