NAME

elk, scheme-elk – extensible Scheme interpreter

SYNOPSIS

elk [**–l** *file*] [**–h** *KBytes*] [**–p** *load-path*] [**–g**] [**–i**] [**–v** *type*] [[**––**] *args*]

elk. . .

DESCRIPTION

Elk (Extension Language Kit) is a Scheme implementation designed as a general extension language for applications written in C or C++. Normally, *Elk* is linked with the application it serves, but a stand-alone version of the Scheme interpreter is installed as well (usually under the name **elk**). This interpreter, together with the standard Scheme toplevel, *Elk* can be used as an ordinary, stand-alone implementation of the Scheme language.

When called without the **–l** option, *Elk* loads the standard "toplevel" to start an interactive session. When called with **–l** *file*, the contents of the specified file is loaded instead. If a '–' is given as a filename argument, *Elk* loads from standard input.

The option **–p** *load-path* can be used to override the standard *load-path*. The argument is a colon-separated list of directories. If this option is not present and the environment variable ELK_LOADPATH is defined, the value of this variable is used to initialize the *load-path*. The value of ELK_LOADPATH has the same format as the argument to the **-p** option.

The **–h** *KBytes* option is used to specify a non-standard heap size. The default heap size is 512 KBytes.

If the option **–i** is specified, symbols are mapped to lower case. Normally, *Elk* is case-sensitive.

The **–g** option causes the interpreter to run the garbage collector each time memory is allocated on the heap. This is useful for writers of extensions who want to test the garbage collect behavior of an extension. Running *Elk* with the **–g** option is likely to reveal GC-related bugs in extensions (such as not properly protected local objects), as it triggers a garbage collection each time an object is allocated on the Scheme heap. A dot is written to standard output each time a garbage collection is performed when **–g** has been specified.

When called with one or more **–v** *type* (''verbose'') options, the interpreter prints additional informational messages to standard output, depending on the value of the *type* argument. If *type* is *load*, the linker command and options are printed each time an object file is loaded; if *type* is *init*, the names of extension initialization and finalization functions are printed as they are called.

The remaining *args* are put into a list of strings, and the Scheme variable **command-line-args** is bound to this list in the global environment. If arguments could be interpreted as options, '−-´ can be used to indicate the end of the options.

FILES

$TMPDIR/ldXXXXXX Temporary files

AUTHOR

Oliver Laumann