S-Lang Library C Programmer's Guide

John E. Davis, davis@space.mit.edu

Sun Jan 25 22:41:18 1998

Preface

S-Lang is an interpreted language that was designed from the start to be easily embedded into a program to provide it with a powerful extension language. Examples of programs that use **S-Lang** as an extension language include the **jed** text editor, the **slrn** newsreader, and **sldxe** (unreleased), a numerical computation program. For this reason, **S-Lang** does not exist as a separate application and many of the examples in this document are presented in the context of one of the above applications.

S-Lang is also a programmer's library that permits a programmer to develop sophisticated platform-independent software. In addition to providing the **S-Lang** extension language, the library provides facilities for screen management, keymaps, low-level terminal I/O, etc. However, this document is concerned only with the extension language and does not address these other features of the **S-Lang** library. For information about the other components of the library, the reader is referred to the **The S-Lang Library Reference**.

A Brief History of S-Lang

I first began working on **S-Lang** sometime during the fall of 1992. At that time I was writing a text editor (**jed**), which I wanted to endow with a macro language. It occured to me that an application-independent language that could be embedded into the editor would prove more useful because I could envision embedding it into other programs. As a result, **S-Lang** was born.

S-Lang was originally a stack language that supported a postscript-like syntax. For that reason, I named it **S-Lang**, where the *S* was supposed to emphasize its stack-based nature. About a year later, I began to work on a preparser that would allow one to write using a more traditional infix syntax making it easier to use for those unfamiliar with stack based languages. Currently, the syntax of the language resembles C, nevertheless some postscript-like features still remain, e.g., the '%' character is still used as a comment delimiter.

Acknowledgements

Since I first released **S-Lang**, I have received a lot feedback about the library and the language from many people. This has given me the opportunity and pleasure to interact with several people to make the library portable and easy to use. In particular, I would like to thank the following individuals:

Luchesar Ionkov `<lionkov@sf.cit.bg>` for his comments and criticisms of the syntax of the language. He was the person who made me realize that the low-level byte-code engine should be totally type-independent. He also improved the tokenizer and preparser and impressed upon me that the language needed a grammar.

Mark Olesen `<olesen@weber.me.queensu.ca>` for his many patches to various aspects of the library and his support on AIX. He also contributed a lot to the pre-processing (`SLprep`) routines.

John Burnell `<j.burnell@irl.cri.nz>` for the OS/2 port of the video and keyboard routines. He also made value suggestions regarding the interpreter interface.

Darrel Hankerson `<hankedr@mail.auburn.edu>` for cleaning up and unifying some of the code and the makefiles.

Dominik Wujastyk `<ucgadkw@ucl.ac.uk>` who was always willing to test new releases of the library.

Michael Elkins `<me@muddcs.cs.hmc.edu>` for his work on the curses emulation.

Ulli Horlacher `<framstag@belwue.de>` and Oezguer Kesim `<kesim@math.fu-berlin.de>` for the **S-Lang** newsgroup and mailing list.

Hunter Goatley, Andy Harper `<Andy.Harper@kcl.ac.uk>`, and Martin P.J. Zinser `<m.zinser@gsi.de>` for their VMS support.

I am also grateful to many other people who send in bug-reports and bug-fixes, for without such community involvement, **S-Lang** would not be as well-tested and stable as it is. Finally, I would like to thank my wife for her support and understanding while I spent long weekend hours developing the library.

Contents

Chapter 1

Introduction

S-Lang is a C programmer's library that includes routines for the rapid development of sophisticated, user friendly, multi-platform applications. The **S-Lang** library includes the following:

- Low level tty input routines for reading single characters at a time.
- Keymap routines for defining keys and manipulating multiple keymaps.
- A high-level keyprocessing interface (`SLkp`) for handling function and arrow keys.
- High level screen management routines for manipulating both monochrome and color terminals. These routines are *very* efficient. (`SLsmg`)
- Low level terminal-independent routines for manipulating the display of a terminal. (`SLtt`)
- Routines for reading single line input with line editing and recall capabilities. (`SLrline`)
- Searching functions: both ordinary searches and regular expression searches. (`SLsearch`)
- An embedded stack-based language interpreter with a C-like syntax.

The library is currently available for OS/2, MSDOS, Unix, and VMS systems. For the most part, the interface to library routines has been implemented in such a way that it appears to be platform independent from the point of view of the application. In addition, care has been taken to ensure that the routines are "independent" of one another as much as possible. For example, although the keymap routines require keyboard input, they are not tied to **S-Lang**'s keyboard input routines—one can use a different keyboard `getkey` routine if one desires. This also means that linking to only part of the **S-Lang** library does not pull the whole library into the application. Thus, **S-Lang** applications tend to be relatively small in comparison to programs that use libraries with similar capabilities.

Chapter 2

Interpreter Interface

The **S-Lang** library provides an interpreter that when embedded into an application, makes the application extensible. Examples of programs that embed the interpreter include the **jed** editor and the **slrn** newsreader.

Embedding the interpreter is easy. The hard part is to decide what application specific built-in or intrinsic functions should be provided by the application. The **S-Lang** library provides some pre-defined intrinsic functions, such as string processing functions, and simple file input-output routines. However, the basic philosophy behind the interpreter is that it is not a standalone program and it derives much of its power from the application that embeds it.

2.1 Embedding the Interpreter

Only one function needs to be called to embed the **S-Lang** interpreter into an application: `SLang_init_slang`. This function initializes the interpreters data structures and adds some intinsic functions:

```
if (-1 == SLang_init_slang ())
  exit (EXIT_FAILURE);
```

This function does not provide file input output intrinsic nor does it provide mathematical functions. To make these as well as some unix system calls available use

```
if ((-1 == SLang_init_slang ())    /* basic interpreter functions */
    || (-1 == SLang_init_slmath ()) /* sin, cos, etc... */
    || (-1 == SLang_init_slunix ()) /* unix system calls */
    || (-1 == SLang_init_slfile ())) /* file I/O */
  exit (EXIT_FAILURE);
```

See the \slang-run-time-library for more information about the intrinsic functions.

2.2 Calling the Interpreter

There are several ways of calling the interpreter. The most common method used by both **jed** and **slrn** is to use the `SLang_load_file` function to interprete a file. For example, **jed** starts by loading a file called `site.sl`:

```
if (-1 == SLang_load_file ("site.sl"))
  SLang_restart (1);
```

The `SLang_load_file` function returns zero upon if successful, or `-1` upon failure. The `SLang_restart` function resets the interpreter back to its default state.

There are several other mechanisms for interacting with the interpreter. For example, the `SLang_load_string` function loads a string into the interpreter and interprets it:

```
if (-1 == SLang_load_string ("message (\"hello\");"))
  SLang_restart (1);
```

Typically, an interactive application will load a file via `SLang_load_file` and then go into a loop that consists of reading lines of input and sending them to the interpreter, e.g.,

```
while (EOF != fgets (buf, sizeof (buf), stdin))
  {
     if (-1 == SLang_load_string (buf))
       SLang_restart (1);
  }
```

Both **jed** and **slrn** use another method of interacting with the interpreter. They read key sequences from the keyboard and map those key sequences to interpreter functions via the **S-Lang** keymap interface.

2.3 Intrinsic Functions

An intrinsic function is simply a function that is written in C and is made available to the interpreter as a built-in function. For this reason, the words 'intrinsic' and 'built-in' are often used interchangeably.

Applications are expected to add application specific functions to the interpreter. For example, **jed** adds nearly 300 editor-specific intrinsic functions. The application designer should think carefully about what intrinsic functions to add to the interpreter.

2.3.1 Restrictions on Intrinsic Functions

Intrinsic functions are required to follow a few rules to cooperate with the interpreter.

Intrinsic function must take only pointer arguments. This is because when the interpreter calls an intrinsic function, it passes value to the function by reference and *not* by value. For example, intrinsic with the declarations:

```
int intrinsic_0 (void);
int intrinsic_1 (char *s);
void intrinsic_2 (char *s, int *i);
void intrinsic_3 (int *i, double *d, double *e);
```

are all valid. However,

```
int invalid_1 (char *s, int len);
```

is not valid since the `len` parameter is not a pointer.

Intrinsic functions can only return `void`, `int`, `double`, or `char *`. A function such as

```
int *invalid (void);
```

is not permitted since it does not return one of these types. The current implementation limits the number of arguments to 7.

Another restriction is that the intrinsic should regard all its parameters as pointers to constant objects and make no attempt to modify the value to which they point. For example,

```
void truncate (char *s)
{
   s[0] = 0;
}
```

is illegal since the function modifies the string `s`.

2.3.2 Adding a New Intrinsic

There are two mechanisms for adding an intrinsic function to the interpreter: `SLadd_intrinsic_function` and `SLadd_intrin_fun_table`.

As an specific example, consider a function that will cause the program to exit via the `exit` C library function. It is not possible to make this function an intrinsic because it does not meet the specifications for an intrinsic function that were described earlier. However, one can call `exit` from a function that is suitable, e.g.,

```
void intrin_exit (int *code)
{
   exit (*code);
}
```

This function may be made available to the interpreter as as an intrinsic via the `SLadd_intrinsic_function` routine:

```
if (-1 == SLadd_intrinsic_function ("exit", (FVOID_STAR) intrin_exit,
                                    SLANG_VOID_TYPE, 1,
                                    SLANG_INT_TYPE))
  exit (EXIT_FAILURE);
```

This statement basically tells the interpreter that `intrin_exit` is a function that returns nothing and takes a single argument: a pointer to an integer (`SLANG_INT_TYPE`). A user can call this function from within the interpreter via

```
message ("Calling the exit function");
exit (0);
```

After printing a message, this will cause the `intrin_exit` function to execute, which in turn calls `exit`.

The most convenient mechanism for adding new intrinsic functions is to create a table of `SLang_Intrin_Fun_Type` objects and add the table via the `SLadd_intrin_fun_table` function. The table will look like:

```
SLang_Intrin_Fun_Type My_Intrinsics [] =
{
 /* table entries */
  MAKE_INTRINSIC_N(...),
  MAKE_INTRINSIC_N(...),
        .
        .
  MAKE_INTRINSIC_N(...),
  SLANG_END_TABLE
};
```

Construction of the table entries may be facilitated using a set of `MAKE_INTRINSIC` macros defined in `slang.h`. The main macro is called `MAKE_INTRINSIC_N` and takes ?? arguments:

```
MAKE_INTRINSIC_N(name, funct-ptr, return-type, num-args,
                 arg-1-type, arg-2-type, ... arg-7-type)
```

Here `name` is the name of the intrinsic function that the interpreter is to give to the function. `func-ptr` is a pointer to the intrinsic function taking `num-args` and returning `ret-type`. The final 7 arguments specifiy the argument types. For example, the `intrin_exit` intrinsic described above may be added to the table using

```
MAKE_INTRINSIC_N("exit", intrin_exit, SLANG_VOID_TYPE, 1,
                 SLANG_INT_TYPE, 0,0,0,0,0,0)
```

While `MAKE_INTRINSIC_N` is the main macro for constructing table entries, `slang.h` defines other macros that may prove useful. In particular, an entry for the `intrin_exit` function may also be created using any of the following forms:

```
MAKE_INTRINSIC_1("exit", intrin_exit, SLANG_VOID_TYPE, SLANG_INT_TYPE)
MAKE_INTRINSIC_I("exit", intrin_exit, SLANG_VOID_TYPE)
```

See `slang.h` for related macros. You are also encouraged to look at, e.g., `slang/src/slstd.c` for a more extensive examples.

2.3.3 More Complicated Intrinsics

The intrinsic functions described in the previous example were functions that took a fixed number of arguments. In this section we explore more complex intrinsics such as those that take a variable number of arguments.

Consider a function that takes two double precision numbers and returns the lesser:

```
double intrin_min (double *a, double *b)
{
   if (*a < *b) return *a;
   return *b;
}
```

This function may be added to a table of intrinsics using

```
MAKE_INTRINSIC_2("min", intrin_min, SLANG_DOUBLE_TYPE,
                 SLANG_DOUBLE_TYPE, SLANG_DOUBLE_TYPE)
```

It is useful to extend this function to take an arbitray number of arguments and return the lesser. Consider the following variant:

```
double intrin_min_n (int *num_ptr)
{
   double min_value, x;
   unsigned int num = (unsigned int) *num_ptr;

   if (-1 == SLang_pop_double (&min_value, NULL, NULL))
     return 0.0;
   num--;

   while (num > 0)
     {
        num--;
        if (-1 == SLang_pop_double (&x, NULL, NULL))
          return 0.0;
        if (x < min_value) min_value = x;
     }
   return min_value;
}
```

Here the number to compare is passed to the function and the actual numbers are removed from the stack via the `SLang_pop_double` function. A suitable table entry for it is

```
MAKE_INTRINSIC_I("min", intrin_min_n, SLANG_DOUBLE_TYPE)
```

This function would be used in an interpreter script via a statement such as

```
variable xmin = min (x0, x1, x2, x3, x4, 5);
```

which computes the smallest of 5 values.

The problem with this intrinsic function is that the user must explicitly specify how many numbers to compare. It would be more convenient to simply use

```
variable xmin = min (x0, x1, x2, x3, x4);
```

An intrinsic function can query the value of the variable `SLang_Num_Function_Args` to obtain the necessary information:

```
double intrin_min (void)
{
   double min_value, x;

   unsigned int num = SLang_Num_Function_Args;

   if (-1 == SLang_pop_double (&min_value, NULL, NULL))
     return 0.0;
   num--;

   while (num > 0)
     {
        num--;
        if (-1 == SLang_pop_double (&x, NULL, NULL))
          return 0.0;
        if (x < min_value) min_value = x;
     }
   return min_value;
}
```

This may be declared as an intrinsic using:

```
MAKE_INTRINSIC_0("min", intrin_min, SLANG_DOUBLE_TYPE)
```

2.4 Intrinsic Variables

It is possible to access an application's global variables from within the interpreter. The current implementation supports the access of variables of type `int`, `char *`, and `double`.

There are two methods of making an intrinsic variable available to the interpreter. The most straight forward method is to use the function `SLadd_intrinsic_variable`:

```
int SLadd_intrinsic_variable (char *name, VOID_STAR addr,
                              unsigned char data_type,
                              int read_only);
```

For example, suppose that `I` is an integer variable, e.g.,

```
int I;
```

One can make it known to the interpreter as `I_Variable` via a statement such as

```
 if (-1 == SLadd_intrinsic_variable ("I_Variable", &I,
                                     SLANG_INT_TYPE, 0))
   exit (EXIT_FAILURE);
```

Similarly, if `S` is declared as

```
char *S;
```

then

```
 if (-1 == SLadd_intrinsic_variable ("S_Variable", &S,
                                     SLANG_STRING_TYPE, 1))
   exit (EXIT_FAILURE);
```

makes `S` available as a *read-only* variable with the name `S_Variable`. Note that if a pointer variable is made available to the interpreter, its value is managed by the interpreter and not the application. For this reason, it is recommended that such variables be declared as *read-only*.

It is important to note that if `S` were declared as an array of characters, e.g.,

```
 char S[256];
```

then it would not be possible to make it directly available to the interpreter. However, one could create a pointer to it, i.e.,

```
 char *S_Ptr = S;
```

and make `S_Ptr` available as a read-only variable.

The most convenient method for adding many intrinsic variables to the interpreter is to create an array of `SLang_Intrin_Var_Type` objects and then add the array via `SLadd_intrin_var_table`. For example, the array

```
static SLang_Intrin_Var_Type Intrin_Vars [] =
{
   MAKE_VARIABLE("I_Variable", &I, SLANG_INT_TYPE, 0),
   MAKE_VARIABLE("S_Variable", &S_Ptr, SLANG_STRING_TYPE, 1),
   SLANG_END_TABLE
};
```

may be added via

```
if (-1 == SLadd_intrin_var_table (Intrin_Vars, NULL))
  exit (EXIT_FAILURE);
```

It should be rather obvious that the arguments to the `MAKE_VARIABLE` macro correspond to the parameters of the `SLadd_intrinsic_variable` function.

Chapter 3

Keyboard Interface

Overview

S-Lang's keyboard interface has been designed to allow an application to read keyboard input from the user in a system-independent manner. The interface consists of a set of low routines for reading single character data as well as a higher level interface (`SLkp`) which utilize **S-Lang**'s keymap facility for reading multi-character sequences.

To initialize the interface, one must first call the function `SLang_init_tty`. Before exiting the program, the function `SLang_reset_tty` must be called to restore the keyboard interface to its original state. Once initialized, the low-level `SLang_getkey` function may be used to read *simgle* keyboard characters from the terminal. An application using the the higher-level `SLkp` interface will read charcters using the `SLkp_getkey` function.

In addition to these basic functions, there are also functions to "unget" keyboard characters, flush the input, detect pending-input with a timeout, etc. These functions are defined below.

3.1 Initializing the Keyboard Interface

The function `SLang_init_tty` must be called to initialize the terminal for single character input. This puts the terminal in a mode usually referred to as "raw" mode.

The prototype for the function is:

```
int SLang_init_tty (int abort_char, int flow_ctrl, int opost);
```

It takes three parameters that are used to specify how the terminal is to be initialized. %Although the **S-Lang** keyboard interface has been %designed to be as system independent as possible, there are semantic % differences.

The first parameter, `abort_char`, is used to specify the interrupt character (`SIGINT`). Under MSDOS, this value corresponds to the scan code of the character that will be used to generate the interrupt. For example, under MSDOS, `34` should be used to make `Ctrl-G` generate an interrupt signal since 34 is the scan code for `G`. On other systems, the value of `abort_char` will simply be the ascii value of the control character that will be used to generate the interrupt signal, e.g., `7` for `Ctrl-G`. If `-1` is passed, the interrupt character will not be changed.

Pressing the interrupt character specified by the first argument will generate a signal (`SIGINT`) that may or not be caught by the application. It is up to the application to catch this signal. **S-Lang** provides the function `Slang_set_abort_signal` to make it easy to facilitate this task.

The second parameter is used to specify whether or not flow control should be used. If this parameter is non-zero, flow control is enabled otherwise it is disabled. Disabling flow control is necessary to pass certain characters to the application (e.g., `Ctrl-S` and `Ctrl-Q`). For some systems such as MSDOS, this parameter is meaningless.

The third parameter, `opost`, is used to turn output processing on or off. If `opost` is zero, output processing is *not* turned on otherwise, output processing is turned on.

The `SLang_init_tty` function returns -1 upon failure. In addition, after it returns, the **S-Lang** global variable `SLang_TT_Baud_Rate` will be set to the baud rate of the terminal if this value can be determined.

Example:

```
if (-1 == SLang_init_tty (7, 0, 0))  /* For MSDOS, use 34 as scan code */
  {
    fprintf (stderr, "Unable to initialize the terminal.\n");
    exit (1);
  }
SLang_set_abort_signal (NULL);
```

Here the terminal is initialized such that flow control and output processing are turned off. In addition, the character `Ctrl-G`[1] has been specified to be the interrupt character. The function `SLang_set_abort_signal` is used to install the default **S-Lang** interrupt signal handler.

3.2 Resetting the Keyboard Interface

The function `SLang_reset_tty` must be called to reset the terminal to the state it was in before the call to `SLang_init_tty`. The prototype for this function is:

```
void SLang_reset_tty (void);
```

Usually this function is only called before the program exits. However, if the program is suspended it should also be called just before suspension.

3.3 Initializing the `SLkp` Routines

Extra initialization of the higher-level `SLkp` functions are required because they are layered on top of the lower level routines. Since the `SLkp_getkey` function is able to process function and arrow keys in a terminal independent manner, it is necessary to call the `SLtt_get_terminfo` function to get information about the escape character sequences that the terminal's function keys send. Once that information is available, the `SLkp_init` function can construct the proper keymaps to process the escape sequences.

[1]For MSDOS systems, use the *scan code* 34 instead of 7 for `Ctrl-G`

This part of the initialization process for an application using this interface will look something like:

```
SLtt_get_terminfo ();
if (-1 == SLkp_init ())
  {
     SLang_doerror ("SLkp_init failed.");
     exit (1);
  }
if (-1 == SLang_init_tty (-1, 0, 1))
  {
     SLang_doerror ("SLang_init_tty failed.");
     exit (1);
  }
```

It is important to check the return status of the `SLkp_init` function which can failed if it cannot allocate enough memory for the keymap.

3.4 Setting the Interrupt Handler

The function `SLang_set_abort_signal` may be used to associate an interrupt handler with the interrupt character that was previously specified by the `SLang_init_tty` function call. The prototype for this function is:

```
void SLang_set_abort_signal (void (*)(int));
```

This function returns nothing and takes a single parameter which is a pointer to a function taking an integer value and returning `void`. If a `NULL` pointer is passed, the default **S-Lang** interrupt handler will be used. The **S-Lang** default interrupt handler under Unix looks like:

```
static void default_sigint (int sig)
{
  SLsignal_intr (SIGINT, default_sigint);
  SLKeyBoard_Quit = 1;
  if (SLang_Ignore_User_Abort == 0) SLang_Error = USER_BREAK;
}
```

It simply sets the global variable `SLKeyBoard_Quit` to one and if the variable `SLang_Ignore_User_Abort` is non-zero, `SLang_Error` is set to indicate a user break condition. (The function `SLsignal_intr` is similar to the standard C `signal` function *except that it will interrupt system calls*. Some may not like this behavior and may wish to call this `SLang_set_abort_signal` with a different handler.)

Although the function expressed above is specific to Unix, the analogous routines for other operating systems are equivalent in functionality even though the details of the implementation may vary drastically (e.g., under MSDOS, the hardware keyboard interrupt `int 9h` is hooked).

3.5 Reading Keyboard Input with SLang_getkey

After initializing the keyboard via `SLang_init_tty`, the **S-Lang** function `SLang_getkey` may be used to read characters from the terminal interface. In addition, the function `SLang_input_pending` may be used to determine whether or not keyboard input is available to be read.

These functions have prototypes:

```
unsigned int SLang_getkey (void);
int SLang_input_pending (int tsecs);
```

The `SLang_getkey` function returns a single character from the terminal. Upon failure, it returns `0xFFFF`. If the interrupt character specified by the `SLang_init_tty` function is pressed while this function is called, the function will return the value of the interrupt character and set the **S-Lang** global variable `SLKeyBoard_Quit` to a non-zero value. In addition, if the default **S-Lang** interrupt handler has been specified by a `NULL` argument to the `SLang_set_abort_signal` function, the global variable `SLang_Error` will be set to `USER_BREAK` *unless* the variable `SLang_Ignore_User_Abort` is non-zero.

The `SLang_getkey` function waits until input is available to be read. The `SLang_input_pending` function may be used to determine whether or not input is ready. It takes a single parameter that indicates the amount of time to wait for input before returning with information regarding the availability of input. This parameter has units of one tenth (1/10) of a second, i.e., to wait one second, the value of the parameter should be `10`. Passing a value of zero causes the function to return right away. `SLang_input_pending` returns a positive integer if input is available or zero if input is not available. It will return -1 if an error occurs.

Here is a simple example that reads keys from the terminal until one presses `Ctrl-G` or until 5 seconds have gone by with no input:

```
#include <stdio.h>
#include "slang.h"
int main ()
{
   int abort_char = 7;  /* For MSDOS, use 34 as scan code */
   unsigned int ch;

   if (-1 == SLang_init_tty (abort_char, 0, 1))
     {
        fprintf (stderr, "Unable to initialize the terminal.\n");
        exit (-1);
     }
   SLang_set_abort_signal (NULL);
   while (1)
     {
        fputs ("\nPress any key.  To quit, press Ctrl-G: ", stdout);
        fflush (stdout);
        if (SLang_input_pending (50) == 0)  /* 50/10 seconds */
          {
             fputs ("Waited too long! Bye\n", stdout);
             break;
```

```
          }

         ch = SLang_getkey ();
         if (SLang_Error == USER_BREAK)
           {
              fputs ("Ctrl-G pressed!  Bye\n", stdout);
              break;
           }
         putc ((int) ch, stdout);
      }
    SLang_reset_tty ();
    return 0;
 }
```

3.6 Reading Keyboard Input with SLkp_getkey

Unlike the low-level function `SLang_getkey`, the `SLkp_getkey` function can read a multi-character sequence associated with function keys. The `SLkp_getkey` function uses `SLang_getkey` and **S-Lang**'s keymap facility to process escape sequences. It returns a single integer which describes the key that was pressed:

```
int SLkp_getkey (void);
```

That is, the `SLkp_getkey` function simple provides a mapping between keys and integers. In this context the integers are called *keysyms*.

For single character input such as generated by the `a` key on the keyboard, the function returns the character that was generated, e.g., `'a'`. For single characters, `SLkp_getkey` will always return an keysym whose value ranges from 0 to 256. For keys that generate multiple character sequences, e.g., a function or arrow key, the function returns an keysym whose value is greater that 256. The actual values of these keysyms are represented as macros defined in the `slang.h` include file. For example, the up arrow key corresponds to the keysym whose value is `SL_KEY_UP`.

Since it is possible for the user to enter a character sequence that does not correspond to any key. If this happens, the special keysym `SL_KEY_ERR` will be returned.

Here is an example of how `SLkp_getkey` may be used by a file viewer:

```
switch (SLkp_getkey ())
  {
     case ' ':
     case SL_KEY_NPAGE:
        next_page ();
        break;
     case 'b':
     case SL_KEY_PPAGE:
        previous_page ();
        break;
     case '\r':
     case SL_KEY_DOWN:
```

```
          next_line ();
          break;
           .
           .
       case SL_KEY_ERR:
       default:
          SLtt_beep ();
    }
```

Unlike its lower-level counterpart, SLang_getkey, there do not yet exist any functions in the library that are capable of "ungetting" keysyms. In particular, the SLang_ungetkey function will not work.

3.7 Buffering Input

S-Lang has several functions pushing characters back onto the input stream to be read again later by SLang_getkey. It should be noted that none of the above functions are designed to push back keysyms read by the SLkp_getkey function. These functions are declared as follows:

```
void SLang_ungetkey (unsigned char ch);
void SLang_ungetkey_string (unsigned char *buf, int buflen);
void SLang_buffer_keystring (unsigned char *buf, int buflen);
```

SLang_ungetkey is the most simple of the three functions. It takes a single character a pushes it back on to the input stream. The next call to SLang_getkey will return this character. This function may be used to *peek* at the character to be read by first reading it and then putting it back.

SLang_ungetkey_string has the same function as SLang_ungetkey except that it is able to push more than one character back onto the input stream. Since this function can push back null (ascii 0) characters, the number of characters to push is required as one of the parameters.

The last of these three functions, SLang_buffer_keystring can handle more than one charater but unlike the other two, it places the characters at the *end* of the keyboard buffer instead of at the beginning.

Note that the use of each of these three functions will cause SLang_input_pending to return right away with a non-zero value.

Finally, the **S-Lang** keyboard interface includes the function SLang_flush_input with prototype

```
void SLang_flush_input (void);
```

It may be used to discard *all* input.

Here is a simple example that looks to see what the next key to be read is if one is available:

```
int peek_key ()
{
   int ch;
   if (SLang_input_pending (0) == 0) return -1;
   ch = SLang_getkey ();
   SLang_ungetkey (ch);
```

```
    return ch;
}
```

3.8 Global Variables

Although the following **S-Lang** global variables have already been mentioned earlier, they are gathered together here for completeness.

`int SLang_Ignore_User_Abort;`\ If non-zero, pressing the interrupt character will not result in `SLang_Error` being set to `USER_BREAK`.

`volatile int SLKeyBoard_Quit;`\ This variable is set to a non-zero value when the interrupt character is pressed. If the interrupt character is pressed when `SLang_getkey` is called, the interrupt character will be returned from `SLang_getkey`.

`int SLang_TT_Baud_Rate;`\ On systems which support it, this variable is set to the value of the terminal's baud rate after the call to `SLang_init_tty`.

Chapter 4

Screen Management

The **S-Lang** library provides two interfaces to terminal independent routines for manipulating the display on a terminal. The highest level interface, known as the `SLsmg` interface is discussed in this section. It provides high level screen management functions more manipulating the display in an optimal manner and is similar in spirit to the `curses` library. The lowest level interface, or the `SLtt` interface, is used by the `SLsmg` routines to actually perform the task of writing to the display. This interface is discussed in another section. Like the keyboard routines, the `SLsmg` routines are *platform independent* and work the same on MSDOS, OS/2, Unix, and VMS.

The screen management, or `SLsmg`, routines are initialized by function `SLsmg_init_smg`. Once initialized, the application uses various `SLsmg` functions to write to a *virtual* display. This does not cause the *physical* terminal display to be updated immediately. The physical display is updated to look like the virtual display only after a call to the function `SLsmg_refresh`. Before exiting, the application using these routines is required to call `SLsmg_reset_smg` to reset the display system.

The following subsections explore **S-Lang**'s screen management system in greater detail.

4.1 Initialization

The function `SLsmg_init_smg` must be called before any other `SLsmg` function can be used. It has the simple prototype:

```
int SLsmg_init_smg (void);
```

It returns non-zero if successful or zero if it cannot allocate space for the virtual display.

For this routine to properly initialize the virtual display, the capabilities of the terminal must be known as well as the size of the *physical* display. For these reasons, the lower level `SLtt` routines come into play. In particular, before the first call to `SLsmg_init_smg`, the application is required to call the function `SLtt_get_terminfo` before calling `SLsmg_init_smg`.

The `SLtt_get_terminfo` function sets the global variables `SLtt_Screen_Rows` and `SLtt_Screen_Cols` to the values appropriate for the terminal. It does this by calling the `SLtt_get_screen_size` function to query the terminal driver for the appropriate values for these variables. From this point on, it is up to the application to maintain the correct values for these variables by calling the

SLtt_get_screen_size function whenever the display size changes, e.g., in response to a SIGWINCH signal. If the screen size changes, the application must also re-initialize the SLsmg routines by calling SLsmg_init_tty again so that the size of the virtual screen can be adjusted accordingly.

Finally, if the application is going to read characters from the keyboard, it is also a good idea to initialize the keyboard routines at this point as well.

4.2 Resetting SLsmg

Before the program exits or suspends, the function SLsmg_reset_tty should be called to shutdown the display system. This function has the prototype

```
void SLsmg_reset_smg (void);
```

This will deallocate any memory allocated for the virtual screen and reset the terminal's display.

Basically, a program that uses the SLsmg screen management functions and **S-Lang**'s keyboard interface will look something like:

```
#include "slang.h"
int main ()
{
   SLtt_get_terminfo ();
   SLang_init_tty (-1, 0, 0);
   SLsmg_init_smg ();

   /* do stuff .... */

   SLsmg_reset_smg ();
   SLang_reset_tty ();
   return 0;
}
```

If this program is compiled and run, all it will do is clear the screen and position the cursor at the bottom of the display. In the following sections, other SLsmg functions will be introduced which may be used to make this simple program do much more.

4.3 SLsmg Functions

In the previous sections, functions for initializing and shutting down the SLsmg routines were discussed. In this section, the rest of the SLsmg functions are presented. These functions act only on the *virtual* display. The *physical* display is updated when the SLsmg_refresh function is called and *not until that time*. This function has the simple prototype:

```
void SLsmg_refresh (void);
```

4.3.1 Positioning the cursor

The SLsmg_gotorc function is used to position the cursor at a given row and column. The prototype for this function is:

```
void SLsmg_gotorc (int row, int col);
```

The origin of the screen is at the top left corner and is given the coordinate (0, 0), i.e., the top row of the screen corresponds to `row = 0` and the first column corresponds to `col = 0`. The last row of the screen is given by `row = SLtt_Screen_Rows - 1`.

It is possible to change the origin of the coordinate system by using the function SLsmg_set_screen_start with prototype:

```
void SLsmg_set_screen_start (int *r, int *c);
```

This function takes pointers to the new values of the first row and first column. It returns the previous values by modifying the values of the integers at the addresses specified by the parameter list. A NULL pointer may be passed to indicate that the origin is to be set to its initial value of 0. For example,

```
int r = 10;
SLsmg_set_screen_start (&r, NULL);
```

sets the origin to (10, 0) and after the function returns, the variable `r` will have the value of the previous row origin.

4.3.2 Writing to the Display

SLsmg has several routines for outputting text to the virtual display. The following points should be understood:

- The text is output at the position of the cursor of the virtual display and the cursor is advanced to the position that corresponds to the end of the text.
- Text does *not* wrap at the boundary of the display— it is trucated. This behavior seems to be more useful in practice since most programs that would use screen management tend to be line oriented.
- Control characters are displayed in a two character sequence representation with ^ as the first character. That is, `Ctrl-X` is output as `^X`.
- The newline character does *not* cause the cursor to advance to the next row. Instead, when a newline character is encountered when outputting text, the output routine will return. That is, outputting a string containing a newline character will only display the contents of the string up to the newline character.

Although the some of the above items might appear to be too restrictive, in practice this is not seem to be the case. In fact, the design of the output routines was influenced by their actual use and modified to simplify the code of the application utilizing them.

`void SLsmg_write_char (char ch);\` Write a single character to the virtual display.

`void SLsmg_write_nchars (char *str, int len);\` Write `len` characters pointed to by `str` to the virtual display.

`void SLsmg_write_string (char *str);\` Write the null terminated string given by pointer `str` to the virtual display. This function is a wrapper around `SLsmg_write_nchars`.

`void SLsmg_write_nstring (char *str, int n);\` Write the null terminated string given by pointer `str` to the virtual display. At most, only `n` characters are written. If the length of the string is less than `n`, then the string will be padded with blanks. This function is a wrapper around `SLsmg_write_nchars`.

`void SLsmg_printf (char *fmt, ...);\` This function is similar to `printf` except that it writes to the `SLsmg` virtual display.

`void SLsmg_vprintf (char *, va_list);\` Like `SLsmg_printf` but uses a variable argument list.

4.3.3 Erasing the Display

The following functions may be used to fill portions of the display with blank characters. The attributes of blank character are the current attributes. (See below for a discussion of character attributes)

`void SLsmg_erase_eol (void);\` Erase line from current position to the end of the line.

`void SLsmg_erase_eos (void);\` Erase from the current position to the end of the screen.

`void SLsmg_cls (void);\` Clear the entire virtual display.

4.3.4 Setting Character Attributes

Character attributes define the visual characteristics the character possesses when it is displayed. Visual characteristics include the foreground and background colors as well as other attributes such as blinking, bold, and so on. Since `SLsmg` takes a different approach to this problem than other screen management libraries an explanation of this approach is given here. This approach has been motivated by experience with programs that require some sort of screen management.

Most programs that use `SLsmg` are composed of specific textual objects or objects made up of line drawing characters. For example, consider an application with a menu bar with drop down menus. The menus might be enclosed by some sort of frame or perhaps a shadow. The basic idea is to associate an integer to each of the objects (e.g., menu bar, shadow, current menu item, etc.) and create a mapping from the integer to the set of attributes. In the terminology of `SLsmg`, the integer is simply called an *object*.

For example, the menu bar might be associated with the object `1`, the drop down menu could be object `2`, the shadow could be object `3`, and so on.

The range of values for the object integer is restricted from 0 up to and including 255 on all systems except MSDOS where the maximum allowed integer is 15[1]. The object numbered zero should not

[1]This difference is due to memory constraints imposed by MSDOS. This restriction might be removed in a future version of the library.

be regarding as an object at all. Rather it should be regarded as all *other* objects that have not explicitly been given an object number. SLsmg, or more precisely SLtt, refers to the attributes of this special object as the *default* or *normal* attributes.

The SLsmg routines know nothing about the mapping of the color to the attributes associated with the color. The actual mapping takes place at a lower level in the SLtt routines. Hence, to map an object to the actual set of attributes requires a call to any of the following SLtt routines:

```
void SLtt_set_color (int obj, char *name, char *fg, char *bg);
void SLtt_set_color_object (int obj, SLtt_Char_Type attr);
void SLtt_set_mono (int obj, char *, SLtt_Char_Type attr);
```

Only the first of these routines will be discussed briefly here. The latter two functions allow more fine control over the object to attribute mapping (such as assigning a "blink" attribute to the object). For a more full explanation on all of these routines see the section about the SLtt interface.

The SLtt_set_color function takes four parameters. The first parameter, obj, is simply the integer of the object for which attributes are to be assigned. The second parameter is currently unused by these routines. The third and forth parameters, fg and bg, are the names of the foreground and background color to be used associated with the object. The strings that one can use for the third and fourth parameters can be any one of the 16 colors:

```
"black"             "gray"
"red"               "brightred"
"green"             "brightgreen"
"brown"             "yellow"
"blue"              "brightblue"
"magenta"           "brightmagenta"
"cyan"              "brightcyan"
"lightgray"         "white"
```

The value of the foreground parameter fg can be anyone of these sixteen colors. However, on most terminals, the background color will can only be one of the colors listed in the first column[2].

Of course not all terminals are color terminals. If the **S-Lang** global variable SLtt_Use_Ansi_Colors is non-zero, the terminal is assumed to be a color terminal. The SLtt_get_terminfo will try to determine whether or not the terminal supports colors and set this variable accordingly. It does this by looking for the capability in the terminfo/termcap database. Unfortunately many Unix databases lack this information and so the SLtt_get_terminfo routine will check whether or not the environment variable COLORTERM exists. If it exists, the terminal will be assumed to support ANSI colors and SLtt_Use_Ansi_Colors will be set to one. Nevertheless, the application should provide some other mechanism to set this variable, e.g., via a command line parameter.

When the SLtt_Use_Ansi_Colors variable is zero, all objects with numbers greater than one will be displayed in inverse video[3].

With this background, the SLsmg functions for setting the character attributes can now be defined. These functions simply set the object attributes that are to be assigned to *subsequent* characters written to the virtual display. For this reason, the new attribute is called the *current* attribute.

[2]This is also true on the Linux console. However, it need not be the case and hopefully the designers of Linux will someday remove this restriction.

[3]This behavior can be modified by using the SLtt_set_mono function call.

`void SLsmg_set_color (int obj);`\ Set the current attribute to those of object `obj`.

`void SLsmg_normal_video (void);`\ This function is equivalent to `SLsmg_set_color (0)`.

`void SLsmg_reverse_video (void);`\ This function is equivalent to `SLsmg_set_color (1)`. On monochrome terminals, it is equivalent to setting the subsequent character attributes to inverse video.

Unfortunately there does not seem to be a standard way for the application or, in particular, the library to determine which color will be used by the terminal for the default background. Such information would be useful in initializing the foreground and background colors associated with the default color object (0). FOr this reason, it is up to the application to provide some means for the user to indicate what these colors are for the particular terminal setup. To facilitate this, the `SLtt_get_terminfo` function checks for the existence of the `COLORFGBG` environment variable. If this variable exists, its value will be used to initialize the colors associated with the default color object. Specifically, the value is assumed to consist of a foreground color name and a background color name separated by a semicolon. For example, if the value of `COLORTERM` is `lightgray;blue`, the default color object will be initialized to represent a `lightgray` foreground upon a `blue` background.

4.3.5 Lines and Alternate Character Sets

The **S-Lang** screen management library also includes routines for turning on and turning off alternate character sets. This is especially useful for drawing horizontal and vertical lines.

`void SLsmg_set_char_set (int flag);`\ If `flag` is non-zero, subsequent write functions will use characters from the alternate character set. If `flag` is zero, the default, or, ordinary character set will be used.

`void SLsmg_draw_hline (int len);`\ Draw a horizontal line from the current position to the column that is `len` characters to the right.

`void SLsmg_draw_vline (int len);`\ Draw a horizontal line from the current position to the row that is `len` rows below.

`void SLsmg_draw_box (int r, int c, int dr, int dc);`\ Draw a box whose upper right corner is at row `r` and column `c`. The box spans `dr` rows and `dc` columns. The current position will be left at row `r` and column `c`.

4.3.6 Miscellaneous Functions

`void SLsmg_touch_lines (int r, int n);`\ Mark screen rows numbered `r`, `r + 1`, ... `r + (n - 1)` as modified. When `SLsmg_refresh` is called, these rows will be completely redrawn.

`unsigned short SLsmg_char_at(void);`\ Returns the character and its attributes object number at the current cursor position. The character itself occupies the lower byte and the object attributes number forms the upper byte. The object returned by this function call should not be written back out via any of the functions that write characters or character strings.

4.4 Variables

The following **S-Lang** global variables are used by the `SLsmg` interface. Some of these have been previously discussed.

`int SLtt_Screen_Rows;`\ `int SLtt_Screen_Cols;`\ The number of rows and columns of the *physical* display. If either of these numbers changes, the functions `SLsmg_reset_smg` and `SLsmg_init_smg` should be called again so that the `SLsmg` routines can re-adjust to the new size.

`int SLsmg_Tab_Width;`\ Set this variable to the tab width that will be used when expanding tab characters. The default is 8.

`int SLsmg_Display_Eight_Bit`\ This variable determines how characters with the high bit set are to be output. Specifically, a character with the high bit set with a value greater than or equal to this value is output as is; otherwise, it will be output in a 7-bit representation. The default value for this variable is `128` for MSDOS and `160` for other systems (ISO-Latin).

`int SLtt_Use_Ansi_Colors;`\ If this value is non-zero, the terminal is assumed to support ANSI colors otherwise it is assumed to be monochrome. The default is 0.

`int SLtt_Term_Cannot_Scroll;`\ If this value is zero, the `SLsmg` will attempt to scroll the physical display to optimize the update. If it is non-zero, the screen management routines will not perform this optimization. For some applications, this variable should be set to zero. The default value is set by the `SLtt_get_terminfo` function.

4.5 Hints for using SLsmg

This section discusses some general design issues that one must face when writing an application that requires some sort of screen management.

Chapter 5

Signal Functions

Almost all non-trivial programs must worry about signals. This is especially true for programs that use the **S-Lang** terminal input/output and screen management routines. Unfortunately, there is no fixed way to handle signals; otherwise, the Unix kernel would take care of all issues regarding signals and the application programmer would never have to worry about them. For this reason, none of the routines in the **S-Lang** library catch signals; however, some of the routines block the delivery of signals during crucial moments. It is up to the application programmer to install handlers for the various signals of interest.

For the interpreter, the most important signal to worry about is `SIGINT`. This signal is usually generated when the user presses `Ctrl-C` at the keyboard. The interpreter checks the value of the `SLang_Error` variable to determine whether or not it should abort the interpreting process and return control back to the application. This means that if `SIGINT` is to be used to abort the interpreter, a signal handler for `SIGINT` should be installed. The handler should set the value of `SLang_Error` to `SL_USER_BREAK`.

Applications that use the `tty getkey` routines or the screen management routines must worry about about signals such as:

```
SIGINT          interrupt
SIGTSTP         stop
SIGQUIT         quit
SIGTTOU         background write
SIGTTIN         background read
SIGWINCH        window resize
```

It is important that handlers be established for these signals while the either the `SLsmg` routines or the `getkey` routines are initialized. The `SLang_init_tty`, `SLang_reset_tty`, `SLsmg_init_smg`, and `SLsmg_reset_smg` functions block these signals from occuring while they are being called.

Since a signal can be delivered at any time, it is important for the signal handler to call only functions that can be called from a signal handler. This usually means that such function must be re-entrant. In particular, the `SLsmg` routines are *not* re-entrant; hence, they should not be called when a signal is being processed unless the application can ensure that the signal was not delivered while an `SLsmg` function was called. This statement applies to many other functions such as `malloc`, or, more generally, any function that calls `malloc`. The upshot is that the signal handler should not attempt to do too much except set a global variable for the application to look at while not in a signal handler.

The **S-Lang** library provides two functions for blocking and unblocking the above signals:

```
int SLsig_block_signals (void);
int SLsig_unblock_signals (void);
```

It should be noted that for every call to `SLsig_block_signals`, a corresponding call should be made to `SLsig_unblock_signals`, e.g.,

```
void update_screen ()
{
   SLsig_block_signals ();

   /* Call SLsmg functions */
       .
       .
   SLsig_unblock_signals ();
}
```

See `demo/pager.c` for examples.

Chapter 6

Searching Functions

The S-Lang library incorporates two types of searches: Regular expression pattern matching and ordinary searching.

6.1 Regular Expressions

!!! No documentation available yet !!!

6.2 Simple Searches

The routines for ordinary searching are defined in the `slsearch.c` file. To use these routines, simply include "slang.h" in your program and simply call the appropriate routines.

The searches can go in either a forward or backward direction and can either be case or case insensitive. The region that is searched may contain null characters (ASCII 0) however, the search string cannot in the current implementation. In addition the length of the string to be found is currently limited to 256 characters.

Before searching, the function `SLsearch_init` must first be called to '`preprocess`' the search string.

6.3 Initialization

The function `SLsearch_init` must be called before a search can take place. Its prototype is:

```
int SLsearch_init (char *key, int dir, int case_sens, SLsearch_Type *st);
```

Here `key` is the string to be searched for. `dir` specifies the direction of the search: a value greater than zero is used for searching forward and a value less than zero is used for searching backward. The parameter `case_sens` specifies whether the search is case sensitive or not. A non-zero value indicates that case is important. `st` is a pointer to a structure of type `SLsearch_Type` defined in "slang.h". This structure is initialized by this routine and must be passed to `SLsearch` when the search is actually performed.

This routine returns the length of the string to be searched for.

6.4 SLsearch

```
Prototype: unsigned char *SLsearch (unsigned char *pmin,
                                    unsigned char *pmax,
                                    SLsearch_Type *st);
```

This function performs the search defined by a previous call to `SLsearch_init` over a region specified by the pointers `pmin` and `pmax`.

It returns a pointer to the start of the match if successful or it will return `NULL` if a match was not found.

Appendix A

Copyright

The **S-Lang** library is distributed under two copyrights: the GNU Genral Public License, and the Artistic License. Any program that uses the interpreter must adhere to rules of one of these licenses.

A.1 The GNU Public License

the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

```
                    GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION
```

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

```
a) You must cause the modified files to carry prominent notices
stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in
whole or in part contains or is derived from the Program or any
part thereof, to be licensed as a whole at no charge to all third
```

```
parties under the terms of this License.

c) If the modified program normally reads commands interactively
when run, you must cause it, when started running for such
interactive use in the most ordinary way, to print or display an
announcement including an appropriate copyright notice and a
notice that there is no warranty (or else, saying that you provide
a warranty) and that users may redistribute the program under
these conditions, and telling the user how to view a copy of this
License.  (Exception: if the Program itself is interactive but
does not normally print such an announcement, your work based on
the Program is not required to print an announcement.)
```

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

```
a) Accompany it with the complete corresponding machine-readable
source code, which must be distributed under the terms of Sections
1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three
years, to give any third party, for a charge no more than your
cost of physically performing source distribution, a complete
machine-readable copy of the corresponding source code, to be
distributed under the terms of Sections 1 and 2 above on a medium
customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer
to distribute corresponding source code.  (This alternative is
allowed only for noncommercial distribution and only if you
received the program in object code or executable form with such
an offer, in accord with Subsection b above.)
```

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

```
                            NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

                     END OF TERMS AND CONDITIONS
```

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) 19yy  <name of author>

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
`Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

A.2 The Artistic License

```
                         The "Artistic License"

                                Preamble
```

The intent of this document is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of artistic control over the development of the package, while giving the users of the package the right to use and distribute the Package in a more-or-less customary fashion, plus the right to make reasonable modifications.

Definitions:

```
        "Package" refers to the collection of files distributed by the
        Copyright Holder, and derivatives of that collection of files
        created through textual modification.

        "Standard Version" refers to such a Package if it has not been
        modified, or has been modified in accordance with the wishes
        of the Copyright Holder as specified below.

        "Copyright Holder" is whoever is named in the copyright or
        copyrights for the package.

        "You" is you, if you're thinking about copying or distributing
        this Package.

        "Reasonable copying fee" is whatever you can justify on the
        basis of media cost, duplication charges, time of people involved,
        and so on.  (You will not be required to justify it to the
        Copyright Holder, but only to the computing community at large
        as a market that must bear the fee.)

        "Freely Available" means that no fee is charged for the item
        itself, though there may be fees involved in handling the item.
        It also means that recipients of the item may redistribute it
        under the same conditions they received it.
```

1. You may make and give away verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.

2. You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.

3. You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:

```
    a) place your modifications in the Public Domain or otherwise make them
```

```
Freely Available, such as by posting said modifications to Usenet or
an equivalent medium, or placing the modifications on a major archive
site such as uunet.uu.net, or by allowing the Copyright Holder to include
your modifications in the Standard Version of the Package.

b) use the modified Package only within your corporation or organization.

c) rename any non-standard executables so the names do not conflict
with standard executables, which must also be provided, and provide
a separate manual page for each non-standard executable that clearly
documents how it differs from the Standard Version.

d) make other distribution arrangements with the Copyright Holder.
```

4. You may distribute the programs of this Package in object code or executable form, provided that you do at least ONE of the following:

```
a) distribute a Standard Version of the executables and library files,
together with instructions (in the manual page or equivalent) on where
to get the Standard Version.

b) accompany the distribution with the machine-readable source of
the Package with your modifications.

c) give non-standard executables non-standard names, and clearly
document the differences in manual pages (or equivalent), together
with instructions on where to get the Standard Version.

d) make other distribution arrangements with the Copyright Holder.
```

5. You may charge a reasonable copying fee for any distribution of this Package. You may charge any fee you choose for support of this Package. You may not charge a fee for this Package itself. However, you may distribute this Package in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution provided that you do not advertise this Package as a product of your own. You may embed this Package's interpreter within an executable of yours (by linking); this shall be construed as a mere form of aggregation, provided that the complete Standard Version of the interpreter is so embedded.

6. The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whomever generated them, and may be sold commercially, and may be aggregated with this Package. If such scripts or library files are aggregated with this Package via the so-called "undump" or "unexec" methods of producing a binary executable image, then distribution of such an image shall neither be construed as a distribution of this Package nor shall it fall under the restrictions of Paragraphs 3 and 4, provided that you do not represent such an executable image as a Standard Version of this Package.

7. C subroutines (or comparably compiled subroutines in other languages) supplied by you and linked into this Package in order to emulate subroutines and variables of the language defined by this Package shall not be considered part of this Package, but are the equivalent of input as in

Paragraph 6, provided these subroutines do not change the language in any way that would cause it to fail the regression tests for the language.

8. Aggregation of this Package with a commercial distribution is always permitted provided that the use of this Package is embedded; that is, when no overt attempt is made to make this Package's interfaces visible to the end user of the commercial distribution. Such use shall not be construed as a distribution of this Package.

9. The name of the Copyright Holder may not be used to endorse or promote products derived from this software without specific prior written permission.

10. THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.