Contents

1 Functions dealing with UTF-8 encoded strings 9
  1.1 SLutf8_skip_char .................................................. 9
  1.2 SLutf8_skip_chars ................................................. 10
  1.3 SLutf8_bskip_char .................................................. 10
  1.4 SLutf8_bskip_chars ................................................ 11
  1.5 SLutf8_decode ....................................................... 11
  1.6 SLutf8_encode ....................................................... 12
  1.7 SLutf8_strlen ........................................................ 12
  1.8 SLutf8_extract_utf8_char ......................................... 13
  1.9 SLutf8_encode_null_terminate .................................. 13
  1.10 SLutf8_strup ....................................................... 14
  1.11 SLutf8_strlo ....................................................... 14
  1.12 SLutf8_subst_wchar .............................................. 14
  1.13 SLutf8_compare ................................................... 15

2 Character classification functions 15
  2.1 SLwchar_toupper ................................................... 15
  2.2 SLwchar_tolower .................................................... 16
  2.3 SLwchar_wcwidth .................................................... 16
  2.4 SLwchar_isalnum ..................................................... 17
  2.5 SLwchar_isalpha ..................................................... 17
  2.6 SLwchar_isblank ..................................................... 17
  2.7 SLwchar_isctrl ....................................................... 18
  2.8 SLwchar_isdigit ...................................................... 18
  2.9 SLwchar_isgraph ...................................................... 19
  2.10 SLwchar_islower ..................................................... 19
  2.11 SLwchar_isprint ..................................................... 19
<table>
<thead>
<tr>
<th>Section</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.12</td>
<td>SLwchar_ispunct</td>
<td>20</td>
</tr>
<tr>
<td>2.13</td>
<td>SLwchar isspace</td>
<td>20</td>
</tr>
<tr>
<td>2.14</td>
<td>SLwchar_isupper</td>
<td>21</td>
</tr>
<tr>
<td>2.15</td>
<td>SLwchar_isxdigit</td>
<td>21</td>
</tr>
<tr>
<td>3</td>
<td>SLsearch interface Functions</td>
<td>21</td>
</tr>
<tr>
<td>3.1</td>
<td>SLsearch_new</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>SLsearch_delete</td>
<td>22</td>
</tr>
<tr>
<td>3.3</td>
<td>SLsearch_forward</td>
<td>22</td>
</tr>
<tr>
<td>3.4</td>
<td>SLsearch_backward</td>
<td>23</td>
</tr>
<tr>
<td>3.5</td>
<td>SLsearch_match_len</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>Screen Management (SLsmsg) functions</td>
<td>24</td>
</tr>
<tr>
<td>4.1</td>
<td>SLsmsg_fill_region</td>
<td>24</td>
</tr>
<tr>
<td>4.2</td>
<td>SLsmsg_set_char_set</td>
<td>24</td>
</tr>
<tr>
<td>4.3</td>
<td>int SLsmsg_Scroll_Hash_Border;</td>
<td>25</td>
</tr>
<tr>
<td>4.4</td>
<td>SLsmsg_suspend_smg</td>
<td>25</td>
</tr>
<tr>
<td>4.5</td>
<td>SLsmsg_resume_smg</td>
<td>26</td>
</tr>
<tr>
<td>4.6</td>
<td>SLsmsg_erase_col</td>
<td>26</td>
</tr>
<tr>
<td>4.7</td>
<td>SLsmsg_gotorc</td>
<td>26</td>
</tr>
<tr>
<td>4.8</td>
<td>SLsmsg_erase_eos</td>
<td>27</td>
</tr>
<tr>
<td>4.9</td>
<td>SLsmsg_reverse_video</td>
<td>27</td>
</tr>
<tr>
<td>4.10</td>
<td>SLsmsg_set_color (int)</td>
<td>27</td>
</tr>
<tr>
<td>4.11</td>
<td>SLsmsg_normal_video</td>
<td>28</td>
</tr>
<tr>
<td>4.12</td>
<td>SLsmsg_printf</td>
<td>28</td>
</tr>
<tr>
<td>4.13</td>
<td>SLsmsg_vprintf</td>
<td>29</td>
</tr>
<tr>
<td>4.14</td>
<td>SLsmsg_write_string</td>
<td>29</td>
</tr>
<tr>
<td>4.15</td>
<td>SLsmsg_write_nstring</td>
<td>29</td>
</tr>
<tr>
<td>4.16</td>
<td>SLsmsg_write_char</td>
<td>30</td>
</tr>
<tr>
<td>4.17</td>
<td>SLsmsg_write_nchars</td>
<td>30</td>
</tr>
<tr>
<td>4.18</td>
<td>SLsmsg_write_wrapped_string</td>
<td>30</td>
</tr>
<tr>
<td>4.19</td>
<td>SLsmsg_cls</td>
<td>31</td>
</tr>
<tr>
<td>4.20</td>
<td>SLsmsg_refresh</td>
<td>31</td>
</tr>
<tr>
<td>4.21</td>
<td>SLsmsg_touch_lines</td>
<td>32</td>
</tr>
<tr>
<td>4.22</td>
<td>SLsmsg_init_smg</td>
<td>32</td>
</tr>
</tbody>
</table>
4.23 SLsmg_reset_smg ......................................................... 32
4.24 SLsmg_char_at ............................................................. 33
4.25 SLsmg_set_screen_start ............................................... 33
4.26 SLsmg_draw_hline ...................................................... 33
4.27 SLsmg_draw_vline ....................................................... 34
4.28 SLsmg_draw_object .................................................... 34
4.29 SLsmg_draw_box ......................................................... 35
4.30 SLsmg_set_color_in_region ............................................ 35
4.31 SLsmg_get_column ...................................................... 35
4.32 SLsmg_get_row .......................................................... 36
4.33 SLsmg_forward ......................................................... 36
4.34 SLsmg_write_color_chars ............................................ 36
4.35 SLsmg_read_raw ........................................................ 37
4.36 SLsmg_write_raw ....................................................... 37

5 Functions that deal with the interpreter ............................... 38
5.1 SLallocate_load_type .................................................... 38
5.2 SLdeallocation_load_type .............................................. 38
5.3 SLang_load_object ..................................................... 38
5.4 SLclass_allocate_class ................................................ 39
5.5 SLclass_register_class ................................................ 39
5.6 SLclass_set_string_function ........................................ 40
5.7 SLclass_set_destroy_function ...................................... 41
5.8 SLclass_set_push_function .......................................... 42
5.9 SLclass_set_pop_function ............................................ 42
5.10 SLclass_get_datatype_name ........................................ 43
5.11 SLang_free_mmt ....................................................... 43
5.12 SLang_object_from_mmt ............................................. 44
5.13 SLang_create_mmt ..................................................... 44
5.14 SLang_push_mmt ....................................................... 44
5.15 SLang_pop_mmt ........................................................ 45
5.16 SLang_inc_mmt ........................................................ 45
5.17 SLadd_intrinsic_function .......................................... 45
5.18 SLadd_intrinsic_var_table ......................................... 46
5.19 Slang_load_file .......................................................... 46
5.20 Slang_restart ........................................................... 47
5.21 Slang_byte_compile_file ............................................ 47
5.22 Slang_autoload .......................................................... 48
5.23 Slang_load_string ..................................................... 48
5.24 Sldo_pop ................................................................. 49
5.25 Sldo_pop_n ............................................................... 49
5.26 Slang_pop_integer ..................................................... 49
5.27 Slpop_string ............................................................ 50
5.28 Slang_pop_string ....................................................... 50
5.29 Slang_pop_slstring ................................................... 51
5.30 Slang_pop_double ...................................................... 51
5.31 Slang_pop_complex .................................................... 52
5.32 Slang_push_complex .................................................. 52
5.33 Slang_push_double .................................................... 52
5.34 Slang_push_string ..................................................... 53
5.35 Slang_push_integer .................................................... 53
5.36 Slang_push_malloced_string ....................................... 54
5.37 Slang_is_defined ...................................................... 54
5.38 Slang_run_hooks ....................................................... 55
5.39 Slang_execute_function ............................................. 55
5.40 Slang_get_function .................................................. 56
5.41 Sexecute_function .................................................... 56
5.42 Slang_peek_at_stack ................................................ 57
5.43 Slang_pop_fileptr .................................................... 58
5.44 SLadd_intrinsic_function ........................................... 59
5.45 SLadd_intrinsic_variable ......................................... 60
5.46 SLclass_add_unary_op ............................................... 61
5.47 SLclass_add_app_unary_op ......................................... 61
5.48 SLclass_add_binary_op ............................................... 61
5.49 SLclass_add_math_op ................................................ 62
5.50 SLclass_add_typecast ................................................ 62
## CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Library Initialization Functions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>SLang_init_slang</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>SLang_init_slfile</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>SLang_init_slmath</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>SLang_init_slunix</td>
<td>63</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Miscellaneous Functions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>SLcurrent_time_string</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>SLatoi</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>SLextract_list_element</td>
<td>65</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Error and Messaging Functions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>SLang_verror</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>SLang_doerror</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>SLang_vmessage</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>SLang_exit_error</td>
<td>67</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>String and Memory Allocation Functions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>SLmake_string</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>SLmake_nstring</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>SLang_create_nslstring</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>SLang_create_slstring</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>SLang_free_slstring</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>SLang_concat_slstrings</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>SLang_create_static_slstring</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>Smalloc</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>Scalloc</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>Sfree</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>Srealloc</td>
<td>71</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Keyboard Input Functions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>SLang_init_tty</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>SLang_reset_tty</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>SLtty_set_suspend_state</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>SLang_getkey</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>SLang_ungetkey_string</td>
<td>74</td>
</tr>
</tbody>
</table>
12.16 `SLtt_end_insert(void);` ........................................ 86
12.17 `SLtt_del_eol(void);` ........................................ 87
12.18 `SLtt_goto_rc (int, int);` ................................ 87
12.19 `SLtt_delete_nlines(int);` .................................. 87
12.20 `SLtt_delete_char(void);` .................................... 88
12.21 `SLtt_erase_line(void);` ....................................... 88
12.22 `SLtt_normal_video(void);` .................................. 88
12.23 `SLtt_cls(void);` ................................................ 89
12.24 `SLtt_bEEP(void);` .............................................. 89
12.25 `SLtt_reverse_index(int);` .................................... 89
12.26 `SLtt_smart_puts(unsigned short *, unsigned short *, int, int);` 90
12.27 `SLtt_write_string (char *);` ................................ 90
12.28 `SLtt_putchar(char);` .......................................... 90
12.29 `SLtt_init_video (void);` ..................................... 91
12.30 `SLtt_reset_video (void);` ..................................... 91
12.31 `SLtt_get_terminfo(void);` ..................................... 91
12.32 `SLtt_get_screen_size (void);` ............................... 92
12.33 `SLtt_set_cursor_visibility (int);` ........................... 92
12.34 `SLtt_initialize (char *);` .................................. 92
12.35 `SLtt_enable_cursor_keys(void);` ........................... 93
12.36 `SLtt_set_term_vtxxx(int *);` ................................ 93
12.37 `SLtt_set_color_esc (int, char *);` ........................... 93
12.38 `SLtt_wide_width(void);` ....................................... 94
12.39 `SLtt_narrow_width(void);` .................................... 94
12.40 `SLtt_set_mouse_mode (int, int);` ........................... 94
12.41 `SLtt_set_alt_char_set (int);` ............................... 95
12.42 `SLtt_write_to_status_line (char *, int);` ................ 95
12.43 `SLtt_disable_status_line (void);` ........................ 95
12.44 `char *SLtt_tgetstr (char *);` .............................. 96
12.45 `SLtt_tgetnum (char *);` ..................................... 96
12.46 `SLtt_tgetflag (char *);` ...................................... 96
12.47 `char *SLtt_tigetent (char *);` ............................... 97
12.48 `char *SLtt_tigetstr (char *, char **);` .................. 97
12.49 `int SLtt_tigetnum (char *, char **);` .................... 97
12.50 SLtt_Char_Type SLtt_get_color_object (int); ................................................................. 98
12.51 void SLtt_set_color_object (int, SLtt_Char_Type); ......................................................... 98
12.52 void SLtt_set_color (int, char *, char *, char *); ........................................................ 98
12.53 void SLtt_set_mono (int, char *, SLtt_Char_Type); .......................................................... 99
12.54 void SLtt_add_color_attribute (int, SLtt_Char_Type); ..................................................... 99
12.55 void SLtt_set_color_fgbg (int, SLtt_Char_Type, SLtt_Char_Type); .................................... 99
12.56 int SLkp_define_keysym (char *, unsigned int); ............................................................. 100
12.57 int SLkp_init (void); ....................................................................................................... 100
12.58 int SLkp_getkey (void); .................................................................................................... 100
12.59 int SLscroll_find_top (SLscroll_Window_Type *); ............................................................... 101
12.60 int SLscroll_find_line_num (SLscroll_Window_Type *); ................................................... 101
12.61 unsigned int SLscroll_next_n (SLscroll_Window_Type *, unsigned int); ................................ 101
12.62 unsigned int SLscroll_prev_n (SLscroll_Window_Type *, unsigned int); ......................... 102
12.63 int SLscroll_pageup (SLscroll_Window_Type *); ................................................................ 102
12.64 int SLscroll_pagedown (SLscroll_Window_Type *); ............................................................ 102
12.65 SLSig_Fun_Type *SLsignal (int, SLSig_Fun_Type *); ......................................................... 103
12.66 SLSig_Fun_Type *SLsignal_intr (int, SLSig_Fun_Type *); .............................................. 103
12.67 int SLsig_block_signals (void); ......................................................................................... 103
12.68 int SLsig_unblock_signals (void); ....................................................................................... 104
12.69 int SLsystem (char *); ..................................................................................................... 104
12.70 void SLadd_at_handler (long *, char *); ........................................................................... 104
12.71 void SLang_define_case(int *, int *); ................................................................................. 105
12.72 void SLang_init_case_tables (void); .................................................................................. 105
12.73 unsigned char *SLang_regexp_match(unsigned char *, unsigned int, SLRegexp_Type*); ...... 105
12.74 int SLang_regexp_compile (SLRegexp_Type*); .................................................................... 106
12.75 char *SLregexp_quote_string (char *, char *, unsigned int); ............................................ 106
12.76 int SLcmd_execute_string (char *, SLcmd_Cmd_Table_Type*); ........................................ 106
12.77 SLcomplex_abs ................................................................................................................ 107
12.78 double *SLcomplex_times (double *, double *, double *); .............................................. 107
12.79 double *SLcomplex_divide (double *, double *, double *); ........................................... 107
12.80 double *SLcomplex_sin (double *, double *); ................................................................ 108
12.81 double *SLcomplex_cos (double *, double *); ................................................................ 108
12.82 double *SLcomplex_tan (double *, double *); ................................................................ 108
1. Functions dealing with UTF-8 encoded strings

1.1 SLutf8_skip_char

Synopsis
Skip past a UTF-8 encoded character

Usage
SLuchar_Type *SLutf8_skip_char (SLuchar_Type *u, SLuchar_Type *umax)
functions dealing with UTF-8 encoded strings

Description

The SLutf8_skip_char function returns a pointer to the character immediately following the UTF-8 encoded character at u. It will make no attempt to examine the bytes at the position umax and beyond. If the bytes at u do not represent a valid or legal UTF-8 encoded sequence, a pointer to the byte following u will be returned.

Notes

Unicode combining characters are treated as distinct characters by this function.

See Also

SLutf8_skip_chars, SLutf8_bskip_char, SLutf8_strlen

1.2 SLutf8_skip_chars

Synopsis

Skip past a specified number of characters in a UTF-8 encoded string

Usage

SLuchar_Type *SLutf8_skip_chars (u, umax, num, dnum, ignore_combining)

SLuchar_Type *u, *umax;
unsigned int num;
unsigned int *dnum;
int ignore_combining;

Description

This functions attempts to skip forward past num UTF-8 encoded characters at u returning the actual number skipped via the parameter dnum. It will make no attempt to examine bytes at umax and beyond. Unicode combining characters will not be counted if ignore_combining is non-zero, otherwise they will be treated as distinct characters. If the input contains an invalid or illegal UTF-8 sequence, then each byte in the sequence will be treated as a single character.

See Also

SLutf8_skip_char, SLutf8_bskip_chars

1.3 SLutf8_bskip_char

Synopsis

Skip backward past a UTF-8 encoded character

Usage

SLuchar_Type *SLutf8_bskip_char (SLuchar_Type *umin, SLuchar_Type *u)

Description

The SLutf8_bskip_char skips backward to the start of the UTF-8 encoded character immediately before the position u. The function will make no attempt to examine characters before the position umin. UTF-8 combining characters are treated as distinct characters.
See Also
SLutf8_bskip_chars, SLutf8_skip_char

1.4 SLutf8_bskip_chars

Synopsis
Skip backward past a specified number of UTF-8 encoded characters

Usage
SLuchar_Type *SLutf8_bskip_chars (umin, u, num, dnum, ignore_combining)

SLuchar_Type *umin, *u;
unsigned int num;
unsigned int *dnum;
int ignore_combining;

Description
This function attempts to skip backward past num UTF-8 encoded characters occurring immediately before u. It returns the actual number skipped via the parameter dnum. No attempt will be made to examine the bytes occurring before umin. Unicode combining characters will not be counted if ignore_combining is non-zero, otherwise they will be treated as distinct characters. If the input contains an invalid or illegal UTF-8 sequence, then each byte in the sequence will be treated as a single character.

See Also
SLutf8_skip_char, SLutf8_bskip_chars

1.5 SLutf8_decode

Synopsis
Decode a UTF-8 encoded character sequence

Usage
SLuchar_Type *SLutf8_decode (u, umax, w, nconsumedp

SLuchar_Type *u, *umax;
SLwchar_Type *w;
unsigned int *nconsumedp;

Description
The SLutf8_decode function decodes the UTF-8 encoded character occurring at u and returns the decoded character via the parameter w. No attempt will be made to examine the bytes at umax and beyond. If the parameter nconsumedp is non-NULL, then the number of bytes consumed by the function will be returned to it. If the sequence at u is invalid or illegal, the function will return NULL and with the number of bytes consumed by the function equal to the size of the invalid sequence. Otherwise the function will return a pointer to byte following encoded sequence.

See Also
SLutf8_decode, SLutf8_strlen, SLutf8_skip_char
1.6  SLutf8_encode

Synopsis
UTF-8 encode a character

Usage
SLuchar_Type *SLutf8_encode (w, u, ulen)

SLwchar_Type w;
SLuchar_Type *u;
unsigned int ulen;

Description
This function UTF-8 encodes the Unicode character represented by w and stored the encoded representation in the buffer of size ulen bytes at u. The function will return NULL if the size of the buffer is too small to represent the UTF-8 encoded character, otherwise it will return a pointer to the byte following encoded representation.

Notes
This function does not null terminate the resulting byte sequence. The function SLutf8_encode_null_terminate may be used for that purpose.

To guarantee that the buffer is large enough to hold the encoded bytes, its size should be at least SLUTF8_MAX_BLEN bytes.

The function will encode illegal Unicode characters, i.e., characters in the range 0xD800-0xFFFF (the UTF-16 surrogates) and 0xFFFE-0xFFFF.

See Also
SLutf8_decode, SLutf8_encode_bytes, SLutf8_encode_null_terminate

1.7  SLutf8_strlen

Synopsis
Determine the number of characters in a UTF-8 sequence

Usage
unsigned int SLutf8_strlen (SLuchar_Type *s, int ignore_combining)

Description
This function may be used to determine the number of characters represented by the null-terminated UTF-8 byte sequence. If the ignore_combining parameter is non-zero, then Unicode combining characters will not be counted.

See Also
SLutf8_skip_chars, SLutf8_decode
1. Functions dealing with UTF-8 encoded strings

1.8 SLutf8_extract_utf8_char

Synopsis
Extract a UTF-8 encoded character

Usage
SLuchar_Type *SLutf8_extract_utf8_char (u, umax, buf)

SLuchar_Type *u, *umax, *buf;

Description
This function extracts the bytes representing UTF-8 encoded character at u and places them in the buffer buf, and then null terminates the result. The buffer is assumed to consist of at least SLUTF8_MAX_BLEN+1 bytes, where the extra byte may be necessary for null termination. No attempt will be made to examine the characters at umax and beyond. If the byte-sequence at u is an illegal or invalid UTF-8 sequence, then the byte at u will be copied to the buffer. The function returns a pointer to the byte following copied bytes.

Notes
One may think of this function as the single byte analogue of

if (u < umax)
{
    buf[0] = *u++;
    buf[1] = 0;
}

See Also
SLutf8_decode, SLutf8_skip_char

1.9 SLutf8_encode_null_terminate

Synopsis
UTF-8 encode a character and null terminate the result

Usage
SLuchar_Type *SLutf8_encode_null_terminate (w, buf)

SLwchar_Type w;
SLuchar_Type *buf;

Description
This function has the same functionality as SLutf8_encode, except that it also null terminates the encoded sequences. The buffer buf, where the encoded sequence is placed, is assumed to consist of at least SLUTF8_MAX_BLEN+1 bytes.

See Also
SLutf8_encode
1. Functions dealing with UTF-8 encoded strings

1.10 SLutf8_strup

Synopsis
Upper case a UTF-8 encoded string

Usage
`SLuchar_Type *SLutf8_strup (SLuchar_Type *u, SLuchar_Type *umax)`

Description
The `SLutf8_strup` function returns the uppercase equivalent of UTF-8 encoded sequence of `umax-u` bytes at `u`. The result will be returned as a null-terminated `SLstring` and should be freed with `SLang_free_slstring` when it is no longer needed. If the function encounters an invalid or illegal byte sequence, then the byte-sequence will be copied as is.

See Also
`SLutf8_strlow`, `SLwchar_toupper`

1.11 SLutf8_strlo

Synopsis
Lowercase a UTF-8 encoded string

Usage
`SLuchar_Type *SLutf8_strlo (SLuchar_Type *u, SLuchar_Type *umax)`

Description
The `SLutf8_strlo` function returns the lowercase equivalent of UTF-8 encoded sequence of `umax-u` bytes at `u`. The result will be returned as a null-terminated `SLstring` and should be freed with `SLang_free_slstring` when it is no longer needed. If the function encounters an invalid or illegal byte sequence, then the byte-sequence will be copied as is.

See Also
`SLutf8_strlow`, `SLwchar_toupper`

1.12 SLutf8_subst_wchar

Synopsis
Replace a character in a UTF-8 encoded string

Usage
`SLstr_Type *SLutf8_subst_wchar (u, umax, wch, nth, ignore_combining)`

`SLuchar_Type *u, *umax;`  
`SLwchar_Type wch;`  
`unsigned int nth;`  
`int ignore_combining;`
Description

The SLutf8_subst_wchar function replaces the UTF-8 sequence representing the nth character of u by the UTF-8 representation of the character wch. If the value of the ignore_combining parameter is non-zero, then combining characters will not be counted when computing the position of the nth character. In addition, if the nth character contains any combining characters, then the byte-sequence associated with those characters will also be replaced.

Since the byte sequence representing wch could be longer than the sequence of the nth character, the function returns a new copy of the resulting string as an SLSTRING. Hence, the calling function should call SLang_free_slstring when the result is no longer needed.

See Also
SLutf8_strup, SLutf8_strlow, SLutf8_skip_chars, SLutf8_strlen

1.13 SLutf8_compare

Synopsis
Compare two UTF-8 encoded sequences

Usage

int SLutf8_compare (a, amax, b, bmax, nchars, case_sensitive)

SLuchar_Type *a, *amax;
SLuchar_Type *b, *bmax;
unsigned int nchars;
int case_sensitive;

Description

This function compares nchars of one UTF-8 encoded character sequence to another by performing a character by character comparison. The function returns 0, +1, or -1 according to whether the string a is equal to, greater than, or less than the string at b. At most nchars characters will be tested. The parameters amax and bmax serve as upper boundaries of the strings a and b, resp.

If the value of the case_sensitive parameter is non-zero, then a case-sensitive comparison will be performed, otherwise characters will be compared in a case-insensitive manner.

Notes

For case-sensitive comparisons, this function is analogous to the standard C library’s strncmp function. However, SLutf8_compare can also cope with invalid or illegal UTF-8 sequences.

See Also
SLutf8_strup, SLutf8_strlen, SLutf8_strlen

2 Character classification functions

2.1 SLwchar_toupper

Synopsis
Uppercase a Unicode character
2. Character classification functions

Usage

SIwchar_Type SIwchar_toupper (SIwchar_Type wc)

Description

SIwchar_toupper returns the uppercase equivalent of the specified character.

Notes

If the library is not in UTF-8 mode, then the current locale will be used.

See Also

SIwchar_tolower, SIwchar_isupper, SIutf8_strup

2.2 SIwchar_tolower

Synopsis

Lowercase a Unicode character

Usage

SIwchar_Type SIwchar_tolower (SIwchar_Type wc)

Description

SIwchar_tolower returns the lowercase equivalent of the specified character.

Notes

If the library is not in UTF-8 mode, then the current locale will be used.

See Also

SIwchar_toupper, SIwchar_islower, SIutf8_strlow

2.3 SIwchar_wcwidth

Synopsis

Determine the displayable width of a wide character

Usage

int SIwchar_wcwidth (SIwchar_Type wc)

Description

This function returns the number of columns necessary to display the specified Unicode character. Combining characters are meant to be combined with other characters and, as such, have 0 width.

Notes

If the library is not in UTF-8 mode, then the current locale will be used.

See Also

SIwchar_isspace, SIwchar_iscntrl
2.4  **SLwchar_isalnum**

**Synopsis**

Determine if a Unicode character is alphanumerical

**Usage**

```c
int SLwchar_isalnum (SLwchar_Type wc)
```

**Description**

SLwchar_isalnum returns a non-zero value if the Unicode character is alphanumerical, otherwise it returns 0.

**Notes**

If the library is not in UTF-8 mode, then the current locale will be used.

**See Also**

SLwchar_isalpha, SLwchar_isdigit, SLwchar_iscntrl

2.5  **SLwchar_isalpha**

**Synopsis**

Determine if a Unicode character is an alphabetic character

**Usage**

```c
int SLwchar_isalpha (SLwchar_Type wc)
```

**Description**

SLwchar_isalpha returns a non-zero value if the Unicode character is alphabetic, otherwise it returns 0.

**Notes**

If the library is not in UTF-8 mode, then the current locale will be used.

**See Also**

SLwchar_isalnum, SLwchar_isalpha, SLwchar_isdigit, SLwchar_iscntrl

2.6  **SLwchar_isblank**

**Synopsis**

Determine if a Unicode character is a blank

**Usage**

```c
int SLwchar_isblank (SLwchar_Type wc)
```

**Description**

SLwchar_isblank returns a non-zero value if the Unicode character is a blank one (space or tab), otherwise it returns 0.
2. Character classification functions

Notes
If the library is not in UTF-8 mode, then the current locale will be used.

See Also
SLwchar_isspace, SLwchar_isalpha, SLwchar_isdigit, SLwchar_iscntrl

2.7 SLwchar_iscntrl

Synopsis
Determine if a Unicode character is a control character

Usage
int SLwchar_iscntrl (SLwchar_Type wc)

Description
SLwchar_isblank returns a non-zero value if the Unicode character is a control character, otherwise it returns 0.

Notes
If the library is not in UTF-8 mode, then the current locale will be used.

See Also
SLwchar_isspace, SLwchar_isalpha, SLwchar_isdigit, SLwchar_isprint

2.8 SLwchar_isdigit

Synopsis
Determine if a Unicode character is a digit

Usage
int SLwchar_isdigit (SLwchar_Type wc)

Description
This function returns a non-zero value if the specified Unicode character is a digit, otherwise it returns 0.

Notes
If the library is not in UTF-8 mode, then the current locale will be used.

See Also
SLwchar_isspace, SLwchar_isalpha, SLwchar_isxdigit, SLwchar_isprint
2.9 SLwchar_isgraph

Synopsis
Determine if a non-space Unicode character is printable

Usage
int SLwchar_isgraph (SLwchar_Type wc)

Description
This function returns a non-zero value if the specified Unicode character is a non-space printable character, otherwise it returns 0.

Notes
If the library is not in UTF-8 mode, then the current locale will be used.

See Also
SLwchar_isspace, SLwchar_isalpha, SLwchar_isdigit, SLwchar_isprint

2.10 SLwchar_islower

Synopsis
Determine if a Unicode character is alphanumeric

Usage
int SLwchar_islower (SLwchar_Type wc)

Description
This function returns a non-zero value if the specified Unicode character is a lowercase one, otherwise it returns 0.

Notes
If the library is not in UTF-8 mode, then the current locale will be used.

See Also
SLwchar_isupper, SLwchar_isspace, SLwchar_isalpha, SLwchar_isdigit, SLwchar_iscntrl

2.11 SLwchar_isprint

Synopsis
Determine if a Unicode character is printable

Usage
int SLwchar_isprint (SLwchar_Type wc)

Description
This function returns a non-zero value if the specified Unicode character is a printable one (includes space), otherwise it returns 0.
2. Character classification functions

Notes
If the library is not in UTF-8 mode, then the current locale will be used.

See Also
SLwchar_isgraph, SLwchar_isspace, SLwchar_isalpha, SLwchar_isdigit

2.12 SLwchar_ispunct

Synopsis
Determine if a Unicode character is a punctuation character

Usage
int SLwchar_ispunct (SLwchar_Type wc)

Description
This function returns a non-zero value if the specified Unicode character is a punctuation character, otherwise it returns 0.

Notes
If the library is not in UTF-8 mode, then the current locale will be used.

See Also
SLwchar_isspace, SLwchar_isalpha, SLwchar_isdigit, SLwchar_isprint

2.13 SLwchar_isisspace

Synopsis
Determine if a Unicode character is a whitespace character

Usage
int SLwchar_isisspace (SLwchar_Type wc)

Description
This function returns a non-zero value if the specified Unicode character is a whitespace character, otherwise it returns 0.

Notes
If the library is not in UTF-8 mode, then the current locale will be used.

See Also
SLwchar_isblank, SLwchar_isalpha, SLwchar_isdigit, SLwchar_isprint
2.14 SLwchar_isupper

Synopsis
Determine if a Unicode character is uppercase

Usage
int SLwchar_isupper (SLwchar_Type wc)

Description
This function returns a non-zero value if the specified Unicode character is an uppercase character, otherwise it returns 0.

Notes
If the library is not in UTF-8 mode, then the current locale will be used.

See Also
SLwchar_islower, SLwchar_isspace, SLwchar_isalpha, SLwchar_isdigit

2.15 SLwchar_isxdigit

Synopsis
Determine if a Unicode character is a hexadecimal digit

Usage
int SLwchar_isxdigit (SLwchar_Type wc)

Description
This function returns a non-zero value if the specified Unicode character is a hexadecimal digit character, otherwise it returns 0.

Notes
If the library is not in UTF-8 mode, then the current locale will be used.

See Also
SLwchar_isdigit, SLwchar_isspace, SLwchar_isalpha, SLwchar_ispunct

3 SLsearch interface Functions

3.1 SLsearch_new

Synopsis
Create an SLsearch_Type object

Usage
SLsearch_Type *SLsearch_new (SLuchar_Type *key, int search_flags)
3. SLsearch interface Functions

Description

The `SLsearch_new` function instantiates an `SLsearch_Type` object for use in ordinary searches (non-regular expression) by the functions in the SLsearch interface. The first argument `key` is a pointer to a null terminated string that specifies the character string to be searched. This character string may not contain any embedded null characters.

The second argument `search_flags` is used to specify how the search is to be performed. It is a bit-mapped integer whose value is constructed by the bitwise-or of zero or more of the following:

- `SLSEARCH_CASELESS`
  The search shall be performed in a case-insensitive manner.

- `SLSEARCH_UTF8`
  Both the search string and the text to be searched is UTF-8 encoded.

Upon success, the function returns the newly created object, and `NULL` otherwise. When the search object is no longer needed, it should be freed via the `SLsearch_delete` function.

See Also

`SLsearch_delete`, `SLsearch_forward`, `SLsearch_backward`

3.2 SLsearch_delete

Synopsis

Free the memory associated with a `SLsearch_Type` object

Usage

`SLsearch_delete (SLsearch_Type *)`

Description

This function should be called to free the memory associated with a `SLsearch_Type` object created by the `SLsearch_new` function. Failure to do so will result in a memory leak.

See Also

`SLsearch_new`, `SLsearch_forward`, `SLsearch_backward`

3.3 SLsearch_forward

Synopsis

Search forward in a buffer

Usage

```c
SLuchar_Type SLsearch_forward (st, pmin, pmax)

   SLsearch_Type *st;
   SLuchar_Type *pmin, *pmax;
```
3. SLsearch interface Functions

Description

The SLsearch_forward function searches forward in the buffer defined by the pointers pmin and pmax. The starting point for the search is at the beginning of the buffer at pmin. At no point will the bytes at pmax and beyond be examined. The first parameter st, obtained by a prior call to SLsearch_new, specifies the object to found. If the object was found, the pointer to the beginning of it will be returned. Otherwise, SLsearch_forward will return NULL. The length of the object may be obtained via the SLsearch_match_len function.

Notes

This function uses the Boyer-Moore search algorithm when possible.

See Also

SLsearch_new, SLsearch_backward, SLsearch_delete, SLsearch_match_len

3.4 SLsearch_backward

Synopsis

Search backward in a buffer

Usage

SLuchar_Type SLsearch_forward (st, pmin, pstart, pmax)

SLsearch_Type *st;
SLuchar_Type *pmin, *pstart, *pmax;

Description

The SLsearch_forward function searches backward in the buffer defined by the pointers pmin and pmax. The starting point for the search is at the position pstart. At no point will the bytes at pmax and beyond be examined. The first parameter st, obtained by a prior call to SLsearch_new, specifies the object to found.

If the object was found, the pointer to the beginning of it will be returned. Otherwise, SLsearch_forward will return NULL. The length of the object may be obtained via the SLsearch_match_len function.

Notes

This function uses the Boyer-Moore search algorithm when possible.

It is possible for the end of match to appear after the point where the search began (pstart).

See Also

SLsearch_new, SLsearch_forward, SLsearch_delete, SLsearch_match_len
3.5 SLsearch_match_len

Synopsis
Get the length of the previous match

Usage
unsigned int SLsearch_match_len (SLsearch_Type *st)

Description
The SLsearch_match_len function returns the length of the match from the most recent search involving the specified SLsearch_Type object. If the most recent search was unsuccessful, the function will return 0.

See Also
SLsearch_forward, SLsearch_backward, SLsearch_new, SLsearch_delete

4 Screen Management (SLsmg) functions

4.1 SLsmg_fill_region

Synopsis
Fill a rectangular region with a character

Usage
void SLsmg_fill_region (r, c, nr, nc, ch)
   int r
   int c
   unsigned int nr
   unsigned int nc
   unsigned char ch

Description
The SLsmg_fill_region function may be used to a rectangular region with the character ch in the current color. The rectangle’s upper left corner is at row r and column c, and spans nr rows and nc columns. The position of the virtual cursor will be left at (r, c).

See Also
SLsmg_write_char, SLsmg_set_color

4.2 SLsmg_set_char_set

Synopsis
Turn on or off line drawing characters

Usage
void SLsmg_set_char_set (int a);
4. Screen Management (SLsmg) functions

Descriptio

SLsmg_set_char_set may be used to select or deselect the line drawing character set as the current character set. If a is non-zero, the line drawing character set will be selected. Otherwise, the standard character set will be selected.

Notes

There is no guarantee that this function will actually enable the use of line drawing characters. All it does is cause subsequent characters to be rendered using the terminal’s alternate character set. Such character sets usually contain line drawing characters.

See Also

SLsmg_write_char, SLtt_get_terminfo

4.3 int SLsmg_Scroll_Hash_Border;

Synopsis

Set the size of the border for the scroll hash

Usage

int SLsmg_Scroll_Hash_Border = 0;

Description

This variable may be used to ignore the characters that occur at the beginning and the end of a row when performing the hash calculation to determine whether or not a line has scrolled. The default value is zero which means that all the characters on a line will be used.

See Also

SLsmg_refresh

4.4 SLsmg_suspend_smg

Synopsis

Suspend screen management

Usage

int SLsmg_suspend_smg (void)

Description

SLsmg_suspend_smg can be used to suspend the state of the screen management facility during suspension of the program. Use of this function will reset the display back to its default state. The function SLsmg_resume_smg should be called after suspension.

It returns zero upon success, or -1 upon error.

This function is similar to SLsmg_reset_smg except that the state of the display prior to calling SLsmg_suspend_smg is saved.

See Also

SLsmg_resume_smg, SLsmg_reset_smg
4. Screen Management (SLsmg) functions

4.5 SLsmg_resume_smg

Synopsis
Resume screen management

Usage
int SLsmg_resume_smg (void)

Description
SLsmg_resume_smg should be called after SLsmg_suspend_smg to redraw the display exactly like it was before SLsmg_suspend_smg was called. It returns zero upon success, or -1 upon error.

See Also
SLsmg_suspend_smg

4.6 SLsmg_erase_eol

Synopsis
Erase to the end of the row

Usage
void SLsmg_erase_eol (void);

Description
SLsmg_erase_eol erases all characters from the current position to the end of the line. The newly created space is given the color of the current color. This function has no effect on the position of the virtual cursor.

See Also
SLsmg_gotorc, SLsmg_erase_eos, SLsmg_fill_region

4.7 SLsmg_gotorc

Synopsis
Move the virtual cursor

Usage
void SLsmg_gotorc (int r, int c)

Description
The SLsmg_gotorc function moves the virtual cursor to the row r and column c. The first row and first column is specified by r = 0 and c = 0.

See Also
SLsmg_refresh
4.8 SLsmg_erase_eos

Synopsis
Erase to the end of the screen

Usage
void SLsmg_erase_eos (void);

Description
The SLsmg_erase_eos is like SLsmg_erase_eol except that it erases all text from the current position to the end of the display. The current color will be used to set the background of the erased area.

See Also
SLsmg_erase_eol

4.9 SLsmg_reverse_video

Synopsis
Set the current color to 1

Usage
void SLsmg_reverse_video (void);

Description
This function is nothing more than SLsmg_set_color(1).

See Also
SLsmg_set_color

4.10 SLsmg_set_color (int)

Synopsis
Set the current color

Usage
void SLsmg_set_color (int c);

Description
SLsmg_set_color is used to set the current color. The parameter c is really a color object descriptor. Actual foreground and background colors as well as other visual attributes may be associated with a color descriptor via the SLtt_set_color function.

Example
This example defines color 7 to be green foreground on black background and then displays some text in this color:
4. Screen Management (SLsmg) functions

```c
SLtt_set_color (7, NULL, "green", "black");
SLsmg_set_color (7);
SLsmg_write_string ("Hello");
SLsmg_refresh ();
```

Notes

It is important to understand that the screen management routines know nothing about the actual colors associated with a color descriptor. Only the descriptor itself is used by the SLsmg routines. The lower level SLtt interface converts the color descriptors to actual colors. Thus

```c
SLtt_set_color (7, NULL, "green", "black");
SLsmg_set_color (7);
SLsmg_write_string ("Hello");
SLtt_set_color (7, NULL, "red", "blue");
SLsmg_write_string ("World");
SLsmg_refresh ();
```

will result in "hello" displayed in red on blue and not green on black.

See Also

SLtt_set_color, SLtt_set_color_object

4.11 SLsmg_normal_video

Synopsis

Set the current color to 0

Usage

```c
void SLsmg_normal_video (void);
```

Description

SLsmg_normal_video sets the current color descriptor to 0.

See Also

SLsmg_set_color

4.12 SLsmg_printf

Synopsis

Format a string on the virtual display

Usage

```c
void SLsmg_printf (char *fmt, ...)
```

Description

SLsmg_printf format a printf style variable argument list and writes it on the virtual display. The virtual cursor will be moved to the end of the string.

See Also

SLsmg_write_string, SLsmg_vprintf
4.13 SLsmg_vprintf

Synopsis
Format a string on the virtual display

Usage
void SLsmg_vprintf (char *fmt, va_list ap)

Description
SLsmg_vprintf formats a string in the manner of vprintf and writes the result to the display. The virtual cursor is advanced to the end of the string.

See Also
SLsmg_write_string, SLsmg_printf

4.14 SLsmg_write_string

Synopsis
Write a character string on the display

Usage
void SLsmg_write_string (char *s)

Description
The function SLsmg_write_string displays the string s on the virtual display at the current position and moves the position to the end of the string.

See Also
SLsmg_printf, SLsmg_write_nstring

4.15 SLsmg_write_nstring

Synopsis
Write the first n characters of a string on the display

Usage
void SLsmg_write_nstring (char *s, unsigned int n);

Description
SLsmg_write_nstring writes the first n characters of s to this virtual display. If the length of the string s is less than n, the spaces will used until n characters have been written. s can be NULL, in which case n spaces will be written.

See Also
SLsmg_write_string, SLsmg_write_nchars
4.16  SLsmg_write_char

Synopsis
Write a character to the virtual display

Usage
void SLsmg_write_char (char ch);

Description
SLsmg_write_char writes the character ch to the virtual display.

See Also
SLsmg_write_nchars, SLsmg_write_string

4.17  SLsmg_write_nchars

Synopsis
Write n characters to the virtual display

Usage
void SLsmg_write_nchars (char *s, unsigned int n);

Description
SLsmg_write_nchars writes at most n characters from the string s to the display. If the length of s is less than n, the whole length of the string will get written.

This function differs from SLsmg_write_nstring in that SLsmg_write_nstring will pad the string to write exactly n characters. SLsmg_write_nchars does not perform any padding.

See Also
SLsmg_write_nchars, SLsmg_write_nstring

4.18  SLsmg_write_wrapped_string

Synopsis
Write a string to the display with wrapping

Usage
void SLsmg_write_wrapped_string (s, r, c, nr, nc, fill)
     char *s
     int r, c
     unsigned int nr, nc
     int fill

Description
SLsmg_write_wrapped_string writes the string s to the virtual display. The string will be confined to the rectangular region whose upper right corner is at row r and column c, and consists of nr rows and nc columns. The string will be wrapped at the boundaries of the box. If fill is non-zero, the last line to which characters have been written will get padded with spaces.
4. Screen Management (SLsmg) functions

Notes
This function does not wrap on word boundaries. However, it will wrap when a newline character is encountered.

See Also
SLsmg_write_string

4.19 SLsmg_cls

Synopsis
Clear the virtual display

Usage
void SLsmg_cls (void)

Description
SLsmg_cls erases the virtual display using the current color. This will cause the physical display to get cleared the next time SLsmg_refresh is called.

Notes
This function is not the same as
SLsmg_gotorc (0,0); SLsmg_erase_eos ();
since these statements do not guarantee that the physical screen will get cleared.

See Also
SLsmg_refresh, SLsmg_erase_eos

4.20 SLsmg_refresh

Synopsis
Update physical screen

Usage
void SLsmg_refresh (void)

Description
The SLsmg_refresh function updates the physical display to look like the virtual display.

See Also
SLsmg_suspend_smg, SLsmg_init_smg, SLsmg_reset_smg
4.21 SLsmg_touch_lines

Synopsis
Mark lines on the virtual display for redisplay

Usage
void SLsmg_touch_lines (int r, unsigned int nr)

Description
SLsmg_touch_lines marks the nr lines on the virtual display starting at row r for redisplay upon the next call to SLsmg_refresh.

Notes
This function should rarely be called, if ever. If you find that you need to call this function, then your application should be modified to properly use the SLsmg screen management routines. This function is provided only for curses compatibility.

See Also
SLsmg_refresh

4.22 SLsmg_init_smg

Synopsis
Initialize the SLsmg routines

Usage
int SLsmg_init_smg (void)

Description
The SLsmg_init_smg function initializes the SLsmg screen management routines. Specifically, this function allocates space for the virtual display and calls SLtt_init_video to put the terminal’s physical display in the proper state. It is up to the caller to make sure that the SLtt routines are initialized via SLtt_get_terminfo before calling SLsmg_init_smg.

This function should also be called any time the size of the physical display has changed so that it can reallocate a new virtual display to match the physical display.

It returns zero upon success, or -1 upon failure.

See Also
SLsmg_reset_smg

4.23 SLsmg_reset_smg

Synopsis
Reset the SLsmg routines

Usage
int SLsmg_reset_smg (void);
Description

SLsmg_reset_smg resets the SLsmg screen management routines by freeing all memory allocated while it was active. It also calls SLtt_reset_video to put the terminal’s display in its default state.

See Also

SLsmg_init_smg

4.24 SLsmg_char_at

Synopsis

Get the character at the current position on the virtual display

Usage

unsigned short SLsmg_char_at(void)

Description

The SLsmg_char_at function returns the character and its color at the current position on the virtual display.

See Also

SLsmg_read_raw, SLsmg_write_char

4.25 SLsmg_set_screen_start

Synopsis

Set the origin of the virtual display

Usage

void SLsmg_set_screen_start (int *r, int *c)

Description

SLsmg_set_screen_start sets the origin of the virtual display to the row *r and the column *c. If either r or c is NULL, then the corresponding value will be set to 0. Otherwise, the location specified by the pointers will be updated to reflect the old origin.

See also slang/demo/pager.c for how this function may be used to scroll horizontally.

See Also

SLsmg_init_smg

4.26 SLsmg_draw_hline

Synopsis

Draw a horizontal line

Usage

void SLsmg_draw_hline (unsigned int len)
4. Screen Management (SLsmg) functions

Description
The `SLsmg_draw_hline` function draws a horizontal line of length `len` on the virtual display. The position of the virtual cursor is left at the end of the line.

See Also
`SLsmg_draw_vline`

4.27 SLsmg_draw_vline

Synopsis
Draw a vertical line

Usage
```c
void SLsmg_draw_vline (unsigned int len);
```

Description
The `SLsmg_draw_vline` function draws a vertical line of length `len` on the virtual display. The position of the virtual cursor is left at the end of the line.

See Also

4.28 SLsmg_draw_object

Synopsis
Draw an object from the alternate character set

Usage
```c
void SLsmg_draw_object (int r, int c, unsigned char obj);
```

Description
The `SLsmg_draw_object` function may be used to place the object specified by `obj` at row `r` and column `c`. The object is really a character from the alternate character set and may be specified using one of the following constants:

- `SLSMG_HLINE_CHAR` - Horizontal line
- `SLSMG_VLINE_CHAR` - Vertical line
- `SLSMG_ULCORN_CHAR` - Upper left corner
- `SLSMG_URCORN_CHAR` - Upper right corner
- `SLSMG_LLCORN_CHAR` - Lower left corner
- `SLSMG_LRCORN_CHAR` - Lower right corner
- `SLSMG_CKBRD_CHAR` - Checkboard character
- `SLSMG_RTEE_CHAR` - Right Tee
- `SLSMG_LTEE_CHAR` - Left Tee
- `SLSMG_UTEE_CHAR` - Up Tee
- `SLSMG_DTEE_CHAR` - Down Tee
- `SLSMG_PLUS_CHAR` - Plus or Cross character
4. Screen Management (SLsmg) functions

See Also
SLsmg_draw_vline, SLsmg_draw_hline, SLsmg_draw_box

4.29 SLsmg_draw_box

Synopsis
Draw a box on the virtual display

Usage
void SLsmg_draw_box (int r, int c, unsigned int dr, unsigned int dc)

Description
SLsmg_draw_box uses the SLsmg_draw_hline and SLsmg_draw_vline functions to draw a rectangular box on the virtual display. The box's upper left corner is placed at row r and column c. The width and length of the box is specified by dc and dr, respectively.

See Also
SLsmg_draw_vline, SLsmg_draw_hline, SLsmg_draw_object

4.30 SLsmg_set_color_in_region

Synopsis
Change the color of a specified region

Usage
void SLsmg_set_color_in_region (color, r, c, dr, dc)

Description
SLsmg_set_color_in_region may be used to change the color of a rectangular region whose upper left corner is given by (r, c), and whose width and height is given by dc and dr, respectively. The color of the region is given by the color parameter.

See Also
SLsmg_draw_box, SLsmg_set_color

4.31 SLsmg_get_column

Synopsis
Get the column of the virtual cursor

Usage
int SLsmg_get_column(void);
4. Screen Management (SLsmg) functions

Description

The `SLsmg_get_column` function returns the current column of the virtual cursor on the virtual display.

See Also

`SLsmg_get_row`, `SLsmg_gotorc`

4.32 SLsmg_get_row

Synopsis

Get the row of the virtual cursor

Usage

```c
int SLsmg_get_row(void);
```

Description

The `SLsmg_get_row` function returns the current row of the virtual cursor on the virtual display.

See Also

`SLsmg_get_column`, `SLsmg_gotorc`

4.33 SLsmg_forward

Synopsis

Move the virtual cursor forward n columns

Usage

```c
void SLsmg_forward (int n);
```

Description

The `SLsmg_forward` function moves the virtual cursor forward n columns.

See Also

`SLsmg_gotorc`

4.34 SLsmg_write_color_chars

Synopsis

Write characters with color descriptors to virtual display

Usage

```c
void SLsmg_write_color_chars (unsigned short *s, unsigned int len)
```

Description

The `SLsmg_write_color_chars` function may be used to write len characters, each with a different color descriptor to the virtual display. Each character and its associated color are encoded as an `unsigned short` such that the lower eight bits form the character and the next eight bits form the color.
4. Screen Management (SLsmg) functions

See Also

SLsmg_char_at, SLsmg_write_raw

4.35 SLsmg_read_raw

Synopsis

Read characters from the virtual display

Usage

unsigned int SLsmg_read_raw (SLsmg_Char_Type *buf, unsigned int len)

Description

SLsmg_read_raw attempts to read len characters from the current position on the virtual display into the buffer specified by buf. It returns the number of characters actually read. This number will be less than len if an attempt is made to read past the right margin of the display.

Notes

The purpose of the pair of functions, SLsmg_read_raw and SLsmg_write_raw, is to permit one to copy the contents of one region of the virtual display to another region.

See Also

SLsmg_char_at, SLsmg_write_raw

4.36 SLsmg_write_raw

Synopsis

Write characters directly to the virtual display

Usage

unsigned int SLsmg_write_raw (unsigned short *buf, unsigned int len)

Description

The SLsmg_write_raw function attempts to write len characters specified by buf to the display at the current position. It returns the number of characters successfully written, which will be less than len if an attempt is made to write past the right margin.

Notes

The purpose of the pair of functions, SLsmg_read_raw and SLsmg_write_raw, is to permit one to copy the contents of one region of the virtual display to another region.

See Also

SLsmg_read_raw
5 Functions that deal with the interpreter

5.1 SLallocate_load_type

Synopsis

Allocate a SLang_Load_Type object

Usage

SLang_Load_Type *SLallocate_load_type (char *name)

Description

The SLallocate_load_type function allocates and initializes space for a SLang_Load_Type object and returns it. Upon failure, the function returns NULL. The parameter name must uniquely identify the object. For example, if the object represents a file, then name could be the absolute path name of the file.

See Also

SLdeallocate_load_type, SLang_load_object

5.2 SLdeallocate_load_type

Synopsis

Free a SLang_Load_Type object

Usage

void SLdeallocate_load_type (SLang_Load_Type *slt)

Description

This function frees the memory associated with a SLang_Load_Type object that was acquired from a call to the SLallocate_load_type function.

See Also

SLallocate_load_type, SLang_load_object

5.3 SLang_load_object

Synopsis

Load an object into the interpreter

Usage

int SLang_load_object (SLang_Load_Type *obj)

Description

The function SLang_load_object is a generic function that may be used to loaded an object of type SLang_Load_Type into the interpreter. For example, the functions SLang_load_file and SLang_load_string are wrappers around this function to load a file and a string, respectively.

See Also

SLang_load_file, SLang_load_string, SLallocate_load_type
5. Functions that deal with the interpreter

5.4 SLclass_allocate_class

Synopsis
Allocate a class for a new data type

Usage
SLang_Class_Type *SLclass_allocate_class (char *name)

Description
The purpose of this function is to allocate and initialize space that defines a new data type or class called name. If successful, a pointer to the class is returned, or upon failure the function returns NULL.

This function does not automatically create the new data type. Callback functions must first be associated with the data type via functions such as SLclass_set_push_function, and the data type must be registered with the interpreter via SLclass_register_class. See the S-Lang library programmer's guide for more information.

See Also
SLclass_register_class, SLclass_set_push_function

5.5 SLclass_register_class

Synopsis
Register a new data type with the interpreter

Usage
int SLclass_register_class (cl, type, sizeof_type, class_type)

SLang_Class_Type *cl
SLtype type
unsigned int sizeof_type
SLclass_Type class_type

Description
The SLclass_register_class function is used to register a new class or data type with the interpreter. If successful, the function returns 0, or upon failure, it returns -1.

The first parameter, cl, must have been previously obtained via the SLclass_allocate_class function.

The second parameter, type specifies the data type of the new class. If set to SLANG_VOID_TYPE then the library will automatically allocate an unused value for the class (the allocated value can then be found using the SLclass_get_class_id function), otherwise a value greater than 255 should be used. The values in the range 0-255 are reserved for internal use by the library.

The size that the data type represents in bytes is specified by the third parameter, sizeof_type. This value should not be confused with the sizeof the structure that represents the data type, unless the data type is of class SLANG_CLASS_TYPE_VECTOR or SLANG_CLASS_TYPE_SCALAR. For pointer objects, the value of this parameter is just sizeof(void *)

The final parameter specifies the class type of the data type. It must be one of the values:
The `SLANG_CLASS_TYPE SCALAR` indicates that the new data type is a scalar. Examples of scalars in `SLANG_INT_TYPE` and `SLANG_DOUBLE_TYPE`.

Setting `class_type` to `SLANG_CLASS_TYPE VECTOR` implies that the new data type is a vector, or a 1-d array of scalar types. An example of a data type of this class is the `SLANG_COMPLEX_TYPE`, which represents complex numbers.

`SLANG_CLASS_TYPE_PTR` specifies the data type is of a pointer type. Examples of data types of this class include `SLANG_STRING_TYPE` and `SLANG_ARRAY_TYPE`. Such types must provide for their own memory management.

Data types of class `SLANG_CLASS_TYPE MMT` are pointer types except that the memory management, i.e., creation and destruction of the type, is handled by the interpreter. Such a type is called a *memory managed type*. An example of this data type is the `SLANG_FILEPTR_TYPE`.

Notes


See Also

`SLclass_allocate_class`, `SLclass_get_class_id`

5.6 `SLclass_set_string_function`

**Synopsis**

Set a data type’s string representation callback

**Usage**

```c
int SLclass_set_string_function (cl, sfun)

SLang_Class_Type *cl
char *(*sfun) (SLtype, VOID_STAR);
```

**Description**

The `SLclass_set_string_function` routine is used to define a callback function, `sfun`, that will be used when a string representation of an object of the data type represented by `cl` is needed. `cl` must have already been obtained via a call to `SLclass_allocate_class`. When called, `sfun` will be passed two arguments: an `SLtype` which represents the data type, and the address of the object for which a string representation is required. The callback function must return a *malloced* string.

Upon success, `SLclass_set_string_function` returns zero, or upon error it returns -1.

**Example**

A callback function that handles both `SLANG_STRING_TYPE` and `SLANG_INT_TYPE` variables looks like:
char *string_and_int_callback (SLtype type, VOID_STAR addr)
{
    char buf[64];

    switch (type)
    {
    case SLANG_STRING_TYPE:
        return SLmake_string (*(char **)addr);

    case SLANG_INTEGER_TYPE:
        sprintf (buf, "%d", *(int *)addr);
        return SLmake_string (buf);
    }
    return NULL;
}

Notes
The default string callback simply returns the name of the data type.

See Also
SLclass_allocate_class, SLclass_register_class

5.7 SLclass_set_destroy_function

Synopsis
Set the destroy method callback for a data type

Usage
int SLclass_set_destroy_function (cl, destroy_fun)
    SLang_Class_Type *cl
    void (*destroy_fun) (SLtype, VOID_STAR);

Description
SLclass_set_destroy_function is used to set the destroy callback for a data type. The data type’s class cl must have been previously obtained via a call to SLclass_allocate_class. When called, destroy_fun will be passed two arguments: an SLtype which represents the data type, and the address of the object to be destroyed.

SLclass_set_destroy_function returns zero upon success, and -1 upon failure.

Example
The destroy method for SLANG_STRING_TYPE looks like:

static void string_destroy (SLtype type, VOID_STAR ptr)
{
    char *s = *(char **) ptr;
    if (s != NULL) SLang_free_slstring (*(char **) s);
}
Notes

Data types of class SLANG_CLASS_TYPE_SCALAR do not require a destroy callback. However, other classes do.

See Also

SLclass_allocate_class, SLclass_register_class

5.8 SLclass_set_push_function

Synopsis

Set the push callback for a new data type

Usage

int SLclass_set_push_function (cl, push_fun)

SLang_Class_Type *cl
int (*push_fun) (SLtype, VOID_STAR);

Description

SLclass_set_push_function is used to set the push callback for a new data type specified by cl, which must have been previously obtained via SLclass_allocate_class.

The parameter push_fun is a pointer to the push callback. It is required to take two arguments: an SLtype representing the data type, and the address of the object to be pushed. It must return zero upon success, or -1 upon failure.

SLclass_set_push_function returns zero upon success, or -1 upon failure.

Example

The push callback for SLANG_COMPLEX_TYPE looks like:

    static int complex_push (SLtype type, VOID_STAR ptr)
    {
        double *z = *(double **) ptr;
        return SLang_push_complex (z[0], z[1]);
    }

See Also

SLclass_allocate_class, SLclass_register_class

5.9 SLclass_set_pop_function

Synopsis

Set the pop callback for a new data type

Usage

int SLclass_set_pop_function (cl, pop_fun)

SLang_Class_Type *cl
int (*pop_fun) (SLtype, VOID_STAR);
Description

SLclass_set_pop_function is used to set the callback for popping an object from the stack for a new data type specified by cl, which must have been previously obtained via SLclass_allocate_class. The parameter pop_fun is a pointer to the pop callback function, which is required to take two arguments: an unsigned character representing the data type, and the address of the object to be popped. It must return zero upon success, or -1 upon failure. SLclass_set_pop_function returns zero upon success, or -1 upon failure.

Example

The pop callback for SLANG_COMPLEX_TYPE looks like:

```c
static int complex_push (SLtype type, VOIDSTAR ptr)
{
  double *z = *(double **) ptr;
  return SLang_pop_complex (&z[0], &z[1]);
}
```

See Also

SLclass_allocate_class, SLclass_register_class

5.10 SLclass_get_datatype_name

Synopsis

Get the name of a data type

Usage

```c
char *SLclass_get_datatype_name (SLtype type)
```

Description

The SLclass_get_datatype_name function returns the name of the data type specified by type. For example, if type is SLANG_INT_TYPE, the string "Integer_Type" will be returned.

This function returns a pointer that should not be modified or freed.

See Also

SLclassAllocateClass, SLclassRegisterClass

5.11 SLang_free_mmt

Synopsis

Free a memory managed type

Usage

```c
void SLang_free_mmt (SLang_MMT_Type *mmt)
```

Description

The SLang_MMT_Type function is used to free a memory managed data type.

See Also

SLangObjectFromMmt, SLangCreateMmt
5. Functions that deal with the interpreter

5.12 SLang_object_from_mmt

Synopsis
Get a pointer to the value of a memory managed type

Usage
VOID_STAR SLang_object_from_mmt (SLang_MMT_Type *mmt)

Description
The SLang_object_from_mmt function returns a pointer to the actual object whose memory is being managed by the interpreter.

See Also
SLang_free_mmt, SLang_create_mmt

5.13 SLang_create_mmt

Synopsis
Create a memory managed data type

Usage
SLang_MMT_Type *SLang_create_mmt (SLtype t, VOID_STAR ptr)

Description
The SLang_create_mmt function returns a pointer to a new memory managed object. This object contains information necessary to manage the memory associated with the pointer ptr which represents the application defined data type of type t.

See Also
SLang_object_from_mmt, SLang_push_mmt, SLang_free_mmt

5.14 SLang_push_mmt

Synopsis
Push a memory managed type

Usage
int SLang_push_mmt (SLang_MMT_Type *mmt)

Description
This function is used to push a memory managed type onto the interpreter stack. It returns zero upon success, or -1 upon failure.

See Also
SLang_create_mmt, SLang_pop_mmt
5. Functions that deal with the interpreter

5.15 SLang_pop_mmt

Synopsis
Pop a memory managed data type

Usage
SLang_MMT_Type *SLang_pop_mmt (SLtype t)

Description
The SLang_pop_mmt function may be used to pop a memory managed type of type t from the stack. It returns a pointer to the memory managed object upon success, or NULL upon failure. The function SLang_object_from_mmt should be used to access the actual pointer to the data type.

See Also
SLang_object_from_mmt, SLang_push_mmt

5.16 SLang_inc_mmt

Synopsis
Increment a memory managed type reference count

Usage
void SLang_inc_mmt (SLang_MMT_Type *mmt);

Description
The SLang_inc_mmt function may be used to increment the reference count associated with the memory managed data type given by mmt.

See Also
SLang_free_mmt, SLang_create_mmt, SLang_pop_mmt, SLang_pop_mmt

5.17 SLadd_intrin_fun_table

Synopsis
Add a table of intrinsic functions to the interpreter

Usage
int SLadd_intrin_fun_table(SLang_Intrin_Fun_Type *tbl, char *pp_name);

Description
The SLadd_intrin_fun_table function adds an array, or table, of SLang_Intrin_Fun_Type objects to the interpreter. The first parameter, tbl specifies the table to be added. The second parameter pp_name, if non-NULL will be added to the list of preprocessor symbols.

This function returns -1 upon failure or zero upon success.
5. Functions that deal with the interpreter

Notes

A table should only be loaded one time and it is considered to be an error on the part of the application if it loads a table more than once.

See Also

SLadd_intrin_var_table, SLadd_intrinsic_function, SLdefine_for_ifdef

5.18 SLadd_intrin_var_table

Synopsis

Add a table of intrinsic variables to the interpreter

Usage

int SLadd_intrin_var_table (SLang_Intrin_Var_Type *tbl, char *pp_name);

Description

The SLadd_intrin_var_table function adds an array, or table, of SLang_Intrin_Var_Type objects to the interpreter. The first parameter, tbl specifies the table to be added. The second parameter pp_name, if non-NULL will be added to the list of preprocessor symbols.

This function returns -1 upon failure or zero upon success.

Notes

A table should only be loaded one time and it is considered to be an error on the part of the application if it loads a table more than once.

See Also

SLadd_intrin_var_table, SLadd_intrinsic_function, SLdefine_for_ifdef

5.19 SLang_load_file

Synopsis

Load a file into the interpreter

Usage

int SLang_load_file (char *fn)

Description

The SLang_load_file function opens the file whose name is specified by fn and feeds it to the interpreter, line by line, for execution. If fn is NULL, the function will take input from stdin.

If no error occurs, it returns 0; otherwise, it returns -1, and sets SLang_Error accordingly. For example, if it fails to open the file, it will return -1 with SLang_Error set to SL_OBJ_NOPEN.

Notes

If the hook SLang_Load_File_Hook declared as

int (*SLang_Load_File_Hook)(char *);
is non-NULL, the function point to by it will be used to load the file. For example, the jed editor uses
this hook to load files via its own routines.

See Also
SLang_load_object, SLang_load_string

5.20 SLang_restart

Synopsis
Reset the interpreter after an error

Usage
void SLang_restart (int full)

Description
The SLang_restart function should be called by the application at top level if an error occurs. If
the parameter full is non-zero, any objects on the S-Lang run time stack will be removed from the
stack; otherwise, the stack will be left intact. Any time the stack is believed to be trashed, this routine
should be called with a non-zero argument (e.g., if setjmp/longjmp is called).

Calling SLang_restart does not reset the global variable SLang_Error to zero. It is up to the appli-
cation to reset that variable to zero after calling SLang_restart.

Example

while (1)
{
    if (SLang_Error)
    {
        SLang_restart (1);
        SLang_Error = 0;
    }
    (void) SLang_load_file (NULL);
}

See Also
SLang_init_slang, SLang_load_file

5.21 SLang_byte_compile_file

Synopsis
Byte-compile a file for faster loading

Usage
int SLang_byte_compile_file(char *fn, int reserved)
Description

The `SLang_byte_compile_file` function “byte-compiles” the file `fn` for faster loading by the interpreter. This produces a new file whose filename is equivalent to the one specified by `fn`, except that a 'c' is appended to the name. For example, if `fn` is set to `init.sl`, then the new file will have the name `init.slc`. The meaning of the second parameter, `reserved`, is reserved for future use. For now, set it to 0.

The function returns zero upon success, or -1 upon error and sets `SLang_Error` accordingly.

See Also

`SLang_load_file`, `SLang_init_slang`

5.22 `SLang_autoload`

Synopsis

Autoload a function from a file

Usage

```c
int SLang autoload(char *funct, char *filename)
```

Description

The `SLang autoload` function may be used to associate a `slang` function name `funct` with the file `filename` such that if `funct` has not already been defined when needed, it will be loaded from `filename`. `SLang autoload` has no effect if `funct` has already been defined. Otherwise it declares `funct` as a user-defined `S-Lang` function. It returns 0 upon success, or -1 upon error.

See Also

`SLang_load_file`, `SLang_is_defined`

5.23 `SLang_load_string`

Synopsis

Interpret a string

Usage

```c
int SLang load_string(char *str)
```

Description

The `SLang load_string` function feeds the string specified by `str` to the interpreter for execution. It returns zero upon success, or -1 upon failure.

See Also

`SLang_load_file`, `SLang_load_object`
5. Functions that deal with the interpreter

5.24 SLdo_pop

Synopsis
Delete an object from the stack

Usage
int SLdo_pop(void)

Description
This function removes an object from the top of the interpreter's run-time stack and frees any memory
associated with it. It returns zero upon success, or -1 upon error (most likely due to a stack-underflow).

See Also
SLdo_pop_n, SLang_pop_integer, SLang_pop_string

5.25 SLdo_pop_n

Synopsis
Delete n objects from the stack

Usage
int SLdo_pop_n (unsigned int n)

Description
The SLdo_pop_n function removes the top n objects from the interpreter's run-time stack and frees all
memory associated with the objects. It returns zero upon success, or -1 upon error (most likely due
to a stack-underflow).

See Also
SLdo_pop, SLang_pop_integer, SLang_pop_string

5.26 SLang_pop_integer

Synopsis
Pop an integer off the stack

Usage
int SLang_pop_integer (int *i)

Description
The SLang_pop_integer function removes an integer from the top of the interpreter's run-time stack
and returns its value via the pointer i. If successful, it returns zero. However, if the top stack item is
not of type SLANG_INT_TYPE, or the stack is empty, the function will return -1 and set SLang_Error
accordingly.

See Also
SLang_push_integer, SLang_pop_double
5. Functions that deal with the interpreter

5.27 SLpop_string

Synopsis
Pop a string from the stack

Usage
int SLpop_string (char **strptr);

Description
The SLpop_string function pops a string from the stack and returns it as a malloced pointer. It is up to the calling routine to free this string via a call to free or SLfree. If successful, SLpop_string returns zero. However, if the top stack item is not of type SLANG_STRING_TYPE, or the stack is empty, the function will return -1 and set SLang_Error accordingly.

Example

    define print_string (void)
    {
        char *s;
        if (-1 == SLpop_string (&s))
            return;
        puts (s, stdout);
        SLfree (s);
    }

Notes
This function should not be confused with SLang_pop_slstring, which pops a hashed string from the stack.

See Also
SLang_pop_slstring, SLfree

5.28 SLang_pop_string

Synopsis
Pop a string from the stack

Usage
int SLang_pop_string(char **strptr, int *do_free)

Description
The SLpop_string function pops a string from the stack and returns it as a malloced pointer via strptr. After the function returns, the integer pointed to by the second parameter will be set to a non-zero value if *strptr should be freed via free or SLfree. If successful, SLpop_string returns zero. However, if the top stack item is not of type SLANG_STRING_TYPE, or the stack is empty, the function will return -1 and set SLang_Error accordingly.
5. Functions that deal with the interpreter

Notes
This function is considered obsolete and should not be used by applications. If one requires a malloced string for modification, SLpop_string should be used. If one requires a constant string that will not be modified by the application, SLang_pop_slstring should be used.

See Also
SLang_pop_slstring, SLpop_string

5.29 SLang_pop_slstring

Synopsis
Pop a hashed string from the stack

Usage
int SLang_pop_slstring (char **s_ptr)

Description
The SLang_pop_slstring function pops a hashed string from the S-Lang run-time stack and returns it via s_ptr. It returns zero if successful, or -1 upon failure. The resulting string should be freed via a call to SLang_free_slstring after use.

Example

```c
void print_string (void)
{
    char *s;
    if (-1 == SLang_pop_slstring (&s))
        return;
    fprintf (stdout, "%s\n", s);
    SLang_free_slstring (s);
}
```

Notes
SLang_free_slstring is the preferred function for popping strings. This is a result of the fact that the interpreter uses hashed strings as the native representation for string data.

One must never free a hashed string using free or SLfree. In addition, one must never make any attempt to modify a hashed string and doing so will result in memory corruption.

See Also
SLang_free_slstring, SLpop_string

5.30 SLang_pop_double

Synopsis
Pop a double from the stack

Usage
int SLang_pop_double (double *dptr)
5. Functions that deal with the interpreter

Description

The `SLang_pop_double` function pops a double precision number from the stack and returns it via `dptr`. This function returns 0 upon success, otherwise it returns -1 and sets `SLang_Error` accordingly.

See Also

`SLang_pop_integer`, `SLang_push_double`

5.31 `SLang_pop_complex`

Synopsis

Pop a complex number from the stack

Usage

```c
int SLang_pop_complex (double *re, double *im)
```

Description

`SLang_pop_complex` pops a complex number from the stack and returns it via the parameters `re` and `im` as the real and imaginary parts of the complex number, respectively. This function automatically converts objects of type `SLANG_DOUBLE_TYPE` and `SLANG_INT_TYPE` to `SLANG_COMPLEX_TYPE`, if necessary. It returns zero upon success, or -1 upon error setting `SLang_Error` accordingly.

See Also

`SLang_pop_integer`, `SLang_pop_double`, `SLang_push_complex`

5.32 `SLang_push_complex`

Synopsis

Push a complex number onto the stack

Usage

```c
int SLang_push_complex (double re, double im)
```

Description

`SLang_push_complex` may be used to push the complex number whose real and imaginary parts are given by `re` and `im`, respectively. It returns zero upon success, or -1 upon error setting `SLang_Error` accordingly.

See Also

`SLang_pop_complex`, `SLang_push_double`

5.33 `SLang_push_double`

Synopsis

Push a double onto the stack
5. Functions that deal with the interpreter

Usage

```
int SLang_push_double(double d)
```

Description

`SLang_push_double` may be used to push the double precision floating point number `d` onto the interpreter's run-time stack. It returns zero upon success, or -1 upon error setting `SLang_Error` accordingly.

See Also

`SLang_pop_double, SLang_push_integer`

5.34 `SLang_push_string`

Synopsis

Push a string onto the stack

Usage

```
int SLang_push_string (char *s)
```

Description

`SLang_push_string` pushes a copy of the string specified by `s` onto the interpreter's run-time stack. It returns zero upon success, or -1 upon error setting `SLang_Error` accordingly.

Notes

If `s` is `NULL`, this function pushes `NULL` (`SLANG_NULL_TYPE`) onto the stack.

See Also

`SLang_push_malloced_string`

5.35 `SLang_push_integer`

Synopsis

Push an integer onto the stack

Usage

```
int SLang_push_integer (int i)
```

Description

`SLang_push_integer` the integer `i` onto the interpreter's run-time stack. It returns zero upon success, or -1 upon error setting `SLang_Error` accordingly.

See Also

`SLang_pop_integer, SLang_push_double, SLang_push_string`
5.36 SLang_push_malloced_string

Synopsis
Push a malloced string onto the stack

Usage
int SLang_push_malloced_string (char *s);

Description
SLang_push_malloced_string may be used to push a malloced string onto the interpreter’s run-time stack. It returns zero upon success, or -1 upon error setting SLang_Error accordingly.

Example
The following example illustrates that it is up to the calling routine to free the string if SLang_push_malloced_string fails:

```c
int push_hello (void)
{
    char *s = malloc (6);
    if (s == NULL) return -1;
    strcpy (s, "hello");
    if (-1 == SLang_push_malloced_string (s))
    {
        free (s);
        return -1;
    }
    return 0;
}
```

Example
The function SLang_create_slstring returns a hashed string. Such a string may not be malloced and should not be passed to SLang_push_malloced_string.

Notes
If s is NULL, this function pushes NULL (SLANG_NULL_TYPE) onto the stack.

See Also
SLang_push_string, SLmake_string

5.37 SLang_is_defined

Synopsis
Check to see if the interpreter defines an object

Usage
int SLang_is_defined (char *nm)
5. Functions that deal with the interpreter

Description
The `SLang_is_defined` function may be used to determine whether or not a variable or function whose name is given by `em` has been defined. It returns zero if no such object has been defined. Otherwise it returns a non-zero value according to the following table:

1 intrinsic function  
2 user-defined slang function  
-1 intrinsic variable  
-2 user-defined global variable

Note that variables correspond to negative numbers and functions are represented by positive numbers.

See Also
`SLadd_intrinsic_function`, `SLang_run_hooks`, `SLang_execute_function`

5.38 SLang_run_hooks

Synopsis
Run a user-defined hook with arguments

Usage
```
int SLang_run_hooks (char *fname, unsigned int n, ...)
```

Description
The `SLang_run_hooks` function may be used to execute a user-defined function named `fname`. Before execution of the function, the `n` string arguments specified by the variable parameter list are pushed onto the stack. If the function `fname` does not exist, `SLang_run_hooks` returns zero; otherwise, it returns 1 upon successful execution of the function, or -1 if an error occurred.

Example
The `jed` editor uses `SLang_run_hooks` to setup the mode of a buffer based on the filename extension of the file associated with the buffer:

```
char *ext = get_filename_extension (filename);
if (ext == NULL) return -1;
if (-1 == SLang_run_hooks ("mode_hook", 1, ext))
    return -1;
return 0;
```

See Also
`SLang_is_defined`, `SLang_execute_function`

5.39 SLang_execute_function

Synopsis
Execute a user or intrinsic function

Usage
```
int SLang_execute_function (char *fname)
```
5. Functions that deal with the interpreter

Description
This function may be used to execute either a user-defined function or an intrinsic function. The name of the function is specified by \texttt{fname}. It returns zero if \texttt{fname} is not defined, or 1 if the function was successfully executed, or -1 upon error.

Notes
The function \texttt{SLexecute_function} may be a better alternative for some uses.

See Also
\texttt{SLang_run_hooks, SLexecute_function, SLang_is_defined}

5.40 SLang_get_function

Synopsis
Get a pointer to a \texttt{S-Lang} function

Usage
\begin{verbatim}
SLang_Name_Type *SLang_get_function (char *fname)
\end{verbatim}

Description
This function returns a pointer to the internal \texttt{S-Lang} table entry of a function whose name is given by \texttt{fname}. It returns \texttt{NULL} upon failure. The value returned by this function can be used \texttt{SLexecute_function} to call the function directly from C.

See Also
\texttt{SLexecute_function}

5.41 SLexecute_function

Synopsis
Execute a \texttt{S-Lang} or intrinsic function

Usage
\begin{verbatim}
int SLexecute_function (SLang_Name_Type *nt)
\end{verbatim}

Description
The \texttt{SLexecute_function} allows an application to call the \texttt{S-Lang} function specified by the \texttt{SLang_Name_Type} pointer \texttt{nt}. This parameter must be non \texttt{NULL} and must have been previously obtained by a call to \texttt{SLang_get_function}.

Example
Consider the \texttt{S-Lang} function:
\begin{verbatim}
define my_fun (x)
{
    return x^2 - 2;
}
\end{verbatim}
Suppose that it is desired to call this function many times with different values of \( x \). There are at least two ways to do this. The easiest way is to use `SLang_execute_function` by passing the string "my_fun". A better way that is much faster is to use `SLexecute_function`:

```c
int sum_a_function (char *fname, double *result)
{
    double sum, x, y;
    SLang_Name_Type *nt;

    if (NULL == (nt = SLang_get_function (fname)))
        return -1;

    sum = 0;
    for (x = 0; x < 10.0; x += 0.1)
    {
        SLang_start_arg_list ();
        if (-1 == SLang_push_double (x))
            return -1;
        SLang_end_arg_list ();
        if (-1 == SLexecute_function (nt))
            return -1;
        if (-1 == SLang_pop_double (&y))
            return -1;
        sum += y;
    }
    return sum;
}
```

Although not necessary in this case, `SLang_start_arg_list` and `SLang_end_arg_list` were used to provide the function with information about the number of parameters passed to it.

**See Also**

`SLang_get_function`, `SLang_start_arg_list`, `SLang_end_arg_list`

### 5.42 SLang_peek_at_stack

**Synopsis**

Find the type of object on the top of the stack

**Usage**

```c
int SLang_peek_at_stack (void)
```

**Description**

The `SLang_peek_at_stack` function is useful for determining the data type of the object at the top of the stack. It returns the data type, or -1 upon a stack-underflow error. It does not remove anything from the stack.

**See Also**

`SLang_pop_string`, `SLang_pop_integer`
5.43 SLang_pop_fileptr

Synopsis
Pop a file pointer

Usage
int SLang_pop_fileptr (SLang_MMT_Type **mmt, FILE **fp)

Description
SLang_pop_fileptr pops a file pointer from the S-Lang run-time stack. It returns zero upon success, or -1 upon failure.

A S-Lang file pointer (SLANG_FILEPTR_TYPE) is actually a memory managed object. For this reason, SLang_pop_fileptr also returns the memory managed object via the argument list. It is up to the calling routine to call SLang_free_mmt to free the object.

Example
The following example illustrates an application defined intrinsic function that writes a user defined double precision number to a file. Note the use of SLang_free_mmt:

```c
int write_double (void)
{
    double t;
    SLang_MMT_Type *mmt;
    FILE *fp;
    int status;

    if (-1 == SLang_pop_double (&d, NULL, NULL))
        return -1;
    if (-1 == SLang_pop_fileptr (&mmt, &fp))
        return -1;
    status = fwrite (&d, sizeof (double), 1, fp);
    SLang_free_mmt (mmt);
    return status;
}
```

This function can be used by a S-Lang function as follows:

```c
define write_some_values ()
{
    variable fp, d;

    fp = fopen ("myfile.dat", "wb");
    if (fp == NULL)
        error ("file failed to open");
    for (d = 0; d < 10.0; d += 0.1)
    {
       if (-1 == write_double (fp, d))
           error ("write failed");
    }
    if (-1 == fclose (fp))
error ("fclose failed");
}

See Also
SLang_free_mmt, SLang_pop_double

5.44 SLadd_intrinsic_function

Synopsis
Add a new intrinsic function to the interpreter

Usage
int SLadd_intrinsic_function (name, f, type, nargs, ...)

char *name
FVOID_STAR f
SLtype type
unsigned int nargs

Description
The SLadd_intrinsic_function function may be used to add a new intrinsic function. The S-Lang name of the function is specified by name and the actual function pointer is given by f, cast to FVOID_STAR. The third parameter, type specifies the return type of the function and must be one of the following values:

SLANG_VOID_TYPE (returns nothing)
SLANG_INT_TYPE (returns int)
SLANG_DOUBLE_TYPE (returns double)
SLANG_STRING_TYPE (returns char *)

The nargs parameter specifies the number of parameters to pass to the function. The variable argument list following nargs must consists of nargs integers which specify the data type of each argument.

The function returns zero upon success or -1 upon failure.

Example
The jed editor uses this function to change the system intrinsic function to the following:

static int jed_system (char *cmd)
{
    if (Jed_Secure_Mode)
    {
        msg_error ("Access denied.");
        return -1;
    }
    return SLsystem (cmd);
}

After initializing the interpreter with SLang_init_slang, jed calls SLadd_intrinsic_function to substitute the above definition for the default S-Lang definition:
if (-1 == SLadd_intrinsic_function ("system", (VOID_STAR)&_ed_system,
SLANG_INT_TYPE, 1,
SLANG_STRING_TYPE))

return -1;

See Also
SLadd_intrinsic_variable, SLadd_intrinsic_array

5.45 SLadd_intrinsic_variable

Synopsis
Add an intrinsic variable to the interpreter

Usage
int SLadd_intrinsic_variable (name, addr, type, rdonly)
    char *name
    VOID_STAR addr
    SLtype type
    int rdonly

Description
The SLadd_intrinsic_variable function adds an intrinsic variable called name to the interpreter. The second parameter addr specifies the address of the variable (cast to VOID_STAR). The third parameter, type, specifies the data type of the variable. If the fourth parameter, rdonly, is non-zero, the variable will interpreted by the interpreter as read-only.

If successful, SLadd_intrinsic_variable returns zero, otherwise it returns -1.

Example
Suppose that My_Global_Int is a global variable (at least not a local one):
    int My_Global_Int;
It can be added to the interpreter via the function call
    if (-1 == SLadd_intrinsic_variable ("MyGlobalInt",
        (VOID_STAR)&My_Global_Int,
        SLANG_INT_TYPE, 0))
        exit (1);

Notes
The current implementation requires all pointer type intrinsic variables to be read-only. For example,
    char *My_Global_String;
is of type SLANG_STRING_TYPE, and must be declared as read-only. Finally, not that
    char My_Global_Char_BUF[256];
is not a SLANG_STRING_TYPE object. This difference is very important because internally the interpreter dereferences the address passed to it to get to the value of the variable.

See Also
SLadd_intrinsic_function, SLadd_intrinsic_array
5.46 SLclass_add_unary_op

Synopsis

??

Usage

int SLclass_add_unary_op (SLtype, int (*)(int, SLtype, VOID_STAR, unsigned int, VOID_STAR), int (*)(int, SLtype, SLtype *));

Description

??

See Also

??

5.47 SLclass_add_app_unary_op

Synopsis

??

Usage

int SLclass_add_app_unary_op (SLtype, int (*)(int, SLtype, VOID_STAR, unsigned int, VOID_STAR), int (*)(int, SLtype, SLtype *));

Description

??

See Also

??

5.48 SLclass_add_binary_op

Synopsis

??

Usage

int SLclass_add_binary_op (SLtype, SLtype, int (*)(int, SLtype, VOID_STAR, unsigned int, SLtype, VOID_STAR, unsigned int, VOID_STAR), int (*)(int, SLtype, SLtype, SLtype *));

Description

??

See Also

??
5.49 SLclass_add_math_op

Synopsis

Usage

```
int SLclass_add_math_op (SLtype, int (*)(int, SLtype, VOID_STAR, unsigned
int, VOID_STAR), int (*)(int, SLtype, SLtype *));
```

Description

See Also

---

5.50 SLclass_add_typecast

Synopsis

Usage

```
int SLclass_add_typecast (SLtype, SLtype int (*)_PROTO((SLtype, VOID_STAR, unsigned
int, SLtype, VOID_STAR)), int);
```

Description

See Also

---

6 Library Initialization Functions

6.1 SLang_init_slang

Synopsis

Initialize the interpreter

Usage

```
int SLang_init_slang (void)
```

Description

The `SLang_init_slang` function must be called by all applications that use the S-Lang interpreter. It initializes the interpreter, defines the built-in data types, and adds a set of core intrinsic functions.

The function returns 0 upon success, or -1 upon failure.

See Also

`SLang_init_slfile`, `SLang_init_slmath`, `SLang_init_slunix`
6. Library Initialization Functions

6.2 SLang_init_slfile

Synopsis

Initialize the interpreter file I/O intrinsics

Usage

int SLang_init_slfile (void)

Description

This function initializes the interpreters file I/O intrinsic functions. This function adds intrinsic functions such as fopen, fclose, and fputs to the interpreter. It returns 0 if successful, or -1 upon error.

Notes

Before this function can be called, it is first necessary to call SLang_init_slang. It also adds the preprocessor symbol __SLFILE__ to the interpreter.

See Also

SLang_init_slang, SLang_init_slunix, SLang_init_slmath

6.3 SLang_init_slmath

Synopsis

Initialize the interpreter math intrinsics

Usage

int SLang_init_slmath (void)

Description

The SLang_init_slmath function initializes the interpreter's mathematical intrinsic functions and makes them available to the language. The intrinsic functions include sin, cos, tan, etc.. It returns 0 if successful, or -1 upon failure.

Notes

This function must be called after SLang_init_slang. It adds the preprocessor symbol __SLMATH__ to the interpreter.

See Also

SLang_init_slang, SLang_init_slfile, SLang_init_slunix

6.4 SLang_init_slunix

Synopsis

Make available some unix system calls to the interpreter

Usage

int SLang_init_slunix (void)
7. Miscellaneous Functions

7.1 SLcurrent_time_string

Synopsis
Get the current time as a string

Usage
char *SLcurrent_time_string (void)

Description
The SLcurrent_time_string function uses the C library function ctime to obtain a string representation of the current date and time in the form
"Wed Dec 10 12:50:28 1997"
However, unlike the ctime function, a newline character is not present in the string.
The returned value points to a statically allocated memory block which may get overwritten on subsequent function calls.

See Also
SLmake_string

7.2 SLatoi

Synopsis
Convert a text string to an integer

Usage
int SLatoi(unsigned char *str

Description
SLatoi parses the string str to interpret it as an integer value. Unlike atoi, SLatoi can also parse strings containing integers expressed in hexadecimal (e.g., "0x7F") and octal (e.g., "012") notation.

See Also
SLang_guess_type
7. Miscellaneous Functions

7.3 SLextract_list_element

Synopsis

Extract a substring of a delimited string

Usage

```c
int SLextract_list_element (dlist, nth, delim, buf, buflen)
```

```c
dlist;
unsigned int nth;
delim;
buf;
unsigned int buflen;
```

Description

SLextract_list_element may be used to obtain the nth element of a list of strings, dlist, that are delimited by the character delim. The routine copies the nth element of dlist to the buffer buf whose size is buflen characters. It returns zero upon success, or -1 if dlist does not contain an nth element.

Example

A delimited list of strings may be turned into an array of strings as follows. For conciseness, all malloc error checking has been omitted.

```c
int list_to_array (char *list, char delim, char ***ap)
{
    unsigned int nth;
    char **a;
    char buf[1024];

    /* Determine the size of the array */
    nth = 0;
    while (0 == SLextract_list_element (list, nth, delim, buf, sizeof(buf)))
        nth++;

    ap = (char **) SLmalloc ((nth + 1) * sizeof (char **));
    nth = 0;
    while (0 == SLextract_list_element (list, nth, delim, buf, sizeof(buf)))
    {
        a[nth] = SLmake_string (buf);
        nth++;
    }
    a[nth] = NULL;
    *ap = a;
    return 0;
}
```

See Also

SLmalloc, SLmake_string
8 Error and Messaging Functions

8.1 SLang_verror

Synopsis
Signal an error with a message

Usage
void SLang_verror (int code, char *fmt, ...);

Description
The SLang_verror function sets SLang_Error to code if SLang_Error is 0. It also displays the error message implied by the printf variable argument list using fmt as the format.

Example

FILE *open_file (char *file)
{
    char *file = "my_file.dat";
    if (NULL == (fp = fopen (file, "w")))
        SLang_verror (SL_INTRINSIC_ERROR, "Unable to open %s", file);
    return fp;
}

See Also
SLang_vmessage, SLang_exit_error

8.2 SLang_doerror

Synopsis
Signal an error

Usage
void SLang_doerror (char *err_str)

Description
The SLang_doerror function displays the string err_str to the error device and signals a S-Lang error.

Notes
SLang_doerror is considered to obsolete. Applications should use the SLang_verror function instead.

See Also
SLang_verror, SLang_exit_error
8.3 SLang_vmessage

Synopsis
Display a message to the message device

Usage
void SLang_vmessage (char *fmt, ...)

Description
This function prints a printf style formatted variable argument list to the message device. The default message device is stdout.

See Also
SLang_error

8.4 SLang_exit_error

Synopsis
Exit the program and display an error message

Usage
void SLang_exit_error (char *fmt, ...)

Description
The SLang_exit_error function terminates the program and displays an error message using a printf type variable argument list. The default behavior to this function is to write the message to stderr and exit with the exit system call.

If the function pointer SLang_Exit_Error_Hook is non-NULL, the function to which it points will be called. This permits an application to perform whatever cleanup is necessary. This hook has the prototype:

void (*SLang_Exit_Error_Hook)(char *, va_list);

See Also
SLang_error, exit

9 String and Memory Allocation Functions

9.1 SLmake_string

Synopsis
Duplicate a string

Usage
char *SLmake_string (char *s)
Description

The `SLmake_string` function creates a new copy of the string `s`, via `malloc`, and returns it. Upon failure it returns `NULL`. Since the resulting string is malloced, it should be freed when no longer needed via a call to either `free` or `SLfree`.

Notes

`SLmake_string` should not be confused with the function `SLang_create_slstring`, which performs a similar function.

See Also

`SLmake_nstring`, `SLfree`, `SLmalloc`, `SLang_create_slstring`

9.2 `SLmake_nstring`

Synopsis

Duplicate a substring

Usage

```
char *SLmake_nstring (char *s, unsigned int n)
```

Description

This function is like `SLmake_string` except that it creates a null terminated string formed from the first `n` characters of `s`. Upon failure, it returns `NULL`, otherwise it returns the new string. When no longer needed, the returned string should be freed with `SLfree`.

See Also

`SLmake_string`, `SLfree`, `SLang_create_nslstring`

9.3 `SLang_create_nslstring`

Synopsis

Created a hashed substring

Usage

```
char *SLang_create_nslstring (char *s, unsigned int n)
```

Description

`SLang_create_nslstring` is like `SLang_create_slstring` except that only the first `n` characters of `s` are used to create the hashed string. Upon error, it returns `NULL`, otherwise it returns the hashed substring. Such a string must be freed by the function `SLang_free_slstring`.

Notes

Do not use `free` or `SLfree` to free the string returned by `SLang_create_slstring` or `SLang_create_nslstring`. Also it is important that no attempt is made to modify the hashed string returned by either of these functions. If one needs to modify a string, the functions `SLmake_string` or `SLmake_nstring` should be used instead.

See Also

`SLang_free_slstring`, `SLang_create_slstring`, `SLmake_nstring`
9.4 SLang_create_slstring

Synopsis
Create a hashed string

Usage
char *SLang_create_slstring (char *s)

Description
The SLang_create_slstring creates a copy of s and returns it as a hashed string. Upon error, the function returns NULL, otherwise it returns the hashed string. Such a string must only be freed via the SLang_free_slstring function.

Notes
Do not use free or SLfree to free the string returned by SLang_create_slstring or SLang_create_nslstring. Also it is important that no attempt is made to modify the hashed string returned by either of these functions. If one needs to modify a string, the functions SLmake_string or SLmake_nstring should be used instead.

See Also
SLang_free_slstring, SLang_create_nslstring, SLmake_string

9.5 SLang_free_slstring

Synopsis
Free a hashed string

Usage
void SLang_free_slstring (char *s)

Description
The SLang_free_slstring function is used to free a hashed string such as one returned by SLang_create_slstring, SLang_create_nslstring, or SLang_create_static_slstring. If s is NULL, the routine does nothing.

See Also
SLang_create_slstring, SLang_create_nslstring, SLang_create_static_slstring

9.6 SLang_concat_slstrings

Synopsis
Concatenate two strings to produce a hashed string

Usage
char *SLang_concat_slstrings (char *a, char *b)
9. String and Memory Allocation Functions

Description
The `SLang_concat_slstrings` function concatenates two strings, `a` and `b`, and returns the result as a hashed string. Upon failure, `NULL` is returned.

Notes
A hashed string can only be freed using `SLang_free_slstring`. Never use `free` or `Sfree` to free a hashed string, otherwise memory corruption will result.

See Also
`SLang_free_slstring`, `SLang_create_slstring`

9.7 `SLang_create_static_slstring`

Synopsis
Create a hashed string

Usage
```c
char *SLang_create_static_slstring (char *s_literal)
```

Description
The `SLang_create_static_slstring` creates a hashed string from the string literal `s_literal` and returns the result. Upon failure it returns `NULL`.

Example
```c
cchar *create_hello (void)
{
    return SLang_create_static_slstring ("hello");
}
```

Notes
This function should only be used with string literals.

See Also
`SLang_create_slstring`, `SLang_create_nslstring`

9.8 `SLmalloc`

Synopsis
Allocate some memory

Usage
```c
char *SLmalloc (unsigned int nbytes)
```

Description
This function uses `malloc` to allocate `nbytes` of memory. Upon error it returns `NULL`; otherwise it returns a pointer to the allocated memory. One should use `Sfree` to free the memory after use.

See Also
`Sfree`, `Srealloc`, `Scalloc`
9.9  SLcalloc

Synopsis
Allocate some memory

Usage
char *SLcalloc (unsigned int num_elem, unsigned int elem_size)

Description
This function uses calloc to allocate memory for num_elem objects with each of size elem_size and returns the result. In addition, the newly allocated memory is zeroed. Upon error it returns NULL; otherwise it returns a pointer to the allocated memory. One should use SLfree to free the memory after use.

See Also
SLmalloc, SLrealloc, SLfree

9.10  SLfree

Synopsis
Free some allocated memory

Usage
void SLfree (char *ptr)

Description
The SLfree function deallocates the memory specified by ptr, which may be NULL in which case the function does nothing.

Notes
Never use this function to free a hashed string returned by one of the family of slstring functions, e.g., SLang_pop_slstring.

See Also
SLmalloc, SLcalloc, SLrealloc, SLmake_string

9.11  SLrealloc

Synopsis
Resize a dynamic memory block

Usage
char *SLrealloc (char *ptr, unsigned int new_size)
10. Keyboard Input Functions

10.1 SLang_init_tty

Synopsis

Initialize the terminal keyboard interface

Usage

int SLang_init_tty (int intr_ch, int no_flow_ctrl, int opost)

Description

SLang_init_tty initializes the terminal for single character input. If the first parameter intr_ch is in the range 0-255, it will be used as the interrupt character, e.g., under Unix this character will generate a SIGINT signal. Otherwise, if it is -1, the interrupt character will be left unchanged.

If the second parameter no_flow_ctrl is non-zero, flow control (XON/XOFF) processing will be enabled.

If the last parameter opost is non-zero, output processing by the terminal will be enabled. If one intends to use this function in conjunction with the S-Lang screen management routines (SLsmg), this parameter should be set to zero.

SLang_init_tty returns zero upon success, or -1 upon error.

Notes

Terminal I/O is a complex subject. The S-Lang interface presents a simplification that the author has found useful in practice. For example, the only special character processing that SLang_init_tty enables is that of the SIGINT character, and the generation of other signals via the keyboard is disabled. However, generation of the job control signal SIGTSTP is possible via the Sltty_set_suspend_state function.

Under Unix, the integer variable SLang_TT_Read_FD is used to specify the input descriptor for the terminal. If SLang_TT_Read_FD represents a terminal device as determined via the isatty system call, then it will be used as the terminal file descriptor. Otherwise, the terminal device /dev/tty will be used as the input device. The default value of SLang_TT_Read_FD is -1 which causes /dev/tty to be used. So, if you prefer to use stdin for input, then set SLang_TT_Read_FD to fileno(stdin) before calling SLang_init_tty.

If the variable SLang_TT_Baud_Rate is zero when this function is called, the function will attempt to determine the baud rate by querying the terminal driver and set SLang_TT_Baud_Rate to that value.
See Also

SLang_reset_tty, SLang_getkey, SLtty_set_suspend_state

10.2 SLang_reset_tty

Synopsis

Reset the terminal

Usage

void SLang_reset_tty (void)

Description

SLang_reset_tty resets the terminal interface back to the state it was in before SLang_init_tty was called.

See Also

SLang_init_tty

10.3 SLtty_set_suspend_state

Synopsis

Enable or disable keyboard suspension

Usage

void SLtty_set_suspend_state (int s)

Description

The SLtty_set_suspend_state function may be used to enable or disable keyboard generation of the SIGTSTP job control signal. If s is non-zero, generation of this signal via the terminal interface will be enabled, otherwise it will be disabled.

This function should only be called after the terminal driver has been initialized via SLang_init_tty. The SLang_init_tty always disables the generation of SIGTSTP via the keyboard.

See Also

SLang_init_tty

10.4 SLang_getkey

Synopsis

Read a character from the keyboard

Usage

unsigned int SLang_getkey (void);
Description

The `SLang_getkey` reads a single character from the terminal and returns it. The terminal must first be initialized via a call to `SLang_init_tty` before this function can be called. Upon success, `SLang_getkey` returns the character read from the terminal, otherwise it returns `SLANG_GETKEY_ERROR`.

See Also

`SLang_init_tty`, `SLang_input_pending`, `SLang_ungetkey`

10.5 `SLang_ungetkey_string`

Synopsis

Unget a key string

Usage

```c
int SLang_ungetkey_string (unsigned char *buf, unsigned int n)
```

Description

The `SLang_ungetkey_string` function may be used to push the `n` characters pointed to by `buf` onto the buffered input stream that `SLgetkey` uses. If there is not enough room for the characters, -1 is returned and none are buffered. Otherwise, it returns zero.

Notes

The difference between `SLang_buffer_keystring` and `SLang_ungetkey_string` is that the `SLang_buffer_keystring` appends the characters to the end of the getkey buffer, whereas `SLang_ungetkey_string` inserts the characters at the beginning of the input buffer.

See Also

`SLang_ungetkey`, `SLang_getkey`

10.6 `SLang_buffer_keystring`

Synopsis

Append a keystring to the input buffer

Usage

```c
int SLang_buffer_keystring (unsigned char *b, unsigned int len)
```

Description

`SLang_buffer_keystring` places the `len` characters specified by `b` at the end of the buffer that `SLang_getkey` uses. Upon success it returns 0; otherwise, no characters are buffered and it returns -1.

Notes

The difference between `SLang_buffer_keystring` and `SLang_ungetkey_string` is that the `SLang_buffer_keystring` appends the characters to the end of the getkey buffer, whereas `SLang_ungetkey_string` inserts the characters at the beginning of the input buffer.

See Also

`SLang_getkey`, `SLang_ungetkey`, `SLang_ungetkey_string`
10.7  SLang_ungetkey

Synopsis
Push a character back onto the input buffer

Usage
int SLang_ungetkey (unsigned char ch)

Description
SLang_ungetkey pushes the character ch back onto the SLgetkey input stream. Upon success, it returns zero, otherwise it returns 1.

Example
This function is implemented as:

```c
int SLang_ungetkey (unsigned char ch)
{
    return SLang_ungetkey_string(&ch, 1);
}
```

See Also
SLang_getkey, SLang_ungetkey_string

10.8  SLang_flush_input

Synopsis
Discard all keyboard input waiting to be read

Usage
void SLang_flush_input (void)

Description
SLang_flush_input discards all input characters waiting to be read by the SLang_getkey function.

See Also
SLang_getkey

10.9  SLang_input_pending

Synopsis
Check to see if input is pending

Usage
int SLang_input_pending (int tsecs)
11. Keymap Functions

11.1 SLkm_define_key

Synopsis

Define a key in a keymap

Description

SLkm_define_key may be used to see if an input character is available to be read without causing
SLlang_getkey to block. It will wait up to tsecs tenths of a second if no characters are immediately available for reading. If tsecs is less than zero, then SLlang_input_pending will wait -tsecs milliseconds for input, otherwise tsecs represents 1/10 of a second intervals.

Notes

Not all systems support millisecond resolution.

See Also

SLlang_getkey

10.10 SLang_set_abort_signal

Synopsis

Set the signal to trap SIGINT

Usage

void SLang_set_abort_signal (void (*f)(int));

Description

SLang_set_abort_signal sets the function that gets triggered when the user presses the interrupt key (SIGINT) to the function f. If f is NULL the default handler will get installed.

Example

The default interrupt handler on a Unix system is:

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

Notes

For Unix programmers, the name of this function may appear misleading since it is associated with SIGINT and not SIGABRT. The origin of the name stems from the original intent of the function: to allow the user to abort the running of a S-Lang interpreter function.

See Also

SLang_init_tty, SLsignal_intr

11 Keymap Functions
11. Keymap Functions

Usage

int SLkm_define_key (char *seq, FVOID_STAR f, SLKeyMap_List_Type *km)

Description

SLkm_define_key associates the key sequence seq with the function pointer f in the keymap specified by km. Upon success, it returns zero, otherwise it returns a negative integer upon error.

See Also

SLkm_define_keysym, SLang_define_key

11.2 SLang_define_key

Synopsis

Define a key in a keymap

Usage

int SLang_define_key(char *seq, char *fun, SLKeyMap_List_Type *km)

Description

SLang_define_key associates the key sequence seq with the function whose name is fun in the keymap specified by km.

See Also

SLkm_define_keysym, SLkm_define_key

11.3 SLkm_define_keysym

Synopsis

Define a keysym in a keymap

Usage

int SLkm_define_keysym (seq, ks, km)

    char *seq;
    unsigned int ks;
    SLKeyMap_List_Type *km;

Description

SLkm_define_keysym associates the key sequence seq with the keysym ks in the keymap km. Keysyms whose value is less than or equal to 0x1000 is reserved by the library and should not be used.

See Also

SLkm_define_key, SLang_define_key
11.4 SLang_undefine_key

Synopsis
Undefined a key from a keymap

Usage
void SLang_undefine_key(char *seq, SLKeyMap_List_Type *km);

Description
SLang_undefine_key removes the key sequence seq from the keymap km.

See Also
SLang_define_key

11.5 SLang_create_keymap

Synopsis
Create a new keymap

Usage
SLKeyMap_List_Type *SLang_create_keymap (name, km)

   char *name;
   SLKeyMap_List_Type *km;

Description
SLang_create_keymap creates a new keymap called name by copying the key definitions from the keymap km. If km is NULL, the newly created keymap will be empty and it is up to the calling routine to initialize it via the SLang_define_key and SLkm_define_keysym functions. SLang_create_keymap returns a pointer to the new keymap, or NULL upon failure.

See Also
SLang_define_key, SLkm_define_keysym

11.6 SLang_do_key

Synopsis
Read a keysequence and return its keymap entry

Usage
SLang_Key_Type *SLang_do_key (kml, getkey)

   SLKeyMap_List_Type *kml;
   int (*getkey)(void);
Description

The `SLang_do_key` function reads characters using the function specified by the `getkey` function pointer and uses the key sequence to return the appropriate entry in the keymap specified by `km`.

`SLang_do_key` returns `NULL` if the key sequence is not defined by the keymap, otherwise it returns a pointer to an object of type `SLang_Key_Type`, which is defined in `slang.h` as

```c
#define SLANG_MAX_KEYMAP_KEY_SEQ 14
typedef struct SLang_Key_Type
{
    struct SLang_Key_Type *next;
    union
    {
        char *s;
        FVOID_STAR f;
        unsigned int keysym;
    }
    f;
    unsigned char type; /* type of function */
#define SLKEY_F_INTERPRET 0x01
#define SLKEY_F_INTRINSIC 0x02
#define SLKEY_F_KEYSYM 0x03
    unsigned char str[SLANG_MAX_KEYMAP_KEY_SEQ + 1]; /* key sequence */
} SLang_Key_Type;
```

The `type` field specifies which field of the union `f` should be used. If `type` is `SLKEY_F_INTERPRET`, then `f.s` is a string that should be passed to the interpreter for evaluation. If `type` is `SLKEY_F_INTRINSIC`, then `f.f` refers to function that should be called. Otherwise, `type` is `SLKEY_F_KEYSYM` and `f.keysym` represents the value of the keysym that is associated with the key sequence.

See Also

`SLkm_define_keysym`, `SLkm_define_key`

11.7 `SLang_find_key_function`

Synopsis

Obtain a function pointer associated with a keymap

Usage

```c
FVOID_STAR SLang_find_key_function (fname, km);
    char *fname;
    SLKeyMap_List_Type *km;
```

Description

The `SLang_find_key_function` routine searches through the `SLKeymap_Function_Type` list of functions associated with the keymap `km` for the function with name `fname`. If a matching function is found, a pointer to the function will be returned, otherwise `SLang_find_key_function` will return `NULL`. 
11. Keymap Functions

See Also
SLang_create_keymap, SLang_find_keymap

11.8 SLang_find_keymap

Synopsis
Find a keymap

Usage
SLKeyMap_List_Type *SLang_find_keymap (char *keymap_name);

Description
The SLang_find_keymap function searches through the list of keymaps looking for one whose name is keymap_name. If a matching keymap is found, the function returns a pointer to the keymap. It returns NULL if no such keymap exists.

See Also
SLang_create_keymap, SLang_find_key_function

11.9 SLang_process_keystring

Synopsis
Un-escape a key-sequence

Usage
char *SLang_process_keystring (char *kseq);

Description
The SLang_process_keystring function converts an escaped key sequence to its raw form by converting two-character combinations such as ^A to the single character Ctrl-A (ASCII 1). In addition, if the key sequence contains constructs such as ^XX, where XX represents a two-character termcap specifier, the termcap escape sequence will be looked up and substituted. Upon success, SLang_process_keystring returns a raw key-sequence whose first character represents the total length of the key-sequence, including the length specifier itself. It returns NULL upon failure.

Example
Consider the following examples:

    SLang_process_keystring ("^X^C");
    SLang_process_keystring ("^[A");

The first example will return a pointer to a buffer of three characters whose ASCII values are given by {3, 24, 3}. Similarly, the second example will return a pointer to the four characters {4, 27, 91, 65}. Finally, the result of

    SLang_process_keystring ("^[^ku");
will depend upon the termcap/terminfo capability "ku", which represents the escape sequence associated with the terminal’s UP arrow key. For an ANSI terminal whose UP arrow produces "ESC [ A", the result will be 5,27,27,91,65.

Notes
SLang_process_keystring returns a pointer to a static area that will be overwritten on subsequent calls.

See Also
SLang_define_key, SLang_make_keystring

11.10 SLang_make_keystring

Synopsis
Make a printable key sequence

Usage
char *SLang_make_keystring (unsigned char *ks);

Description
The SLang_make_keystring function takes a raw key sequence ks and converts it to a printable form by converting characters such as ASCII 1 (ctl-A) to ^A. That is, it performs the opposite function of SLang_process_keystring.

Notes
This function returns a pointer to a static area that will be overwritten on the next call to SLang_make_keystring.

See Also
SLang_process_keystring

12 Undocumented Functions

The following functions are not yet documented:

12.1 SLprep_open_prep

Synopsis
??

Usage
int SLprep_open_prep (SLPreprocess_Type *);

Description
??

See Also
??
12.2 SLprep_close_prep

Synopsis

??

Usage

void SLprep_close_prep (SLPreprocess_Type *);

Description

??

See Also

??

12.3 SLprep_line_ok

Synopsis

??

Usage

int SLprep_line_ok (char *, SLPreprocess_Type *);

Description

??

See Also

??

12.4 SLdefine_for_ifdef

Synopsis

??

Usage

int SLdefine_for_ifdef (char *);

Description

??

See Also

??
12.5 `SLang_Read_Line_Type` * `SLang_rline_save_line`
   `(SLang_RLine_Info_Type *)`

Synopsis

Usage

```
SLang_Read_Line_Type * SLang_rline_save_line (SLang_RLine_Info_Type *);
```

Description

See Also

??

12.6 `int SLang_init_readline (SLang_RLine_Info_Type *)`;

Synopsis

Usage

```
int SLang_init_readline (SLang_RLine_Info_Type *);
```

Description

See Also

??

12.7 `int SLang_read_line (SLang_RLine_Info_Type *)`;

Synopsis

Usage

```
int SLang_read_line (SLang_RLine_Info_Type *);
```

Description

See Also

??
12.8 int SLang_rline_insert (char *);

Synopsis

??

Usage

int SLang_rline_insert (char *);

Description

??

See Also

??

12.9 void SLrline_redraw (SLang_RLine_Info_Type *);

Synopsis

??

Usage

void SLrline_redraw (SLang_RLine_Info_Type *);

Description

??

See Also

??

12.10 int SLtt_flush_output (void);

Synopsis

??

Usage

int SLtt_flush_output (void);

Description

??

See Also

??
12.11 void SLtt_set_scroll_region(int, int);

Synopsis

??

Usage

void SLtt_set_scroll_region(int, int);

Description

??

See Also

??

12.12 void SLtt_reset_scroll_region(void);

Synopsis

??

Usage

void SLtt_reset_scroll_region(void);

Description

??

See Also

??

12.13 void SLtt_reverse_video (int);

Synopsis

??

Usage

void SLtt_reverse_video (int);

Description

??

See Also

??
12.14 void SLtt_bold_video (void);

Synopsis
??

Usage
void SLtt_bold_video (void);

Description
??

See Also
??

12.15 void SLtt_begin_insert(void);

Synopsis
??

Usage
void SLtt_begin_insert(void);

Description
??

See Also
??

12.16 void SLtt_end_insert(void);

Synopsis
??

Usage
void SLtt_end_insert(void);

Description
??

See Also
??
12.17 void SLtt_del_eol(void);

Synopsis
??

Usage
void SLtt_del_eol(void);

Description
??

See Also
??

12.18 void SLtt_goto_rc (int, int);

Synopsis
??

Usage
void SLtt_goto_rc (int, int);

Description
??

See Also
??

12.19 void SLtt_delete_nlines(int);

Synopsis
??

Usage
void SLtt_delete_nlines(int);

Description
??

See Also
??
12.20 void SLtt_delete_char(void);

Synopsis
??

Usage
void SLtt_delete_char(void);

Description
??

See Also
??

12.21 void SLtt_erase_line(void);

Synopsis
??

Usage
void SLtt_erase_line(void);

Description
??

See Also
??

12.22 void SLtt_normal_video(void);

Synopsis
??

Usage
void SLtt_normal_video(void);

Description
??

See Also
??
12.23  void SLtt_cls(void);

Synopsis
??
Usage
    void SLtt_cls(void);
Description
??
See Also
??

12.24  void SLtt_beep(void);

Synopsis
??
Usage
    void SLtt_beep(void);
Description
??
See Also
??

12.25  void SLtt_reverse_index(int);

Synopsis
??
Usage
    void SLtt_reverse_index(int);
Description
??
See Also
??
12.26  void SLtt_smart_puts(unsigned short *, unsigned short *, int, int);

Synopsis

??

Usage

void SLtt_smart_puts(unsigned short *, unsigned short *, int, int);

Description

??

See Also

??

12.27  void SLtt_write_string (char *);

Synopsis

??

Usage

void SLtt_write_string (char *);

Description

??

See Also

??

12.28  void SLtt_putchar(char);

Synopsis

??

Usage

void SLtt_putchar(char);

Description

??

See Also

??
12.29  int SLtt_init_video (void);

Synopsis

??

Usage

int SLtt_init_video (void);

Description

??

See Also

??

12.30  int SLtt_reset_video (void);

Synopsis

??

Usage

int SLtt_reset_video (void);

Description

??

See Also

??

12.31  void SLtt_get_terminfo(void);

Synopsis

??

Usage

void SLtt_get_terminfo(void);

Description

??

See Also

??
12.32  void SLtt_get_screen_size (void);

Synopsis

??

Usage

   void SLtt_get_screen_size (void);

Description

??

See Also

??

12.33  int SLtt_set_cursor_visibility (int);

Synopsis

??

Usage

   int SLtt_set_cursor_visibility (int);

Description

??

See Also

??

12.34  int SLtt_initialize (char *);

Synopsis

??

Usage

   int SLtt_initialize (char *);

Description

??

See Also

??
12.35   void SLtt_enable_cursor_keys(void);

Synopsis
??

Usage
   void SLtt_enable_cursor_keys(void);

Description
??

See Also
??

12.36   void SLtt_set_term_vtxxx(int *);

Synopsis
??

Usage
   void SLtt_set_term_vtxxx(int *);

Description
??

See Also
??

12.37   void SLtt_set_color_esc (int, char *);

Synopsis
??

Usage
   void SLtt_set_color_esc (int, char *);

Description
??

See Also
??
12.38  void SLtt_wide_width(void);

Synopsis
  
Usage
  void SLtt_wide_width(void);

Description
  
See Also
  
12.39  void SLtt_narrow_width(void);

Synopsis
  
Usage
  void SLtt_narrow_width(void);

Description
  
See Also
  
12.40  int SLtt_set_mouse_mode (int, int);

Synopsis
  
Usage
  int SLtt_set_mouse_mode (int, int);

Description
  
See Also
  
12.41 void SLtt_set_alt_char_set (int);

Synopsis

??

Usage

void SLtt_set_alt_char_set (int);

Description

??

See Also

??

12.42 int SLtt_write_to_status_line (char *, int);

Synopsis

??

Usage

int SLtt_write_to_status_line (char *, int);

Description

??

See Also

??

12.43 void SLtt_disable_status_line (void);

Synopsis

??

Usage

void SLtt_disable_status_line (void);

Description

??

See Also

??
12.44 char *SLtt_tgetstr (char *);  

Synopsis  
??  

Usage  
char *SLtt_tgetstr (char *);  

Description  
??  

See Also  
??  

12.45 int SLtt_tgetnum (char *);  

Synopsis  
??  

Usage  
int SLtt_tgetnum (char *);  

Description  
??  

See Also  
??  

12.46 int SLtt_tgetflag (char *);  

Synopsis  
??  

Usage  
int SLtt_tgetflag (char *);  

Description  
??  

See Also  
??
12.47  char *SLtt_tigetent (char *);

Synopsis

??

Usage

char *SLtt_tigetent (char *);

Description

??

See Also

??

12.48  char *SLtt_tigetstr (char *, char **);

Synopsis

??

Usage

char *SLtt_tigetstr (char *, char **);

Description

??

See Also

??

12.49  int SLtt_tigetnum (char *, char **);

Synopsis

??

Usage

int SLtt_tigetnum (char *, char **);

Description

??

See Also

??
12.50  SLtt_Char_Type SLtt_get_color_object (int);

Synopsis
??

Usage
SLtt_Char_Type SLtt_get_color_object (int);

Description
??

See Also
??

12.51  void SLtt_set_color_object (int, SLtt_Char_Type);

Synopsis
??

Usage
void SLtt_set_color_object (int, SLtt_Char_Type);

Description
??

See Also
??

12.52  void SLtt_set_color (int, char *, char *, char *);

Synopsis
??

Usage
void SLtt_set_color (int, char *, char *, char *);

Description
??

See Also
??
12.53  void SLtt_set_mono (int, char *, SLtt_Char_Type);

Synopsis
   ??
Usage
   void SLtt_set_mono (int, char *, SLtt_Char_Type);
Description
   ??
See Also
   ??

12.54  void SLtt_add_color_attribute (int, SLtt_Char_Type);

Synopsis
   ??
Usage
   void SLtt_add_color_attribute (int, SLtt_Char_Type);
Description
   ??
See Also
   ??

12.55  void SLtt_set_color_fgbg (int, SLtt_Char_Type, SLtt_Char_Type);

Synopsis
   ??
Usage
   void SLtt_set_color_fgbg (int, SLtt_Char_Type, SLtt_Char_Type);
Description
   ??
See Also
   ??
12.56 int SLkp_define_keysym (char *, unsigned int);

Synopsis
??

Usage
   int SLkp_define_keysym (char *, unsigned int);

Description
??

See Also
??

12.57 int SLkp_init (void);

Synopsis
??

Usage
   int SLkp_init (void);

Description
??

See Also
??

12.58 int SLkp_getkey (void);

Synopsis
??

Usage
   int SLkp_getkey (void);

Description
??

See Also
??
12.59  int SLscroll_find_top (SLscroll_Window_Type *);

Synopsis

??

Usage

int SLscroll_find_top (SLscroll_Window_Type *);

Description

??

See Also

??

12.60  int SLscroll_find_line_num (SLscroll_Window_Type *);

Synopsis

??

Usage

int SLscroll_find_line_num (SLscroll_Window_Type *);

Description

??

See Also

??

12.61  unsigned int SLscroll_next_n (SLscroll_Window_Type *, unsigned int);

Synopsis

??

Usage

unsigned int SLscroll_next_n (SLscroll_Window_Type *, unsigned int);

Description

??

See Also

??
12.62 unsigned int SLscroll_prev_n (SLscroll_Window_Type *, unsigned int);

Synopsis
??

Usage
unsigned int SLscroll_prev_n (SLscroll_Window_Type *, unsigned int);

Description
??

See Also
??

12.63 int SLscroll_pageup (SLscroll_Window_Type *);

Synopsis
??

Usage
int SLscroll_pageup (SLscroll_Window_Type *);

Description
??

See Also
??

12.64 int SLscroll_pagedown (SLscroll_Window_Type *);

Synopsis
??

Usage
int SLscroll_pagedown (SLscroll_Window_Type *);

Description
??

See Also
??
12.65  SLSig_Fun_Type *SLsignal (int, SLSig_Fun_Type *);

Synopsis

??

Usage

SLSig_Fun_Type *SLsignal (int, SLSig_Fun_Type *);

Description

??

See Also

??

12.66  SLSig_Fun_Type *SLsignal_intr (int, SLSig_Fun_Type *);

Synopsis

??

Usage

SLSig_Fun_Type *SLsignal_intr (int, SLSig_Fun_Type *);

Description

??

See Also

??

12.67  int SLsig_block_signals (void);

Synopsis

??

Usage

int SLsig_block_signals (void);

Description

??

See Also

??
12.68 int SLsig_unblock_signals (void);

Synopsis

??

Usage

int SLsig_unblock_signals (void);

Description

??

See Also

??

12.69 int SLsystem (char *);

Synopsis

??

Usage

int SLsystem (char *);

Description

??

See Also

??

12.70 void SLadd_at_handler (long *, char *);

Synopsis

??

Usage

void SLadd_at_handler (long *, char *);

Description

??

See Also

??
12.71 void SLang_define_case(int *, int *);

Synopsis
??

Usage
void SLang_define_case(int *, int *);

Description
??

See Also
??

12.72 void SLang_init_case_tables (void);

Synopsis
??

Usage
void SLang_init_case_tables (void);

Description
??

See Also
??

12.73 unsigned char *SLang_regexp_match(unsigned char *, unsigned int, SLRegexp_Type *);

Synopsis
??

Usage
unsigned char *SLang_regexp_match(unsigned char *, unsigned int, SLRegexp_Type *);

Description
??

See Also
??
12.74 int SLang_regexp_compile (SLRegexp_Type *);

Synopsis

Usage

int SLang_regexp_compile (SLRegexp_Type *);

Description

See Also

12.75 char *SLregexp_quote_string (char *, char *, unsigned int);

Synopsis

Usage

char *SLregexp_quote_string (char *, char *, unsigned int);

Description

See Also

12.76 int SLcmd_execute_string (char *, SLcmd_Cmd_Table_Type *);

Synopsis

Usage

int SLcmd_execute_string (char *, SLcmd_Cmd_Table_Type *);

Description

See Also

??
12.77  **SLcomplex_abs**

Synopsis

Returns the norm of a complex number

Usage

```c
double SLcomplex_abs (double *z)
```

Description

The `SLcomplex_abs` function returns the absolute value or the norm of the complex number given by `z`.

See Also

`SLcomplex_times`

12.78  `double *SLcomplex_times (double *, double *, double *)`;

Synopsis

```
??
```

Usage

```
double *SLcomplex_times (double *, double *, double *);
```

Description

```
??
```

See Also

```
??
```

12.79  `double *SLcomplex_divide (double *, double *, double *)`;

Synopsis

```
??
```

Usage

```
double *SLcomplex_divide (double *, double *, double *);
```

Description

```
??
```

See Also

```
??
```
12.80  double *SLcomplex_sin (double *, double *);

Synopsis

??

Usage

double *SLcomplex_sin (double *, double *);

Description

??

See Also

??

12.81  double *SLcomplex_cos (double *, double *);

Synopsis

??

Usage

double *SLcomplex_cos (double *, double *);

Description

??

See Also

??

12.82  double *SLcomplex_tan (double *, double *);

Synopsis

??

Usage

double *SLcomplex_tan (double *, double *);

Description

??

See Also

??
12.83 double *SLcomplex_asin (double *, double *);  

Synopsis  
??  

Usage  
double *SLcomplex_asin (double *, double *);  

Description  
??  

See Also  
??  

12.84 double *SLcomplex_acos (double *, double *);  

Synopsis  
??  

Usage  
double *SLcomplex_acos (double *, double *);  

Description  
??  

See Also  
??  

12.85 double *SLcomplex_atan (double *, double *);  

Synopsis  
??  

Usage  
double *SLcomplex_atan (double *, double *);  

Description  
??  

See Also  
??
12.86  double *SLcomplex_exp (double *, double *);  

Synopsis

??

Usage

double *SLcomplex_exp (double *, double *);

Description

??

See Also

??

12.87  double *SLcomplex_log (double *, double *);  

Synopsis

??

Usage

double *SLcomplex_log (double *, double *);

Description

??

See Also

??

12.88  double *SLcomplex_log10 (double *, double *);  

Synopsis

??

Usage

double *SLcomplex_log10 (double *, double *);

Description

??

See Also

??
12.89  double *SLcomplex_sqrt (double *, double *);

Synopsis
??

Usage
    double *SLcomplex_sqrt (double *, double *);

Description
??

See Also
??

12.90  double *SLcomplex_sinh (double *, double *);

Synopsis
??

Usage
    double *SLcomplex_sinh (double *, double *);

Description
??

See Also
??

12.91  double *SLcomplex_cosh (double *, double *);

Synopsis
??

Usage
    double *SLcomplex_cosh (double *, double *);

Description
??

See Also
??
12.92  double *SLcomplex_tanh (double *, double *);

Synopsis

??

Usage

double *SLcomplex_tanh (double *, double *);

Description

??

See Also

??

12.93  double *SLcomplex_pow (double *, double *, double *);

Synopsis

??

Usage

double *SLcomplex_pow (double *, double *, double *);

Description

??

See Also

??

12.94  double SLmath_hypot (double x, double y);

Synopsis

??

Usage

double SLmath_hypot (double x, double y);

Description

??

See Also

??

extern double *SLcomplex_asinh (double *, double *);
12.95  double *SLcomplex_acosh (double *, double *);

Synopsis

??

Usage

double *SLcomplex_acosh (double *, double *);

Description

??

See Also

??

12.96  double *SLcomplex_atanh (double *, double *);

Synopsis

??

Usage

double *SLcomplex_atanh (double *, double *);

Description

??

See Also

??

12.97  char *SLdebug_malloc (unsigned long);

Synopsis

??

Usage

char *SLdebug_malloc (unsigned long);

Description

??

See Also

??
12.98  char *SLdebug_calloc (unsigned long, unsigned long);

Synopsis

Usage

char *SLdebug_calloc (unsigned long, unsigned long);

Description

See Also

12.99  char *SLdebug_realloc (char *, unsigned long);

Synopsis

Usage

char *SLdebug_realloc (char *, unsigned long);

Description

See Also

12.100 void SLdebug_free (char *);

Synopsis

Usage

void SLdebug_free (char *);

Description

See Also
12.101 void SLmalloc_dump_statistics (void);

Synopsis

??

Usage

void SLmalloc_dump_statistics (void);

Description

??

See Also

??

12.102 char *SLstrcpy(register char *, register char *);

Synopsis

??

Usage

char *SLstrcpy(register char *, register char *);

Description

??

See Also

??

12.103 int SLstrcmp(register char *, register char *);

Synopsis

??

Usage

int SLstrcmp(register char *, register char *);

Description

??

See Also

??
12.104 char *SLstrncpy(char *, register char *, register int);

Synopsis
??

Usage
  char *SLstrncpy(char *, register char *, register int);

Description
??

See Also
??

12.105 void SLmemset (char *, char, int);

Synopsis
??

Usage
  void SLmemset (char *, char, int);

Description
??

See Also
??

12.106 void SLexpand_escaped_string (register char *, register char *, register char *);

Synopsis
??

Usage
  void SLexpand_escaped_string (register char *, register char *, register char *);

Description
??

See Also
??
12.107  void SLmake_lut (unsigned char *, unsigned char *, unsigned char);

Synopsis

??

Usage

void SLmake_lut (unsigned char *, unsigned char *, unsigned char);

Description

??

See Also

??

12.108  int SLang_guess_type (char *);

Synopsis

??

Usage

int SLang_guess_type (char *);

Description

??

See Also

??