URL: http://cxc.harvard.edu/ciao3.4/slsh.html
Last modified: December 2006

AHELP for CIAO 3.4

slsh

Context: slang

Jump to: Description Examples CHANGES FUNCTIONS AND VARIABLES AVAILABLE IN SLSH STARTING SLSH ACKNOWLEDGEMENTS See Also

Synopsis

Evaluate and run S−Lang code.

Syntax

```
slsh [OPTIONS] [[-|file] [arguments...]]
slsh --help
```

Description

The slsh program evaluates the S−Lang code given to it, either as a file or read from STDIN (the "−" option). All functions and variables in the S−Lang Run−Time library (see "ahelp slangrtl") can be used, and slsh adds several additional functions, as described in the "Functions and Variables available in slsh" section. A more−complete description of how slsh starts up can be found in the "Starting slsh" section below.

Command−line options

The following options can be used with slsh:

Option	Description
−−help	Print usage information.
−−version	Show slsh version information.
−g	Compile with debugging code, tracebacks, etc.
−n	Don't load personal init file.
−i init−file	Use the specified file instead of the default.
−v	Show verbose loading messages.

Example 1

```
unix% slsh myprog.sl
```

This command will execute the code in the file myprog.sl. Note that the files do not have to end in ".sl", but it is a useful idiom. If the contents of myprog.sl were:

```
  vmessage( "The time is %s.", time() );
```

then the output would look something like:

```
  unix% slsh myprog.sl
  The time is Fri Mar 19 11:28:40 2004
```

Both the vmessage() and time() functions are part of the S−Lang Run−Time Library, and so are available for use by slsh (see "ahelp slangrtl" for more information).

Example 2

If you start a S−Lang script with the line

```
#!/usr/bin/env slsh
```

and set it to be an executable, then you can run the script without having to explicitly call slsh. So if the file myprog.sl looked like:

```
  #!/usr/bin/env slsh
  vmessage( "The time is %s.", time() );
```

then the script could be run by just saying:

```
  unix% ./myprog.sl
  The time is Fri Mar 19 11:28:40 2004.
```

assuming that

```
chmod u+x myprog.sl
```

had previously been called. The following examples will use this method.

Example 3

Code that uses CIAO modules − such as paramio, varmm, or sherpa − can also be run using slsh. All that needs to be done is to load the required modules before using any functions they define. In the following example we use the region module to calculate the area of a simple region.

```
  unix% cat reg.sl
  #!/usr/bin/env slsh
  require("region");
  variable reg = regParse( "circle(4300.45,3274.22,60.3)" );
  vmessage( "The region area is: %7.2f", regArea(reg) );
```

which, when run, produces

```
  unix% ./reg.sl
```

```
The region area is: 11423.11
```

Example 4

Command−line arguments that are not recognised by slsh are made available to the S−Lang code via the __argc and __argv variables. These are analogous to the argc and argv variables of the main() routine in C code. We can use these variables to enhance the previous example to allow regions to be specified on the command line, rather than being written into the code.

```
unix% cat reg2.sl
#!/usr/bin/env slsh

% check called correctly
%
if ( 2 != __argc ) {
  () = fprintf( stderr, "Usage: %s <region>\n", __argv[0] );
  exit(1);
}

require("region");

% has the user given us a valid region?
%
variable reg = regParse( __argv[1] );
if ( NULL == reg ) {
  () = fprintf( stderr, "Did not recognise %s as a region!\n",
                __argv[1] );
  exit(1);
}

% print out the region area
%
vmessage( "The region area is: %7.2f", regArea(reg) );
```

The code has been enhanced to check that the correct number of arguments has been given and that the user−supplied argument is recognised as a region. As an example of its use:

```
unix% ./reg2.sl
Usage: ./reg2.sl <region>
unix% ./reg2.sl "circle(4200,3200,60.3)"
The region area is: 11423.11
unix% ./reg2.sl "annulus(4200,3200,10,60.3)"
The region area is: 11108.95
```

CHANGES

CIAO 3.2

The CIAO environment is now set up so that the require() and provide() routines, amongst others, are available to code evaluated by slsh. This is described in the "Using slsh in the CIAO environment" part of the "STARTING SLSH" section below.

FUNCTIONS AND VARIABLES AVAILABLE IN SLSH

When evaluating S−Lang code by slsh you can take advantage of the following functions and variables: __argc; __argv; exit(); atexit(); and stat_mode_to_string().

__argc

This is a read−only integer variable which contains the number of arguments on the command line. It is analogous to the argc argument of the main() function in the C language.

__argv

This is a read−only String_Type array which contains the arguments on the command line. It is analogous to the argv argument of the main() function in the C language.

exit(status)

This function terminates the slsh program and uses the value of the argument (status) as the exit status. All hooks set up by the atexit() function are called before slsh exits.

atexit(function)

The atexit() function tells the system to call the supplied function (which should be sent in as a S−Lang reference) when the interpreter is about to exit due to a call to exit(). These functions are not called if the code finishes without explicitly calling exit().

The functions are pushed onto a stack which is then popped from on exit, so the function used in the last call to atexit() will be the first one called.

stat_mode_to_string(mode)

This function converts the mode of a file (the st_mode field of the structure returned by the stat_file() function) to a string in the format used by "ls −l"; e.g. "−rw−r−−r−−" for a file which everyone can read but only the owner can change.

STARTING SLSH

Upon startup, the program will try to load slsh.rc as follows: If either SLSH_CONF_DIR or SLSH_LIB_DIR environment variables exist, then look in the corresponding directories for slsh.rc. Otherwise look in:

```
  /usr/local/etc/
  /usr/local/etc/slsh/
  /etc/
  /etc/slsh/
```

The slsh.rc file may load other files from slsh's library directory in the manner described below.

Once slsh.rc is loaded, slsh will load $HOME/.slshrc if present. Finally, it will load the script specified on the command line. If the name of the script is "−", then it will be read from stdin.

When a script loads a file via the built−in "evalfile" function or the "require" function (autoloaded by slsh.rc), the file is searched for along the SLSH_PATH as specified in the Makefile. An alternate path may be specified by the SLSH_PATH environment variable.

The search path may be queried and set during run time via set the get_slang_load_path and set_slang_load_path functions, e.g.,

```
set_slang_load_path("/home/bill/lib/slsh:/usr/share/slsh");
```

Using slsh in the CIAO environment

The CIAO environment sets up the following environment variables:

Environment variable	Value
SLSH_CONF_DIR	$ASCDS_INSTALL/etc
SLSH_PATH	$ASCDS_INSTALL/share/slsh
SLANG_SCRIPT_PATH	$ASCDS_INSTALL/share/slsh/local−packages:$ASCDS_INSTALL/contrib/share/slsh/local−
SLANG_MODULE_PATH	$ASCDS_INSTALL/lib/slang/modules:$ASCDS_INSTALL/contrib/lib/slang/modules

The file $ASCDS_INSTALL/etc/slsh.rc, which is automatically loaded by slsh, makes several functions available − these are provide(), require(), reverse(), shift(), prepend_to_slang_load_path(), append_to_slang_load_path(), and _featurep() − and sets up the search path for the CIAO modules.

ACKNOWLEDGEMENTS

The information in this page was partly taken from documentation in version 1.4.9 of the S−Lang distribution, available from the S−Lang home page.

See Also

slangrtl
    slangrtl