Preface

This document describes the intrinsic functions that are available to any application that embeds the S-Lang interpreter. In addition, slsh defines a number of useful functions that are also available to conforming S-Lang applications. Those functions are described in The SLSH Library Reference.
## Contents

1 Data Types ......................................................... 1
   1.1 Assoc_Type .................................................... 1
   1.2 File_Type ..................................................... 2
   1.3 List_Type ...................................................... 3
   1.4 String_Type ................................................... 3
   1.5 Struct_Type .................................................... 4

2 Array Functions .................................................. 7
   2.1 all .............................................................. 7
   2.2 any .............................................................. 8
   2.3 array_info ...................................................... 8
   2.4 array_map ....................................................... 9
   2.5 array_reverse .................................................. 10
   2.6 array_shape .................................................... 11
   2.7 array_sort ..................................................... 11
   2.8 array_swap ..................................................... 13
   2.9 cumsum ........................................................ 14
   2.10 get_default_sort_method ..................................... 14
   2.11 init_char_array .............................................. 15
   2.12 _isnull ........................................................ 15
   2.13 length ........................................................ 16
   2.14 max ........................................................... 16
   2.15 maxabs ........................................................ 17
   2.16 min ............................................................ 17
   2.17 minabs ........................................................ 18
   2.18 _reshape ....................................................... 18
2.19 reshape .............................................. 19
2.20 set_default_sort_method ............................ 20
2.21 sum .................................................. 20
2.22 sumsq .............................................. 20
2.23 transpose .......................................... 21
2.24 where .............................................. 21
2.25 wherediff .......................................... 22
2.26 wherefirst ......................................... 23
2.27 wherefirstmax ...................................... 24
2.28 wherefirstmin ...................................... 24
2.29 welast .............................................. 24
2.30 werelastmax ........................................ 25
2.31 werelastmin ........................................ 25
2.32 wherenot .......................................... 26

3 Associative Array Functions .......................... 27
  3.1 assoc_delete_key .................................... 27
  3.2 assoc_get_keys ..................................... 27
  3.3 assoc_get_values ................................... 28
  3.4 assoc_key_exists ................................... 28

4 Functions that Operate on Strings .................. 29
  4.1 count_char_occurrences ............................. 29
  4.2 create_delimited_string ......................... 29
  4.3 extract_element ................................... 30
  4.4 glob_to_regexp .................................... 31
  4.5 is_list_element ................................... 32
  4.6 is_substr ......................................... 32
  4.7 make_printable_string ............................. 33
  4.8 printf ............................................ 33
  4.9 strbskipchar ...................................... 33
  4.10 sprintf ........................................... 34
  4.11 sscanf ........................................... 36
  4.12 strbyteslen ...................................... 37
  4.13 strbytesub ....................................... 38
| 4.14 | strcat | 38 |
| 4.15 | strcharlen | 39 |
| 4.16 | strchop | 39 |
| 4.17 | strchopr | 40 |
| 4.18 | strcmp | 40 |
| 4.19 | strcompress | 41 |
| 4.20 | string_match | 41 |
| 4.21 | string_match_nth | 42 |
| 4.22 | string_matches | 43 |
| 4.23 | strjoin | 43 |
| 4.24 | strlen | 44 |
| 4.25 | strlow | 44 |
| 4.26 | strnbytecmp | 45 |
| 4.27 | strncharcmp | 45 |
| 4.28 | strncmp | 46 |
| 4.29 | strreplace | 47 |
| 4.30 | strskipbytes | 47 |
| 4.31 | strskipchar | 48 |
| 4.32 | strsub | 49 |
| 4.33 | strtok | 50 |
| 4.34 | strtrans | 51 |
| 4.35 | strtrim | 52 |
| 4.36 | strtrim_beg | 52 |
| 4.37 | strtrim_end | 53 |
| 4.38 | strup | 53 |
| 4.39 | str_delete_chars | 54 |
| 4.40 | str_quote_string | 55 |
| 4.41 | str_replace | 55 |
| 4.42 | str_uncomment_string | 56 |
| 4.43 | substr | 57 |
| 4.44 | substrbytes | 57 |

5 Functions that Operate on Binary Strings

5.1 array_to_bstring | 59
<table>
<thead>
<tr>
<th>Section</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>bstring_to_array</td>
<td>59</td>
</tr>
<tr>
<td>5.3</td>
<td>bstrcat</td>
<td>60</td>
</tr>
<tr>
<td>5.4</td>
<td>bstrjoin</td>
<td>60</td>
</tr>
<tr>
<td>5.5</td>
<td>bstrlen</td>
<td>60</td>
</tr>
<tr>
<td>5.6</td>
<td>count_byte_occurrences</td>
<td>61</td>
</tr>
<tr>
<td>5.7</td>
<td>is_substrbytes</td>
<td>61</td>
</tr>
<tr>
<td>5.8</td>
<td>pack</td>
<td>62</td>
</tr>
<tr>
<td>5.9</td>
<td>pad_pack_format</td>
<td>63</td>
</tr>
<tr>
<td>5.10</td>
<td>sizeof_pack</td>
<td>64</td>
</tr>
<tr>
<td>5.11</td>
<td>unpack</td>
<td>64</td>
</tr>
<tr>
<td>6</td>
<td>Functions that Manipulate Structures</td>
<td>65</td>
</tr>
<tr>
<td>6.1</td>
<td>__add_binary</td>
<td>65</td>
</tr>
<tr>
<td>6.2</td>
<td>__add_string</td>
<td>66</td>
</tr>
<tr>
<td>6.3</td>
<td>__add_typecast</td>
<td>67</td>
</tr>
<tr>
<td>6.4</td>
<td>__add_unary</td>
<td>67</td>
</tr>
<tr>
<td>6.5</td>
<td>get_struct_field</td>
<td>68</td>
</tr>
<tr>
<td>6.6</td>
<td>get_struct_field_names</td>
<td>68</td>
</tr>
<tr>
<td>6.7</td>
<td>__is_struct_type</td>
<td>69</td>
</tr>
<tr>
<td>6.8</td>
<td>is_struct_type</td>
<td>69</td>
</tr>
<tr>
<td>6.9</td>
<td>__push_struct_field_values</td>
<td>70</td>
</tr>
<tr>
<td>6.10</td>
<td>set_struct_field</td>
<td>70</td>
</tr>
<tr>
<td>6.11</td>
<td>set_struct_fields</td>
<td>71</td>
</tr>
<tr>
<td>7</td>
<td>Functions that Create and Manipulate Lists</td>
<td>73</td>
</tr>
<tr>
<td>7.1</td>
<td>list_append</td>
<td>73</td>
</tr>
<tr>
<td>7.2</td>
<td>list_concat</td>
<td>73</td>
</tr>
<tr>
<td>7.3</td>
<td>list_delete</td>
<td>74</td>
</tr>
<tr>
<td>7.4</td>
<td>list_insert</td>
<td>74</td>
</tr>
<tr>
<td>7.5</td>
<td>list_join</td>
<td>75</td>
</tr>
<tr>
<td>7.6</td>
<td>list_new</td>
<td>75</td>
</tr>
<tr>
<td>7.7</td>
<td>list_pop</td>
<td>75</td>
</tr>
<tr>
<td>7.8</td>
<td>list_reverse</td>
<td>76</td>
</tr>
<tr>
<td>7.9</td>
<td>list_to_array</td>
<td>76</td>
</tr>
</tbody>
</table>
## 8 Informational Functions

8.1 `add_doc_file` ......................................................... 77
8.2 `_apropos` ................................................................. 77
8.3 `__FILE__` ................................................................. 78
8.4 `_function_name` ......................................................... 79
8.5 `__get_defined_symbols` ............................................ 79
8.6 `get_doc_files` ......................................................... 79
8.7 `get_doc_string_from_file` ....................................... 80
8.8 `__get_namespaces` ..................................................... 80
8.9 `is_defined` .............................................................. 80
8.10 `__is_initialized` .................................................... 81
8.11 `_NARGS` ................................................................. 82
8.12 `set_doc_files` ......................................................... 82
8.13 `_slang_doc_dir` ...................................................... 83
8.14 `_slang_version` ....................................................... 83
8.15 `_slang_version_string` ............................................. 83

## 9 Mathematical Functions

9.1 `abs` ........................................................................ 85
9.2 `acos` .................................................................. 85
9.3 `acosh` .................................................................. 86
9.4 `asin` .................................................................... 86
9.5 `asinh` .................................................................. 86
9.6 `atan` .................................................................... 87
9.7 `atan2` .................................................................. 87
9.8 `atanh` .................................................................. 87
9.9 `ceil` ...................................................................... 88
9.10 `Conj` ................................................................... 88
9.11 `cos` ..................................................................... 88
9.12 `cosh` ................................................................... 89
9.13 `__diff` .................................................................. 89
9.14 `exp` ..................................................................... 89
9.15 `expm1` ................................................................ 90
9.16 `feqs` ...................................................................... 90
CONTENTS

9.17 fgteqs ................................................. 91
9.18 floor ................................................. 91
9.19 fteqs ................................................. 91
9.20 fneqs ................................................. 92
9.21 get_float_format ................................. 92
9.22 hypot ................................................. 92
9.23 Imag ................................................. 93
9.24 isinf ................................................. 93
9.25 isnan ................................................. 94
9.26 _isneg ................................................. 94
9.27 _isnongeq ........................................... 95
9.28 _ispos ................................................. 95
9.29 log .................................................. 95
9.30 log10 ................................................ 96
9.31 log1p ................................................ 96
9.32 _max ................................................ 96
9.33 _min ................................................ 97
9.34 mul2 ................................................ 97
9.35 nint ................................................ 98
9.36 polynom ............................................. 98
9.37 Real ................................................ 99
9.38 round ............................................... 99
9.39 set_float_format ................................. 99
9.40 sign ................................................ 100
9.41 sin ................................................ 100
9.42 sincos .............................................. 101
9.43 sinh ................................................ 101
9.44 sqr ................................................ 101
9.45 sqrt ................................................ 102
9.46 tan ................................................ 102
9.47 tanh ................................................ 102

10 Message and Error Functions ............................. 105
10.1 errno ................................................. 105
## CONTENTS

10.2 **errno_string** .................................................. 106  
10.3 **error** ............................................................ 107  
10.4 **__get_exception_info** ........................................ 107  
10.5 **message** ....................................................... 108  
10.6 **new_exception** ............................................... 109  
10.7 **usage** .......................................................... 109  
10.8 **verror** .......................................................... 110  
10.9 **vmessage** ...................................................... 111

### 11 Time and Date Functions  
11.1 **ctime** .......................................................... 113  
11.2 **gmtime** .......................................................... 113  
11.3 **localtime** .......................................................... 114  
11.4 **mktime** ........................................................... 114  
11.5 **strftime** .......................................................... 115  
11.6 **_tic** ............................................................... 116  
11.7 **tic** ............................................................... 116  
11.8 **_time** ............................................................. 117  
11.9 **time** ............................................................. 117  
11.10 **timegm** ........................................................... 117  
11.11 **times** ............................................................ 118  
11.12 **_toc** ............................................................. 118  
11.13 **toc** ............................................................... 119

### 12 Data-Type Conversion Functions  
12.1 **atof** .............................................................. 121  
12.2 **atoi** .............................................................. 121  
12.3 **atol** .............................................................. 122  
12.4 **atoll** .............................................................. 122  
12.5 **char** .............................................................. 123  
12.6 **define_case** .................................................... 123  
12.7 **double** ........................................................... 124  
12.8 **int** ............................................................... 124  
12.9 **integer** .......................................................... 125
<table>
<thead>
<tr>
<th>12.10</th>
<th>isalnum, isalpha, isascii, isblank, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit</th>
<th>125</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.11</td>
<td>_slang_guess_type</td>
<td>126</td>
</tr>
<tr>
<td>12.12</td>
<td>string</td>
<td>126</td>
</tr>
<tr>
<td>12.13</td>
<td>tolower</td>
<td>127</td>
</tr>
<tr>
<td>12.14</td>
<td>toupper</td>
<td>127</td>
</tr>
<tr>
<td>12.15</td>
<td>typecast</td>
<td>128</td>
</tr>
<tr>
<td>12.16</td>
<td>_typeof</td>
<td>128</td>
</tr>
<tr>
<td>12.17</td>
<td>typeof</td>
<td>129</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>13</th>
<th>Stdio File I/O Functions</th>
<th>131</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.1</td>
<td>clearerr</td>
<td>131</td>
</tr>
<tr>
<td>13.2</td>
<td>fclose</td>
<td>131</td>
</tr>
<tr>
<td>13.3</td>
<td>fdopen</td>
<td>132</td>
</tr>
<tr>
<td>13.4</td>
<td>feof</td>
<td>133</td>
</tr>
<tr>
<td>13.5</td>
<td>ferror</td>
<td>133</td>
</tr>
<tr>
<td>13.6</td>
<td>fflush</td>
<td>133</td>
</tr>
<tr>
<td>13.7</td>
<td>fgets</td>
<td>134</td>
</tr>
<tr>
<td>13.8</td>
<td>fgetslines</td>
<td>135</td>
</tr>
<tr>
<td>13.9</td>
<td>fopen</td>
<td>136</td>
</tr>
<tr>
<td>13.10</td>
<td>fprintf</td>
<td>137</td>
</tr>
<tr>
<td>13.11</td>
<td>fputs</td>
<td>137</td>
</tr>
<tr>
<td>13.12</td>
<td>fputslines</td>
<td>138</td>
</tr>
<tr>
<td>13.13</td>
<td>fread</td>
<td>138</td>
</tr>
<tr>
<td>13.14</td>
<td>fread_bytes</td>
<td>139</td>
</tr>
<tr>
<td>13.15</td>
<td>fseek</td>
<td>139</td>
</tr>
<tr>
<td>13.16</td>
<td>ftell</td>
<td>140</td>
</tr>
<tr>
<td>13.17</td>
<td>fwrite</td>
<td>140</td>
</tr>
<tr>
<td>13.18</td>
<td>pclose</td>
<td>141</td>
</tr>
<tr>
<td>13.19</td>
<td>popen</td>
<td>141</td>
</tr>
<tr>
<td>13.20</td>
<td>printf</td>
<td>142</td>
</tr>
<tr>
<td>13.21</td>
<td>setvbuf</td>
<td>143</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>14</th>
<th>Low-level POSIX I/O functions</th>
<th>145</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.1</td>
<td>close</td>
<td>145</td>
</tr>
</tbody>
</table>
14.2 _close ................................................................. 145
14.3 dup_fd ................................................................. 146
14.4 dup2_fd ................................................................. 146
14.5 _fileno ................................................................. 146
14.6 fileno ................................................................. 147
14.7 isnatty ................................................................. 147
14.8 lseek ................................................................. 147
14.9 open ................................................................. 148
14.10 read ................................................................. 149
14.11 write ................................................................. 149

15 Signal Functions ..................................................... 151
15.1 alarm ................................................................. 151
15.2 getitimer .............................................................. 152
15.3 setitimer .............................................................. 153
15.4 signal ................................................................. 153
15.5 sigprocmask .......................................................... 154
15.6 sigsuspend ........................................................... 155

16 Directory Functions .................................................. 157
16.1 access ................................................................. 157
16.2 chdir ................................................................. 157
16.3 chmod ................................................................. 158
16.4 chown ................................................................. 158
16.5 getcwd ................................................................. 159
16.6 hardlink .............................................................. 159
16.7 lchown ................................................................. 159
16.8 listdir ................................................................. 160
16.9 lstat_file ............................................................. 160
16.10 mkdir ................................................................. 161
16.11 readlink ............................................................. 161
16.12 remove ............................................................... 162
16.13 rename ............................................................... 162
16.14 rmdir ................................................................. 162
16.15 stat_file ............................................................. 163
<table>
<thead>
<tr>
<th>Section</th>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.16</td>
<td>stat_is</td>
<td>164</td>
</tr>
<tr>
<td>16.17</td>
<td>stat_mode_to_string</td>
<td>164</td>
</tr>
<tr>
<td>16.18</td>
<td>symlink</td>
<td>165</td>
</tr>
<tr>
<td>16.19</td>
<td>utime</td>
<td>165</td>
</tr>
<tr>
<td>17.1</td>
<td>path_basename</td>
<td>167</td>
</tr>
<tr>
<td>17.2</td>
<td>path_basename_sans_extname</td>
<td>167</td>
</tr>
<tr>
<td>17.3</td>
<td>path_concat</td>
<td>168</td>
</tr>
<tr>
<td>17.4</td>
<td>path_dirname</td>
<td>168</td>
</tr>
<tr>
<td>17.5</td>
<td>path_extname</td>
<td>168</td>
</tr>
<tr>
<td>17.6</td>
<td>path_get_delimiter</td>
<td>169</td>
</tr>
<tr>
<td>17.7</td>
<td>path_is_absolute</td>
<td>169</td>
</tr>
<tr>
<td>17.8</td>
<td>path_sans_extname</td>
<td>169</td>
</tr>
<tr>
<td>18.1</td>
<td>getegid</td>
<td>171</td>
</tr>
<tr>
<td>18.2</td>
<td>geteuid</td>
<td>171</td>
</tr>
<tr>
<td>18.3</td>
<td>getgid</td>
<td>172</td>
</tr>
<tr>
<td>18.4</td>
<td>getpgid</td>
<td>172</td>
</tr>
<tr>
<td>18.5</td>
<td>getpgrp</td>
<td>172</td>
</tr>
<tr>
<td>18.6</td>
<td>getpid</td>
<td>173</td>
</tr>
<tr>
<td>18.7</td>
<td>getppid</td>
<td>173</td>
</tr>
<tr>
<td>18.8</td>
<td>getpriority</td>
<td>173</td>
</tr>
<tr>
<td>18.9</td>
<td>getrusage</td>
<td>174</td>
</tr>
<tr>
<td>18.10</td>
<td>getsid</td>
<td>175</td>
</tr>
<tr>
<td>18.11</td>
<td>getuid</td>
<td>175</td>
</tr>
<tr>
<td>18.12</td>
<td>kill</td>
<td>176</td>
</tr>
<tr>
<td>18.13</td>
<td>killpg</td>
<td>176</td>
</tr>
<tr>
<td>18.14</td>
<td>mkfifo</td>
<td>177</td>
</tr>
<tr>
<td>18.15</td>
<td>setgid</td>
<td>177</td>
</tr>
<tr>
<td>18.16</td>
<td>setpgid</td>
<td>178</td>
</tr>
<tr>
<td>18.17</td>
<td>setpriority</td>
<td>178</td>
</tr>
<tr>
<td>18.18</td>
<td>setsid</td>
<td>179</td>
</tr>
<tr>
<td>18.19</td>
<td>setuid</td>
<td>179</td>
</tr>
</tbody>
</table>
CONTENTS

18.20 sleep .................................................. 180
18.21 system .................................................. 180
18.22 system_intr ........................................... 180
18.23 umask ................................................... 181
18.24 uname .................................................. 181

19 Eval Functions .............................................. 183
  19.1 $ .......................................................... 183
  19.2 autoload ................................................ 183
  19.3 byte_compile_file ..................................... 184
  19.4 eval ..................................................... 184
  19.5 evalfile ............................................... 185
  19.6 get_slang_load_path .................................. 186
  19.7 set_slang_load_path .................................. 186

20 Qualifier Functions ....................................... 189
  20.1 qualifier ............................................. 189
  20.2 __qualifiers .......................................... 190
  20.3 qualifier_exists ...................................... 191

21 Module Functions ......................................... 193
  21.1 get_import_module_path ................................ 193
  21.2 import ................................................ 193
  21.3 set_import_module_path ............................... 194

22 Debugging Functions ...................................... 195
  22.1 _bfeof_info .......................................... 195
  22.2 _boseos_info ......................................... 195
  22.3 _clear_error .......................................... 196
  22.4 _get_frame_info ...................................... 197
  22.5 _get_frame_variable ................................ 198
  22.6 _set_bof_handler ..................................... 198
  22.7 _set_bos_handler .................................... 199
  22.8 _set_eof_handler ................................... 199
  22.9 _set_eos_handler ................................... 200
  22.10 _slangtrace ......................................... 200
22.11_traceback .................................................. 201
22.12_trace_function ........................................... 201
22.13_use_frame_namespace ................................. 201

23 Stack Functions ............................................. 203
  23.1 dup ..................................................... 203
  23.2 exch ................................................... 203
  23.3 pop ...................................................... 204
  23.4 __pop_args ............................................. 204
  23.5 __pop_list ............................................. 205
  23.6 __pop_n ............................................... 206
  23.7 _print_stack ........................................... 206
  23.8 __push_args .......................................... 206
  23.9 __push_list ........................................... 207
  23.10_stkdepth .............................................. 207
  23.11_stk_reverse ......................................... 207
  23.12_stk_roll ............................................. 208

24 Functions that deal with the S-Lang readline interface .... 209
  24.1 rline_bolp ............................................. 209
  24.2 rline_call ............................................. 209
  24.3 rline_del .............................................. 210
  24.4 rline_eolp ............................................. 210
  24.5 rline_getkey .......................................... 211
  24.6 rline_get_edit_width ................................ 211
  24.7 rline_get_history ................................... 211
  24.8 rline_get_line ....................................... 212
  24.9 rline_get_point ...................................... 212
  24.10 rline_input_pending ................................ 213
  24.11 rline_ins ........................................... 213
  24.12 rline_setkey ........................................ 213
  24.13 rline_set_completion_callback ..................... 214
  24.14 rline_set_history .................................. 215
  24.15 rline_set_line ...................................... 215
  24.16 rline_set_list_completions_callback ................ 216
24.17 rline_set_point ........................................ 216
24.18 rline_unsetkey ........................................ 217

25 Miscellaneous Functions .................................. 219
  25.1 _autoDeclare ........................................ 219
  25.2 __class_id .......................................... 220
  25.3 __class_type .................................... 220
  25.4 current_namespace ................................ 220
  25.5 __datatype ........................................ 221
  25.6 _eqs ............................................... 221
  25.7 get_environ ........................................ 222
  25.8 getenv ............................................. 222
  25.9 __get_reference .................................. 223
  25.10 implements ....................................... 223
  25.11 __is_callable .................................... 224
  25.12 __is_datatype_numeric............................ 225
  25.13 __is_numeric .................................... 225
  25.14 __is_same ......................................... 225
  25.15 putenv ............................................ 226
  25.16 __set_argc_argv ................................... 226
  25.17 slang_install_prefix .............................. 227
  25.18 slang_utf8_ok ..................................... 227
  25.19 __tmp .............................................. 228
  25.20 __uninitialize ................................... 228
  25.21 use_namespace .................................... 229
Chapter 1

Data Types

1.1 Assoc_Type

Synopsis

An associative array or hash type

Description

An Assoc_Type object is like an array except that it is indexed using strings and not integers. Unlike an Array_Type object, the size of an associative array is not fixed, but grows as objects are added to the array. Another difference is that ordinary arrays represent ordered objects; however, the ordering of the elements of an Assoc_Type object is unspecified.

An Assoc_Type object whose elements are of some data-type d may be created using using

\[ A = \text{Assoc}_\text{Type}[d]; \]

For example,

\[ A = \text{Assoc}_\text{Type}[\text{Int}\_\text{Type}]; \]

will create an associative array of integers. To create an associative array capable of storing an arbitrary type, use the form

\[ A = \text{Assoc}_\text{Type}[]; \]

An optional parameter may be used to specify a default value for array elements. For example,

\[ A = \text{Assoc}_\text{Type}[\text{Int}\_\text{Type}, -1]; \]

creates an integer-valued associative array with a default element value of -1. Then \( A["\text{foo}"] \) will return -1 if the key "foo" does not exist in the array. Default values are available only if the type was specified when the associative array was created.

The following functions may be used with associative arrays:

\[
\begin{align*}
\text{assoc_get_keys} \\
\text{assoc_get_values} \\
\text{assoc_key_exists} \\
\text{assoc_delete_key}
\end{align*}
\]
Chapter 1. Data Types

The length function may be used to obtain the number of elements in the array.

The foreach construct may be used with associative arrays via one of the following forms:

```plaintext
foreach k,v (A) {...}
foreach k (A) using ("keys") { ... }
foreach v (A) using ("values") { ... }
foreach k,v (A) using ("keys", "values") { ... }
```

In all the above forms, the loop is over all elements of the array such that \( v=A[k] \).

See Also

1.3 (List_Type), ?? (Array_Type), 1.5 (Struct_Type)

1.2 File_Type

Synopsis

A type representing a C stdio object

Description

An File_Type is the interpreter's representation of a C stdio FILE object and is usually created using the fopen function, i.e.,

```plaintext
fp = fopen ("file.dat", "r");
```

Functions that utilize the File_Type include:

```plaintext
fopen
fclose
fgets
fputs
ferror
feof
fflush
fprintf
fseek
ftell
fread
fwrite
fread_bytes
```

The foreach construct may be used with File_Type objects via one of the following forms:

```plaintext
foreach line (fp) {...}
foreach byte (A) using ("char") { ... }  % read bytes
foreach line (A) using ("line") { ... }  % read lines (default)
foreach line (A) using ("wsline") { ... }  % whitespace stripped from lines
```

See Also

1.3 (List_Type), ?? (Array_Type), 1.5 (Struct_Type)
1.3 List_Type

Synopsis
A list object

Description
An object of type List_Type represents a list, which is defined as an ordered heterogeneous collection of objects. A list may be created using, e.g.,

```plaintext
empty_list = {};
list_with_4_items = {[1:10], "three", 9, {1,2,3}};
```

Note that the last item of the list in the last example is also a list. A List_Type object may be manipulated by the following functions:

- list_new
- list_insert
- list_append
- list_delete
- list_reverse
- list_pop

A List_Type object may be indexed using an array syntax with the first item on the list given by an index of 0. The length function may be used to obtain the number of elements in the list.

A copy of the list may be created using the @ operator, e.g., `copy = @list`.

The `foreach` statement may be used with a List_Type object to loop over its elements:

```plaintext
foreach elem (list) {}
```

See Also
?? (Array_Type), 1.1 (Assoc_Type), 1.5 (Struct_Type)

1.4 String_Type

Synopsis
A string object

Description
An object of type String_Type represents a string of bytes or characters, which in general have different semantics depending upon the UTF-8 mode.

The string obeys byte-semantics when indexed as an array. That is, `S[0]` will return the first byte of the string `S`. For character semantics, the nth character in the string may be obtained using substr function.

The `foreach` statement may be used with a String_Type object `S` to loop over its bytes:

```plaintext
foreach b (S) {}
foreach b (S) using ("bytes") {}
```
To loop over its characters, the following form may be used:

```c
foreach c (s) using ("chars") {...}
```

When UTF-8 mode is not in effect, the byte and character forms will produce the same sequence. Otherwise, the string will be decoded to generate the (wide) character sequence. If the string contains an invalid UTF-8 encoded character, successive bytes of the invalid sequence will be returned as negative integers. For example, "a\xAB\x{AB}" specifies a string composed of the character a, a byte 0xAB, and the character 0xAB. In this case,

```c
foreach c ("a\xAB\x{AB}") {...}
```

will produce the integer-valued sequence 'a', -0xAB, 0xAB.

See Also

?? (Array_Type), 25.18 (_slang_utf8_ok)

### 1.5 Struct_Type

**Synopsis**

A structure datatype

**Description**

A Struct_Type object with fields f1, f2, ..., fN may be created using

```c
s = struct { f1, f2, ..., fN };
```

The fields may be accessed via the "dot" operator, e.g.,

```c
s.f1 = 3;
if (s12.f1 == 4) s.f1++;
```

By default, all fields will be initialized to NULL.

A structure may also be created using the dereference operator (@):

```c
s = @Struct_Type ("f1", "f2", ..., "fN");
s = @Struct_Type (["f1", "f2", ..., "fN"]);
```

Functions for manipulating structure fields include:

- `push_struct_field_values`
- `get_struct_field`
- `get_struct_field_names`
- `set_struct_field`
- `set_struct_fields`

The `foreach` loop may be used to loop over elements of a linked list. Suppose that first structure in the list is called root, and that the child field is used to form the chain. Then one may walk the list using:
foreach s (root) using ("child")
{
  % s will take on successive values in the list
  .
  .
}

The loop will terminate when the last elements child field is NULL. If no “linking” field is specified, the field name will default to next.

User-defined data types are similar to the Struct_Type. A type, e.g., Vector_Type may be created using:

typedef struct { x, y, z } Vector_Type;

Objects of this type may be created via the @ operator, e.g.,

v = @Vector_Type;

It is recommended that this be used in a function for creating such types, e.g.,

define vector (x, y, z)
{
  variable v = @Vector_Type;
  v.x = x;
  v.y = y;
  v.z = z;
  return v;
}

The action of the binary and unary operators may be defined for such types. Consider the "+" operator. First define a function for adding two Vector_Type objects:

static define vector_add (v1, v2)
{
  return vector (v1.x+v2.x, v1.y+v2.y, v1.z, v2.z);
}

Then use

__add_binary ("+", Vector_Type, &vector_add, Vector_Type, Vector_Type);

to indicate that the function is to be called whenever the "+" binary operation between two Vector_Type objects takes place, e.g.,

V1 = vector (1, 2, 3);
V2 = vector (8, 9, 1);
V3 = V1 + V2;

will assigned the vector (9, 11, 4) to V3. Similarly, the "*" operator between scalars and vectors may be defined using:

static define vector_scalar_mul (v, a)
{
  return vector (a*v.x, a*v.y, a*v.z);
}
static define scalar_vector_mul (a, v) 
{
  return vector_scalar_mul (v, a);
}
__add_binary("*", Vector_Type, &scalar_vector_mul, Any_Type, Vector_Type);
__add_binary("*", Vector_Type, &vector_scalar_mul, Vector_Type, Any_Type);

Related functions include:

__add_unary
__add_string
__add_destroy

See Also

1.3 (List_Type), 1.1 (Assoc_Type)
Chapter 2

Array Functions

2.1 all

Synopsis
Tests if all elements of an array are non-zero

Usage
Char_Type all (Array_Type a [,Int_Type dim])

Description
The all function examines the elements of a numeric array and returns 1 if all elements are non-zero, otherwise it returns 0. If a second argument is given, then it specifies the dimension of the array over which the function is to be applied. In this case, the result will be an array with the same shape as the input array minus the specified dimension.

Example
Consider the 2-d array

\[
\begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
6 & 7 & 8 & 9 & 10
\end{array}
\]

generated by

\[
a = \_reshape ([1:10], [2, 5]);
\]

Then all(a) will return 1, and all(a>3, 0) will return a 1-d array

\[
[0, 0, 0, 1, 1]
\]

Similarly, all(a>3, 1) will return the 1-d array

\[
[0, 1]
\]

See Also
2.24 (where), 2.2 (any), 2.25 (wherediff)
2.2 any

Synopsis
Test if any element of an array is non-zero

Usage
Char_Type any (Array_Type a [,Int_Type dim])

Description
The any function examines the elements of a numeric array and returns 1 if any element is both non-zero and not a NaN, otherwise it returns 0. If a second argument is given, then it specifies the dimension of the array to be tested.

Example
Consider the 2-d array

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
6 & 7 & 8 & 9 \\
\end{array}
\]
generated by

\[
a = \text{reshape} ([1:10], [2, 5]);
\]

Then any(a==3) will return 1, and any(a==3, 0) will return a 1-d array with elements:

\[
0 0 1 0 0
\]

See Also
2.1 (all), 2.24 (where), 2.25 (wherediff)

2.3 array_info

Synopsis
Returns information about an array

Usage
(Array_Type, Integer_Type, DataType_Type) array_info (Array_Type a)

Description
The array_info function returns information about the array a. It returns three values: an 1-d integer array specifying the size of each dimension of a, the number of dimensions of a, and the data type of a.

Example
The array_info function may be used to find the number of rows of an array:


```c

2.4. array_map

define num_rows (a)
{
    variable dims, num_dims, data_type;

    (dims, num_dims, data_type) = array_info (a);
    return dims [0];
}

See Also

12.17 (typeof), 2.6 (array_shape), 2.13 (length), 2.19 (reshape), 2.18 (_reshape)

2.4 array_map

Synopsis

Apply a function to each element of an array

Usage

Array_Type array_map (type, func, args...)

Usage

(Array_Type, ...) array_map (type, ..., func, args...)

DataType_Type type, ...;
Ref_Type func;

Description

The array_map function may be used to apply a function to each element of an array and returns the resulting values as an array of the specified type. The type parameter indicates what kind of array should be returned and generally corresponds to the return type of the function. If the function returns multiple values, then the type of each return value must be given. The first array-valued argument is used to determine the dimensions of the resulting array(s). If any subsequent arguments correspond to an array of the same size, then those array elements will be passed in parallel with the elements of the first array argument.

To use array_map with functions that return no value, either omit the type argument, or explicitly indicate that it returns no value using the Void_Type type.

Example

The first example illustrates how to apply the strlen function to an array of strings.

S = ["", "Train", "Subway", "Car"];
L = array_map (Integer_Type, &strlen, S);

This is equivalent to:

S = ["", "Train", "Subway", "Car"];
L = Integer_Type [length (S)];
for (i = 0; i < length (S); i++) L[i] = strlen (S[i]);

Now consider an example involving the strcat function:

```
files = ["slang", "slstring", "slarray"]; exts = ".c";
cfiles = array_map (String_Type, &strcat, files, exts);
% ==> cfiles = ["slang.c", "slstring.c", "slarray.c"];

exts = [".a",".b",".c"];
xfiles = array_map (String_Type, &strcat, files, exts);
% ==> xfiles = ["slang.a", "slstring.b", "slarray.c"];

Here is an example of its application to a function that returns 3 values. Suppose A is an array of arrays whose types and sizes are arbitrary, and we wish to find the indices of A that contain arrays of type String_Type. For this purpose, the array_info function will be used:

(dims, ndims, types) = array_map (Array_Type, Int_Type, DataType_Type, &array_info, A);
i = where (types == String_Type);

The message function prints a string and returns no value. This example shows how it may be used to print an array of strings:

a = ["Line 1", "Line 2", "Line 3"];
array_map (&message, a); % Form 1
array_map (Void_Type, &message, a); % Form 2

Notes

Many mathematical functions already work transparently on arrays. For example, the following two statements produce identical results:

B = sin (A);
B = array_map (Double_Type, &sin, A);

Notes

A number of the string functions have been vectorized, including the strlen function. This means that there is no need to use the array_map function with the strlen function.

See Also

2.3 (array_info), 4.24 (strlen), 4.14 (strcat), 9.41 (sin)

2.5 array_reverse

Synopsis

Reverse the elements of an array

Usage

array_reverse (Array_Type a [,Int_Type i0, Int_Type i1] [,Int_Type dim])

Description
2.6. 

In its simplest form, the `array_reverse` function reverses the elements of an array. If passed 2 or 4 arguments, `array_reverse` reverses the elements of the specified dimension of a multi-dimensional array. If passed 3 or 4 arguments, the parameters `i0` and `i1` specify a range of elements to reverse.

Example

If `a` is a one dimensional array, then

```c
array_reverse (a, i, j);
a[[i:j]] = a[[j:i:-1]];
```

are equivalent to one another. However, the form using `array_reverse` is about 10 times faster than the version that uses explicit array indexing.

See Also

2.8 (array_swap), 2.23 (transpose)

2.6 array_shape

Synopsis

Get the shape or dimensions of an array

Usage

```c
dims = array_shape (Array_Type a)
```

Description

This function returns an array representing the dimensionality or shape of a specified array. The `array_info` function also returns this information but for many purposes the `array_shape` function is more convenient.

See Also

2.3 (array_info), 2.19 (reshape)

2.7 array_sort

Synopsis

Sort an array or opaque object

Usage

```c
Array_Type array_sort (obj [, &func [, n]])
```

Description

The `array_sort` function may be used to sort an object and returns an integer index array that represents the result of the sort as a permutation.

If a single parameter is passed, that parameter must be an array, which will be sorted into ascending order using a built-in type-specific comparison function.
If two parameters are passed (\texttt{obj} and \texttt{func}), then the first parameter must be the array to be sorted, and the second is a reference to the comparison function. In this case, the comparison function represented by \texttt{func} must take two arguments representing two array elements to be compared, and must return an integer that represents the result of the comparison. The return value must be less than zero if the first parameter is less than the second, zero if they are equal, and a value greater than zero if the first is greater than the second.

If three parameters are passed, then the first argument will be regarded as an opaque object by the sorting algorithm. For this reason, the number of elements represented by the object must also be passed to \texttt{array_sort} function as the third function argument. The second function argument must be a reference to comparison function. In this case, the comparison function will be passed three values: the opaque object, the (0-based) index of the first element to be compared, and the (0-based) index of the second element. The return value must be less than zero if the value of the element at the first index considered to be less than the value of the element at the second index, zero if the values are equal, and a value greater than zero if the first value is greater than the second.

\texttt{array_sort} sorts the array \texttt{a} into ascending order and returns an integer array that represents the result of the sort. If the optional second parameter \texttt{f} is present, the function specified by \texttt{f} will be used to compare elements of \texttt{a}; otherwise, a built-in sorting function will be used.

The integer array returned by this function is simply an index array that indicates the order of the sorted object. The input object \texttt{obj} is not changed.

\textbf{Qualifiers}

By default, elements are sorted in ascending order. The \texttt{dir} qualifier may be used to specify the sort direction. Specifically if \texttt{dir} \geq 0, the sort will be an ascending one, otherwise it will be descending.

The \texttt{method} qualifier may be used to select between the available sorting algorithms. There are currently two algorithms supported: merge-sort and quick-sort. Using \texttt{method="msort"} will cause the merge-sort algorithm to be used. The quick-sort algorithm may be selected using \texttt{method="qsort"}.

\textbf{Example}

An array of strings may be sorted using the \texttt{strcmp} function since it fits the specification for the sorting function described above:

\begin{verbatim}
A = ["gamma", "alpha", "beta"];
I = array_sort (A, &strcmp);
\end{verbatim}

Alternatively, one may use

\begin{verbatim}
variable I = array_sort (A);
\end{verbatim}

to use the built-in comparison function.

After the \texttt{array_sort} has executed, the variable \texttt{I} will have the values [2, 0, 1]. This array can be used to re-shuffle the elements of \texttt{A} into the sorted order via the array index expression \texttt{A = A[I]}. This operation may also be written:

\begin{verbatim}
A = A[array_sort(A)];
\end{verbatim}
Example

A homogeneous list may be sorted by using the opaque form of the `array_sort` function:

```c
private define cmp_function (s, i, j)
{
    if (s[i] > s[j]) return 1;
    if (s[i] < s[j]) return -1;
    return 0;
}

list = {};
% fill list ....
% now sort it
i = array_sort (list, &cmp_function, length(list));

% Create a new sorted list
list = list[i];
```

Alternatively one may first convert it to an array and use the built-in comparison function:

```c
a = list_to_array (list);
i = array_sort(a);

% Rearrange the elements
list[*] = a[i];
```

to get the effect of an "in-place" sort.

Notes

The default sorting algorithm is merge-sort. It has an $N \times \log(N)$ worst-case runtime compared to quick-sort’s worst-case $N^2$ runtime. The primary advantage of quick-sort is that it uses $O(1)$ additional memory, whereas merge-sort requires $O(N)$ additional memory.

A stable sorting algorithm is one that preserves the order of equal elements. Merge-sort is an inherently stable algorithm, whereas quick-sort is not. Nevertheless, the slang library ensures the stability of the results because it uses the indices themselves as tie-breakers. As a result, the following two statements may not produce the same results:

```c
i = array_sort (a; dir=-1);
i = array_reverse (array_sort (a; dir=1));
```

See Also

2.20 (set_default_sort_method), 2.10 (get_default_sort_method), 4.18 (strcmp), 7.9 (list_to_array)

2.8 array_swap

Synopsis

Swap elements of an array
Chapter 2. Array Functions

Usage

array_swap (Array_Type a, Int_Type i, Int_Type j)

Description

The array_swap function swaps the specified elements of an array. It is equivalent to

(a[i], a[j]) = (a[j], a[i]);

excetpt that it executes several times faster than the above construct.

See Also

2.5 (array_reverse), 2.23 (transpose)

2.9 cumsum

Synopsis

Compute the cumulative sum of an array

Usage

result = cumsum (Array_Type a [, Int_Type dim])

Description

The cumsum function performs a cumulative sum over the elements of a numeric array and returns the result. If a second argument is given, then it specifies the dimension of the array to be summed over. For example, the cumulative sum of [1,2,3,4], is the array [1,1+2,1+2+3,1+2+3+4], i.e., [1,3,6,10].

See Also

2.21 (sum), 2.22 (sumsq)

2.10 get_default_sort_method

Synopsis

Get the default sorting method

Usage

String_Type get_default_sort_method ()

Description

This function may be used to get the default sorting method used by array_sort. It will return one of the following strings:

"msort" Merge-Sort
"qsort" Quick-Sort

See Also

2.20 (set_default_sort_method), 2.7 (array_sort)
2.11 init_char_array

Synopsis
Initialize an array of characters

Usage
init_char_array (Array_Type a, String_Type s)

Description
The init_char_array function may be used to initialize a Char_Type array a by setting the
elements of the array a to the corresponding bytes of the string s.

Example
The statements

variable a = Char_Type [10];
init_char_array (a, "HelloWorld");

creates an character array and initializes its elements to the bytes in the string "HelloWorld".

Notes
The character array must be large enough to hold all the characters of the initialization string.
This function uses byte-semantics.

See Also
5.2 (bstring_to_array), 4.24 (strlen), 4.14 (strcat)

2.12 _isnull

Synopsis
Check an array for NULL elements

Usage
Char_Type[] = _isnull (a[])

Description
This function may be used to test for the presence of NULL elements of an array. Specifically,
it returns a Char_Type array of with the same number of elements and dimensionality of the
input array. If an element of the input array is NULL, then the corresponding element of the
output array will be set to 1, otherwise it will be set to 0.

Example
Set all NULL elements of a string array A to the empty string "":

A[where(_isnull(A))] = "";

Notes
It is important to understand the difference between A==NULL and _isnull(A). The latter
tests all elements of A against NULL, whereas the former only tests A itself.
See Also
2.24 (where), 2.4 (array_map)

2.13 length

Synopsis
Get the length of an object

Usage
Integer_Type length (obj)

Description
The length function may be used to get information about the length of an object. For simple scalar data-types, it returns 1. For arrays, it returns the total number of elements of the array.

Notes
If obj is a string, length returns 1 because a String_Type object is considered to be a scalar. To get the number of characters in a string, use the strlen function.

See Also
2.3 (array_info), 2.6 (array_shape), 12.17 (typeof), 4.24 (strlen)

2.14 max

Synopsis
Get the maximum value of an array

Usage
result = max (Array_Type a [,Int_Type dim])

Description
The max function examines the elements of a numeric array and returns the value of the largest element. If a second argument is given, then it specifies the dimension of the array to be searched. In this case, an array of dimension one less than that of the input array will be returned with the corresponding elements in the specified dimension replaced by the maximum value in that dimension.

Example
Consider the 2-d array

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
6 & 7 & 8 & 9 & 10 \\
\end{array}
\]

generated by

\[
a = \_reshape ([1:10], [2, 5]);
\]
Then \( \max(a) \) will return 10, and \( \max(a, 0) \) will return a 1-d array with elements

\[
\begin{array}{cccccc}
6 & 7 & 8 & 9 & 10 \\
\end{array}
\]

Notes

This function ignores NaNs in the input array.

See Also

2.16 (min), 2.15 (maxabs), 2.21 (sum), 2.19 (reshape)

2.15 maxabs

Synopsis

Get the maximum absolute value of an array

Usage

result = maxabs (Array_Type a [,Int_Type dim])

Description

The \( \text{maxabs} \) function behaves like the \( \max \) function except that it returns the maximum absolute value of the array. That is, \( \text{maxabs}(x) \) is equivalent to \( \max(\text{abs}(x)) \). See the documentation for the \( \max \) function for more information.

See Also

2.16 (min), 2.14 (max), 2.17 (minabs)

2.16 min

Synopsis

Get the minimum value of an array

Usage

result = min (Array_Type a [,Int_Type dim])

Description

The \( \min \) function examines the elements of a numeric array and returns the value of the smallest element. If a second argument is given, then it specifies the dimension of the array to be searched. In this case, an array of dimension one less than that of the input array will be returned with the corresponding elements in the specified dimension replaced by the minimum value in that dimension.

Example

Consider the 2-d array

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
6 & 7 & 8 & 9 & 10 \\
\end{array}
\]
generated by

\[ a = \text{\_reshape ([1:10], [2, 5]);} \]

Then \( \min(a) \) will return 1, and \( \min(a, 0) \) will return a 1-d array with elements

\[
\begin{bmatrix}
1 & 2 & 3 & 4 & 5
\end{bmatrix}
\]

Notes

This function ignores NaNs in the input array.

See Also

2.14 (max), 2.21 (sum), 2.19 (reshape)

2.17 minabs

Synopsis

Get the minimum absolute value of an array

Usage

\[
\text{result} = \text{minabs (Array\_Type \ a \ [,Int\_Type \ dim])}
\]

Description

The \text{minabs} function behaves like the \text{min} function except that it returns the minimum absolute value of the array. That is, \( \text{minabs}(x) \) is equivalent to \( \min(\text{abs}(x)) \). See the documentation for the \text{min} function for more information.

See Also

2.16 (min), 2.14 (max), 2.15 (maxabs)

2.18 _reshape

Synopsis

Copy an array to a new shape

Usage

\[
\text{Array\_Type \ _reshape (Array\_Type \ A, \ Array\_Type \ I)}
\]

Description

The \_reshape function creates a copy of an array \( A \), reshapes it to the form specified by \( I \) and returns the result. The elements of \( I \) specify the new dimensions of the copy of \( A \) and must be consistent with the number of elements \( A \).

Example

If \( A \) is a 100 element 1-d array, a new 2-d array of size 20 by 5 may be created from the elements of \( A \) by
B = _reshape (A, [20, 5]);

Notes
The reshape function performs a similar function to _reshape. In fact, the _reshape function could have been implemented via:

```c
define _reshape (a, i)
{
    a = @a; % Make a new copy
    reshape (a, i);
    return a;
}
```

See Also
2.19 (reshape), 2.6 (array_shape), 2.3 (array_info)

2.19 reshape

Synopsis
Reshape an array

Usage
reshape (Array_Type A, Array_Type I)

Description
The reshape function changes the shape of A to have the shape specified by the 1-d integer array I. The elements of I specify the new dimensions of A and must be consistent with the number of elements A.

Example
If A is a 100 element 1-d array, it can be changed to a 2-d 20 by 5 array via

```c
reshape (A, [20, 5]);
```

However, reshape(A, [11,5]) will result in an error because the [11,5] array specifies 55 elements.

Notes
Since reshape modifies the shape of an array, and arrays are treated as references, then all references to the array will reference the new shape. If this effect is unwanted, then use the _reshape function instead.

See Also
2.18 (_reshape), 2.3 (array_info), 2.6 (array_shape)


2.20  set_default_sort_method

Synopsis

Set the default sorting method

Usage

set_default_sort_method (String_Type method)

Description

This function may be used to set the default sorting method used by array_sort. The following methods are supported:

"msort"  Merge-Sort
"qsort"  Quick-Sort

See Also

2.10 (get_default_sort_method), 2.7 (array_sort)

2.21  sum

Synopsis

Sum over the elements of an array

Usage

result = sum (Array_Type a [, Int_Type dim])

Description

The sum function sums over the elements of a numeric array and returns its result. If a second argument is given, then it specifies the dimension of the array to be summed over. In this case, an array of dimension one less than that of the input array will be returned.

If the input array is an integer type, then the resulting value will be a Double_Type. If the input array is a Float_Type, then the result will be a Float_Type.

Example

The mean of an array a of numbers is

\[
\frac{\text{sum}(a)}{\text{length}(a)}
\]

See Also

2.9 (cumsum), 2.22 (sumsq), 2.23 (transpose), 2.19 (reshape)

2.22  sumsq

Synopsis

Sum over the squares of the elements of an array
2.23 transpose

Usage

result = sumsq (Array_Type a [, Int_Type dim])

Description

The sumsq function sums over the squares of the elements of a numeric array and returns its result. If a second argument is given, then it specifies the dimension of the array to be summed over. In this case, an array of dimension one less than that of the input array will be returned.

If the input array is an integer type, then the resulting value will be a Double_Type. If the input array is a Float_Type, then the result will be a Float_Type.

For complex arrays, the sum will be over the squares of the moduli of the complex elements.

See Also

2.9 (cumsum), 2.22 (sumsq), 9.22 (hypot), 2.23 (transpose), 2.19 (reshape)

2.23 transpose

Synopsis

Transpose an array

Usage

Array_Type transpose (Array_Type a)

Description

The transpose function returns the transpose of a specified array. By definition, the transpose of an array, say one with elements a[i,j,...,k] is an array whose elements are a[k,...,j,i].

See Also

2.18 (_reshape), 2.19 (reshape), 2.21 (sum), 2.3 (array_info), 2.6 (array_shape)

2.24 where

Usage

Array_Type where (Array_Type a [, Ref_Type jp])

Description

The where function examines a numeric array a and returns an integer array giving the indices of a where the corresponding element of a is non-zero. The function accepts an optional Ref_Type argument that will be set to complement set of indices, that is, the indices where a is zero. In fact

i = where (a);

j = where (not a);

and

i = where (a, &j);
are equivalent, but the latter form is preferred since it executes about twice as fast as the former.

The `where` function can also be used with relational operators and with the boolean binary `or` and `and` operators, e.g.,

```
    a = where (array == "a string");
    a = where (array <= 5);
    a = where (2 <= array <= 10);
    a = where ((array == "a string") or (array == "another string"));
```

Using in the last example the short-circuiting `||` and `&&` operators, will result in a `TypeMismatchError` exception.

Although this function may appear to be simple or even trivial, it is arguably one of the most important and powerful functions for manipulating arrays.

Example

Consider the following:

```
variable X = [0.0:10.0:0.01];
variable A = sin (X);
variable I = where (A < 0.0);
A[I] = cos (X) [I];
```

Here the variable `X` has been assigned an array of doubles whose elements range from 0.0 through 10.0 in increments of 0.01. The second statement assigns `A` to an array whose elements are the `sin` of the elements of `X`. The third statement uses the `where` function to get the indices of the elements of `A` that are less than 0. Finally, the last statement replaces those elements of `A` by the cosine of the corresponding elements of `X`.

Notes

Support for the optional argument was added to version 2.1.0.

See Also

2.26 (wherefirst), 2.29 (wherelast), 2.32 (wherenot), 2.25 (wherediff), 2.3 (array_info), 2.6 (array_shape), 2.12 (_isnull)

2.25 wherediff

Synopsis

Get the indices where adjacent elements differ

Usage

```
Array_Type wherediff (Array_Type A [, Ref_Type jp])
```

Description

This function returns an array of the indices where adjacent elements of the array `A` differ. If the optional second argument is given, it must be a reference to a variable whose value will be set to the complement indices (those where adjacent elements are the same).
The returned array of indices will consist of those elements $i$ where $A[i] \neq A[i-1]$. Since no element precedes the 0th element, $A[0]$ differs from its non-existing preceeding element; hence the index 0 will a member of the returned array.

**Example**

Suppose that $A = [1, 1, 3, 0, 0, 4, 7, 7]$. Then,

$$i = \text{wherediff}(A, &j);$$

will result in $i = [0, 2, 3, 5, 6]$ and $j = [1, 4, 7]$.

**Notes**

Higher dimensional arrays are treated as a 1-d array of contiguous elements.

**See Also**

2.24 (where), 2.32 (wherenot)

### 2.26 wherefirst

**Synopsis**

Get the index of the first non-zero array element

**Usage**

```c
Int_Type wherefirst (Array_Type a [,start_index])
```

**Description**

The `wherefirst` function returns the index of the first non-zero element of a specified array. If the optional parameter `start_index` is given, the search will take place starting from that index. If a non-zero element is not found, the function will return `NULL`.

**Notes**

The single parameter version of this function is equivalent to

```
define wherefirst (a)
  { variable i = where (a);
    if (length(i))
      return i[0];
    else
      return NULL;
  }
```

**See Also**

2.24 (where), 2.29 (wherelast), ?? (wherfirstmin), ?? (wherfirstmax)
2.27 wherefirstmax

Synopsis
Get the index of the first maximum array value

Usage
Int_Type wherefirstmax (Array_Type a)

Description
This function is equivalent to

\[ \text{index} = \text{wherefirst} (a == \text{max}(a)); \]

It executes about 3 times faster, and does not require the creation of temporary arrays.

See Also
2.26 (wherefirst), 2.27 (wherefirstmax), 2.31 (wherelastmin), 2.16 (min), 2.14 (max)

2.28 wherefirstmin

Synopsis
Get the index of the first minimum array value

Usage
Int_Type wherefirstmin (Array_Type a)

Description
This function is equivalent to

\[ \text{index} = \text{wherefirst} (a == \text{min}(a)); \]

It executes about 3 times faster, and does not require the creation of temporary arrays.

See Also
2.26 (wherefirst), 2.31 (wherelastmin), 2.27 (wherefirstmax), 2.16 (min), 2.14 (max)

2.29 wherelast

Synopsis
Get the index of the last non-zero array element

Usage
Int_Type wherelast (Array_Type a[, start_index])

Description
The wherelast function returns the index of the last non-zero element of a specified array. If the optional parameter start_index is given, the backward search will take place starting from that index. If a non-zero element is not found, the function will return NULL.
Notes

The single parameter version of this function is equivalent to

```
define wherelast (a)
{
    variable i = where (a);
    if (length(i))
        return i[-1];
    else
        return NULL;
}
```

See Also

2.24 (where), 2.26 (wherefirst), 2.31 (wherelastmin), 2.30 (wherelastmax)

2.30 wherelastmax

Synopsis

Get the index of the last maximum array value

Usage

```
Int_Type wherelastmax (Array_Type a)
```

Description

This function is equivalent to

```
index = wherelast (a == max(a));
```

It executes about 3 times faster, and does not require the creation of temporary arrays.

See Also

2.29 (wherelast), 2.28 (wherefirstmin), 2.31 (wherelastmin), 2.16 (min), 2.14 (max)

2.31 wherelastmin

Synopsis

Get the index of the last minimum array value

Usage

```
Int_Type wherelastmin (Array_Type a)
```

Description

This function is equivalent to

```
index = wherelast (a == min(a));
```

It executes about 3 times faster, and does not require the creation of temporary arrays.

See Also

2.29 (wherelast), 2.28 (wherefirstmin), 2.30 (wherelastmax), 2.16 (min), 2.14 (max)
2.32 where\texttt{not}

Synopsis

Get indices where a numeric array is 0

Usage

\texttt{Array\_Type where\texttt{not} (Array\_Type a)}

Description

This function is equivalent to \texttt{where(not a)}. See the documentation for \texttt{where} for more information.

See Also

2.24 (where), 2.25 (wherediff), 2.26 (wherefirst), 2.29 (wherelast)
3.1 assoc_delete_key

Synopsis
Delete a key from an Associative Array

Usage
assoc_delete_key (Assoc_Type a, String_Type k)

Description
The assoc_delete_key function deletes a key given by k from the associative array a. If the specified key does not exist in a, then this function has no effect.

See Also
3.4 (assoc_key_exists), 3.2 (assoc_get_keys)

3.2 assoc_get_keys

Synopsis
Return all the key names of an Associative Array

Usage
String_Type[] assoc_get_keys (Assoc_Type a)

Description
This function returns all the key names of an associative array a as an ordinary one dimensional array of strings. If the associative array contains no keys, an empty array will be returned.

See Also
3.3 (assoc_get_values), 3.4 (assoc_key_exists), 3.1 (assoc_delete_key), 2.13 (length)
3.3 assoc_get_values

Synopsis

Return all the values of an Associative Array

Usage

`Array_Type assoc_get_keys (Assoc_Type a)`

Description

This function returns all the values in the associative array `a` as an array of proper type. If the associative array contains no keys, an empty array will be returned.

Example

Suppose that `a` is an associative array of type `Integer_Type`, i.e., it was created via

```c
variable a = Assoc_Type[Integer_Type];
```

Then the following may be used to print the values of the array in ascending order:

```c
define print_sorted_values (a)
{
    variable v = assoc_get_values (a);
    variable i = array_sort (v);
    v = v[i];
    foreach (v)
    {
        variable vi = ();
        () = fprintf (stdout, "%d\n", vi);
    }
}
```

See Also

3.2 (assoc_get_keys), 3.4 (assoc_key_exists), 3.1 (assoc_delete_key), 2.7 (array_sort)

3.4 assoc_key_exists

Synopsis

Check to see whether a key exists in an Associative Array

Usage

`Integer_Type assoc_key_exists (Assoc_Type a, String_Type k)`

Description

The `assoc_key_exists` function may be used to determine whether or not a specified key `k` exists in an associative array `a`. It returns 1 if the key exists, or 0 if it does not.

See Also

3.2 (assoc_get_keys), 3.3 (assoc_get_values), 3.1 (assoc_delete_key)
Chapter 4

Functions that Operate on Strings

4.1 count_char_occurrences

Synopsis

Count the number of occurrences of a character in a string

Usage

UInt_Type count_char_occurrences (str, ch)

Description

This function returns the number of times the specified character ch occurs in the string str.

Notes

If UTF-8 mode is in effect, then the character may correspond to more than one byte. In such a case, the function returns the number of such byte-sequences in the string. To count actual bytes, use the count_byte_occurrences function.

See Also

5.6 (count_byte_occurrences)

4.2 create_delimited_string

Synopsis

Concatenate strings using a delimiter

Usage

String_Type create_delimited_string (delim, s_1, s_2, ..., s_n, n)

String_Type delim, s_1, ..., s_n
Int_Type n
Description

`create_delimited_string` performs a concatenation operation on the `n` strings `s_1, \ldots, s_n`, using the string `delim` as a delimiter. The resulting string is equivalent to one obtained via

\[ s_1 + delim + s_2 + delim + \ldots + s_n \]

Example

\[
create_delimited_string ("/", "user", "local", "bin", 3);
\]

will produce "usr/local/bin".

Notes

New code should use the `strjoin` function, which performs a similar task.

See Also

4.23 (strjoin), 4.5 (is_list_element), 4.3 (extract_element), 4.16 (strchop), 4.14 (strcat)

4.3 extract_element

Synopsis

Extract the `nth` element of a string with delimiters

Usage

\[
String_Type extract_element (String_Type list, Int_Type nth, Int_Type delim)
\]

Description

The `extract_element` function may be used to extract the `nth` substring of a string delimited by the character given by the `delim` parameter. If the string contains fewer than the requested substring, the function will return `NULL`. Substring elements are numbered from 0.

Example

The expression

\[
extract_element ("element 0, element 1, element 2", 1, ",")
\]

returns the string " element 1", whereas

\[
extract_element ("element 0, element 1, element 2", 1, ")"
\]

returns "0,"

The following function may be used to compute the number of elements in the list:

\[
define num_elements (list, delim)
\
\{
\quad variable nth = 0;
\quad while (NULL != extract_element (list, nth, delim))
\quad \quad nth++;
\quad return nth;
\}\n\]
Alternatively, the `strchop` function may be more useful. In fact, `extract_element` may be expressed in terms of the function `strchop` as

```c
#define extract_element (list, nth, delim)
{
    list = strchop(list, delim, 0);
    if (nth >= length (list))
        return NULL;
    else
        return list[nth];
}
```

and the `num_elements` function used above may be recoded more simply as:

```c
#define num_elements (list, delim)
{
    return length (strchop (length, delim, 0));
}
```

**Notes**

New code should make use of the `List_Type` object for lists.

**See Also**

4.5 (is_list_element), 4.6 (is_substr), 4.33 (strtok), 4.16 (strchop), 4.2 (create_delimited_string)

### 4.4 glob_to_regexp

**Synopsis**

Convert a globbing expression to a regular expression

**Usage**

```c
String_Type glob_to_regexp (String_Type g)
```

**Description**

This function may be used to convert a so-called globbing expression to a regular expression. A globbing expression is frequently used for matching filenames where '?' represents a single character and '*' represents 0 or more characters.

**Notes**

The `slsh` program that is distributed with the `S-Lang` library includes a function called `glob` that is a wrapper around `glob_to_regexp` and `listdir`. It returns a list of filenames matching a globbing expression.

**See Also**

4.20 (string_match), 16.8 (listdir)
4.5  is_list_element

Synopsis
Test whether a delimited string contains a specific element

Usage
Int_Type is_list_element (String_Type list, String_Type elem, Int_Type delim)

Description
The is_list_element function may be used to determine whether or not a delimited list of
substring, list, contains the element elem. If elem is not an element of list, the function
will return zero, otherwise, it returns 1 plus the matching element number.

Example
The expression

    is_list_element ("element 0, element 1, element 2", "0," , ' ');

returns 2 since "0," is element number one of the list (numbered from zero).

See Also
4.3 (extract_element), 4.6 (is_substr), 4.2 (create_delimited_string)

4.6  is_substr

Synopsis
Test for a specified substring within a string

Usage
Int_Type is_substr (String_Type a, String_Type b)

Description
This function may be used to determine if a contains the string b. If it does not, the function
returns 0; otherwise it returns the position of the first occurrence of b in a expressed in terms
of characters, not bytes.

Notes
This function regards the first character of a string to be given by a position value of 1.
The distinction between characters and bytes is significant in UTF-8 mode.
This function has been vectorized in the sense that if an array of strings is passed for either
of the string-valued arguments, then a corresponding array of integers will be returned. If two
arrays are passed then the arrays must have the same length.

See Also
4.43 (substr), 4.20 (string_match), 4.29 (strreplace)
4.7  make_printable_string

Synopsis
Format a string suitable for parsing

Usage
String_Type make_printable_string(String_Type str)

Description
This function formats a string in such a way that it may be used as an argument to the eval function. The resulting string is identical to str except that it is enclosed in double quotes and the backslash, newline, control, and double quote characters are expanded.

See Also
19.4 (eval), 4.40 (str_quote_string)

4.8  Sprintf

Synopsis
Format objects into a string (deprecated)

Usage
String_Type Sprintf (String_Type format, ..., Int_Type n)

Description
This function performs a similar task as the sprintf function but requires an additional argument that specifies the number of items to format. For this reason, the sprintf function should be used.

See Also
4.10 (sprintf), 12.12 (string), 4.11 (sscanf), 10.9 (vmessage)

4.9  strbskipchar

Synopsis
Get an index to the previous character in a UTF-8 encoded string

Usage
(p1, wch) = strbskipchar (str, p0 [,skip_combining])

Description
This function moves backward from the 0-based byte-offset p0 in the string str to the previous character in the string. It returns the byte-offset (p1) of the previous character and the decoded character value at that byte-offset.
The optional third argument specifies the handling of combining characters. If it is non-zero, combining characters will be ignored, otherwise a combining character will not be treated differently from other characters. The default is to ignore such characters.

If the byte-offset p0 corresponds to the end of the string (p0=0), then (p0,0) will be returned. Otherwise if the byte-offset specifies a value that lies outside the string, an IndexError exception will be thrown. Finally, if the byte-offset corresponds to an illegally coded character, the character returned will be the negative byte-value at the position.

See Also

4.31 (strskipchar), 4.30 (strskipbytes)

4.10 sprintf

Synopsis

Format objects into a string

Usage

String_Type sprintf (String fmt, ...)

Description

The sprintf function formats a string from a variable number of arguments according to the format specification string \texttt{fmt}.

The format string is a C library \texttt{sprintf} style format descriptor. Briefly, the format string may consist of ordinary characters (not including the \texttt{\%} character), which are copied into the output string as-is, and conversion specification sequences introduced by the \texttt{\%} character. The number of additional arguments passed to the \texttt{sprintf} function must be consistent with the number required by the format string.

The \texttt{\%} character in the format string starts a conversion specification that indicates how an object is to be formatted. Usually the percent character is followed immediately by a conversion specification character. However, it may optionally be followed by flag characters, field width characters, and precision modifiers, as described below.

The character immediately following the \texttt{\%} character may be one or more of the following flag characters:

- Use left-justification
- Use alternate form for formatting.
- Use 0 padding
- Precede a number by a plus or minus sign.
- (space) Use a blank instead of a plus sign.

The flag characters (if any) may be followed by an optional field width specification string represented by one or more digit characters. If the size of the formatted object is less than the field width, it will be right-justified in the specified field width, unless the \texttt{-} flag was given, in which case it will be left justified.

If the next character in the control sequence is a period, then it introduces a precision specification sequence. The precision is given by the digit characters following the period. If none
are given the precision is taken to be 0. The meaning of the precision specifier depends upon
the type of conversion: For integer conversions, it gives the minimum number digits to appear
in the output. For e and f floating point conversions, it gives the number of digits to appear
after the decimal point. For the g floating point conversion, it gives the maximum number
of significant digits to appear. Finally for the s and S conversions it specifies the maximum
number of characters to be copied to the output string.

The next character in the sequence may be a modifier that controls the size of object to be
formatted. It may consist of the following characters:

- h This character is ignored in the current implementation.
- l The integer is be formatted as a long integer, or a
  character as a wide character.

Finally the conversion specification sequence ends with the conversion specification character
that describes how the object is to be formatted:

- s as a string
- f as a floating point number
- e as a float using exponential form, e.g., 2.345e08
- g format as e or f, depending upon its value
- c as a character
- b as a byte
- % a literal percent character
- d as a signed decimal integer
- u as an unsigned decimal integer
- o as an octal integer
- X,x as hexadecimal
- B as a binary integer
- S convert object to a string and format accordingly

The S conversion specifier is a S-Lang extension which will cause the corresponding object
to be converted to a string using the string function, and then converted as s formatted as
string. In fact, sprintf("%S",x) is equivalent to sprintf("%s",string(x)).

Example

```
sprintf("%s","hello")
==> "hello"
sprintf("%s %s","hello", "world")
==> "hello world"
sprintf("Agent %.3d",7)
==> "Agent 007"
sprintf("%S",PI)
==> "3.141592653589793"
sprintf("%g",PI)
==> "3.14159"
sprintf("%.2g",PI)
==> "3.1"
sprintf("%.2e",PI)
==> "3.14e+00"
sprintf("%.2f",PI)
==> "3.14"
sprintf("%X 8.2f","PI")
==> "| 3.14|
```

```python
Example

```
sprintf("%S",{1,2,3})
==> "List_Type with 3 elements"
sprintf("%S",1+2i)
==> "(1 + 2i)"
```
Notes

The `set_float_format` function controls the format for the S conversion of floating point numbers.

See Also

12.12 (string), 4.11 (sscanf), 10.5 (message), 5.8 (pack), 9.39 (set_float_format)

4.11 sscanf

Synopsis

Parse a formatted string

Usage

```
Int_Type sscanf (s, fmt, r1, ... rN)
```

```
String_Type s, fmt;
Ref_Type r1, ..., rN
```

Description

The `sscanf` function parses the string `s` according to the format `fmt` and sets the variables whose references are given by `r1`, ..., `rN`. The function returns the number of references assigned, or throws an exception upon error.

The format string `fmt` consists of ordinary characters and conversion specifiers. A conversion specifier begins with the special character `%` and is described more fully below. A white space character in the format string matches any amount of whitespace in the input string. Parsing of the format string stops whenever a match fails.

The `%` character is used to denote a conversion specifier whose general form is given by `%[+-][width][type]format` where the brackets indicate optional items. If `*` is present, then the conversion will be performed but no assignment to a reference will be made. The width specifier specifies the maximum field width to use for the conversion. The type modifier is used to indicate the size of the object, e.g., a short integer, as follows.

If `type` is given as the character `h`, then if the format conversion is for an integer (dioux), the object assigned will be a short integer. If `type` is 1, then the conversion will be to a long integer for integer conversions, or to a double precision floating point number for floating point conversions.

The format specifier is a character that specifies the conversion:

- `%` Matches a literal percent character. No assignment is performed.
- `d` Matches a signed decimal integer.
- `D` Matches a long decimal integer (equiv to `ld`)
- `u` Matches an unsigned decimal integer.
- `U` Matches an unsigned long decimal integer (equiv to `lu`)
- `i` Matches either a hexadecimal integer, decimal integer, or octal integer.
- `I` Equivalent to `li`.
4.12. strbytelen

**Synopsis**

Get the number of bytes in a string

**Usage**

\[
\text{Int}\_\text{Type}\ \text{strbytelen} (\text{String}\_\text{Type}\ s)
\]

**Description**

This function returns the number of bytes in a string. In UTF-8 mode, this value is generally different from the number of characters in a string. For the latter information, the \text{strlen} or \text{strcharlen} functions should be used.

**Notes**

This function has been vectorized in the sense that if an array of strings is passed to the function, then a corresponding array of integers will be returned.

**See Also**

\text{4.24} (strlen), \text{4.15} (strcharlen), \text{2.13} (length)
4.13 strbytesub

Synopsis
Replace a byte with another in a string.

Usage
String_Type strsub (String_Type s, Int_Type pos, UChar_Type b)

Description
The strbytesub function may be used to substitute the byte b for the byte at byte position pos of the string s. The resulting string is returned.

Notes
The first byte in the string s is specified by pos equal to 1. This function uses byte semantics, not character semantics.

See Also
4.32 (strsub), 4.6 (is_substr), 4.29 (strreplace), 4.12 (strbytelen)

4.14 strcat

Synopsis
Concatenate strings

Usage
String_Type strcat (String_Type a_1, ..., String_Type a_N)

Description
The strcat function concatenates its N string arguments a_1, ... a_N together and returns the result.

Example

strcat ("Hello", " ", "World");

produces the string "Hello World".

Notes
This function is equivalent to the binary operation a_1+...+a_N. However, strcat is much faster making it the preferred method to concatenate strings.

See Also
4.10 (printf), 4.23 (strjoin)
4.15 strcharlen

Synopsis
Get the number of characters in a string including combining characters

Usage
Int_Type strcharlen (String_Type s)

Description
The strcharlen function returns the number of characters in a string. If the string contains
combining characters, then they are also counted. Use the strlen function to obtain the
character count ignoring combining characters.

Notes
This function has been vectorized in the sense that if an array of strings is passed to the
function, then a corresponding array of integers will be returned.

See Also
4.24 (strlen), 4.12 (strbytelen)

4.16 strchop

Synopsis
Chop or split a string into substrings.

Usage
String_Type[] strchop (String_Type str, Int_Type delim, Int_Type quote)

Description
The strchop function may be used to split-up a string str that consists of substrings delimited
by the character specified by delim. If the integer quote is non-zero, it will be taken as a
quote character for the delimiter. The function returns the substrings as an array.

Example
The following function illustrates how to sort a comma separated list of strings:

define sort_string_list (a)
{
    variable i, b, c;
    b = strchop (a, ",", 0);

    i = array_sort (b);
    b = b[i];  % rearrange

    % Convert array back into comma separated form
    return strjoin (b, ",");
}
4.17 strchopr

Synopsis
Chop or split a string into substrings.

Usage
String_Type[ ] strchopr (String_Type str, String_Type delim, String_Type quote)

Description
This routine performs exactly the same function as strchop except that it returns the substrings in the reverse order. See the documentation for strchop for more information.

See Also
4.16 (strchop), 4.33 (strtok), 4.23 (strjoin)

4.18 strcmp

Synopsis
Compare two strings

Usage
Int_Type strcmp (String_Type a, String_Type b)

Description
The strcmp function may be used to perform a case-sensitive string comparison, in the lexicographic sense, on strings a and b. It returns 0 if the strings are identical, a negative integer if a is less than b, or a positive integer if a is greater than b.

Example
Thestrup function may be used to perform a case-insensitive string comparison:

define case_insensitive_strcmp (a, b)
{
    return strcmp (strup(a), strup(b));
}

Notes
One may also use one of the binary comparison operators, e.g., a > b.
This function has been vectorized in the sense that if an array of strings is passed to the function, then a corresponding array of integers will be returned.

See Also
4.38 (strup), 4.28 (strncmp)
4.19  strcompress

Synopsis
Remove excess whitespace characters from a string

Usage
String_Type strcompress (String_Type s, String_Type white)

Description
The strcompress function compresses the string s by replacing a sequence of one or more
characters from the set white by the first character of white. In addition, it also removes all
leading and trailing characters from s that are part of white.

Example
The expression

    strcompress (";,apple,,cherry;,banana", ",;");

returns the string "apple,cherry,banana".

Notes
This function has been vectorized in the sense that if an array of strings is passed as the
first argument then a corresponding array of strings will be returned. Array values are not
supported for the remaining arguments.

See Also
4.35 (strtrim), 4.34 (strtrans), 4.39 (str_delete_chars)

4.20  string_match

Synopsis
Match a string against a regular expression

Usage
Int_Type string_match(String_Type str, String_Type pat [,Int_Type pos])

Description
The string_match function returns zero if str does not match the regular expression specified
by pat. This function performs the match starting at the first byte of the string. The optional
pos argument may be used to specify a different byte offset (numbered from 1). This function
returns the position in bytes (numbered from 1) of the start of the match in str. The exact
substring matched may be found using string_match_nth.

Notes
Positions in the string are specified using byte-offsets not character offsets. The value returned
by this function is measured from the beginning of the string str.
The function is not yet UTF-8 aware. If possible, consider using the pcre module for better, more sophisticated regular expressions.

The pos argument was made optional in version 2.2.3.

See Also

4.22 (string_matches), 4.21 (string_match_nth), 4.18 (strcmp), 4.28 (strncmp)

4.21 string_match_nth

Synopsis

Get the result of the last call to string_match

Usage

(Int_Type pos, Int_Type len) = string_match_nth(Int_Type nth)

Description

The string_match_nth function returns two integers describing the result of the last call to string_match. It returns both the zero-based byte-position of the nth submatch and the length of the match.

By convention, nth equal to zero means the entire match. Otherwise, nth must be an integer with a value 1 through 9, and refers to the set of characters matched by the nth regular expression enclosed by the pairs \( (, \)\).

Example

Consider:

```c
variable matched, pos, len;
matched = string_match("hello world", "\([a-z]+\) \([a-z]+\)"R, 1);
if (matched)
  (pos, len) = string_match_nth(2);
```

This will set matched to 1 since a match will be found at the first byte position, pos to 6 since w is offset 6 bytes from the beginning of the string, and len to 5 since "world" is 5 bytes long.

Notes

The position offset is not affected by the value of the offset parameter to the string_match function. For example, if the value of the last parameter to the string_match function had been 3, pos would still have been set to 6.

The string_matches function may be used as an alternative to string_match_nth.

See Also

4.20 (string_match), 4.22 (string_matches)
4.22 string_matches

Synopsis

Match a string against a regular expression and return the matches

Usage

String_Type[] string_matches(String_Type str, String_Type pat [,Int_Type pos])

Description

The string_matches function combines the functionality of string_match and string_match_nth. Like string_match, it matches the string str against the regular expression pat. If the string does not match the pattern the function will return NULL. Otherwise, the function will return an array of strings whose ith element is the string that corresponds to the return value of the string_match_nth function.

Example

strs = string_matches ("p0.5keV_27deg.dat",
"p\([0-9.]+\)keV\([0-9.]+\)deg\.[dat]"R, 1);
% ==> strs[0] = "p0.5keV_27deg.dat"
% strs[1] = "0.5"
% strs[2] = "27"

strs = string_matches ("q0.5keV_27deg.dat",
"p\([0-9.]+\)keV\([0-9.]+\)deg\.[dat]"R);
% ==> strs = NULL

Notes

The function is not yet UTF-8 aware. If possible, consider using the pcre module for better, more sophisticated regular expressions.

The pos argument was made optional in version 2.2.3.

See Also

4.20 (string_match), 4.21 (string_match_nth), 4.18 (strcmp), 4.28 (strncmp)

4.23 strjoin

Synopsis

Concatenate elements of a string array

Usage

String_Type strjoin (Array_Type a [, String_Type delim])

Description

The strjoin function operates on an array of strings by joining successive elements together separated with the optional delimiter delim. If delim is not specified, then empty string "" will be used resulting in a concatenation of the elements.
Example

Suppose that

days = ["Sun","Mon","Tue","Wed","Thu","Fri","Sat","Sun"];

Then strjoin (days,"+") will produce "Sun+Mon+Tue+Wed+Thu+Fri+Sat+Sun". Similarly, strjoin (["","",""], "X") will produce "XX".

See Also
4.16 (strchop), 4.14 (strcat)

4.24 strlen

Synopsis
Compute the length of a string

Usage
Int_Type strlen (String_Type a)

Description
The strlen function may be used to compute the character length of a string ignoring the presence of combining characters. The strcharlen function may be used to count combining characters as distinct characters. For byte-semantics, use the strbytelen function.

Example
After execution of

    variable len = strlen ("hello");

len will have a value of 5.

Notes
This function has been vectorized in the sense that if an array of strings is passed to the function, then a corresponding array of integers will be returned.

See Also
4.12 (strbytelen), 4.15 (strcharlen), 5.5 (bstrlen), 2.13 (length), 4.43 (substr)

4.25 strlow

Synopsis
Convert a string to lowercase

Usage
String_Type strlow (String_Type s)
4.26. strnbytecmp

Description

The strlow function takes a string s and returns another string identical to s except that all upper case characters that are contained in s are converted to lower case.

Example

The function

```
define Strcmp (a, b)
{
    return strcmp (strlow (a), strlow (b));
}
```

performs a case-insensitive comparison operation of two strings by converting them to lower case first.

Notes

This function has been vectorized in the sense that if an array of strings is passed to the function, then a corresponding array of strings will be returned.

See Also

4.38 (strup), 12.13 (tolower), 4.18 (strcmp), 4.35 (strtrim), 12.6 (define_case)

4.26  strnbytecmp

Synopsis

Compare the first n bytes of two strings

Usage

```
Int_Type strnbytecmp (String_Type a, String_Type b, Int_Type n)
```

Description

This function compares the first n bytes of the strings a and b. See the documentation for strcmp for information about the return value.

Notes

This function has been vectorized in the sense that if an array of strings is passed for either of the string-valued arguments, then a corresponding array of integers will be returned. If two arrays are passed then the arrays must have the same length.

See Also

4.28 (strncmp), 4.27 (strncharcmp), 4.18 (strcmp)

4.27  strncharcmp

Synopsis

Compare the first n characters of two strings
Usage

```c
Int_Type strncharcmp (String_Type a, String_Type b, Int_Type n)
```

Description

This function compares the first \( n \) characters of the strings \( a \) and \( b \) counting combining characters as distinct characters. See the documentation for `strcmp` for information about the return value.

Notes

This function has been vectorized in the sense that if an array of strings is passed for either of the string-valued arguments, then a corresponding array of integers will be returned. If two arrays are passed then the arrays must have the same length.

See Also

4.28 (strncmp), 4.26 (strnbytecmp), 4.18 (strcmp)

4.28 `strncmp`

Synopsis

Compare the first few characters of two strings

Usage

```c
Int_Type strncmp (String_Type a, String_Type b, Int_Type n)
```

Description

This function behaves like `strcmp` except that it compares only the first \( n \) characters in the strings \( a \) and \( b \). See the documentation for `strcmp` for information about the return value.

In counting characters, combining characters are not counted, although they are used in the comparison. Use the `strncharcmp` function if you want combining characters to be included in the character count. The `strnbytecmp` function should be used to compare bytes.

Example

The expression

```c
strncmp ("apple", "appliance", 3);
```

will return zero since the first three characters match.

Notes

This function uses character semantics.

This function has been vectorized in the sense that if an array of strings is passed for either of the string-valued arguments, then a corresponding array of integers will be returned. If two arrays are passed then the arrays must have the same length.

See Also

4.18 (strcmp), 4.24 (strlen), 4.27 (strncharcmp), 4.26 (strnbytecmp)
4.29  strreplace

Synopsis
Replace one or more substrings

Usage
(new,n) = strreplace(a, b, c, max_n)

Usage
new = strreplace(a, b, c)

Description
The strreplace function may be used to replace one or more occurrences of b in a with c. This function supports two calling interfaces.

The first form may be used to replace a specified number of substrings. If max_n is positive, then the first max_n occurrences of b in a will be replaced. Otherwise, if max_n is negative, then the last abs(max_n) occurrences will be replaced. The function returns the resulting string and an integer indicating how many replacements were made.

The second calling form may be used to replace all occurrences of b in a with c. In this case, only the resulting string will be returned.

Example
The following function illustrates how strreplace may be used to remove all occurrences of a specified substring:

```c
define delete_substrings (a, b)
{
    return strreplace (a, b, "");
}
```

See Also
4.6 (is_substr), 4.32 (strsub), 4.35 (strtrim), 4.34 (strtrans), 4.39 (str_delete_chars)

4.30  strskipbytes

Synopsis
Skip a range of bytes in a byte string

Usage
Int_Type strskipbytes (str, range [n0 [,nmax]])

String_Type s;
String_Type range;
Int_Type n0, nmax;


Description

This function skips over a range of bytes in a string \texttt{str}. The byte range to be skipped is specified by the \texttt{range} parameter. Optional start (\texttt{n0}) and stop (\texttt{max}) (0-based) parameters may be used to specify the part of the input string to be processed. The function returns a 0-based offset from the beginning of the string where processing stopped.

See the documentation for the \texttt{strtrans} function for the format of the range parameter.

See Also

4.31 (\texttt{strskipchar}), 4.9 (\texttt{strbskipchar}), 4.34 (\texttt{strtrans})

4.31 \texttt{strskipchar}

Synopsis

Get an index to the next character in a UTF-8 encoded string

Usage

\begin{verbatim}
(p1, wch) = strskipchar (str, p0 [,skip_combining])
\end{verbatim}

Description

This function decodes the character at the 0-based byte-offset \texttt{p0} in the string \texttt{str}. It returns the byte-offset (\texttt{p1}) of the next character in the string and the decoded character at byte-offset \texttt{p0}.

The optional third argument specifies the handling of combining characters. If it is non-zero, combining characters will be ignored, otherwise a combining character will not be treated differently from other characters. The default is to ignore such characters.

If the byte-offset \texttt{p0} corresponds to the end of the string, then (\texttt{p0},0) will be returned. Otherwise if the byte-offset specifies a value that lies outside the string, an \texttt{IndexError} exception will be thrown. Finally, if the byte-offset corresponds to an illegally coded character, the character returned will be the negative byte-value at the position.

Example

The following is an example of a function that skips alphanumeric characters and returns the new byte-offset.

\begin{verbatim}
private define skip_word_chars (line, p)
{
    variable p1 = p, ch;
    do
    {
        p = p1;
        (p1, ch) = strskipchar (line, p, 1);
    } while (isalnum(ch));
    return p;
}
\end{verbatim}
4.32. strsub

Notes

In non-UTF-8 mode (_slang_utf8_ok=0), this function is equivalent to:

```c
#define strskipchar (s, p)
{
    if ((p < 0) || (p > strlen(s)))
        throw IndexError;
    if (p == strlen(s))
        return (p, s[p]);
    return (p+1, s[p]);
}
```

It is important to understand that the above code relies upon byte-semantics, which are invalid for multi-byte characters.

See Also

4.9 (strbskipchar), 4.30 (strskipbytes)

4.32  strsub

Synopsis

Replace a character with another in a string.

Usage

```c
String_Type strsub (String_Type s, Int_Type pos, Int_Type ch)
```

Description

The `strsub` function may be used to substitute the character `ch` for the character at character position `pos` of the string `s`. The resulting string is returned.

Example

```c
define replace_spaces_with_comma (s)
{
    variable n;
    while (n = is_substr (s, " "), n) s = strsub (s, n, ',');
    return s;
}
```

For uses such as this, the `strtrans` function is a better choice.

Notes

The first character in the string `s` is specified by `pos` equal to 1. This function uses character semantics, not byte semantics.

See Also

4.6 (is_substr), 4.29 (strreplace), 4.24 (strlen)
4.33 `strtok`

Synopsis

Extract tokens from a string

Usage

```
String_Type[] strtok (String_Type str [,String_Type white])
```

Description

`strtok` breaks the string `str` into a series of tokens and returns them as an array of strings. If the second parameter `white` is present, then it specifies the set of characters that are to be regarded as whitespace when extracting the tokens, and may consist of the whitespace characters or a range of such characters. If the first character of `white` is `''`, then the whitespace characters consist of all characters except those in `white`. For example, if `white` is "\t\n,;". Then those characters specify the whitespace characters. However, if `white` is given by "^a-zA-Z0-9_", then any character is a whitespace character except those in the ranges a-z, A-Z, 0-9, and the underscore character. To specify the hyphen character as a whitespace character, then it should be the first character of the whitespace string. In addition to ranges, the whitespace specifier may also include character classes:

- \w matches a unicode "word" character, taken to be alphanumeric.
- \a alphabetic character, excluding digits
- \s matches whitespace
- \l matches lowercase
- \u matches uppercase
- \d matches a digit
- \^ matches a ^ character
- \ matches a backslash
- ^ matches a ^ character

If the second parameter is not present, then it defaults to "\s".

Example

The following example may be used to count the words in a text file:

```
define count_words (file)
{
    variable fp, line, count;

    fp = fopen (file, "r");
    if (fp == NULL) return -1;

    count = 0;
    while (-1 != fgets (&line, fp))
    {
        line = strtok (line, "^\a");
        count += length (line);
    }
    () = fclose (fp);
    return count;
}
```
Here a word was assumed to consist only of alphabetic characters.

See Also

4.16 (strchop), 4.19 (strcompress), 4.23 (strjoin)

4.34 strtrans

Synopsis

Replace characters in a string

Usage

String_Type strtrans (str, old_set, new_set)

String_Type str, old_set, new_set;

Description

The strtrans function may be used to replace all the characters from the set old_set with the corresponding characters from new_set in the string str. If new_set is empty, then the characters in old_set will be removed from str.

If new_set is not empty, then old_set and new_set must be commensurate. Each set may consist of character ranges such as A-Z and character classes:

\, matches a punctuation character
\7 matches any 7bit ascii character
\\ matches a backslash
\^ matches the ^ character
\a matches an alphabetic character, excluding digits
\c matches a control character
\d matches a digit
\g matches a graphic character
\l matches lowercase
\p matches a printable character
\s matches whitespace
\u matches uppercase
\w matches a unicode "word" character, taken to be alphanumeric.
\x matches hex digit (a-fA-F0-9)

If the first character of a set is ^ then the set is taken to be the complement set.

Example

str = strtrans (str, "\\u", "\\l"); % lower-case str
str = strtrans (str, "0-9", " "); % Replace anything but 0-9 by space
str = strtrans (str, "\\0-9", " "); % Replace ' ' and 0-9 by a space

Notes

This function has been vectorized in the sense that if an array of strings is passed as the first argument then a corresponding array of strings will be returned. Array values are not supported for the remaining arguments.
4.35  strtrim

Synopsis

Remove whitespace from the ends of a string

Usage

String_Type strtrim (String_Type s [,String_Type w])

Description

The `strtrim` function removes all leading and trailing whitespace characters from the string `s` and returns the result. The optional second parameter specifies the set of whitespace characters. If the argument is not present, then the set defaults to "\s". The whitespace specification may consist of character ranges such as A–Z and character classes:

\w matches a unicode "word" character, taken to be alphanumeric.
\a alphabetic character, excluding digits
\s matches whitespace
\l matches lowercase
\u matches uppercase
\d matches a digit
\^ matches a backslash
\^ matches a ^ character

If the first character of a set is ^ then the set is taken to be the complement set.

Notes

This function has been vectorized in the sense that if the first argument is an array of strings, then a corresponding array of strings will be returned. An array value for the optional whitespace argument is not supported.

See Also

4.36 (strtrim_beg), 4.37 (strtrim_end), 4.19 (strcompress)
4.37. strtrim_end

Description

The strtrim_beg function removes all leading whitespace characters from the string s and returns the result. The optional second parameter specifies the set of whitespace characters. See the documentation for the strtrim function for more information about the whitespace parameter.

Notes

This function has been vectorized in the sense that if the first argument is an array of strings, then a corresponding array of strings will be returned. An array value for the optional whitespace argument is not supported.

See Also

4.35 (strtrim), 4.37 (strtrim_end), 4.19 (strcompress)

4.37 strtrim_end

Synopsis

Remove trailing whitespace from a string

Usage

String_Type strtrim_end (String_Type s [,String_Type w])

Description

The strtrim_end function removes all trailing whitespace characters from the string s and returns the result. The optional second parameter specifies the set of whitespace characters. See the documentation for the strtrim function for more information about the whitespace parameter.

Notes

This function has been vectorized in the sense that if the first argument is an array of strings, then a corresponding array of strings will be returned. An array value for the optional whitespace argument is not supported.

See Also

4.35 (strtrim), 4.36 (strtrim_beg), 4.19 (strcompress)

4.38 strup

Synopsis

Convert a string to uppercase

Usage

String_Type strup (String_Type s)
Chapter 4. Functions that Operate on Strings

Description

The *strup* function takes a string `s` and returns another string identical to `s` except that all lower case characters that contained in `s` are converted to upper case.

Example

The function

```c
#define Strcmp (a, b)
{
    return strcmp (strup (a), strup (b));
}
```

performs a case-insensitive comparison operation of two strings by converting them to upper case first.

Notes

This function has been vectorized in the sense that if an array of strings is passed to the function, then a corresponding array of strings will be returned.

See Also

4.25 (strlwr), 12.14 (toupper), 4.18 (strcmp), 4.35 (strtrim), 12.6 (define_case), 4.34 (strtrans)

4.39 *str_delete_chars*

Synopsis

Delete characters from a string

Usage

```c
String_Type str_delete_chars (String_Type str [, String_Type del_set])
```

Description

This function may be used to delete the set of characters specified by the optional argument `del_set` from the string `str`. If `del_set` is not given, "\s" will be used. The modified string is returned.

The set of characters to be deleted may include ranges such as A-Z and characters classes:

- \w matches a unicode "word" character, taken to be alphanumeric.
- \a alphabetic character, excluding digits
- \s matches whitespace
- \l matches lowercase
- \u matches uppercase
- \d matches a digit
- \\ matches a backslash
- ^ matches a ^ character

If the first character of `del_set` is ^, then the set is taken to be the complement of the remaining string.

Example
str = str_delete_chars (str, "^A-Za-z");

will remove all characters except A-Z and a-z from str. Similarly,

str = str_delete_chars (str, "^\a");

will remove all but the alphabetic characters.

Notes
This function has been vectorized in the sense that if an array of strings is passed as the first argument then a corresponding array of strings will be returned. Array values are not supported for the remaining arguments.

See Also
4.34 (strtrans), 4.29 (strreplace), 4.19 (stcompress)

4.40 str_quote_string

Synopsis
Escape characters in a string.

Usage
String_Type str_quote_string(String_Type str, String_Type qlis, Int_Type quote)

Description
The str_quote_string returns a string identical to str except that all characters contained in the string qlis are escaped with the quote character, including the quote character itself. This function is useful for making a string that can be used in a regular expression.

Example
Execution of the statements

node = "Is it [the coat] really worth $100?";
tag = str_quote_string (node, "\^\$\[.\]*/?", '\');

will result in tag having the value:

Is it \[the coat\] really worth \$100\?

See Also
4.42 (str_uncomment_string), 4.7 (make_printable_string)

4.41 str_replace

Synopsis
Replace a substring of a string (deprecated)
Chapter 4. Functions that Operate on Strings

### 4.29 str_replace

**Usage**

```
Int_Type str_replace (String_Type a, String_Type b, String_Type c)
```

**Description**

The `str_replace` function replaces the first occurrence of `b` in `a` with `c` and returns an integer that indicates whether a replacement was made. If `b` does not occur in `a`, zero is returned. However, if `b` occurs in `a`, a non-zero integer is returned as well as the new string resulting from the replacement.

**Notes**

This function has been superseded by `strreplace`. It should no longer be used.

**See Also**

4.29 (strreplace)

### 4.42 str_uncomment_string

**Synopsis**

Remove comments from a string

**Usage**

```
String_Type str_uncomment_string(String_Type s, String_Type beg, String_Type end)
```

**Description**

This function may be used to remove simple forms of comments from a string `s`. The parameters, `beg` and `end`, are strings of equal length whose corresponding characters specify the begin and end comment characters, respectively. It returns the uncommented string.

**Example**

The expression

```
str_uncomment_string ("Hello (testing) 'example' World", "'(", "')")
```

returns the string "Hello World".

**Notes**

This routine does not handle multi-character comment delimiters and it assumes that comments are not nested.

**See Also**

4.40 (str_quote_string), 4.39 (str_delete_chars), 4.34 (strtrans)
### 4.43 substr

**Synopsis**

Extract a substring from a string

**Usage**

```c
String_Type substr (String_Type s, Int_Type n, Int_Type len)
```

**Description**

The `substr` function returns a substring with character length `len` of the string `s` beginning at the character position `n`. If `len` is -1, the entire length of the string `s` will be used for `len`. The first character of `s` is given by `n` equal to 1.

**Example**

```c
substr ("To be or not to be", 7, 5);
```

returns "or no"

**Notes**

This function assumes character semantics and not byte semantics. Use the `substrbytes` function to extract bytes from a string.

**See Also**

4.6 (is_substr), 4.44 (substrbytes), 4.24 (strlen)

### 4.44 substrbytes

**Synopsis**

Extract a byte sequence from a string

**Usage**

```c
String_Type substrbytes (String_Type s, Int_Type n, Int_Type len)
```

**Description**

The `substrbytes` function returns a substring with byte length `len` of the string `s` beginning at the byte position `n`, counting from 1. If `len` is -1, the entire byte-length of the string `s` will be used for `len`. The first byte of `s` is given by `n` equal to 1.

**Example**

```c
substrbytes ("To be or not to be", 7, 5);
```

returns "or no"

**Notes**

In many cases it is more convenient to use array indexing rather than the `substrbytes` function. In fact `substrbytes(s, i+1, -1)` is equivalent to `s[[i:]]`.

The function `substr` may be used if character semantics are desired.
See Also

4.43 (substr), 4.12 (strbytelen)
Chapter 5

Functions that Operate on Binary Strings

5.1 array_to_bstring

Synopsis
Convert an array to a binary string

Usage
BString_Type array_to_bstring (Array_Type a)

Description
The array_to_bstring function returns the elements of an array a as a binary string.

See Also
5.2 (bstring_to_array), 2.11 (init_char_array)

5.2 bstring_to_array

Synopsis
Convert a binary string to an array of bytes

Usage
UChar_Type[] bstring_to_array (BString_Type b)

Description
The bstring_to_array function returns an array of unsigned characters whose elements correspond to the bytes in the binary string.

See Also
5.1 (array_to_bstring), 2.11 (init_char_array)
5.3 bstrcat

Synopsis

Concatenate binary strings

Usage

String_Type bstrcat (BString_Type a_1, ..., BString_Type a_N)

Description

The bstrcat function concatenates its N binary string arguments a_1, ... a_N together and returns the result.

Notes

This function will produce a result that is identical to that of strcat if the input strings do not contain null characters.

See Also

4.14 (strcat), 5.4 (bstrjoin)

5.4 bstrjoin

Synopsis

Concatenate elements of an array of BString_Type objects

Usage

String_Type bstrjoin (Array_Type a [, BString_Type delim])

Description

The bstrjoin function operates on an array of binary strings by joining successive elements together separated with the optional delimiter delim. If delim is not specified, then empty string "" will be used resulting in a concatenation of the elements.

See Also

5.3 (bstrcat), 4.23 (strjoin)

5.5 bstrlen

Synopsis

Get the length of a binary string

Usage

UInt_Type bstrlen (BString_Type s)

Description

The bstrlen function may be used to obtain the length of a binary string. A binary string differs from an ordinary string (a C string) in that a binary string may include null characters.
5.6 count_byte_occurrences

Example

```c
s = "hello\0";
len = bstrlen (s);  // len = 6
len = strlen (s);   // len = 5
```

See Also

4.24 (strlen), 2.13 (length)

5.6 count_byte_occurrences

Synopsis

Count the number of occurrences of a byte in a binary string

Usage

```c
UInt_Type count_byte_occurrences (bstring, byte)
```

Description

This function returns the number of times the specified byte occurs in the binary string **bstr**.

Notes

This function uses byte-semantics. If character semantics are desired, use the `count_char_occurrences` function.

See Also

4.1 (count_char_occurrences)

5.7 is_substrbytes

Synopsis

test if a binary string contains a series of bytes

Usage

```c
Int_Type is_substrbytes (a, b [,ofs])
```

Description

This function may be used to see if the binary string **a** contains the byte-sequence given by the binary string **b**. If **b** is contained in **a**, then a ones-based offset of the first occurrence of **b** in **a** is returned. Otherwise, the function will return 0 to indicate that **a** does not contain **b**.

An optional 1-based parameter **ofs** may be passed to the function to indicate where in **a** the search is to start. The returned value is still a 1-based offset from the beginning of **a** where **b** is located.

Notes

Support for the optional argument was added in version 2.3.0.

See Also

4.6 (is_substr), 5.6 (count_byte_occurrences)
5.8 pack

Synopsis
    Pack objects into a binary string

Usage
    BString_Type pack (String_Type fmt, ...)

Description
    The pack function combines zero or more objects (represented by the ellipses above) into a binary string according to the format string fmt.

    The format string consists of one or more data-type specification characters defined by the following table:

    | Character | Data Type          |
    |-----------|--------------------|
    | c         | signed byte       |
    | C         | unsigned byte     |
    | h         | short             |
    | H         | unsigned short    |
    | i         | int               |
    | I         | unsigned int      |
    | l         | long              |
    | L         | unsigned long     |
    | m         | long long         |
    | M         | unsigned long long|
    | j         | 16 bit int        |
    | J         | 16 bit unsigned int|
    | k         | 32 bit int        |
    | K         | 32 bit unsigned int|
    | q         | 64 bit int        |
    | Q         | 64 bit unsigned int|
    | f         | float             |
    | d         | double            |
    | F         | 32 bit float      |
    | D         | 64 bit float      |
    | s         | character string, null padded |
    | S         | character string, space padded |
    | z         | character string, null padded |
    | x         | a null pad character |

    A decimal length specifier may follow the data-type specifier. With the exception of the s and S specifiers, the length specifier indicates how many objects of that data type are to be packed or unpacked from the string. When used with the s, S, or z specifiers, it indicates the field width to be used. If the length specifier is not present, the length defaults to one.

    When packing, unlike the s specifier, the z specifier guarantees that at least one null byte will be written even if the field has to be truncated to do so.

    With the exception of c, C, s, S, and x, each of these may be prefixed by a character that indicates the byte-order of the object:

        >        big-endian order (network order)
< little-endian order
= native byte-order

The default is to use native byte order.

When unpacking via the `unpack` function, if the length specifier is greater than one, then an array of that length will be returned. In addition, trailing whitespace and null characters are stripped when unpacking an object given by the `S` specifier. Trailing null characters will be stripped from an object represented by the `z` specifier. No such stripping is performed by the `s` specifier.

Example

```c
a = pack ("cc", 'A', 'B'); % ==> a = "AB";
a = pack ("c2", 'A', 'B'); % ==> a = "AB";
a = pack ("xxccc", 'A', 'B'); % ==> a = "\0\0A\0\0B";
a = pack ("h2", 'A', 'B'); % ==> a = "\0A\0B" or "\0B\0A"
a = pack (">h2", 'A', 'B'); % ==> a = "\0xA\0xB"
a = pack ("<h2", 'A', 'B'); % ==> a = "\0B\0A"
a = pack ("s4", "AB", "CD"); % ==> a = "AB\0\0"
a = pack ("s4s2", "AB", "CD"); % ==> a = "AB\0\0CD"
a = pack ("S4", "AB", "CD"); % ==> a = "AB\0"
a = pack ("S4S2", "AB", "CD"); % ==> a = "AB CD"
a = pack ("z4", "AB"); % ==> a = "AB\0\0"
a = pack ("z4", "ABCDEFG"); % ==> a = "ABCDEFG"
a = pack ("z4", "ABCDEFG"); % ==> a = "ABC0"
```

See Also

5.11 (unpack), 5.10 (sizeof_pack), 5.9 (pad_pack_format), 4.10 (sprintf)

5.9 pad_pack_format

Synopsis

Add padding to a pack format

Usage

```c
BString_Type pad_pack_format (String_Type fmt)
```

Description

The `pad_pack_format` function may be used to add the appropriate padding characters to the format `fmt` such that the data types specified by the format will be properly aligned on word boundaries. This is especially important when reading or writing files that assume the native alignment.

See Also

5.8 (pack), 5.11 (unpack), 5.10 (sizeof_pack)
5.10 sizeof_pack

Synopsis

Compute the size implied by a pack format string

Usage

UInt_Type sizeof_pack (String_Type fmt)

Description

The sizeof_pack function returns the size of the binary string represented by the format string fmt. This information may be needed when reading a structure from a file.

See Also

5.8 (pack), 5.11 (unpack), 5.9 (pad_pack_format)

5.11 unpack

Synopsis

Unpack Objects from a Binary String

Usage

(... = unpack (String_Type fmt, BString_Type s)

Description

The unpack function unpacks objects from a binary string s according to the format fmt and returns the objects to the stack in the order in which they were unpacked. See the documentation of the pack function for details about the format string.

Example

(x,y) = unpack ("cc", "AB"); \% \Rightarrow x = 'A', y = 'B'
x = unpack ("c2", "AB"); \% \Rightarrow x = ['A', 'B']
x = unpack ("x<H", "0\xAB\xCD"); \% \Rightarrow x = 0xCDABuh
x = unpack ("xxs4", "a b c\0d e f"); \% \Rightarrow x = "b c\0"
x = unpack ("xxS4", "a b c\0d e f"); \% \Rightarrow x = "b c"

See Also

5.8 (pack), 5.10 (sizeof_pack), 5.9 (pad_pack_format)
Chapter 6

Functions that Manipulate Structures

6.1 __add_binary

Synopsis

Extend a binary operation to a user defined type

Usage

__add_binary(op, return_type, binary_funct, lhs_type, rhs_type)

    String_Type op;
    Ref_Type binary_funct;
    DataType_Type return_type, lhs_type, rhs_type;

Description

The __add_binary function is used to specify a function to be called when a binary operation takes place between specified data types. The first parameter indicates the binary operator and must be one of the following:

    "or", "and", "&", "|", "xor", "shl", "shr", "mod"

The second parameter (binary_funct) specifies the function to be called when the binary function takes place between the types lhs_type and rhs_type. The return_type parameter stipulates the return values of the function and the data type of the result of the binary operation.

The data type for lhs_type or rhs_type may be left unspecified by using Any_Type for either of these values. However, at least one of the parameters must correspond to a user-defined datatype.

Example

This example defines a vector data type and extends the "*" operator to the new type:

typedef struct { x, y, z } Vector_Type;
define vector (x, y, z)
variable v = @Vector_Type;
v.x = x;
v.y = y;
v.z = z;
return v;
}
static define vector_scalar_mul (v, a)
{
    return vector (a*v.x, a*v.y, a*v.z);
}
static define scalar_vector_mul (a, v)
{
    return vector_scalar_mul (v, a);
}
static define dotprod (v1,v2)
{
    return v1.x*v2.x + v1.y*v2.y + v1.z*v2.z;
}
__add_binary ("*", Vector_Type, &scalar_vector_mul, Any_Type, Vector_Type);
__add_binary ("*", Vector_Type, &scalar_vector_mul, Any_Type, Vector_Type);
__add_binary ("*", Double_Type, &dotprod, Vector_Type, Vector_Type);

See Also
6.4 (__add_unary), 6.2 (__add_string), ?? (__add_destroy)

6.2 __add_string

Synopsis
Specify a string representation for a user-defined type

Usage
__add_string (DataType_Type user_type, Ref_Type func)

Description
The __add_string function specifies a function to be called when a string representation is required for the specified user-defined datatype.

Example
Consider the Vector_Type object defined in the example for the __add_binary function.

static define vector_string (v)
{
    return sprintf ("[%S,%S,%S]", v.x, v.y, v.z);
}
__add_string (Vector_Type, &vector_string);

Then
v = vector (3, 4, 5);
vmessage ("v=%S", v);
6.3  __add_typecast

will generate the message:

\[ v = [3, 4, 5] \]

See Also

6.4 (__add_unary), 6.1 (__add_binary), ?? (__add_destroy), 6.3 (__add_typecast)

6.3  __add_typecast

Synopsis

Add a typecast-function for a user-defined type

Usage

__add_typecast (DataType_Type user_type, DataType_Type totype, Ref_Type func)

Description

The __add_typecast function specifies a function to be called to typecast the user-defined type to an object of type totype. The function must be defined to take a single argument (the user-type to be converted) and must return an object of type totype.

See Also

6.4 (__add_unary), 6.1 (__add_binary), ?? (__add_destroy), 6.2 (__add_string)

6.4  __add_unary

Synopsis

Extend a unary operator to a user-defined type

Usage

__add_unary (op, return_type, unary_func, user_type)

\[
\begin{align*}
\text{String_Type} & \quad \text{op}; \\
\text{Ref_Type} & \quad \text{unary_func}; \\
\text{DataType_Type} & \quad \text{return_type}, \text{user_type};
\end{align*}
\]

Description

The __add_unary function is used to define the action of an unary operation on a user-defined type. The first parameter op must be a valid unary operator

"-", "not", "-"

or one of the following:

"++", "--",    

The third parameter, unary_func specifies the function to be called to carry out the specified unary operation on the data type user_type. The result of the operation is indicated by the value of the return_type parameter and must also be the return type of the unary function.
Example

The example for the __add_binary function defined a Vector_Type object. Here, the unary "-" and "abs" operators are extended to this type:

```c
static define vector_chs (v)
{
    variable v1 = @Vector_Type;
    v1.x = -v.x;
    v1.y = -v.y;
    v1.z = -v.z;
    return v1;
}
static define vector_abs (v)
{
    return sqrt (v.x*v.x + v.y*v.y + v.z*v.z);
}
__add_unary ("-", Vector_Type, &vector_chs, Vector_Type);
__add_unary ("abs", Double_Type, &vector_abs, Vector_Type);
```

See Also

6.1 (__add_binary), 6.2 (__add_string), ?? (__add_destroy)

6.5 get_struct_field

Synopsis

Get the value associated with a structure field

Usage

```c
x = get_struct_field (Struct_Type s, String field_name)
```

Description

The get_struct_field function gets the value of the field whose name is specified by field_name of the structure s. If the specified name is not a field of the structure, the function will throw an InvalidParmError exception.

See Also

6.10 (set_struct_field), 6.6 (get_struct_field_names), 2.3 (array_info)

6.6 get_struct_field_names

Synopsis

Retrieve the field names associated with a structure

Usage

```c
String_Type[] = get_struct_field_names (Struct_Type s)
```
6.7. _is_struct_type

Description

The get_struct_field_names function returns an array of strings whose elements specify the names of the fields of the struct s.

Example

The following example illustrates how the get_struct_field_names function may be used in conjunction with the get_struct_field function to print the value of a structure.

```c
define print_struct (s)
{
    variable name, value;

    foreach (get_struct_field_names (s))
    {
        name = ();
        value = get_struct_field (s, name);
        vmessage ("s.%s = %s\n", name, string (value));
    }
}
```

See Also

6.9 (_push_struct_field_values), 6.5 (get_struct_field)

6.7 _is_struct_type

Synopsis

Determine whether or not an object is a structure

Usage

Integer_Type _is_struct_type (X)

Description

The _is_struct_type function returns 1 if the parameter refers to a structure or a user-defined type, or to an array of structures or user-defined types. If the object is neither, 0 will be returned.

See Also

12.17 (typeof), 12.16 (_typeof), 6.8 (is_struct_type)

6.8 is_struct_type

Synopsis

Determine whether or not an object is a structure

Usage

Integer_Type is_struct_type (X)
Description

The `is_struct_type` function returns 1 if the parameter refers to a structure or a user-defined type. If the object is neither, 0 will be returned.

See Also

12.17 (typeof), 12.16 (_typeof), 6.7 (_is_struct_type)

6.9 _push_struct_field_values

Synopsis

Push the values of a structure’s fields onto the stack

Usage

`Integer_Type num = _push_struct_field_values (Struct_Type s)`

Description

The `_push_struct_field_values` function pushes the values of all the fields of a structure onto the stack, returning the number of items pushed. The fields are pushed such that the last field of the structure is pushed first.

See Also

6.6 (get_struct_field_names), 6.5 (get_struct_field)

6.10 set_struct_field

Synopsis

Set the value associated with a structure field

Usage

`set_struct_field (s, field_name, field_value)`

```c
Struct_Type s;
String_Type field_name;
Generic_Type field_value;
```

Description

The `set_struct_field` function sets the value of the field whose name is specified by `field_name` of the structure `s` to `field_value`.

See Also

6.5 (get_struct_field), 6.6 (get_struct_field_names), 6.11 (set_struct_fields), 2.3 (array_info)
6.11  set_struct_fields

Synopsis
Set the fields of a structure

Usage
set_struct_fields (Struct_Type s, ...)

Description
The set_struct_fields function may be used to set zero or more fields of a structure. The fields are set in the order in which they were created when the structure was defined.

Example

    variable s = struct { name, age, height };
    set_struct_fields (s, "Bill", 13, 64);

See Also
6.10 (set_struct_field), 6.6 (get_struct_field_names)
Chapter 7

Functions that Create and Manipulate Lists

7.1 list_append

Synopsis
Append an object to a list

Usage
list_append (List_Type list, object [,Int_Type nth])

Description
The list_append function is like list_insert except this function appends the object to the list. The optional argument nth may be used to specify where the object is to be appended. See the documentation on list_insert for more details.

See Also
7.2 (list_concat), 7.4 (list_insert), 7.5 (list_join), 7.3 (list_delete), 7.7 (list_pop), 7.6 (list_new), 7.8 (list_reverse)

7.2 list_concat

Synopsis
Concatenate two lists to form a third

Usage
List_Type = list_concat (List_Type a, List_Type b)

Description
This function creates a new list that is formed by concatenating the two lists a and b together. Neither of the input lists are modified by this operation.
7.3 list_delete

Synopsis
Remove an item from a list

Usage
list_delete (List_Type list, Int_Type nth)

Description
This function removes the nth item in the specified list. The first item in the list corresponds
to a value of nth equal to zero. If nth is negative, then the indexing is with respect to the end
of the list with the last item corresponding to nth equal to -1.

See Also
7.4 (list_insert), 7.1 (list_append), 7.7 (list_pop), 7.6 (list_new), 7.8 (list_reverse)

7.4 list_insert

Synopsis
Insert an item into a list

Usage
list_insert (List_Type list, object [,Int_Type nth])

Description
This function may be used to insert an object into the specified list. With just two arguments,
the object will be inserted at the beginning of the list. The optional third argument, nth, may
be used to specify the insertion point. The first item in the list corresponds to a value of nth
equal to zero. If nth is negative, then the indexing is with respect to the end of the list with
the last item given by a value of nth equal to -1.

Notes
It is important to note that

    list_insert (list, object, 0);

is not the same as

    list = {object, list}

since the latter creates a new list with two items, object and the old list.

See Also
7.1 (list_append), 7.7 (list_pop), 7.3 (list_delete), 7.6 (list_new), 7.8 (list_reverse)
7.5  list_join

Synopsis
  Join the elements of a second list onto the end of the first

Usage
  list_join (List_Type a, List_Type b)

Description
  This function modifies the list a by appending the elements of b to it.

See Also
  7.2 (list_concat), 7.1 (list_append), 7.4 (list_insert)

7.6  list_new

Synopsis
  Create a new list

Usage
  List_Type list_new ()

Description
  This function creates a new empty List_Type object. Such a list may also be created using
  the syntax
    list = {};

See Also
  7.3 (list_delete), 7.4 (list_insert), 7.1 (list_append), 7.8 (list_reverse), 7.7 (list_pop)

7.7  list_pop

Synopsis
  Extract an item from a list

Usage
  object = list_pop (List_Type list [, Int_Type nth])

Description
  The list_pop function returns an object from a list deleting the item from the list in the
  process. If the second argument is present, then it may be used to specify the position in the
  list where the item is to be obtained. If called with a single argument, the first item in the list
  will be used.

See Also
  7.3 (list_delete), 7.4 (list_insert), 7.1 (list_append), 7.8 (list_reverse), 7.6 (list_new)
7.8 list_reverse

Synopsis
Reverse a list

Usage
list_reverse (List_Type list)

Description
This function may be used to reverse the items in list.

Notes
This function does not create a new list. The list passed to the function will be reversed upon
return from the function. If it is desired to create a separate reversed list, then a separate copy
should be made, e.g.,

```
rev_list = @list;
list_reverse (rev_list);
```

See Also
7.6 (list_new), 7.4 (list_insert), 7.1 (list_append), 7.3 (list_delete), 7.7 (list_pop)

7.9 list_to_array

Synopsis
Convert a list into an array

Usage
Array_Type list_to_array (List_Type list [,DataType_Type type])

Description
The list_to_array function converts a list of objects into an array of the same length and
returns the result. The optional argument may be used to specify the array’s data type. If no
type is given, list_to_array tries to find the common data type of all list elements. This
function will generate an exception if the list is empty and no type has been specified, or the
objects in the list cannot be converted to a common type.

Notes
A future version of this function may produce an Any_Type array for an empty or heterogeneous
list.

See Also
2.13 (length), 12.15 (typecast), 23.5 (__pop_list), 12.17 (typeof), 2.7 (array_sort)
Chapter 8

Informational Functions

8.1 add_doc_file

Synopsis

Make a documentation file known to the help system

Usage

add_doc_file (String_Type file)

Description

The add_doc_file is used to add a documentation file to the system. Such files are searched by the get_doc_string_from_file function. The file must be specified using the full path.

See Also

8.12 (set_doc_files), 8.6 (get_doc_files), 8.7 (get_doc_string_from_file)

8.2 _apropos

Synopsis

Generate a list of functions and variables

Usage

Array_Type _apropos (String_Type ns, String_Type s, Integer_Type flags)

Description

The _apropos function may be used to get a list of all defined objects in the namespace ns whose name matches the regular expression s and whose type matches those specified by flags. It returns an array of strings containing the names matched.

The third parameter flags is a bit mapped value whose bits are defined according to the following table
## Example

```plaintext
define apropos (s)
{
    variable n, name, a;
    a = _apropos ("Global", s, 0xF);

    vmmessage ("Found %d matches:", length (a));
    foreach name (a)
        message (name);
}
```

prints a list of all matches.

### Notes

If the namespace specifier `ns` is the empty string "", then the namespace will default to the static namespace of the current compilation unit.

### See Also

8.9 (is_defined), 4.10 (sprintf), 8.8 (_get_namespaces)

### 8.3 __FILE__

#### Synopsis

Path of the compilation unit

#### Usage

```plaintext
String_Type __FILE__
```

#### Description

Every private namespace has `__FILE__` variable associated with it. If the namespace is associated with a file, then the value of this variable will be equal to the pathname of the file. If the namespace is associated with a string, such as one passed to the `eval` function, then the value of this variable will be "***string***";

#### Notes

In the case of a file, the pathname may be an absolute path or a relative one. If it is a relative one, then it will be relative to the directory from where the file was loaded, i.e., the value returned by the `getcwd` function.
8.4 __function_name

Synopsis
Returns the name of the currently executing function

Usage
String_Type __function_name ()

Description
This function returns the name of the currently executing function. If called from top-level, it
returns the empty string.

See Also
22.12 (_trace_function), 8.9 (is_defined)

8.5 __get_defined_symbols

Synopsis
Get the symbols defined by the preprocessor

Usage
Int_Type __get_defined_symbols ()

Description
The __get_defined_symbols functions is used to get the list of all the symbols defined by
the S-Lang preprocessor. It pushes each of the symbols on the stack followed by the number
of items pushed.

See Also
8.9 (is_defined), 8.2 (_apropos), 8.8 (_get_namespaces)

8.6 get_doc_files

Synopsis
Get the list of documentation files

Usage
String_Type[] = get_doc_files ()

Description
The get_doc_files function returns the internal list of documentation files as an array of
strings.

See Also
8.12 (set_doc_files), 8.1 (add_doc_file), 8.7 (get_doc_string_from_file)
8.7 get_doc_string_from_file

Synopsis
Read documentation from a file

Usage
String_Type get_doc_string_from_file ([String_Type f,] String_Type t)

Description
If called with two arguments, get_doc_string_from_file opens the documentation file f and searches it for topic t. Otherwise, it will search an internal list of documentation files looking for the documentation associated with the topic t. If found, the documentation for t will be returned, otherwise the function will return NULL.

Files may be added to the internal list via the add_doc_file or set_doc_files functions.

See Also
8.1 (add_doc_file), 8.12 (set_doc_files), 8.6 (get_doc_files), 8.13 (slang_doc_dir)

8.8 _get_namespaces

Synopsis
Returns a list of namespace names

Usage
String_Type [] _get_namespaces ()

Description
This function returns a string array containing the names of the currently defined namespaces.

See Also
8.2 (_apropos), 25.21 (use_namespace), 25.10 (implements), 8.5 (_get_defined_symbols)

8.9 is_defined

Synopsis
Determine if a variable or function is defined

Usage
Integer_Type is_defined (String_Type name)

Description
This function is used to determine whether or not a function or variable of the given name has been defined. If the specified name has not been defined, the function returns 0. Otherwise, it returns a non-zero value that depends on the type of object attached to the name. Specifically, it returns one of the following values:
Example

Consider the function:

```lisp
define runhooks (hook)
{
  if (2 == is_defined(hook)) eval(hook);
}
```

This function could be called from another S-Lang function to allow customization of that function, e.g., if the function represents a mode, the hook could be called to setup keybindings for the mode.

See Also
12.17 (typeof), 19.4 (eval), 19.2 (autoload), 25.9 (__get_reference), 8.10 (__is_initialized)

8.10 __is_initialized

Synopsis

Determine whether or not a variable has a value

Usage

```lisp
Integer_Type __is_initialized (Ref_Type r)
```

Description

This function returns non-zero of the object referenced by r is initialized, i.e., whether it has a value. It returns 0 if the referenced object has not been initialized.

Example

The function:

```lisp
define zero ()
{
  variable f;
  return __is_initialized (&f);
}
```

will always return zero, but

```lisp
define one ()
{
  variable f = 0;
  return __is_initialized (&f);
}
```

will return one.
8.11 _NARGS

Synopsis

The number of parameters passed to a function

Usage

Integer_Type _NARGS The value of the _NARGS variable represents the number of arguments passed to the function. This variable is local to each function.

Example

This example uses the _NARGS variable to print the list of values passed to the function:

```slsh
define print_values ()
{
  variable arg;

  if (_NARGS == 0)
  {
    message ("Nothing to print");
    return;
  }
  foreach arg (__pop_args (_NARGS))
  {
    vmessage ("Argument value is: %s", arg.value);
  }
}
```

See Also

23.4 (__pop_args), 23.8 (__push_args), 12.17 (typeof)

8.12 set_doc_files

Synopsis

Set the internal list of documentation files

Usage

set_doc_files (String_Type[] list)

Description

The set_doc_files function may be used to set the internal list of documentation files. It takes a single parameter, which is required to be an array of strings. The internal file list is set to the files specified by the elements of the array.

Example

The following example shows how to add all the files in a specified directory to the internal list. It makes use of the glob function that is distributed as part of slsh.
8.13  _slang_doc_dir

Synopsis

Installed documentation directory

Usage

String_Type _slang_doc_dir

Description

The _slang_doc_dir variable is a read-only variable that specifies the compile-time installation location of the S-Lang documentation.

See Also

8.6 (get_doc_files), 8.1 (add_doc_file), 8.7 (get_doc_string_from_file)

8.14  _slang_version

Synopsis

The S-Lang library version number

Usage

Integer_Type _slang_version

Description

_slang_version is a read-only variable that gives the version number of the S-Lang library.

See Also

8.15 (_slang_version_string)

8.15  _slang_version_string

Synopsis

The S-Lang library version number as a string

Usage

String_Type _slang_version_string
Description

_slang_version_string is a read-only variable that gives a string representation of the version number of the S-Lang library.

See Also

8.14 (_slang_version)
Chapter 9

Mathematical Functions

9.1 abs

Synopsis
Compute the absolute value of a number

Usage
\[ y = \text{abs}(x) \]

Description
The abs function returns the absolute value of an arithmetic type. If its argument is a complex number (Complex_Type), then it returns the modulus. If the argument is an array, a new array will be created whose elements are obtained from the original array by using the abs function.

See Also
9.40 (sign), 9.44 (sqr)

9.2 acos

Synopsis
Compute the arc-cosine of a number

Usage
\[ y = \text{acos}(x) \]

Description
The acos function computes the arc-cosine of a number and returns the result. If its argument is an array, the acos function will be applied to each element and the result returned as an array.

See Also
9.11 (cos), 9.6 (atan), 9.3 (acosh), 9.12 (cosh)
9.3  acosh

Synopsis
   Compute the inverse cosh of a number

Usage
   \( y = \text{acosh} \, (x) \)

Description
   The \text{acosh} function computes the inverse hyperbolic cosine of a number and returns the result. If its argument is an array, the \text{acosh} function will be applied to each element and the result returned as an array.

See Also
   9.11 (cos), 9.6 (atan), 9.3 (acosh), 9.12 (cosh)

9.4  asin

Synopsis
   Compute the arc-sine of a number

Usage
   \( y = \text{asin} \, (x) \)

Description
   The \text{asin} function computes the arc-sine of a number and returns the result. If its argument is an array, the \text{asin} function will be applied to each element and the result returned as an array.

See Also
   9.11 (cos), 9.6 (atan), 9.3 (acosh), 9.12 (cosh)

9.5  asinh

Synopsis
   Compute the inverse-sinh of a number

Usage
   \( y = \text{asinh} \, (x) \)

Description
   The \text{asinh} function computes the inverse hyperbolic sine of a number and returns the result. If its argument is an array, the \text{asinh} function will be applied to each element and the result returned as an array.

See Also
   9.11 (cos), 9.6 (atan), 9.3 (acosh), 9.12 (cosh)
9.6 atan

Synopsis
Compute the arc-tangent of a number

Usage
\[ y = \text{atan} \ (x) \]

Description
The \text{atan} function computes the arc-tangent of a number and returns the result. If its argument is an array, the \text{atan} function will be applied to each element and the result returned as an array.

See Also
9.7 (atan2), 9.11 (cos), 9.3 (acosh), 9.12 (cosh)

9.7 atan2

Synopsis
Compute the arc-tangent of the ratio of two variables

Usage
\[ z = \text{atan2} \ (y, \ x) \]

Description
The \text{atan2} function computes the arc-tangent of the ratio \( y/x \) and returns the result as a value that has the proper sign for the quadrant where the point \( (x,y) \) is located. The returned value \( z \) will satisfy \((-\pi < z <= \pi)\). If either of the arguments is an array, an array of the corresponding values will be returned.

See Also
9.22 (hypot), 9.11 (cos), 9.6 (atan), 9.3 (acosh), 9.12 (cosh)

9.8 atanh

Synopsis
Compute the inverse-tanh of a number

Usage
\[ y = \text{atanh} \ (x) \]

Description
The \text{atanh} function computes the inverse hyperbolic tangent of a number and returns the result. If its argument is an array, the \text{atanh} function will be applied to each element and the result returned as an array.
9.9 ceil

Synopsis
Round x up to the nearest integral value

Usage
\[ y = \text{ceil} \ (x) \]

Description
This function rounds its numeric argument up to the nearest integral value. If the argument is an array, the corresponding array will be returned.

See Also
9.18 (floor), 9.38 (round)

9.10 Conj

Synopsis
Compute the complex conjugate of a number

Usage
\[ z1 = \text{Conj} \ (z) \]

Description
The \text{Conj} function returns the complex conjugate of a number. If its argument is an array, the \text{Conj} function will be applied to each element and the result returned as an array.

See Also
9.37 (Real), 9.23 (Imag), 9.1 (abs)

9.11 cos

Synopsis
Compute the cosine of a number

Usage
\[ y = \cos \ (x) \]

Description
The \text{cos} function computes the cosine of a number and returns the result. If its argument is an array, the \text{cos} function will be applied to each element and the result returned as an array.
9.12  \textit{cosh}

Synopsis

Compute the hyperbolic cosine of a number

Usage

\[ y = \text{cosh} \ (x) \]

Description

The \texttt{cosh} function computes the hyperbolic cosine of a number and returns the result. If its argument is an array, the \texttt{cosh} function will be applied to each element and the result returned as an array.

See Also

9.11 (cos), 9.6 (atan), 9.3 (acosh), 9.12 (cosh), 9.42 (sincos)

9.13  \textit{diff}

Synopsis

Compute the absolute difference of two values

Usage

\[ y = \text{diff} \ (x, y) \]

Description

The \texttt{diff} function returns a floating point number equal to the absolute value of the difference of its two arguments. If either argument is an array, an array of the corresponding values will be returned.

See Also

9.1 (abs)

9.14  \textit{exp}

Synopsis

Compute the exponential of a number

Usage

\[ y = \exp \ (x) \]
Description

The \texttt{exp} function computes the exponential of a number and returns the result. If its argument is an array, the \texttt{exp} function will be applied to each element and the result returned as an array.

See Also

9.15 (expm1), 9.11 (cos), 9.6 (atan), 9.3 (acosh), 9.12 (cosh)

9.15 expm1

Synopsis

Compute \( \exp(x)-1 \)

Usage

\[ y = \text{expm1}(x) \]

Description

The \texttt{expm1} function computes \( \exp(x)-1 \) and returns the result. If its argument is an array, the \texttt{expm1} function will be applied to each element and the results returned as an array.

This function should be called whenever \( x \) is close to 0 to avoid the numerical error that would arise in a naive computation of \( \exp(x)-1 \).

See Also

9.15 (expm1), 9.31 (log1p), 9.11 (cos), 9.6 (atan), 9.3 (acosh), 9.12 (cosh)

9.16 feqs

Synopsis

Test the approximate equality of two numbers

Usage

\[ \text{Char\_Type \hspace{1mm} feqs \hspace{1mm} (a, b \hspace{1mm} [,reldiff \hspace{1mm} [,absdiff]])} \]

Description

This function compares two floating point numbers \( a \) and \( b \), and returns a non-zero value if they are equal to within a specified tolerance; otherwise 0 will be returned. If either is an array, a corresponding boolean array will be returned.

The tolerances are specified as relative and absolute differences via the optional third and fourth arguments. If no optional arguments are present, the tolerances default to \( \text{reldiff}=0.01 \) and \( \text{absdiff}=1e-6 \). If only the relative difference has been specified, the absolute difference \( (\text{absdiff}) \) will be taken to be 0.0.

For the case when \( |b| \geq |a| \), \( a \) and \( b \) are considered to be equal to within the specified tolerances if either \( |b-a| \leq \text{absdiff} \) or \( |b-a|/|b| \leq \text{reldiff} \) is true.

See Also

9.20 (fneqs), 9.17 (fgteqs), 9.19 (flteqs)
9.17 fgteqs

Synopsis

Compare two numbers using specified tolerances.

Usage

Char_Type fgteqs (a, b [,reldiff [,absdiff]])

Description

This function is functionally equivalent to:

(a >= b) or feqs(a,b,...)

See the documentation of feqs for more information.

See Also

9.16 (feqs), 9.20 (fneqs), 9.17 (fgteqs)

9.18 floor

Synopsis

Round x down to the nearest integer

Usage

y = floor (x)

Description

This function rounds its numeric argument down to the nearest integral value. If the argument
is an array, the corresponding array will be returned.

See Also

9.9 (ceil), 9.38 (round), 9.35 (nint)

9.19 flteqs

Synopsis

Compare two numbers using specified tolerances.

Usage

Char_Type flteqs (a, b [,reldiff [,absdiff]])

Description

This function is functionally equivalent to:

(a <= b) or feqs(a,b,...)

See the documentation of feqs for more information.

See Also

9.16 (feqs), 9.20 (fneqs), 9.17 (fgteqs)
9.20  fneqs

Synopsis
    Test the approximate inequality of two numbers

Usage
    Char_Type fneqs (a, b [,reldiff [,absdiff]])

Description
    This function is functionally equivalent to:
        not fneqs(a,b,...)

See Also
    9.16 (feqs), 9.17 (fgteqs), 9.19 (flteqs)

9.21  get_float_format

Synopsis
    Get the format for printing floating point values.

Usage
    String_Type get_float_format ()

Description
    The get_float_format retrieves the format string used for printing single and double precision
    floating point numbers. See the documentation for the set_float_format function for more
    information about the format.

See Also
    9.39 (set_float_format)

9.22  hypot

Synopsis
    Compute sqrt(x1^2+x2^2+...+xN^2)

Usage
    r = hypot (x1 [,x2,...,xN])

Description
    If given two or more arguments, x1,...,xN, the hypot function computes the quantity
    sqrt(x1^2+...+xN^2) using an algorithm that tries to avoid arithmetic overflow. If any
    of the arguments is an array, an array of the corresponding values will be returned.

    If given a single array argument x, the hypot function computes sqrt(sumsq(x)), where
    sumsq(x) computes the sum of the squares of the elements of x.
Example

A vector in Euclidean 3 dimensional space may be represented by an array of three values representing the components of the vector in some orthogonal cartesian coordinate system. Then the length of the vector may be computed using the \texttt{hypot} function, e.g.,

\begin{verbatim}
A = [2,3,4];
len_A = hypot (A);
\end{verbatim}

The dot-product or scalar-product between two such vectors A and B may be computed using the \texttt{sum(A*B)}. It is well known that this is also equal to the product of the lengths of the two vectors and the cosine of the angle between them. Hence, the angle between the vectors A and B may be computed using

\begin{verbatim}
ahat = A/hypot(A);
bhat = B/hypot(B);
theta = acos(sum(ahat*bhat));
\end{verbatim}

Here, \texttt{ahat} and \texttt{bhat} are the unit vectors associated with the vectors A and B, respectively. Unfortunately, the above method for computing the angle between the vectors is numerically unstable when A and B are nearly parallel. An alternative method is to use:

\begin{verbatim}
ahat = A/hypot(A);
bhat = B/hypot(B);
ab = sum(ahat*bhat);
theta = atan2(hypot(bhat - ab*ahat), ab);
\end{verbatim}

See Also

9.7 (atan2), 9.11 (cos), 9.6 (atan), 9.3 (acosh), 9.12 (cosh), 2.21 (sum), 2.22 (sumsq)

9.23 Imag

Synopsis

Compute the imaginary part of a number

Usage

\begin{verbatim}
i = Imag (z)
\end{verbatim}

Description

The \texttt{Imag} function returns the imaginary part of a number. If its argument is an array, the \texttt{Imag} function will be applied to each element and the result returned as an array.

See Also

9.37 (Real), 9.10 (Conj), 9.1 (abs)

9.24 isinf

Synopsis

Test for infinity
Chapter 9. Mathematical Functions

Usage

\[ y = \text{isinf}(x) \]

Description

This function returns 1 if \( x \) corresponds to an IEEE infinity, or 0 otherwise. If the argument is an array, an array of the corresponding values will be returned.

See Also

9.25 (isnan), ?? (_Inf)

9.25 isnan

Synopsis

isnan

Usage

\[ y = \text{isnan}(x) \]

Description

This function returns 1 if \( x \) corresponds to an IEEE NaN (Not a Number), or 0 otherwise. If the argument is an array, an array of the corresponding values will be returned.

See Also

9.24 (isinf), ?? (_NaN)

9.26 _isneg

Synopsis

Test if a number is less than 0

Usage

Char_Type _isneg(x)

Description

This function returns 1 if a number is less than 0, and zero otherwise. If the argument is an array, then the corresponding array of boolean (Char_Type) values will be returned.

See Also

9.28 (_ispos), 9.27 (_isnonneg)
9.27 _isnonneg

Synopsis
Test if a number is greater than or equal to 0

Usage
Char_Type _isnonneg(x)

Description
This function returns 1 if a number is greater than or equal to 0, and zero otherwise. If the argument is an array, then the corresponding array of boolean (Char_Type) values will be returned.

See Also
9.26 (_isneg), 9.28 (_ispos)

9.28 _ispos

Synopsis
Test if a number is greater than 0

Usage
Char_Type _ispos(x)

Description
This function returns 1 if a number is greater than 0, and zero otherwise. If the argument is an array, then the corresponding array of boolean (Char_Type) values will be returned.

See Also
9.26 (_isneg), 9.27 (_isnonneg)

9.29 log

Synopsis
Compute the logarithm of a number

Usage
y = log (x)

Description
The log function computes the natural logarithm of a number and returns the result. If its argument is an array, the log function will be applied to each element and the result returned as an array.

See Also
9.11 (cos), 9.6 (atan), 9.3 (acosh), 9.12 (cosh), 9.31 (log1p)
9.30 log10

Synopsis
Compute the base-10 logarithm of a number

Usage
\[ y = \log_{10}(x) \]

Description
The log10 function computes the base-10 logarithm of a number and returns the result. If its argument is an array, the log10 function will be applied to each element and the result returned as an array.

See Also
9.11 (cos), 9.6 (atan), 9.3 (acosh), 9.12 (cosh)

9.31 log1p

Synopsis
Compute the logarithm of 1 plus a number

Usage
\[ y = \log_{1p}(x) \]

Description
The log1p function computes the natural logarithm of 1.0 plus \( x \) returns the result. If its argument is an array, the log1p function will be applied to each element and the results returned as an array.
This function should be used instead of \( \log(1+x) \) to avoid numerical errors whenever \( x \) is close to 0.

See Also
9.29 (log), 9.15 (expm1), 9.11 (cos), 9.6 (atan), 9.3 (acosh), 9.12 (cosh)

9.32 _max

Synopsis
Compute the maximum of two or more numeric values

Usage
\[ z = \text{\_max}(x_1, \ldots, x_N) \]

Description
The _max function returns a floating point number equal to the maximum value of its arguments. If any of the arguments are arrays (of equal length), an array of the corresponding values will be returned.
9.33  _min

**Notes**

This function returns a floating point result even when the arguments are integers.

**See Also**

9.33 (_min), 2.16 (min), 2.14 (max)

9.33  _min

**Synopsis**

Compute the minimum of two or more numeric values

**Usage**

\[ z = _\text{min} (x_1, \ldots, x_M) \]

**Description**

The _min function returns a floating point number equal to the minimum value of its arguments. If any of the arguments are arrays (of equal length), an array of the corresponding values will be returned.

**Notes**

This function returns a floating point result even when the arguments are integers.

**See Also**

2.16 (min), 9.32 (_max), 2.14 (max)

9.34  mul2

**Synopsis**

Multiply a number by 2

**Usage**

\[ y = \text{mul2}(x) \]

**Description**

The mul2 function multiplies an arithmetic type by two and returns the result. If its argument is an array, a new array will be created whose elements are obtained from the original array by using the mul2 function.

**See Also**

9.44 (sqr), 9.1 (abs)
9.35 nint

Synopsis
Round to the nearest integer

Usage
i = nint(x)

Description
The nint rounds its argument to the nearest integer and returns the result. If its argument is an array, a new array will be created whose elements are obtained from the original array elements by using the nint function.

See Also
9.38 (round), 9.18 (floor), 9.9 (ceil)

9.36 polynom

Synopsis
Evaluate a polynomial

Usage
Double_Type polynom([a0,a1,...aN], x [,use_factorial])

Description
The polynom function returns the value of the polynomial expression
\[ a_0 + a_1 x + a_2 x^2 + \ldots + a_N x^N \]

where the coefficients are given by an array of values [a0,...,aN]. If x is an array, the function will return a corresponding array. If the value of the optional use_factorial parameter is non-zero, then each term in the sum will be normalized by the corresponding factorial, i.e.,
\[ a_0/0! + a_1 x/1! + a_2 x^2/2! + \ldots + a_N x^N/N! \]

Notes
Prior to version 2.2, this function had a different calling syntax and was less useful. The polynom function does not yet support complex-valued coefficients.
For the case of a scalar value of x and a small degree polynomial, it is more efficient to use an explicit expression.

See Also
9.14 (exp)
9.37  Real

Synopsis
   Compute the real part of a number

Usage
   \( r = \text{Real} \ (z) \)

Description
   The \text{Real} function returns the real part of a number. If its argument is an array, the \text{Real} function will be applied to each element and the result returned as an array.

See Also
   9.23 (Imag), 9.10 (Conj), 9.1 (abs)

9.38  round

Synopsis
   Round to the nearest integral value

Usage
   \( y = \text{round} \ (x) \)

Description
   This function rounds its argument to the nearest integral value and returns it as a floating point result. If the argument is an array, an array of the corresponding values will be returned.

See Also
   9.18 (floor), 9.9 (ceil), 9.35 (nint)

9.39  set_float_format

Synopsis
   Set the format for printing floating point values.

Usage
   set_float_format (String_Type fmt)

Description
   The \text{set_float_format} function is used to set the floating point format to be used when floating point numbers are printed. The routines that use this are the traceback routines and the \text{string} function, any anything based upon the \text{string} function. The default value is "%f", which causes the number to be displayed with enough significant digits such that \( x = \text{atof} \ (\text{string} \ (x)) \).

Example
set_float_format ("%S"); % default
s = string (PI); % --> s = "3.141592653589793"
set_float_format ("%16.10f");
s = string (PI); % --> s = "3.1415926536"
set_float_format ("%10.6e");
s = string (PI); % --> s = "3.141593e+00"

See Also
9.21 (get_float_format), 12.12 (string), 4.10 (sprintf), 12.1 (atof), 12.7 (double)

9.40 sign

Synopsis
Compute the sign of a number

Usage
y = sign(x)

Description
The sign function returns the sign of an arithmetic type. If its argument is a complex number (Complex_Type), the sign will be applied to the imaginary part of the number. If the argument is an array, a new array will be created whose elements are obtained from the original array by using the sign function.

When applied to a real number or an integer, the sign function returns -1, 0, or +1 according to whether the number is less than zero, equal to zero, or greater than zero, respectively.

See Also
9.1 (abs)

9.41 sin

Synopsis
Compute the sine of a number

Usage
y = sin (x)

Description
The sin function computes the sine of a number and returns the result. If its argument is an array, the sin function will be applied to each element and the result returned as an array.

See Also
9.11 (cos), 9.6 (atan), 9.3 (acosh), 9.12 (cosh), 9.42 (sincos)
9.42  sincos

Synopsis
Compute the sine and cosine of a number

Usage
(s, c) = sincos (x)

Description
The sincos function computes the sine and cosine of a number and returns the result. If its argument is an array, the sincos function will be applied to each element and the result returned as an array.

See Also
9.41 (sin), 9.11 (cos)

9.43  sinh

Synopsis
Compute the hyperbolic sine of a number

Usage
y = sinh (x)

Description
The sinh function computes the hyperbolic sine of a number and returns the result. If its argument is an array, the sinh function will be applied to each element and the result returned as an array.

See Also
9.11 (cos), 9.6 (atan), 9.3 (acosh), 9.12 (cosh)

9.44  sqr

Synopsis
Compute the square of a number

Usage
y = sqr(x)

Description
The sqr function returns the square of an arithmetic type. If its argument is a complex number (Complex_Type), then it returns the square of the modulus. If the argument is an array, a new array will be created whose elements are obtained from the original array by using the sqr function.
Notes

For real scalar numbers, using \( x^* x \) instead of \( \text{sqr}(x) \) will result in faster executing code. However, if \( x \) is an array, then \( \text{sqr}(x) \) will execute faster.

See Also

9.1 (abs), 9.34 (mul2)

9.45 sqrt

Synopsis

Compute the square root of a number

Usage

\[ y = \text{sqrt}(x) \]

Description

The \( \text{sqrt} \) function computes the square root of a number and returns the result. If its argument is an array, the \( \text{sqrt} \) function will be applied to each element and the result returned as an array.

See Also

9.44 (sqr), 9.11 (cos), 9.6 (atan), 9.3 (acosh), 9.12 (cosh)

9.46 tan

Synopsis

Compute the tangent of a number

Usage

\[ y = \text{tan}(x) \]

Description

The \( \text{tan} \) function computes the tangent of a number and returns the result. If its argument is an array, the \( \text{tan} \) function will be applied to each element and the result returned as an array.

See Also

9.11 (cos), 9.6 (atan), 9.3 (acosh), 9.12 (cosh)

9.47 tanh

Synopsis

Compute the hyperbolic tangent of a number
Usage

\[ y = \tanh(x) \]

Description

The \texttt{tanh} function computes the hyperbolic tangent of a number and returns the result. If its argument is an array, the \texttt{tanh} function will be applied to each element and the result returned as an array.

See Also

\texttt{9.11 (cos), 9.6 (atan), 9.3 (acosh), 9.12 (cosh)}
Chapter 10

Message and Error Functions

10.1   errno

Synopsis
Error code set by system functions

Usage
Int_Type errno

Description
A system function can fail for a variety of reasons. For example, a file operation may fail
because lack of disk space, or the process does not have permission to perform the operation.
Such functions will return -1 and set the variable errno to an error code describing the reason
for failure.

Particular values of errno may be specified by the following symbolic constants (read-only
variables) and the corresponding errno_string value:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2BIG</td>
<td>&quot;Arg list too long&quot;</td>
</tr>
<tr>
<td>EACCES</td>
<td>&quot;Permission denied&quot;</td>
</tr>
<tr>
<td>EBADF</td>
<td>&quot;Bad file number&quot;</td>
</tr>
<tr>
<td>EBUSY</td>
<td>&quot;Mount device busy&quot;</td>
</tr>
<tr>
<td>ECHILD</td>
<td>&quot;No children&quot;</td>
</tr>
<tr>
<td>EEXIST</td>
<td>&quot;File exists&quot;</td>
</tr>
<tr>
<td>EFAULT</td>
<td>&quot;Bad address&quot;</td>
</tr>
<tr>
<td>EFBIG</td>
<td>&quot;File too large&quot;</td>
</tr>
<tr>
<td>EINTR</td>
<td>&quot;Interrupted system call&quot;</td>
</tr>
<tr>
<td>EINVAL</td>
<td>&quot;Invalid argument&quot;</td>
</tr>
<tr>
<td>EIO</td>
<td>&quot;I/O error&quot;</td>
</tr>
<tr>
<td>EISDIR</td>
<td>&quot;Is a directory&quot;</td>
</tr>
<tr>
<td>ELOOP</td>
<td>&quot;Too many levels of symbolic links&quot;</td>
</tr>
<tr>
<td>ENFILE</td>
<td>&quot;Too many open files&quot;</td>
</tr>
<tr>
<td>EMLINK</td>
<td>&quot;Too many links&quot;</td>
</tr>
<tr>
<td>ENAMETOOLONG</td>
<td>&quot;File name too long&quot;</td>
</tr>
<tr>
<td>ENFILE</td>
<td>&quot;File table overflow&quot;</td>
</tr>
</tbody>
</table>

105
Chapter 10. Message and Error Functions

Example

The `mkdir` function will attempt to create a directory. If it fails, the function will throw an IOError exception with a message containing the string representation of the `errno` value.

```c
if (-1 == mkdir (dir))
    throw IOError, sprintf ("mkdir %s failed: %s", dir, errno_string (errno));
```

See Also

10.2 (errno_string), 10.3 (error), 16.10 (mkdir)

10.2  `errno_string`

Synopsis

Return a string describing an `errno`.

Usage

```c
String_Type errno_string ( [Int_Type err ])
```

Description

The `errno_string` function returns a string describing the integer `errno` code `err`. If the `err` parameter is omitted, the current value of `errno` will be used. See the description for `errno` for more information.

Example

The `errno_string` function may be used as follows:

```c
define sizeof_file (file)
{
    variable st = stat_file (file);
    if (st == NULL)
```
throw IOError, sprintf("%s: %s", file, errno_string (errno));
    return st.st_size;
  }

See Also

10.1 (errno), 16.15 (stat_file)

10.3 error

Synopsis

Generate an error condition (deprecated)

Usage

error (String_Type msg)

Description

This function has been deprecated in favor of throw.

The error function generates a S-Lang RunTimeError exception. It takes a single string parameter which is displayed on the stderr output device.

Example

define add_txt_extension (file)
  {
    if (typeof (file) != String_Type)
      error ("add_extension: parameter must be a string");
    file += ".txt";
    return file;
  }

See Also

10.8 (verror), 10.5 (message)

10.4 __get_exception_info

Synopsis

Get information about the current exception

Usage

Struct_Type __get_exception_info ()

Description

This function returns information about the currently active exception in the form as a structure with the following fields:
### Chapter 10. Message and Error Functions

<table>
<thead>
<tr>
<th>variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error</td>
<td>The current exception, e.g., RunTimeError</td>
</tr>
<tr>
<td>descr</td>
<td>A description of the exception</td>
</tr>
<tr>
<td>file</td>
<td>Name of the file generating the exception</td>
</tr>
<tr>
<td>line</td>
<td>Line number where the exception originated</td>
</tr>
<tr>
<td>function</td>
<td>Function where the exception originated</td>
</tr>
<tr>
<td>object</td>
<td>A user-defined object thrown by the exception</td>
</tr>
<tr>
<td>message</td>
<td>A user-defined message</td>
</tr>
<tr>
<td>traceback</td>
<td>Traceback messages</td>
</tr>
</tbody>
</table>

If no exception is active, NULL will be returned.

This same information may also be obtained via the optional argument to the *try* statement:

```c
variable e = NULL;
try (e)
{
    do_something ();
}
finally
{
    if (e != NULL)
        vmessage ("An error occurred: %s", e.message);
}
```

**See Also**

10.3 (error)

### 10.5 message

**Synopsis**

Print a string onto the message device

**Usage**

```c
message (String_Type s)
```

**Description**

The *message* function will print the string specified by *s* onto the message device.

**Example**

```c
define print_current_time ()
{
    message (time ());
}
```

**Notes**

The message device will depend upon the application. For example, the output message device for the *jed* editor corresponds to the line at the bottom of the display window. The default message device is the standard output device.

**See Also**

10.9 (vmessage), 4.10 (strftime), 10.3 (error)
10.6 new_exception

Synopsis

Create a new exception

Usage

new_exception (String_Type name, Int_Type baseclass, String_Type descr)

Description

This function creates a new exception called name subclassed upon baseclass. The description of the exception is specified by descr.

Example

new_exception ("MyError", RunTimeError, "My very own error");
try
{
    if (something_is_wrong ())
        throw MyError;
}
catch RunTimeError;

In this case, catching RunTimeError will also catch MyError since it is a subclass of RunTimeError.

See Also

10.3 (error), 10.8 (verror)

10.7 usage

Synopsis

Generate a usage error

Usage

usage (String_Type msg)

Description

The usage function generates a UsageError exception and displays msg to the message device.

Example

Suppose that a function called plot plots an array of x and y values. Then such a function could be written to issue a usage message if the wrong number of arguments was passed:

define plot ()
{
    variable x, y;

    if (_NARGS != 2)
        usage ("plot (x, y)");
\[(x, y) = ();\]
\%
Now do the hard part
.
.
}

See Also

10.3 (error), 10.5 (message)

10.8 \texttt{verror}

Synopsis

Generate an error condition (deprecated)

Usage

\texttt{verror (String\_Type fmt, \ldots)}

Description

This function has been deprecated in favor of \texttt{throw}.

The \texttt{verror} function performs the same role as the \texttt{error} function. The only difference is that instead of a single string argument, \texttt{verror} takes a \texttt{sprintf} style argument list.

Example

\begin{verbatim}
define open_file (file)
{
variable fp;

fp = fopen (file, "r");
if (fp == NULL) verror ("Unable to open \%s", file);
return fp;
}
\end{verbatim}

Notes

In the current implementation, the \texttt{verror} function is not an intrinsic function. Rather it is a predefined \texttt{S-Lang} function using a combination of \texttt{sprintf} and \texttt{error}.

To generate a specific exception, a \texttt{throw} statement should be used. In fact, a \texttt{throw} statement such as:

\begin{verbatim}
if (fp == NULL)
    throw OpenError, "Unable to open $file$";
\end{verbatim}

is preferable to the use of \texttt{verror} in the above example.

See Also

10.3 (error), 4.8 (Sprintf), 10.9 (vmessage)
10.9 vminessage

Synopsis

Print a formatted string onto the message device

Usage

vmessage (String_Type fmt, ...)

Description

The vmessage function formats a sprintf style argument list and displays the resulting string onto the message device.

Notes

In the current implementation, the vmessage function is not an intrinsic function. Rather it is a predefined S-Lang function using a combination of sprintf and message.

See Also

10.5 (message), 4.10 (sprintf), 4.8 (sprintf), 10.8 (verror)
Chapter 11

Time and Date Functions

11.1 ctime

Synopsis
Convert a calendar time to a string

Usage
String_Type ctime(Long_Type secs)

Description
This function returns a string representation of the time as given by secs seconds since 00:00:00 UTC, Jan 1, 1970.

See Also
11.9 (time), 11.5 (strptime), 11.8 (_time), 11.3 (localtime), 11.2 (gmtime)

11.2 gmtime

Synopsis
Break down a time in seconds to the GMT timezone

Usage
Struct_Type gmtime (Long_Type secs)

Description
The gmtime function is exactly like localtime except that the values in the structure it returns are with respect to GMT instead of the local timezone. See the documentation for localtime for more information.

Notes
On systems that do not support the gmtime C library function, this function is the same as localtime.
11.3 localtime

Synopsis
Break down a time in seconds to the local timezone

Usage
Struct_Type localtime (Long_Type secs)

Description
The localtime function takes a parameter secs representing the number of seconds since 00:00:00, January 1 1970 UTC and returns a structure containing information about secs in the local timezone. The structure contains the following Int_Type fields:

- tm_sec The number of seconds after the minute, normally in the range 0 to 59, but can be up to 61 to allow for leap seconds.
- tm_min The number of minutes after the hour, in the range 0 to 59.
- tm_hour The number of hours past midnight, in the range 0 to 23.
- tm_mday The day of the month, in the range 1 to 31.
- tm_mon The number of months since January, in the range 0 to 11.
- tm_year The number of years since 1900.
- tm_wday The number of days since Sunday, in the range 0 to 6.
- tm_yday The number of days since January 1, in the range 0 to 365.
- tm_isdst A flag that indicates whether daylight saving time is in effect at the time described. The value is positive if daylight saving time is in effect, zero if it is not, and negative if the information is not available.

See Also
11.2 (gmtime), 11.8 (_time), 11.1 (ctime), 11.4 (mktime)

11.4 mktime

Synopsis
Convert a time-structure to seconds

Usage
secs = mktime (Struct_Type tm)

Description
The mktime function is essentially the inverse of the localtime function. See the documentation for that function for more details.
11.5  strftime

Synopsis

Format a date and time string

Usage

    str = strftime (String_Type format [,Struct_Type tm])

Description

The `strftime` creates a date and time string according to a specified format. If called with a single argument, the current local time will be used as the reference time. If called with two arguments, the second argument specifies the reference time, and must be a structure with the same fields as the structure returned by the `localtime` function.

The format string may be composed of one or more of the following format descriptors:

- `%A` full weekday name (Monday)
- `%a` abbreviated weekday name (Mon)
- `%B` full month name (January)
- `%b` abbreviated month name (Jan)
- `%c` standard date and time representation
- `%d` day-of-month (01-31)
- `%H` hour (24 hour clock) (00-23)
- `%I` hour (12 hour clock) (01-12)
- `%j` day-of-year (001-366)
- `%m` month (01-12)
- `%p` local equivalent of AM or PM
- `%S` second (00-59)
- `%U` week-of-year, first day Sunday (00-53)
- `%W` week-of-year, first day Monday (00-53)
- `%w` weekday (0-6, Sunday is 0)
- `%x` standard date representation
- `%X` standard time representation
- `%y` year without century (00-99)
- `%Y` year with century
- `%z` timezone name
- `%Z` percent sign

as well as any others provided by the C library. The actual values represented by the format descriptors are locale-dependent.

Example

    message (strftime ("Today is %A, day %j of the year"));
    tm = localtime (0);
    message (strftime ("Unix time 0 was on a %A", tm));
11.6 _tic

Synopsis
Reset the CPU timer

Usage
_tic ()

Description
The _tic function resets the internal CPU timer. The _toc may be used to read this timer. See the documentation for the _toc function for more information.

See Also
11.12 (_toc), 11.11 (times), 11.7 (tic), 11.13 (toc)

11.7 tic

Synopsis
Reset the interval timer

Usage
void tic ()

Description
The tic function resets the internal interval timer. The toc may be used to read the interval timer.

Example
The tic/toc functions may be used to measure execution times. For example, at the slsh prompt, they may be used to measure the speed of a loop:

```
slsh> tic; loop (500000); toc;
0.06558
```

Notes
On Unix, this timer makes use of the C library gettimeofday function.

See Also
11.13 (toc), 11.12 (_toc), 11.6 (_tic), 11.11 (times)
11.8 _time

Synopsis

Get the current calendar time in seconds

Usage

Long_Type _time ()

Description

The _time function returns the number of elapsed seconds since 00:00:00 UTC, January 1, 1970. A number of functions (ctime, gmtime, localtime, etc.) are able to convert such a value to other representations.

See Also

11.1 (ctime), 11.9 (time), 11.3 (localtime), 11.2 (gmtime)

11.9 time

Synopsis

Return the current date and time as a string

Usage

String_Type time ()

Description

This function returns the current time as a string of the form:

    Sun Apr 21 13:34:17 1996

See Also

11.5 (strftime), 11.1 (ctime), 10.5 (message), 4.43 (substr)

11.10 timegm

Synopsis

Convert a time structure for the GMT timezone to seconds

Usage

Long_Type secs = timegm(Struct_Type tm)

Description

timegm is the inverse of the gmtime function.

See Also

11.2 (gmtime), 11.4 (mktime), 11.3 (localtime)
11.11 times

Synopsis
Get process times

Usage
Struct_Type times ()

Description
The times function returns a structure containing the following fields:

- tms_utime (user time)
- tms_stime (system time)
- tms_cutime (user time of child processes)
- tms_cstime (system time of child processes)

Notes
Not all systems support this function.

See Also
11.6 (_tic), 11.12 (_toc), 11.8 (_time)

11.12 _toc

Synopsis
Get the elapsed CPU time for the current process

Usage
Double_Type _toc ()

Description
The _toc function returns the elapsed CPU time in seconds since the last call to _tic. The CPU time is the amount of time the CPU spent running the code of the current process.

Notes
This function may not be available on all systems.

The implementation of this function is based upon the times system call. The precision of the clock is system dependent and may not be very accurate for small time intervals. For this reason, the tic/toc functions may be more useful for small time-intervals.

See Also
11.6 (_tic), 11.7 (tic), 11.13 (toc), 11.11 (times), 11.8 (_time)
11.13 toc

Synopsis
   Read the interval timer

Usage
   Double_Type toc ()

Description
   The toc function returns the elapsed time in seconds since the last call to tic. See the
documentation for the tic function for more information.

See Also
   11.7 (tic), 11.6 (_tic), 11.12 (_toc), 11.11 (times), 11.8 (_time)
Chapter 12

Data-Type Conversion Functions

12.1 atof

Synopsis
Convert a string to a double precision number

Usage
Double_Type atof (String_Type s)

Description
This function converts a string s to a double precision value and returns the result. It performs no error checking on the format of the string. The function _slang_guess_type may be used to check the syntax of the string.

Example

    define error_checked_atof (s)
    {
        if (__is_datatype_numeric (_slang_guess_type (s)))
            return atof (s);
        throw InvalidParmError, "$s is not a double";
    }

See Also
12.15 (typecast), 12.7 (double), 12.11 (_slang_guess_type)

12.2 atoi

Synopsis
Convert a string to an integer

Usage
Int_Type atoi (String_Type str)
Description
The `atoi` function converts a string to an `Int_Type` using the standard C library function of the corresponding name.

Notes
This function performs no syntax checking upon its argument.

See Also
12.9 (integer), 12.3 (atol), 12.4 (atoll), 12.1 (atof), 4.11 (scanf)

12.3 `atol`

Synopsis
Convert a string to an long integer

Usage
`Long_Type atol(String_Type str)`

Description
The `atol` function converts a string to a `Long_Type` using the standard C library function of the corresponding name.

Notes
This function performs no syntax checking upon its argument.

See Also
12.9 (integer), 12.2 (atoi), 12.4 (atoll), 12.1 (atof), 4.11 (scanf)

12.4 `atoll`

Synopsis
Convert a string to a long long

Usage
`LLong_Type atoll(String_Type str)`

Description
The `atoll` function converts a string to a `LLong_Type` using the standard C library function of the corresponding name.

Notes
This function performs no syntax checking upon its argument. Not all platforms provide support for the long long data type.

See Also
12.9 (integer), 12.2 (atoi), 12.3 (atol), 12.1 (atof), 4.11 (scanf)
12.5 char

Synopsis

Convert a character code to a string

Usage

String_Type char (Integer_Type c)

Description

The char function converts an integer character code (ascii) value c to a string of unit character length such that the first character of the string is c. For example, char('a') returns the string "a".

If UTF-8 mode is in effect (slang_utf8_ok is non-zero), the resulting single character may be represented by several bytes.

If the character code c is less than 0, then byte-semantics will be used with the resulting string consisting of a single byte whose value is that of -c&0xFF.

Notes

A better name should have been chosen for this function.

See Also

12.9 (integer), 12.12 (string), ?? (typedef), 4.10 (sprintf), 5.8 (pack)

12.6 define_case

Synopsis

Define upper-lower case conversion

Usage

define_case (Integer_Type ch_up, Integer_Type ch_low)

Description

This function defines an upper and lowercase relationship between two characters specified by the arguments. This relationship is used by routines which perform uppercase and lowercase conversions. The first integer ch_up is the ascii value of the uppercase character and the second parameter ch_low is the ascii value of its lowercase counterpart.

Notes

This function has no effect in UTF-8 mode.

See Also

4.25 (strlow), 4.38 (strup)
12.7 double

Synopsis
Convert an object to double precision

Usage
Double_Type double (x)

Description
The double function typecasts an object x to double precision. For example, if x is an array of integers, an array of double types will be returned. If an object cannot be converted to Double_Type, a type-mismatch error will result.

Notes
The double function is equivalent to the typecast operation

typecast (x, Double_Type)

To convert a string to a double precision number, use the atof function.

See Also
12.15 (typecast), 12.1 (atof), 12.8 (int)

12.8 int

Synopsis
Typecast an object to an integer

Usage
Int_Type int (s)

Description
This function performs a typecast of an object s to an object of Integer_Type. If s is a string, it returns returns the ascii value of the first bytes of the string s. If s is Double_Type, int truncates the number to an integer and returns it.

Example
int can be used to convert single byte strings to integers. As an example, the intrinsic function isdigit may be defined as

define isdigit (s)
{
    if ((int (s) >= '0') and (int (s) <= '9')) return 1;
    return 0;
}

Notes
This function is equivalent to typecast (s, Integer_Type);

See Also
12.15 (typecast), 12.7 (double), 12.9 (integer), 12.5 (char), ?? (isdigit), ?? (isxdigit)
12.9  integer

Synopsis
  Convert a string to an integer

Usage
  Integer_Type integer (String_Type s)

Description
  The integer function converts a string representation of an integer back to an integer. If the string does not form a valid integer, a SyntaxError will be thrown.

Example
  integer ("1234") returns the integer value 1234.

Notes
  This function operates only on strings and is not the same as the more general typecast operator.

See Also
  12.15 (typecast), 12.11 (_slang_guess_type), 12.12 (string), 4.10 (sprintf), 12.5 (char)

12.10  isalnum, isalpha, isascii, isblank, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit

Synopsis
  Character classification functions

Usage
  Char_Type isalnum(wch)  Char_Type isalpha(wch)  Char_Type isascii(wch)
  Char_Type isblank(wch)  Char_Type iscntrl(wch)  Char_Type isdigit(wch)
  Char_Type isgraph(wch)  Char_Type islower(wch)  Char_Type isprint(wch)
  Char_Type ispunct(wch)  Char_Type isspace(wch)  Char_Type isupper(wch)
  Char_Type isxdigit(wch)

Description
  These functions return a non-zero value if the character given by wch is a member of the character class represented by the function, according to the table below. Otherwise, 0 will be returned to indicate that the character is not a member of the class. If the parameter wch is a string, then the first character (not necessarily a byte) of the string will be used.

    isalnum : alphanumeric character, equivalent to isalpha or isdigit
    isalpha : alphabetic character
    isascii : 7-bit unsigned ascii character
    isblank : space or a tab
    iscntrl : control character
    isdigit : digit 0-9
12.11 _slang_guess_type

Synopsis

Guess the data type that a string represents

Usage

DataType_Type _slang_guess_type (String_Type s)

Description

This function tries to determine whether its argument s represents an integer (short, int, long), floating point (float, double), or a complex number. If it appears to be none of these, then a string is assumed. It returns one of the following values depending on the format of the string s:

- Short_Type : short integer (e.g., "2h")
- UShort_Type : unsigned short integer (e.g., "2hu")
- Integer_Type : integer (e.g., "2")
- UInteger_Type : unsigned integer (e.g., "2")
- Long_Type : long integer (e.g., "2l")
- ULong_Type : unsigned long integer (e.g., "2l")
- Float_Type : float (e.g., "2.0f")
- Double_Type : double (e.g., "2.0")
- Complex_Type : imaginary (e.g., "2i")
- String_Type : Anything else. (e.g., "2foo")

For example, _slang_guess_type("1e2") returns Double_Type but _slang_guess_type("e12") returns String_Type.

See Also

12.9 (integer), 12.12 (string), 12.7 (double), 12.1 (atof), 25.12 (__is_datatype_numeric)
12.13 tolower

Usage

String_Type string (obj)

Description

The string function may be used to convert an object obj of any type to its string representation. For example, string(12.34) returns "12.34".

Example

```c
define print Anything (anything)
{
    message (string (anything));
}
```

Notes

This function is not the same as typecasting to a String_Type using the typecast function.

See Also

12.15 (typecast), 4.10 (printf), 12.9 (integer), 12.5 (char)

12.14 toupper

Synopsis

Convert a character to uppercase.

Usage

Integer_Type toupper (Integer_Type ch)

Description

This function takes an integer ch and returns its uppercase equivalent.

See Also

12.13 (tolower), 4.38 (strup), 4.25 (strlow), 12.8 (int), 12.5 (char), 12.6 (define_case)
12.15 typecast

Synopsis

Convert an object from one data type to another.

Usage

typecast (x, new_type)

Description

The typecast function performs a generic typecast operation on x to convert it to new_type. If x represents an array, the function will attempt to convert all elements of x to new_type. Not all objects can be converted and a type-mismatch error will result upon failure.

Example

    define to_complex (x)
    {
        return typecast (x, Complex_Type);
    }

defines a function that converts its argument, x to a complex number.

See Also

12.8 (int), 12.7 (double), 12.17 (typeof)

12.16 _typeof

Synopsis

Get the data type of an object

Usage

DataType_Type _typeof (x)

Description

This function is similar to the typeof function except in the case of arrays. If the object x is an array, then the data type of the array will be returned. Otherwise _typeof returns the data type of x.

Example

    if (Integer_Type == _typeof (x))
        message ("x is an integer or an integer array");

See Also

12.17 (typeof), 23 (array_info), 12.11 (_slang_guess_type), 12.15 (typecast)
12.17 `typeof`

**Synopsis**
Get the data type of an object

**Usage**

```c
DataType_Type typeof (x)
```

**Description**
This function returns the data type of `x`.

**Example**

```c
if (Integer_Type == typeof (x)) message ("x is an integer");
```

**See Also**

12.16 (`_typeof`), 6.8 (is_struct_type), 2.3 (array_info), 12.11 (`_slang_guess_type`), 12.15 (typecast)
Chapter 13

Stdio File I/O Functions

13.1 clearerr

Synopsis
Clear the error of a file stream

Usage
clearerr (File_Type fp)

Description
The clearerr function clears the error and end-of-file flags associated with the open file stream fp.

See Also
13.5 (ferror), 13.4 (feof), 13.9 (fopen)

13.2 fclose

Synopsis
Close a file

Usage
Integer_Type fclose (File_Type fp)

Description
The fclose function may be used to close an open file pointer fp. Upon success it returns zero, and upon failure it sets errno and returns -1. Failure usually indicates that the file system is full or that fp does not refer to an open file.

Notes
Many C programmers call fclose without checking the return value. The S-Lang language requires the programmer to explicitly handle any value returned by a function. The simplest way to handle the return value from fclose is to call it via:
() = fclose (fp);

See Also
13.9 (fopen), 13.7 (fgets), 13.6 (flush), 13.18 (pclose), 10.1 (errno)

13.3 fdopen

Synopsis

Convert a FD_Type file descriptor to a stdio File_Type object

Usage

File_Type fdopen (FD_Type, String_Type mode)

Description

The fdopen function creates and returns a stdio File_Type object from the open FD_Type
descriptor fd. The mode parameter corresponds to the mode parameter of the fopen function
and must be consistent with the mode of the descriptor fd. The function returns NULL upon
failure and sets errno.

Notes

Since the stdio File_Type object created by this function is derived from the FD_Type descrip-
tor, the FD_Type is regarded as more fundamental than the File_Type object. This means
that the descriptor must be in scope while the File_Type object is used. In particular, if
the descriptor goes out of scope, the descriptor will get closed causing I/O to the File_Type
object to fail, e.g.,

```c
fd = open ("/path/to/file", O_RDONLY);
fp = fdopen (fd);
fd = 0;        % This will cause the FD_Type descriptor to go out of
              % scope. Any I/O on fp will now fail.
```

Calling the fclose function on the File_Type object will cause the underlying descriptor to
close.

Any stdio File_Type object created by the fdopen function will remain associated with the
FD_Type descriptor, unless the object is explicitly removed via fclose. This means that code
such as

```c
fd = open (...
loop (50)
{
   fp = fdopen (fd, ...);
   .
   .
}
```

will result in 50 File_Type objects attached to fd after the loop has terminated.

See Also
14.6 (fileno), 13.9 (fopen), 14.9 (open), 14.1 (close), 13.2 (fclose), 14.3 (dup_fd)
13.4 \texttt{feof}

Synopsis
Get the end-of-file status

Usage
\begin{verbatim}
Integer_Type feof (File_Type fp)
\end{verbatim}

Description
This function may be used to determine the state of the end-of-file indicator of the open file descriptor \( fp \). It returns zero if the indicator is not set, or non-zero if it is. The end-of-file indicator may be cleared by the \texttt{clearerr} function.

See Also
13.5 (ferror), 13.1 (clearerr), 13.9 (fopen)

13.5 \texttt{ferror}

Synopsis
Determine the error status of an open file descriptor

Usage
\begin{verbatim}
Integer_Type ferror (File_Type fp)
\end{verbatim}

Description
This function may be used to determine the state of the error indicator of the open file descriptor \( fp \). It returns zero if the indicator is not set, or non-zero if it is. The error indicator may be cleared by the \texttt{clearerr} function.

See Also
13.4 (feof), 13.1 (clearerr), 13.9 (fopen)

13.6 \texttt{fflush}

Synopsis
Flush an output stream

Usage
\begin{verbatim}
Integer_Type fflush (File_Type fp)
\end{verbatim}

Description
The \texttt{fflush} function may be used to update the stdio output stream specified by \( fp \). It returns 0 upon success, or -1 upon failure and sets \texttt{errno} accordingly. In particular, this function will fail if \( fp \) does not represent an open output stream, or if \( fp \) is associated with a disk file and there is insufficient disk space.
Example

This example illustrates how to use the `fflush` function without regard to the return value:

```c
() = fputs ("Enter value> ", stdout);
() = fflush (stdout);
```

See Also

13.9 (fopen), 13.2 (fclose)

13.7 fgets

Synopsis

Read a line from a file

Usage

```c
Integer_Type fgets (SLang_Ref_Type ref, File_Type fp)
```

Description

`fgets` reads a line from the open file specified by `fp` and places the characters in the variable whose reference is specified by `ref`. It returns -1 if `fp` is not associated with an open file or an attempt was made to read at the end the file; otherwise, it returns the number of characters read.

Example

The following example returns the lines of a file via a linked list:

```c
define read_file (file)
{
    variable buf, fp, root, tail;
    variable list_type = struct { text, next };

    root = NULL;

    fp = fopen(file, "r");
    if (fp == NULL)
        throw OpenError, "fopen failed to open $file for reading"$;
    while (-1 != fgets (&buf, fp))
    {
        if (root == NULL)
            {
                root = @list_type;
                tail = root;
            }
        else
            {
                tail.next = @list_type;
                tail = tail.next;
            }
        tail.text = buf;
```
13.8. fgetslines

Synopsis
Read lines as an array from an open file

Usage
String_Type[] fgetslines (File_Type fp [,Int_Type num])

Description
The fgetslines function reads a specified number of lines as an array of strings from the file associated with the file pointer fp. If the number of lines to be read is left unspecified, the function will return the rest of the lines in the file. If the file is empty, an empty string array will be returned. The function returns NULL upon error.

Example
The following function returns the number of lines in a file:

```c
define count_lines_in_file (file)
{
    variable fp, lines;

    fp = fopen (file, "r");
    if (fp == NULL)
        return -1;

    lines = fgetslines (fp);
    if (lines == NULL)
        return -1;

    return length (lines);
}
```

Note that the file was implicitly closed when the variable fp goes out of scope (in the case, when the function returns).

See Also
13.7 (fgets), 13.13 (fread), 13.9 (fopen), 13.12 (fputslines)
13.9 fopen

Synopsis

Open a file

Usage

File_Type fopen (String_Type f, String_Type m)

Description

The fopen function opens a file \( f \) according to the mode string \( m \). Allowed values for \( m \) are:

- "r" Read only
- "w" Write only
- "a" Append
- "r+" Reading and writing at the beginning of the file.
- "w+" Reading and writing. The file is created if it does not exist; otherwise, it is truncated.
- "a+" Reading and writing at the end of the file. The file is created if it does not already exist.

In addition, the mode string can also include the letter 'b' as the last character to indicate that the file is to be opened in binary mode.

Upon success, fopen returns a File_Type object which is meant to be used by other operations that require an open file pointer. Upon failure, the function returns NULL.

Example

The following function opens a file in append mode and writes a string to it:

```define append_string_to_file (str, file)
{
  variable fp = fopen (file, "a");
  if (fp == NULL)
    throw OpenError, "$file could not be opened$";
  () = fputs (str, fp);
  () = fclose (fp);
}
```

Note that the return values from fputs and fclose were ignored.

Notes

There is no need to explicitly close a file opened with fopen. If the returned File_Type object goes out of scope, the interpreter will automatically close the file. However, explicitly closing a file with fclose and checking its return value is recommended.

See Also

13.2 (fclose), 13.7 (fgets), 13.11 (fputs), 13.19 (popen)
13.10  fprintf

Synopsis
Create and write a formatted string to a file

Usage
Int_Type fprintf (File_Type fp, String_Type fmt, ...)

Description
fprintf formats the objects specified by the variable argument list according to the format
fmt and write the result to the open file pointer fp.
The format string obeys the same syntax and semantics as the sprintf format string. See the
description of the sprintf function for more information.
fprintf returns the number of bytes written to the file, or -1 upon error.

See Also
13.11 (fputs), 13.20 (printf), 13.17 (fwrite), 10.5 (message)

13.11  fputs

Synopsis
Write a string to an open stream

Usage
Integer_Type fputs (String_Type s, File_Type fp)

Description
The fputs function writes the string s to the open file pointer fp. It returns -1 upon failure
and sets errno, otherwise it returns the length of the string.

Example
The following function opens a file in append mode and uses the fputs function to write to it.

define append_string_to_file (str, file)
{
    variable fp;
    fp = fopen (file, "a");
    if (fp == NULL)
        throw OpenError, "Unable to open $file"$;
    if (-1 == fputs (str, fp))
        || (-1 == fclose (fp)))
        throw WriteError, "Error writing to $file"$;
}

Notes
One must not disregard the return value from the fputs function. Doing so may lead to a
stack overflow error.
To write an object that contains embedded null characters, use the fwrite function.
See Also

13.2 (fclose), 13.9 (fopen), 13.7 (fgets), 13.17 (fwrite)

13.12 fputslines

Synopsis
Write an array of strings to an open file

Usage
Int_Type fputslines (String_Type[] a, File_Type fp)

Description
The fputslines function writes an array of strings to the specified file pointer. It returns the number of elements successfully written. Any NULL elements in the array will be skipped.

Example

if (length (lines) != fputslines (lines, fp))
  throw WriteError;

See Also

13.11 (fputs), 13.8 (fgetslines), 13.9 (fopen)

13.13 fread

Synopsis
Read binary data from a file

Usage
UInt_Type fread (Ref_Type b, DataType_Type t, UInt_Type n, File_Type fp)

Description
The fread function may be used to read n objects of type t from an open file pointer fp. Upon success, it returns the number of objects read from the file and places the objects in variable specified by b. Upon error or end-of-file, it returns -1 and sets errno accordingly.

If more than one object is read from the file, those objects will be placed in an array of the appropriate size.

Example

The following example illustrates how to read 50 integers from a file:

define read_50_ints_from_a_file (file)
{
  variable fp, n, buf;

  fp = fopen (file, "rb");
if (fp == NULL)
    throw OpenError;
int n = fread (&buf, Int_Type, 50, fp);
if (n == -1)
    throw ReadError, "fread failed";
() = fclose (fp);
return buf;
}

Notes

Use the pack and unpack functions to read data with a specific byte-ordering.

The fread_bytes function may be used to read a specified number of bytes in the form of a binary string (BString_Type).

If an attempt is made to read at the end of a file, the function will return -1. To distinguish this condition from a system error, the fseek function should be used. This distinction is particularly important when reading from a socket or pipe.

See Also

13.14 (fread_bytes), 13.17 (fwrite), 13.7 (fgets), 13.4 (feof), 13.5 (ferror), 13.9 (fopen), 5.8 (pack), 5.11 (unpack)

13.14 fread_bytes

Synopsis

Read bytes from a file as a binary-string

Usage

UInt_Type fread_bytes (Ref_Type b, UInt_Type n, File_Type fp)

Description

The fread_bytes function may be used to read n bytes from from an open file pointer fp. Upon success, it returns the number of bytes read from the file and assigns to the variable attached to the reference b a binary string formed from the bytes read. Upon error or end of file, the function returns -1 and sets errno accordingly.

Notes

Use the pack and unpack functions to read data with a specific byte-ordering.

See Also

13.13 (fread), 13.17 (fwrite), 13.7 (fgets), 13.9 (fopen), 5.8 (pack), 5.11 (unpack)

13.15 fseek

Synopsis

Reposition a stdio stream
Chapter 13. Stdio File I/O Functions

13.16 ftell

Synopsis
Obtain the current position in an open stream

Usage
LLong_Type ftell (File_Type fp)

Description
The ftell function may be used to obtain the current position in the stream associated with the open file pointer fp. It returns the position of the pointer measured in bytes from the beginning of the file. Upon error, it returns -1 and sets errno accordingly.

See Also
13.15 (fseek), 13.9 (fopen)

13.17 fwrite

Synopsis
Write binary data to a file

Usage
UInt_Type fwrite (b, File_Type fp)

Description
The fwrite function may be used to write the object represented by b to an open file. If b is a string or an array, the function will attempt to write all elements of the object to the file.
It returns the number of elements successfully written, otherwise it returns -1 upon error and sets errno accordingly.

**Example**

The following example illustrates how to write an integer array to a file. In this example, `fp` is an open file descriptor:

```plaintext
variable a = [1:50]; % 50 element integer array
if (50 != fwrite (a, fp))
    throw WriteError;
```

Here is how to write the array one element at a time:

```plaintext
variable ai, a = [1:50];
foreach ai (a)
{
    if (1 != fwrite(ai, fp))
        throw WriteError;
}
```

**Notes**

Not all data types may be supported the `fwrite` function. It is supported by all vector, scalar, and string objects.

**See Also**

13.13 (fread), 13.11 (fputs), 13.9 (fopen), 5.8 (pack), 5.11 (unpack)

### 13.18 pclose

**Synopsis**

Close a process pipe

**Usage**

```plaintext
Integer_Type pclose (File_Type fp)
```

**Description**

The `pclose` function waits for the process associated with `fp` to exit and then returns the exit status of the command.

**See Also**

13.19 (popen), 13.2 (fclose)

### 13.19 popen

**Synopsis**

Open a pipe to a process
Chapter 13. Stdio File I/O Functions

Usage

File_Type popen (String_Type cmd, String_Type mode)

Description

The `popen` function executes a process specified by `cmd` and opens a unidirectional pipe to the newly created process. The `mode` indicates whether or not the pipe is open for reading or writing. Specifically, if `mode` is "r", then the pipe is opened for reading, or if `mode` is "w", then the pipe will be open for writing.

Upon success, a `File_Type` pointer will be returned, otherwise the function failed and `NULL` will be returned.

Notes

This function is not available on all systems.

The `process` module’s `new_process` function provides a much more secure and powerful interface to process I/O.

See Also

?? (new_process), 13.18 (pclose), 13.9 (open)

13.20 `printf`

Synopsis

Create and write a formatted string to stdout

Usage

Int_Type printf (String_Type fmt, ...)

Description

`printf` formats the objects specified by the variable argument list according to the format `fmt` and write the result to `stdout`. This function is equivalent to `fprintf` used with the `stdout` file pointer. See `fprintf` for more information.

`printf` returns the number of bytes written or -1 upon error.

Notes

Many C programmers do not check the return status of the `printf` C library function. Make sure that if you do not care about whether or not the function succeeds, then code it as in the following example:

```c
() = printf ("%s laid %d eggs\n", chicken_name, num_egg);
```

See Also

13.11 (fputs), 13.10 (fprintf), 13.17 (fwrite), 10.5 (message)
13.21 setvbuf

Synopsis

Usage

\[\text{Int} \_\text{Type setvbuf (File} \_\text{Type fp, Int} \_\text{Type mode, Int} \_\text{Type size)}\]

Description

The setvbuf function may be used to control how the stdio stream specified by the open File_Type object is buffered.

The mode argument must be one of the following values:

- \text{IONBF} : unbuffered
- \text{IOFBF} : fully buffered
- \text{IOLBF} : line buffered

The size argument controls the size of the buffer. If size is 0, then the function will not change the size of the buffer, only the mode. Otherwise, size is expected to be larger than 0 and a buffer of the requested size will be allocated for the stream.

Notes

This function must be used only after the stream has been opened and before any other operations have been performed on the stream.

See Also

13.9 (fopen), 13.2 (fclose), 13.6 (fflush)
Chapter 14

Low-level POSIX I/O functions

14.1 close

Synopsis

Close an open file descriptor

Usage

Int_Type close (FD_Type fd)

Description

The close function is used to close an open file descriptor created by the open function. Upon success 0 is returned, otherwise the function returns -1 and sets errno accordingly.

See Also

14.9 (open), 14.2 (_close), 13.2 (fclose), 14.10 (read), 14.11 (write)

14.2 _close

Synopsis

Close an open file descriptor

Usage

Int_Type _close (Int_Type fd)

Description

The _close function is used to close the underlying integer open file descriptor. Upon success 0 is returned, otherwise the function returns -1 and sets errno accordingly.

See Also

14.9 (open), 14.5 (_fileno), 14.1 (close), 13.2 (fclose), 14.10 (read), 14.11 (write)
14.3  dup_fd

Synopsis
Duplicate a file descriptor

Usage
FD_Type dup_fd (FD_Type fd)

Description
The dup_fd function duplicates a specified file descriptor and returns the duplicate. If the
function fails, NULL will be returned and errno set accordingly.

Notes
This function is essentially a wrapper around the POSIX dup function.

See Also
14.9 (open), 14.1 (close)

14.4  dup2_fd

Synopsis
Duplicate a file descriptor

Usage
Int_Type dup2_fd (FD_Type fd, int newfd)

Description
The dup2_fd function makes newfd a copy of the specified file descriptor fd. Upon success
returns newfd, otherwise it returns -1 and sets errno accordingly.

See Also
23.1 (dup), 14.9 (open), 14.1 (close), 14.2 (_close), 14.5 (_fileno), 14.10 (read)

14.5  _fileno

Synopsis
Get the underlying integer file descriptor

Usage
Int_Type _fileno (File_Type|FD_Type fp)

Description
The _fileno function returns the underlying integer descriptor for a specified stdio File_Type
or FD_Type object. Upon failure it returns -1 and sets errno accordingly.

See Also
14.6 (fileno), 13.9 (fopen), 14.9 (open), 13.2 (fclose), 14.1 (close), 14.3 (dup_fd)
14.6 fileno

Synopsis
Convert a stdio File_Type object to a FD_Type descriptor

Usage
FD_Type fileno (File_Type fp)

Description
The fileno function returns the FD_Type descriptor associated with the stdio File_Type file pointer. Upon failure, NULL is returned.

Notes
Closing the resulting file descriptor will have no effect.

See Also
13.9 (fopen), 14.9 (open), 13.2 (fclose), 14.1 (close), 14.3 (dup_fd), 14.5 (_fileno)

14.7 isatty

Synopsis
Determine if an open file descriptor refers to a terminal

Usage
Int_Type isatty (FD_Type or File_Type fd)

Description
This function returns 1 if the file descriptor fd refers to a terminal; otherwise it returns 0. The object fd may either be a File_Type stdio descriptor or a lower-level FD_Type object.

See Also
13.9 (fopen), 13.2 (fclose), 14.6 (fileno)

14.8 lseek

Synopsis
Reposition a file descriptor’s file pointer

Usage
Long_Type lseek (FD_Type fd, LLong_Type ofs, int mode) The lseek function repositions the file pointer associated with the open file descriptor fd to the offset ofs according to the mode parameter. Specifically, mode must be one of the values:
SEEK_SET Set the offset to ofs from the beginning of the file
SEEK_CUR Add ofs to the current offset
SEEK_END Add ofs to the current file size
Upon error, lseek returns -1 and sets errno. If successful, it returns the new filepointer offset.

Notes
Not all file descriptors are capable of supporting the seek operation, e.g., a descriptor associated with a pipe.
By using SEEK_END with a positive value of the ofs parameter, it is possible to position the file pointer beyond the current size of the file.

See Also
13.15 (fseek), 13.16 (ftell), 14.9 (open), 14.1 (close)

14.9 open

Synopsis
Open a file

Usage
FD_Type open (String_Type filename, Int_Type flags [,Int_Type mode])

Description
The open function attempts to open a file specified by the filename parameter according to the flags parameter, which must be one of the following values:

0_RDONLY (read-only)
0_WRONLY (write-only)
0_RDWR (read/write)

In addition, flags may also be bitwise-or'd with any of the following:

0_BINAURY (open the file in binary mode)
0_TEXT (open the file in text mode)
0_CREAT (create the file if it does not exist)
0_EXCL (fail if the file already exists)
0_NOCTTY (do not make the device the controlling terminal)
0_TRUNC (truncate the file if it exists)
0_APPEND (open the file in append mode)
0_NONBLOCK (open the file in non-blocking mode)

Some of these flags make sense only when combined with other flags. For example, if O_CREAT is used, then O_CREAT must also be specified, otherwise unpredictable behavior may result.

If 0_CREAT is used for the flags parameter then the mode parameter must be present. mode specifies the permissions to use if a new file is created. The actual file permissions will be affected by the process’s umask via mode & ~umask. The mode parameter’s value is constructed via bitwise-or of the following values:

S_IRWXU (Owner has read/write/execute permission)
S_IRUSR (Owner has read permission)
S_IWUSR (Owner has write permission)
S_IXUSR (Owner has execute permission)
S_IRWXG (Group has read/write/execute permission)
S_IRGRP (Group has read permission)
S_IWGRP (Group has write permission)
S_IXGRP (Group has execute permission)
S_IRWXO (Others have read/write/execute permission)
S_IROTH (Others have read permission)
S_IWOTH (Others have write permission)
S_IXOTH (Others have execute permission)

Upon success open returns a file descriptor object (FD_Type), otherwise NULL is returned and errno is set.

Notes

If you are not familiar with the open system call, then it is recommended that you use fopen instead and use the higher level stdio interface.

See Also

13.9 (fopen), 14.1 (close), 14.10 (read), 14.11 (write), 16.15 (stat_file)

14.10 read

Synopsis

Read from an open file descriptor

Usage

UInt_Type read (FD_Type fd, Ref_Type buf, UInt_Type num)

Description

The read function attempts to read at most num bytes into the variable indicated by buf from the open file descriptor fd. It returns the number of bytes read, or -1 upon failure and sets errno. The number of bytes read may be less than num, and will be zero if an attempt is made to read past the end of the file.

Notes

read is a low-level function and may return -1 for a variety of reasons. For example, if non-blocking I/O has been specified for the open file descriptor and no data is available for reading then the function will return -1 and set errno to EAGAIN.

See Also

13.13 (fread), 14.9 (open), 14.1 (close), 14.11 (write)

14.11 write

Synopsis

Write to an open file descriptor
Usage

UInt_Type write (FD_Type fd, BString_Type buf)

Description

The write function attempts to write the bytes specified by the buf parameter to the open file descriptor fd. It returns the number of bytes successfully written, or -1 and sets errno upon failure. The number of bytes written may be less than length(buf).

See Also

14.10 (read), 13.17 (fwrite), 14.9 (open), 14.1 (close)
Chapter 15

Signal Functions

15.1 alarm

Synopsis
Schedule an alarm signal

Usage
alarm (UInt_Type secs [, Ref_Type secs_remaining])

Description
The alarm function schedules the delivery of a SIGALRM signal in secs seconds. Any previously scheduled alarm will be canceled. If secs is zero, then no new alarm will be scheduled. If the second argument is present, then it must be a reference to a variable whose value will be set upon return to the number of seconds remaining for a previously scheduled alarm to take place.

Example
This example demonstrates how the alarm function may be used to read from stdin within a specified amount of time:

```c
# define sigalrm_handler (sig)
{
    throw ReadError, "Read timed out";
}
#define read_or_timeout (secs)
{
    variable line, err;
    signal (SIGALRM, &sigalrm_handler);
    () = fputs ("Enter some text> ", stdout); () = fflush (stdout);
    alarm (secs);
    try (err)
    {
        if (-1 == fgets (&line, stdin))
            throw ReadError, "Failed to read from stdin";
    }
```
catch IOError:
    {
        message (err.message);
        return NULL;
    }
return line;

Notes

Some operating systems may implement the sleep function using alarm. As a result, it is not a good idea to mix calls to alarm and sleep.
The default action for SIGALRM is to terminate the process. Hence, if alarm is called it is wise to establish a signal handler for SIGALRM.

See Also

15.4 (signal), 18.20 (sleep), 15.3 (setitimer), 15.2 (getitimer)

15.2 getitimer

Synopsis

Get the value of an interval timer

Usage

(secs, period) = getitimer (Int_Type timer)

Description

This function returns the value of the specified interval timer as a pair of double precision values: period and secs.
The value of secs indicates the number of seconds remaining before the timer expires. A value of 0 for secs indicates that the timer is inactive. The value of period indicates the periodicity of the timer. That is, when the timer goes off, it will automatically be reset to go off again after period seconds.

There are 3 interval timers available: ITIMER_REAL, ITIMER_VIRTUAL, and ITIMER_PROF.
The ITIMER_REAL timer operates in real time and when the time elapses, a SIGALRM will be sent to the process.
The ITIMER_VIRTUAL timer operates in the virtual time of the process; that is, when process is actively running. When it elapses, SIGVTALRM will be sent to the process.
The ITIMER_PROF operates when the process is actively running, or when the kernel is performing a task on behalf of the process. It sends a SIGPROF signal to the process.

Notes

The interaction between these timers and the sleep and alarm functions is OS dependent.
The resolution of a timer is system dependent; typical values are on the order of milliseconds.

See Also

15.3 (setitimer), 15.1 (alarm), 15.4 (signal)
15.3 setitimer

Synopsis
Set the value of an interval timer

Usage
setitimer (Int_Type timer, secs [, period] [,old_secs, &old_period])

Description
This function sets the value of a specified interval timer, and optionally returns the previous value. The value of the timer argument must be one of the 3 interval timers ITIMER_REAL, ITIMER_VIRTUAL, or ITIMER_PROF. See the documentation for the getitimer function for information about the semantics of these timers.

The value of the secs parameter specifies the expiration time for the timer. If this value is 0, the timer will be disabled. Unless a non-zero value for the optional period parameter is given, the timer will be disabled after it expires. Otherwise, the timer will reset to go off with a period of period seconds.

The final two optional arguments are references to variables that will be set to the previous values associated with the timer.

See Also
15.2 (getitimer), 15.1 (alarm), 15.4 (signal)

15.4 signal

Synopsis
Establish a signal handler

Usage
signal (Int_Type sig, Ref_Type func [,Ref_Type old_func])

Description
The signal function assigns the signal handler represented by func to the signal sig. Here func is usually reference to a function that takes an integer argument (the signal) and returns nothing, e.g.,

   define signal_handler (sig)
   {
     return;
   }

Alternatively, func may be given by one of the symbolic constants SIG_IGN or SIG_DFL to indicate that the signal is to be ignored or given its default action, respectively.

The first parameter, sig, specifies the signal to be handled. The actual supported values vary with the OS. Common values on Unix include SIGHUP, SIGINT, and SIGTERM.

If a third argument is present, then it must be a reference to a variable whose value will be set to the value of the previously installed handler.
Example

This example establishes a handler for SIGTSTP.

```c
static define sig_suspend (); % forward declaration
static define sig_suspend (sig)
{
    message ("SIGTSTP received-- stopping");
    signal (sig, SIG_DFL);
    () = kill (getpid(), SIGSTOP);
    message ("Resuming");
    signal (sig, &sig_suspend);
}
signal (SIGTSTP, &sig_suspend);
```

Notes

Currently the signal interface is supported only on systems that implement signals according to the POSIX standard.

Once a signal has been received, it will remain blocked until after the signal handler has completed. This is the reason SIGSTOP was used in the above signal handler instead of SIGTSTP.

See Also

15.1 (alarm), 15.6 (sigsuspend), 15.5 (sigprocmask)

15.5 sigprocmask

Synopsis

Change the list of currently blocked signals

Usage

```c
sigprocmask (Int_Type how, Array_Type mask [,Ref_Type old_mask])
```

Description

The `sigprocmask` function may be used to change the list of signals that are currently blocked. The first parameter indicates how this is accomplished. Specifically, `how` must be one of the following values: SIG_BLOCK, SIG_UNBLOCK, or SIG_SETMASK.

If `how` is SIG_BLOCK, then the set of blocked signals will be the union the current set with the values specified in the `mask` argument.

If `how` is SIG_UNBLOCK, then the signals specified by the `mask` parameter will be removed from the currently blocked set.

If `how` is SIG_SETMASK, then the set of blocked signals will be set to those given by the `mask`.

If a third argument is present, then it must be a reference to a variable whose value will be set to the previous signal mask.

See Also

15.4 (signal), 15.6 (sigsuspend), 15.1 (alarm)
15.6 sigsuspend

Synopsis
Suspend the process until a signal is delivered

Usage
sigsuspend ([Array_Type signal_mask])

Description
The sigsuspend function suspends the current process until a signal is received. An optional array argument may be passed to the function to specify a list of signals that should be temporarily blocked while waiting for a signal.

Example
The following example pauses the current process for 10 seconds while blocking the SIGHUP and SIGINT signals.

```c
static variable Tripped;
define sigalrm_handler (sig)
{
    Tripped = 1;
}
signal (SIGALRM, &sigalrm_handler);
Tripped = 0;
alarm (10);
while (Tripped == 0) sigsuspend ([SIGHUP, SIGINT]);
```

Note that in this example the call to sigsuspend was wrapped in a while-loop. This was necessary because there is no guarantee that another signal would not cause sigsuspend to return.

See Also
15.4 (signal), 15.1 (alarm), 15.5 (sigprocmask)
Chapter 16

Directory Functions

16.1  access

Synopsis

Check to see if a file is accessible

Usage

Int_Type access (String_Type pathname, Int_Type mode)

Description

This function checks to see if the current process has access to the specified pathname. The
mode parameter determines the type of desired access. Its value is given by the bitwise-or of
one or more of the following constants:

  R_OK  Check for read permission
  W_OK  Check for write permission
  X_OK  Check for execute permission
  F_OK  Check for existence

The function will return 0 if process has the requested access permissions to the file, otherwise
it will return -1 and set errno accordingly.

Access to a file depend not only upon the file itself, but also upon the permissions of each of
the directories in the pathname. The checks are done using the real user and group ids of the
process, and not using the effective ids.

See Also

  16.15 (stat_file)

16.2  chdir

Synopsis

Change the current working directory
16.3 **chmod**

**Synopsis**
Change the mode of a file

**Usage**

```
Int_Type chmod (String_Type file, Int_Type mode)
```

**Description**

The `chmod` function changes the permissions of the specified file to those given by `mode`. It returns 0 upon success, or -1 upon failure setting `errno` accordingly. See the system specific documentation for the C library function `chmod` for a discussion of the `mode` parameter.

**See Also**

16.4 (chown), 16.15 (stat_file)

16.4 **chown**

**Synopsis**
Change the owner of a file

**Usage**

```
Int_Type chown (String_Type file, Int_Type uid, Int_Type gid)
```

**Description**

The `chown` function is used to change the user-id and group-id of file to `uid` and `gid`, respectively. It returns 0 upon success and -1 upon failure, with `errno` set accordingly.

**Notes**

On most systems, only the superuser can change the ownership of a file.

Some systems do not support this function.

**See Also**

16.7 (lchown), 16.3 (chmod), 16.15 (stat_file)
16.5 getcwd

Synopsis
Get the current working directory

Usage
String_Type getcwd ()

Description
The getcwd function returns the absolute pathname of the current working directory. If an error occurs or it cannot determine the working directory, it returns NULL and sets errno accordingly.

Notes
Under Unix, OS/2, and MSDOS, the pathname returned by this function includes the trailing slash character. It may also include the drive specifier for systems where that is meaningful.

See Also
16.10 (mkdir), 16.2 (chdir), 10.1 (errno)

16.6 hardlink

Synopsis
Create a hard-link

Usage
Int_Type hardlink (String_Type oldpath, String_Type newpath)

Description
The hardlink function creates a hard-link called newpath to the existing file oldpath. If the link was successfully created, the function will return 0. Upon error, the function returns -1 and sets errno accordingly.

Notes
Not all systems support the concept of a hard-link.

See Also
16.18 (symlink)

16.7 lchown

Synopsis
Change the owner of a file

Usage
Int_Type lchown (String_Type file, Int_Type uid, Int_Type gid)
Description

The lchown function is like chown, except that it does not dereference a symbolic link. Hence, it may be used to change the ownership of a symbolic link itself, and not to what it references. See the documentation for the chown function for more details.

See Also

16.4 (chown), 16.3 (chmod), 16.15 (stat_file)

16.8 listdir

Synopsis

Get a list of the files in a directory

Usage

String_Type[] listdir (String_Type dir)

Description

The listdir function returns the directory listing of all the files in the specified directory dir as an array of strings. It does not return the special files ".." and "." as part of the list.

See Also

16.15 (stat_file), 16.16 (stat_is), 2.13 (length)

16.9 lstat_file

Synopsis

Get information about a symbolic link

Usage

Struct_Type lstat_file (String_Type file)

Description

The lstat_file function behaves identically to stat_file but if file is a symbolic link, lstat_file returns information about the link itself, and not the file that it references. See the documentation for stat_file for more information.

Notes

On systems that do not support symbolic links, there is no difference between this function and the stat_file function.

See Also

16.15 (stat_file), 16.16 (stat_is), 16.17 (stat_mode_to_string), 16.11 (readlink)
16.10 mkdir

Synopsis

Create a new directory

Usage

`Int_Type mkdir (String_Type dir [,Int_Type mode])`

Description

The `mkdir` function creates a directory whose name is specified by the `dir` parameter with permissions given by the optional `mode` parameter. Upon success `mkdir` returns 0, or it returns -1 upon failure setting `errno` accordingly. In particular, if the directory already exists, the function will fail and set `errno` to `EEXIST`.

Example

The following function creates a new directory, if it does not already exist (indicated by `errno==EEXIST`).

```c
define my_mkdir (dir)
{
    if (0 == mkdir (dir)) return;
    if (errno == EEXIST) return;
    throw IOError,
        sprintf ("mkdir %s failed: %s", dir, errno_string (errno));
}
```

Notes

The `mode` parameter may not be meaningful on all systems. On systems where it is meaningful, the actual permissions on the newly created directory are modified by the process’s umask.

See Also

`16.14 (rmdir), 16.5 (getcwd), 16.2 (chdir), 13.9 (fopen), 10.1 (errno)`

16.11 readlink

Synopsis

`String_Type readlink (String_Type path)`

Usage

Get the value of a symbolic link

Description

The `readlink` function returns the value of a symbolic link. Upon failure, `NULL` is returned and `errno` set accordingly.

Notes

Not all systems support this function.
See Also

16.18 (symlink), 16.9 (lstat_file), 16.15 (stat_file), 16.16 (stat_is)

16.12 remove

Synopsis

Delete a file

Usage

Int_Type remove (String_Type file)

Description

The remove function deletes a file. It returns 0 upon success, or -1 upon error and sets errno accordingly.

See Also

16.13 (rename), 16.14 (rmdir)

16.13 rename

Synopsis

Rename a file

Usage

Int_Type rename (String_Type old, String_Type new)

Description

The rename function renames a file from old to new moving it between directories if necessary. This function may fail if the directories are not on the same file system. It returns 0 upon success, or -1 upon error and sets errno accordingly.

See Also

16.12 (remove), 10.1 (errno)

16.14 rmdir

Synopsis

Remove a directory

Usage

Int_Type rmdir (String_Type dir)

Description

The rmdir function deletes the specified directory. It returns 0 upon success or -1 upon error and sets errno accordingly.
Notes
The directory must be empty before it can be removed.

See Also
16.13 (rename), 16.12 (remove), 16.10 (mkdir)

16.15 stat_file

Synopsis
Get information about a file

Usage
Struct_Type stat_file (String_Type file)

Description
The stat_file function returns information about file through the use of the system stat
call. If the stat call fails, the function returns NULL and sets errno accordingly. If it is successful,
it returns a stat structure with the following integer-value fields:

st_dev
st_ino
st_mode
st_nlink
st_uid
st_gid
st_rdev
st_size
st_atime
st_mtime
st_ctime

See the C library documentation of stat for a discussion of the meanings of these fields.

Example
The following example shows how the stat_file function may be used to get the size of a file:

define file_size (file)
{
    variable st;
    st = stat_file(file);
    if (st == NULL)
        throw IOError, "Unable to stat $file";
    return st.st_size;
}

See Also
16.9 (lstat_file), 16.16 (stat_is), 16.17 (stat_mode_to_string), 16.19 (utime)
16.16  stat_is

Synopsis
Parse the st_mode field of a stat structure

Usage
Char_Type stat_is (String_Type type, Int_Type st_mode)

Description
The stat_is function returns a boolean value according to whether or not the st_mode parameter is of the specified type. Specifically, type must be one of the strings:

   "sock"  (socket)
   "fifo"  (fifo)
   "blk"   (block device)
   "chr"   (character device)
   "reg"   (regular file)
   "lnk"   (link)
   "dir"   (dir)

It returns a non-zero value if st_mode corresponds to type.

Example
The following example illustrates how to use the stat_is function to determine whether or not a file is a directory:

define is_directory (file)
{
    variable st;

    st = stat_file (file);
    if (st == NULL) return 0;
    return stat_is ("dir", st.st_mode);
}

See Also
16.15 (stat_file), 16.9 (lstat_file), 16.17 (stat_mode_to_string)

16.17  stat_mode_to_string

Synopsis
Format the file type code and access permission bits as a string

Usage
String_Type stat_mode_to_string (Int_Type st_mode)

Description
The stat_mode_to_string function returns a 10 characters string that indicates the type and permissions of a file as represented by the st_mode parameter. The returned string consists of the following characters:
"s"  (socket)
"p"  (fifo)
"b"  (block device)
"c"  (character device)
"-"  (regular file)
"l"  (link)
"d"  (dir)

The access permission bit is one of the following characters:

"s"  (set-user-id)
"w"  (writable)
"x"  (executable)
"r"  (readable)

Notes

This function is an slsh intrinsic. As such, it is not part of S-Lang proper.

See Also

16.15 (stat_file), 16.9 (lstat_file), 16.16 (stat_is)

16.18 symlink

Synopsis

Create a symbolic link

Usage

Int_Type symlink (String_Type oldpath, String_Type new_path)

Description

The symlink function may be used to create a symbolic link named new_path for oldpath. If successful, the function returns 0, otherwise it returns -1 and sets errno appropriately.

Notes

This function is not supported on all systems and even if supported, not all file systems support the concept of a symbolic link.

See Also

16.11 (readlink), 16.6 (hardlink)

16.19 utime

Synopsis

Change a file's last access and modification times

Usage

Int_Type utime(String_Type file, Double_Type actime, Double_Type modtime)
Chapter 16. Directory Functions

Description

This function may be used to change the last access (actime) and last modification (modtime) times associated with the specified file. If successful, the function returns 0; otherwise it returns -1 and sets errno accordingly.

Notes

The utime function will call the C library utimes function if available, which permits microsecond accuracy. Otherwise, it will truncate the time arguments to integers and call the utime function.

See Also

16.15 (stat_file)
Chapter 17

Functions that Parse Filenames

17.1 path_basename

Synopsis
Get the basename part of a filename

Usage
String_Type path_basename (String_Type filename)

Description
The path_basename function returns the basename associated with the filename parameter. The basename is the non-directory part of the filename, e.g., on unix c is the basename of /a/b/c.

See Also
17.4 (path_dirname), 17.5 (path_extname), 17.3 (path_concat), 17.7 (path_is_absolute)

17.2 path_basename_sans_extname

Synopsis
Get the basename part of a filename but without the extension

Usage
String_Type path_basename_sans_extname (String_Type path)

Description
The path_basename_sans_extname function returns the basename associated with the filename parameter, omitting the extension if present. The basename is the non-directory part of the filename, e.g., on unix c is the basename of /a/b/c.

See Also
17.4 (path_dirname), 17.1 (path_basename), 17.5 (path_extname), 17.3 (path_concat), 17.7 (path_is_absolute)
17.3  path_concat

Synopsis
   Combine elements of a filename

Usage
   String_Type path_concat (String_Type dir, String_Type basename)

Description
   The path_concat function combines the arguments dir and basename to produce a filename. For example, on Unix if dir is x/y and basename is z, then the function will return x/y/z.

See Also
   17.4 (path_dirname), 17.1 (path_basename), 17.5 (path_extname), 17.7 (path_is_absolute)

17.4  path_dirname

Synopsis
   Get the directory name part of a filename

Usage
   String_Type path_dirname (String_Type filename)

Description
   The path_dirname function returns the directory name associated with a specified filename.

Notes
   On systems that include a drive specifier as part of the filename, the value returned by this function will also include the drive specifier.

See Also
   17.1 (path_basename), 17.5 (path_extname), 17.3 (path_concat), 17.7 (path_is_absolute)

17.5  path_extname

Synopsis
   Return the extension part of a filename

Usage
   String_Type path_extname (String_Type filename)

Description
   The path_extname function returns the extension portion of the specified filename. If an extension is present, this function will also include the dot as part of the extension, e.g., if filename is "file.c", then this function will return "\'.c\". If no extension is present, the function returns an empty string "".


17.6  path_get_delimiter

Synopsis
Get the value of a search-path delimiter

Usage
Char_Type path_get_delimiter ()

Description
This function returns the value of the character used to delimit fields of a search-path.

See Also
19.7 (set_slang_load_path), 19.6 (get_slang_load_path)

17.7  path_is_absolute

Synopsis
Determine whether or not a filename is absolute

Usage
Int_Type path_is_absolute (String_Type filename)

Description
The path_is_absolute function will return non-zero if filename refers to an absolute file- name, otherwise it returns zero.

See Also
17.4 (path_dirname), 17.1 (path_basename), 17.5 (path_extname), 17.3 (path_concat)

17.8  path_sans_extname

Synopsis
Strip the extension from a filename

Usage
String_Type path_sans_extname (String_Type filename)

Notes
Under VMS, the file version number is not returned as part of the extension.

See Also
17.8 (path_sans_extname), 17.4 (path_dirname), 17.1 (path_basename), 17.3 (path_concat), 17.7 (path_is_absolute)
Description

The `path_sans_extname` function removes the file name extension (including the dot) from the filename and returns the result.

See Also

17.2 (path_basename_sans_extname), 17.5 (path_extname), 17.1 (path_basename), 17.4 (path_dirname), 17.3 (path_concat)
Chapter 18

System Call Functions

18.1 getegid

Synopsis
   Get the effective group id of the current process

Usage
   Int_Type getegid ()

Description
   The getegid function returns the effective group ID of the current process.

Notes
   This function is not supported by all systems.

See Also
   18.3 (getgid), 18.2 (geteuid), 18.15 (setgid)

18.2 geteuid

Synopsis
   Get the effective user-id of the current process

Usage
   Int_Type geteuid ()

Description
   The geteuid function returns the effective user-id of the current process.

Notes
   This function is not supported by all systems.

See Also
   18.11 (getuid), 18.19 (setuid), 18.15 (setgid)
18.3 getgid

Synopsis
Get the group id of the current process

Usage
`Integer_Type getgid ()`

Description
The `getgid` function returns the real group id of the current process.

Notes
This function is not supported by all systems.

See Also
18.6 (getpid), 18.7 (getppid)

18.4 getpgid

Synopsis
Get the process group id

Usage
`Int_Type getpgid (Int_Type pid)`

Description
The `getpgid` function returns the process group id of the process whose process is `pid`. If `pid` is 0, then the current process will be used.

Notes
This function is not supported by all systems.

See Also
18.5 (getpgrp), 18.6 (getpid), 18.7 (getppid)

18.5 getpgrp

Synopsis
Get the process group id of the calling process

Usage
`Int_Type getpgrp ()`

Description
The `getpgrp` function returns the process group id of the current process.
18.6  getpid

Synopsis

Get the current process id

Usage

Integer_Type getpid ()

Description

The getpid function returns the current process identification number.

See Also

18.4 (getpgid), 18.6 (getpid), 18.7 (getppid)

18.7  getppid

Synopsis

Get the parent process id

Usage

Integer_Type getppid ()

Description

The getppid function returns the process identification number of the parent process.

Notes

This function is not supported by all systems.

See Also

18.6 (getpid), 18.3 (getgid)

18.8  getpriority

Synopsis

Get a process’s scheduling priority

Usage

result = getpriority (which, who)
Description

The `setpriority` function may be used to obtain the kernel's scheduling priority for a process, process group, or a user depending upon the values of the `which` and `who` parameters. Specifically, if the value of `which` is `PRIO_PROCESS`, then the value of `who` specifies the process id of the affected process. If `which` is `PRIO_PGRP`, then `who` specifies a process group id. If `which` is `PRIO_USER`, then the value of `who` is interpreted as a user id. For the latter two cases, where `which` refers to a set of processes, the value returned corresponds to the highest priority of a process in the set. A value of 0 may be used for `who` to denote the process id, process group id, or real user ID of the current process.

Upon success, the function returns the specified priority value. If an error occurs, the function will return `NULL` with `errno` set accordingly.

See Also

18.17 (setpriority), 18.6 (getpid), 18.7 (getppid)

18.9 getrusage

Synopsis

Get process resource usage

Usage

`Struct_Type getrusage ([Int_Type who])`

Description

This function returns a structure whose fields contain information about the resource usage of calling process, summed over all threads of the process. The optional integer argument `who` may be used to obtain resource usage of child processes, or of the calling thread itself. Specifically, the optional integer argument `who` may take on one of the following values:

RUSAGE_SELF (default)
RUSAGE_CHILDREN

If `RUSAGE_CHILDREN` is specified, then the process information will be the sum of all descendants of the calling process that have terminated and have been waited for (via, e.g., `waitpid`). It will not contain any information about child processes that have not terminated.

The structure that is returned will contain the following fields:

ru_utime
ru_stime
ru_majflt
ru_minflt
ru_nivcsw
ru_nvcsw
ru_rss
ru_idrss
ru_ixrss
ru_idrss
ru_isrss integral unshared stack size
ru_swap swaps
ru_msgsnd IPC messages sent
ru_msgrcv IPC messages received
ru_nsignals signals received

Some of the fields may not be supported for a particular OS or kernel version. For example, on Linux the 2.6.32 kernel supports only the following fields:

ru_utimesecs
ru_stimesecs
ru_maxrss (since Linux 2.6.32)
ru_minflt
ru_majflt
ru_inblock (since Linux 2.6.22)
ru_outblock (since Linux 2.6.22)
ru_nvcsw (since Linux 2.6)
ru_nivcsw (since Linux 2.6)

Notes
The underlying system call returns the CPU user and system times as C struct timeval objects. For convenience, the interpreter interface represents these objects as double precision floating point values.

See Also
11.11 (times)

18.10 getsid

Synopsis
get the session id of a process

Usage
Int_Type getsid ([Int_Type pid])

Description
The getsid function returns the session id of the current process. If the optional integer pid argument is given, then the function returns the session id of the specified process id.

See Also
18.18 (setsid), 18.6 (getpid), 18.6 (getpid)

18.11 getuid

Synopsis
Get the user-id of the current process
Usage

Int_Type getuid ()

Description

The getuid function returns the user-id of the current process.

Notes

This function is not supported by all systems.

See Also

18.11 (getuid), 18.1 (getegid)

18.12 kill

Synopsis

Send a signal to a process

Usage

Integer_Type kill (Integer_Type pid, Integer_Type sig)

Description

This function may be used to send a signal given by the integer sig to the process specified by pid. The function returns zero upon success or -1 upon failure setting errno accordingly.

Example

The kill function may be used to determine whether or not a specific process exists:

```c
#define process_exists (pid)
{
    if (-1 == kill (pid, 0))
        return 0;  /* Process does not exist */
    return 1;
}
```

Notes

This function is not supported by all systems.

See Also

18.13 (killpg), 18.6 (getpid)

18.13 killpg

Synopsis

Send a signal to a process group

Usage

Integer_Type killpg (Integer_Type pgid, Integer_Type sig)
Description

This function may be used to send a signal given by the integer `sig` to the process group specified by `pgrp`. The function returns zero upon success or -1 upon failure setting `errno` accordingly.

Notes

This function is not supported by all systems.

See Also

18.12 (kill), 18.6 (getpid)

18.14 mkfifo

Synopsis

Create a named pipe

Usage

```
Int_Type mkfifo (String_Type name, Int_Type mode)
```

Description

The `mkfifo` attempts to create a named pipe with the specified name and mode (modified by the process’s umask). The function returns 0 upon success, or -1 and sets `errno` upon failure.

Notes

Not all systems support the `mkfifo` function and even on systems that do implement the `mkfifo` system call, the underlying file system may not support the concept of a named pipe, e.g., an NFS filesystem.

See Also

16.15 (stat_file)

18.15 setgid

Synopsis

Set the group-id of the current process

Usage

```
Int_Type setgid (Int_Type gid)
```

Description

The `setgid` function sets the effective group-id of the current process. It returns zero upon success, or -1 upon error and sets `errno` appropriately.

Notes

This function is not supported by all systems.

See Also

18.3 (getgid), 18.19 (setuid)
18.16 setpgid

Synopsis
Set the process group-id

Usage
Int_Type setpgid (Int_Type pid, Int_Type gid)

Description
The setpgid function sets the group-id gid of the process whose process-id is pid. If pid is 0, then the current process-id will be used. If pgid is 0, then the pid of the affected process will be used.

If successful 0 will be returned, otherwise the function will return -1 and set errno accordingly.

Notes
This function is not supported by all systems.

See Also
18.15 (setgid), 18.19 (setuid)

18.17 setpriority

Synopsis
Set the scheduling priority for a process

Usage
Int_Type setpriority (which, who, prio)

Description
The setpriority function may be used to set the kernel’s scheduling priority for a process, process group, or a user depending upon the values of the which and who parameters. Specifically, if the value of which is PRI0_PROCESS, then the value of who specifies the process id of the affected process. If which is PRI0_PGRP, then who specifies a process group id. If which is PRI0_USER, then the value of who is interpreted as a user id. A value of 0 may be used for who to denote the process id, process group id, or real user ID of the current process.

Upon success, the setpriority function returns 0. If an error occurs, -1 is returned and errno will be set accordingly.

Example
The getpriority and setpriority functions may be used to implement a nice function for incrementing the priority of the current process as follows:

```c
define nice (dp)
{
    variable p = getpriority (PRI0_PROCESS, 0);
    if (p == NULL)
        return -1;
```
variable s = setpriority (PRI0_PROCESS, 0, p + dp);
    if (s == -1)
        return -1;
    return getpriority (PRI0_PROCESS, 0);
}

Notes
Priority values are sometimes called "nice" values. The actual range of priority values is system dependent but commonly range from -20 to 20, with -20 being the highest scheduling priority, and +20 the lowest.

See Also
18.8 (getpriority), 18.6 ( getpid )

18.18 setsid

Synopsis
Create a new session for the current process

Usage
    Int_Type setsid ()

Description
If the current process is not a session leader, the setsid function will create a new session and make the process the session leader for the new session. It returns the the process group id of the new session.
Upon failure, -1 will be returned and errno set accordingly.

See Also
18.10 (getsid), 18.16 (setpgid)

18.19 setuid

Synopsis
Set the user-id of the current process

Usage
    Int_Type setuid (Int_Type id)

Description
The setuid function sets the effective user-id of the current process. It returns zero upon success, or -1 upon error and sets errno appropriately.

Notes
This function is not supported by all systems.

See Also
18.15 (setgid), 18.16 (setpgid), 18.11 (getuid), 18.2 (geteuid)
18.20 sleep

Synopsis

Pause for a specified number of seconds

Usage

sleep (Double_Type n)

Description

The sleep function delays the current process for the specified number of seconds. If it is interrupted by a signal, it will return prematurely.

Notes

Not all system support sleeping for a fractional part of a second.

18.21 system

Synopsis

Execute a shell command

Usage

Integer_Type system (String_Type cmd)

Description

The system function may be used to execute the string expression cmd in an inferior shell. This function is an interface to the C system function which returns an implementation-defined result. On Linux, it returns 127 if the inferior shell could not be invoked, -1 if there was some other error, otherwise it returns the return code for cmd.

Example

    define dir ()
    {
        () = system ("DIR");
    }

displays a directory listing of the current directory under MSDOS or VMS.

See Also

18.22 (system_intr), ?? (new_process), 13.19 (popen)

18.22 system_intr

Synopsis

Execute a shell command
18.23. umask

Usage

Integer_Type system_intr (String_Type cmd)

Description

The `system_intr` function performs the same task as the `system` function, except that the SIGINT signal will not be ignored by the calling process. This means that if a **S-Lang** script calls `system_intr` function, and Ctrl-C is pressed, both the command invoked by the `system_intr` function and the script will be interrupted. In contrast, if the command were invoked using the `system` function, only the command called by it would be interrupted, but the script would continue executing.

See Also

18.21 (system), ?? (new_process), 13.19 (popen)

18.23 umask

Synopsis

Set the file creation mask

Usage

Int_Type umask (Int_Type m)

Description

The `umask` function sets the file creation mask to the value of `m` and returns the previous mask.

See Also

16.15 (stat_file)

18.24 uname

Synopsis

Get the system name

Usage

Struct_Type uname ()

Description

The `uname` function returns a structure containing information about the operating system. The structure contains the following fields:

- **sysname** (Name of the operating system)
- **nodename** (Name of the node within the network)
- **release** (Release level of the OS)
- **version** (Current version of the release)
- **machine** (Name of the hardware)
Notes

Not all systems support this function.

See Also

25.8 (getenv)
Chapter 19

Eval Functions

19.1 \$_$

Synopsis
Expand the dollar-escaped variables in a string

Usage
String_Type _$(String_Type s)

Description
This function expands the dollar-escaped variables in a string and returns the resulting string.

Example
Consider the following code fragment:

```plaintext
private variable Format = "/tmp/foo-$time.$pid";
define make_filename ()
{
    variable pid = getpid ();
    variable time = _time ();
    return _$(Format);
}
```

Note that the variable Format contains dollar-escaped variables, but because the $ suffix was omitted from the string literal, the variables are not expanded. Instead expansion is deferred until execution of the make_filename function through the use of the _$ function.

See Also
19.4 (eval), 25.8 (getenv)

19.2 autoload

Synopsis
Load a function from a file
Chapter 19. Eval Functions

Usage

autoload (String_Type funct, String_Type file)

Description

The autoload function is used to declare funct to the interpreter and indicate that it should be loaded from file when it is actually used. If funct contains a namespace prefix, then the file will be loaded into the corresponding namespace. Otherwise, if the autoload function is called from an execution namespace that is not the Global namespace nor an anonymous namespace, then the file will be loaded into the execution namespace.

Example

Suppose bessel_j0 is a function defined in the file bessel.sl. Then the statement

   autoload ("bessel_j0", "bessel.sl");

will cause bessel.sl to be loaded prior to the execution of bessel_j0.

See Also

19.5 (evaf ile), 21.2 (import)

19.3 byte_compile_file

Synopsis

Compile a file to byte-code for faster loading.

Usage

byte_compile_file (String_Type file, Int_Type method)

Description

The byte_compile_file function byte-compiles file producing a new file with the same name except a 'c' is added to the output file name. For example, file is "site.sl", then this function produces a new file named site.slc.

Notes

The method parameter is not used in the current implementation, but may be in the future. For now, set it to 0.

See Also

19.5 (evaf ile)

19.4 eval

Synopsis

Interpret a string as S-Lang code.

Usage

eval (String_Type expression [,String_Type namespace])
Description
The `eval` function parses a string as S-Lang code and executes the result. If called with the optional namespace argument, then the string will be evaluated in the specified namespace. If that namespace does not exist it will be created first.

This is a useful function in many contexts including those where it is necessary to dynamically generate function definitions.

Example
```
if (0 == is_defined("my_function"))
    eval("define my_function () { message("my_function"); }");
```

See Also
8.9 (is_defined), 19.2 (autoload), 19.5 (evalfile)

19.5 evalfile

Synopsis
Interpret a file containing S-Lang code

Usage
```
Int_Type evalfile (String_Type file [, String_Type namespace])
```

Description
The `evalfile` function loads `file` into the interpreter and executes it. If called with the optional namespace argument, the file will be loaded into the specified namespace, which will be created if necessary. If given no namespace argument and the file has already been loaded, then it will be loaded again into an anonymous namespace. A namespace argument given by the empty string will also cause the file to be loaded into a new anonymous namespace.

If no errors were encountered, 1 will be returned; otherwise, a S-Lang exception will be thrown and the function will return zero.

Example
```
define load_file (file)
{
    try
    {
        () = evalfile (file);
    }
    catch AnyError;
}
```

Notes
For historical reasons, the return value of this function is not really useful.

The file is searched along an application-defined load-path. The `get_slang_load_path` and `set_slang_load_path` functions may be used to set and query the path.
19.6 get_slang_load_path

Synopsis
Get the value of the interpreter's load-path

Usage
String_Type get_slang_load_path ()

Description
This function retrieves the value of the delimiter-separated search path used for loading files. The delimiter is OS-specific and may be queried using the path_get_delimiter function.

Notes
Some applications may not support the built-in load-path searching facility provided by the underlying library.

See Also
19.7 (set_slang_load_path), 17.6 (path_get_delimiter)

19.7 set_slang_load_path

Synopsis
Set the value of the interpreter's load-path

Usage
set_slang_load_path (String_Type path)

Description
This function may be used to set the value of the delimiter-separated search path used by the evalfile and autoload functions for locating files. The delimiter is OS-specific and may be queried using the path_get_delimiter function.

Example

```c
public define prepend_to_slang_load_path (p)
{
    variable s = stat_file (p);
    if (s == NULL) return;
    if (0 == stat_is ("dir", s.st_mode))
        return;

    p = sprintf ("%s%c%s", p, path_get_delimiter (), get_slang_load_path ());
    set_slang_load_path (p);
}
```
Notes

Some applications may not support the built-in load-path searching facility provided by the underlying library.

See Also

19.6 (get_slang_load_path), 17.6 (path_get_delimiter), 19.5 (evalfile), 19.2 (autoload)
Chapter 20

Qualifier Functions

20.1 qualifier

Synopsis
Get the value of a qualifier

Usage
value = qualifier (String_Type name [,default_value])

Description
This function may be used to get the value of a qualifier. If the specified qualifier does not exist, NULL will be returned, unless a default value has been provided.

Example

```c
define echo (text)
{
    variable fp = qualifier ("out"; stdout);
    () = fputs (text, fp);
}

echo ("hello"); % writes hello to stdout
echo ("hello"; out=stderr); % writes hello to stderr
```

Notes
Since NULL is a valid value for a qualifier, this function is unable to distinguish between a non-existent qualifier and one whose value is NULL. If such a distinction is important, the qualifier_exists function can be used. For example,

```c
define echo (text)
{
    variable fp = stdout;
    if (qualifier_exists ("use_stderr"))
        fp = stderr;
    () = fputs (text, fp);
}

echo ("hello"; use_stderr); % writes hello to stderr
```
In this case, no value was provided for the `use_stderr` qualifier: it exists but has a value of `NULL`.

See Also

20.3 (qualier_exists), 20.2 (__qualiers)

20.2 __qualiers

Synopsis

Get the active set of qualifiers

Usage

\[
\text{Struct\_Type \_\_qualifiers ()}
\]

Description

This function returns the set of qualifiers associated with the current execution context. If qualifiers are active, then the result is a structure representing the names of the qualifiers and their corresponding values. Otherwise `NULL` will be returned.

One of the main uses of this function is to pass the current set of qualifiers to another another function. For example, consider a plotting application with a function called `lineto` that sets the pen-color before drawing the line to the specified point:

\[
define \text{lineto} (x, y)
{
    \%
    \text{The color may be specified by a qualifier, defaulting to black}
    \text{variable color = qualifier ("color", "black");}
    \text{set\_pen\_color (color);}
    \%
    \%
}
\]

The `lineto` function permits the color to be specified by a qualifier. Now consider a function that make use of `lineto` to draw a line segment between two points:

\[
define \text{line\_segment} (x0, y0, x1, y1)
{
    \text{moveto (x0, y0);}
    \text{lineto (x1, y1; color=qualifier("color", "black");)}
}
\]

\[
\text{line\_segment (1,1, 10,10; color="blue");}
\]

Note that in this implementation of `line\_segment`, the `color` qualifier was explicitly passed to the `lineto` function. However, this technique does not scale well. For example, the `lineto` function might also take a qualifier that specifies the line-style, to be used as

\[
\text{line\_segment (1,1, 10,10; color="blue", linestyle="solid");}
\]

But the above implementation of `line\_segment` does not pass the `linestyle` qualifier. In such a case, it is preferable to pass all the qualifiers, e.g.,
define line_segment (x0, y0, x1, y1)
{
    moveto (x0, y0);
    lineto (x1, y1 ;; __qualifiers());
}

Note the use of the double-semi colon in the lineto statement. This tells the parser that the
qualifiers are specified by a structure-valued argument and not a set of name-value pairs.

See Also
20.1 (qualifier), 20.3 (qualifier_exists)

20.3 qualifier_exists

Synopsis
Check for the existence of a qualifier

Usage
Int_Type qualifier_exists (String_Type name)

Description
This function will return 1 if a qualifier of the specified name exists, or 0 otherwise.

See Also
20.1 (qualifier), 20.2 (__qualifiers)
Chapter 21

Module Functions

21.1 get_import_module_path

Synopsis
Get the search path for dynamically loadable objects

Usage
String_Type get_import_module_path ()

Description
The get_import_module_path may be used to get the search path for dynamically shared objects. Such objects may be made accessible to the application via the import function.

See Also
21.2 (import), 21.3 (set_import_module_path)

21.2 import

Synopsis
Dynamically link to a specified module

Usage
import (String_Type module [, String_Type namespace])

Description
The import function causes the run-time linker to dynamically link to the shared object specified by the module parameter. It searches for the shared object as follows: First a search is performed along all module paths specified by the application. Then a search is made along the paths defined via the set_import_module_path function. If not found, a search is performed along the paths given by the $SLANG_MODULE_PATH$ environment variable. Finally, a system dependent search is performed (e.g., using the $LD_LIBRARY_PATH$ environment variable).
The optional second parameter may be used to specify a namespace for the intrinsic functions and variables of the module. If this parameter is not present, the intrinsic objects will be placed into the active namespace, or global namespace if the active namespace is anonymous.

This function throws an ImportError if the specified module is not found.

Notes

The import function is not available on all systems.

See Also

21.3 (set_import_module_path), 25.21 (use_namespace), 25.4 (current_namespace), 25.8 (getenv), 19.5 (evalfile)

21.3 set_import_module_path

Synopsis

Set the search path for dynamically loadable objects

Usage

set_import_module_path (String_Type path_list)

Description

The set_import_module_path may be used to set the search path for dynamically shared objects. Such objects may be made accessible to the application via the import function.

The actual syntax for the specification of the set of paths will vary according to the operating system. Under Unix, a colon character is used to separate paths in path_list. For win32 systems a semi-colon is used. The path_get_delimiter function may be used to get the value of the delimiter.

See Also

21.2 (import), 21.1 (get_import_module_path), 17.6 (path_get_delimiter)
Chapter 22

Debugging Functions

22.1 _bofeof_info

Synopsis
Control the generation of function callback code

Usage
Int_Type _bofeof_info

Description
This value of this variable dictates whether or not the S-Lang interpreter will generate code to call the beginning and end of function callback handlers. The value of this variable is local to the compilation unit, but is inherited by other units loaded by the current unit.

If the value of this variable is 1 when a function is defined, then when the function is executed, the callback handlers defined via _set_bof_handler and _set_eof_handler will be called.

See Also
22.6 (_set_bof_handler), 22.8 (_set_eof_handler), 22.2 (_boseos_info)

22.2 _boseos_info

Synopsis
Control the generation of BOS/EOS callback code

Usage
Int_Type _boseos_info

Description
This value of this variable dictates whether or not the S-Lang interpreter will generate code to call the beginning and end of statement callback handlers. The value of this variable is local to the compilation unit, but is inherited by other units loaded by the current unit.

The lower 8 bits of _boseos_info controls the generation of code for callbacks as follows:
Chapter 22. Debugging Functions

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No code for making callbacks will be produced.</td>
</tr>
<tr>
<td>1</td>
<td>Callback generation will take place for all non-branching and looping statements.</td>
</tr>
<tr>
<td>2</td>
<td>Same as for 1 with the addition that code will also be generated for branching statements (if, !if, loop, ...)</td>
</tr>
<tr>
<td>3</td>
<td>Same as 2, but also including break and continue statements.</td>
</tr>
</tbody>
</table>

A non-branching statement is one that does not effect chain of execution. Branching statements include all looping statements, conditional statement, break, continue, and return.

If bit 0x100 is set, callbacks will be generated for preprocessor statements.

**Example**

Consider the following:

```plaintext
_boseos_info = 1;
define foo ()
{
    if (some_expression)
        some_statement;
}
_boseos_info = 2;
define bar ()
{
    if (some_expression)
        some_statement;
}
```

The function `foo` will be compiled with code generated to call the BOS and EOS handlers when `some_statement` is executed. The function `bar` will be compiled with code to call the handlers for both `some_expression` and `some_statement`.

**Notes**

The `sldb` debugger and `slsh`’s `stkcheck.s1` make use of this facility.

**See Also**

22.7 (`_set_bos_handler`), 22.9 (`_set_eos_handler`), 22.1 (`_boeoof_info`)

### 22.3 `_clear_error`

**Synopsis**

Clear an error condition (deprecated)

**Usage**

```plaintext
清楚_error ()
```
Description

This function has been deprecated. New code should make use of try-catch exception handling. This function may be used in error-blocks to clear the error that triggered execution of the error block. Execution resumes following the statement, in the scope of the error-block, that triggered the error.

Example

Consider the following wrapper around the putenv function:

```c
define try_putenv (name, value)
{
    variable status;
    ERROR_BLOCK
    {
        _clear_error ();
        status = -1;
    }
    status = 0;
    putenv (sprintf ("%s=%s", name, value);
    return status;
}
```

If `putenv` fails, it generates an error condition, which the `try_putenv` function catches and clears. Thus `try_putenv` is a function that returns -1 upon failure and 0 upon success.

See Also

22.12 (_trace_function), 22.10 (_slangtrace), 22.11 (_traceback)

22.4 _get_frame_info

Synopsis

Get information about a stack frame

Usage

Struct_Type _get_frame_info (Integer_Type depth)

Description

_get_frame_info returns a structure with information about the function call stack from of depth depth. The structure contains the following fields:

- **file**: The file that contains the code of the stack frame.
- **line**: The line number the file the stack frame is in.
- **function**: The name of the function containing the code of the stack frame; it might be NULL if the code isn't inside a function.
- **locals**: Array of String_Type containing the names of variables local to the stack frame; it might be NULL if the stack frame doesn't belong to a function.
- **namespace**: The namespace the code of this stack frame is in.
See Also

22.5 (_get_frame_variable), 22.13 (_use_frame_namespace)

22.5 _get_frame_variable

Synopsis
Get the value of a variable local to a stack frame

Usage
Any_Type _get_frame_variable (Integer_Type depth, String_Type name)

Description
This function returns value of the variable name in the stack frame at depth depth. This might not only be a local variable but also variables from outer scopes, e.g., a variable private to the namespace.

If no variable with this name is found an UndefinedNameError will be thrown. An VariableUninitializedError will be generated if the variable has no value.

See Also
22.4 (_get_frame_info), 22.13 (_use_frame_namespace)

22.6 _set_bof_handler

Synopsis
Set the beginning of function callback handler

Usage
_set_bof_handler (Ref_Type func)

Description
This function is used to set the function to be called prior to the execution of the body S-Lang function but after its arguments have been evaluated, provided that function was defined with _bofeof_info set appropriately. The callback function must be defined to take a single parameter representing the name of the function and must return nothing.

Example

private define bof_handler (fun)
{
  () = fputs ("About to execute $fun $", stdout);
}
_set_bos_handler (&bof_handler);

See Also
22.8 (_set_eof_handler), 22.2 (_boseos_info), 22.7 (_set_bos_handler)
22.7 _set_bos_handler

Synopsis

Set the beginning of statement callback handler

Usage

_set_bos_handler (Ref_Type func)

Description

This function is used to set the function to be called prior to the beginning of a statement. The function will be passed two parameters: the name of the file and the line number of the statement to be executed. It should return nothing.

Example

private define bos_handler (file, line)
{
  () = fputs ("About to execute $file:$line
", stdout);
}
_set_bos_handler (&bos_handler);

Notes

The beginning and end of statement handlers will be called for statements in a file only if that file was compiled with the variable _boseos_info set to a non-zero value.

See Also

22.9 (_set_eos_handler), 22.2 (_boseos_info), 22.1 (_bofeof_info)

22.8 _set_eof_handler

Synopsis

Set the beginning of function callback handler

Usage

_set_eof_handler (Ref_Type func)

Description

This function is used to set the function to be called at the end of execution of a S-Lang function, provided that function was compiled with _bofeof_info set accordingly. The callback function will be passed no parameters and it must return nothing.

Example

private define eof_handler ()
{
  () = fputs ("Done executing the function\n", stdout);
}
_set_eof_handler (&eof_handler);
See Also

22.6 (_set_bof_handler), 22.1 (_bofeof_info), 22.2 (_booseos_info)

22.9 _set_eos_handler

Synopsis
Set the end of statement callback handler

Usage
_set_eos_handler (Ref_Type func)

Description
This function is used to set the function to be called at the end of a statement. The function will be passed no parameters and it should return nothing.

Example

private define eos_handler ()
{
    () = fputs ("Done executing the statement\n", stdout);
}
_set_eos_handler (&eos_handler);

Notes
The beginning and end of statement handlers will be called for statements in a file only if that file was compiled with the variable _booseos_info set to a non-zero value.

See Also
22.7 (_set_bos_handler), 22.2 (_booseos_info), 22.1 (_bofeof_info)

22.10 _slangtrace

Synopsis
Turn function tracing on or off

Usage
Integer_Type _slangtrace

Description
The _slangtrace variable is a debugging aid that when set to a non-zero value enables tracing when function declared by _trace_function is entered. If the value is greater than zero, both intrinsic and user defined functions will get traced. However, if set to a value less than zero, intrinsic functions will not get traced.

See Also
22.12 (_trace_function), 22.11 (_traceback), 23.7 (_print_stack)
22.11 _traceback

Synopsis

Generate a traceback upon error

Usage

Integer_Type _traceback

Description

_traceback is an intrinsic integer variable whose bitmapped value controls the generation of
the call-stack traceback upon error. When set to 0, no traceback will be generated. Otherwise
its value is the bitwise-or of the following integers:

<table>
<thead>
<tr>
<th>Integer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a full traceback</td>
</tr>
<tr>
<td>2</td>
<td>Omit local variable information</td>
</tr>
<tr>
<td>4</td>
<td>Generate just one line of traceback</td>
</tr>
</tbody>
</table>

The default value of this variable is 4.

Notes

Running slsh with the -g option causes this variable to be set to 1.

See Also

22.2 (_boseos_info)

22.12 _trace_function

Synopsis

Set the function to trace

Usage

_trace_function (String_Type f)

Description

_trace_function declares that the S-Lang function with name f is to be traced when it is
called. Calling _trace_function does not in itself turn tracing on. Tracing is turned on only
when the variable _slangtrace is non-zero.

See Also

22.10 (_slangtrace), 22.11 (_traceback)

22.13 _use_frame_namespace

Synopsis

Selects the namespace of a stack frame
Usage

_use_frame_namespace (Integer_Type depth)

Description

This function sets the current namespace to the one belonging to the call stack frame at depth depth.

See Also

22.4 (_get_frame_info), 22.5 (_get_frame_variable)
Chapter 23

Stack Functions

23.1 dup

Synopsis

Duplicate the value at the top of the stack

Usage

dup ()

Description

This function returns an exact duplicate of the object on top of the stack. For some objects such as arrays or structures, it creates a new reference to the object. However, for simple scalar S-Lang types such as strings, integers, and doubles, it creates a new copy of the object.

See Also

23.3 (pop), 12.17 (typeof)

23.2 exch

Synopsis

Exchange two items on the stack

Usage

exch ()

Description

The exch swaps the two top items on the stack.

See Also

23.3 (pop), 23.11 (_stk_reverse), 23.12 (_stk_roll)
23.3 pop

Synopsis
Discard an item from the stack

Usage
pop ()

Description
The pop function removes the top item from the stack.

See Also
23.6 (__pop_n), 23.4 (__pop_args)

23.4 __pop_args

Synopsis
Remove n function arguments from the stack

Usage
args = __pop_args(Integer_Type n)

Description
This function, together with the companion function __push_args, is useful for creating a function that takes a variable number of arguments, as well as passing the arguments of one function to another function.

__pop_args removes the specified number of values from the stack and returns them as an array of structures of the corresponding length. Each structure in the array consists of a single field called value, which represents the value of the argument.

Example
Consider the following function. It prints all its arguments to stdout separated by spaces:

```c
#define print_args ()
{
    variable i;
    variable args = __pop_args (_NARGS);

    for (i = 0; i < _NARGS; i++)
    {
      () = fputs (string (args[i].value), stdout);
      () = fputs (" ", stdout);
    }
    () = fputs ("\n", stdout);
    () = fflush (stdout);
}
```
Now consider the problem of defining a function called `ones` that returns a multi-dimensional array with all the elements set to 1. For example, `ones(10)` should return a 1-d array of 10 ones, whereas `ones(10,20)` should return a 10x20 array.

```plaintext
define ones ()
{
  !if (_NARGS) return 1;
  variable a;

  a = __pop_args (_NARGS);
  return @Array_Type (Integer_Type, [__push_args (a)]) + 1;
}
```

Here, `__push_args` was used to push the arguments passed to the `ones` function onto the stack to be used when dereferencing `Array_Type`.

**Notes**

This function has been superseded by the `__pop_list` function, which returns the objects as a list instead of an array of structures.

**See Also**

23.8 (`__push_args`), 23.5 (`__pop_list`), 23.9 (`__push_list`), 12.17 (`typeof`), 23.6 (`__pop_n`)

### 23.5 __pop_list

**Synopsis**

Convert items on the stack to a `List_Type`

**Usage**

```plaintext
List_Type = __pop_list (Int_Type n)
```

**Description**

This function removes a specified number of items from the stack and converts returns them in the form of a list.

**Example**

```plaintext
define print_args ()
{
  variable list = __pop_list (_NARGS);
  variable i;
  _for i (0, length(list)-1, 1)
  {
    vmessage ("arg[%d]: %S", i, list[i]);
  }
}
```

**See Also**

23.9 (`__push_list`)
23.6 _pop_n

Synopsis
Remove objects from the stack

Usage
_pop_n (Integer_Type n);

Description
The _pop_n function removes the specified number of objects from the top of the stack.

See Also
23.10 (_stkdepth), 23.3 (pop)

23.7 _print_stack

Synopsis
Print the values on the stack.

Usage
_print_stack ()

Description
This function dumps out what is currently on the S-Lang stack. It does not alter the stack and it is usually used for debugging purposes.

See Also
23.10 (_stkdepth), 12.12 (string), 10.5 (message)

23.8 __push_args

Synopsis
Move n function arguments onto the stack

Usage
__push_args (Struct_Type args);

Description
This function together with the companion function __pop_args is useful for the creation of functions that take a variable number of arguments. See the description of __pop_args for more information.

Notes
This function has been superseded by the __push_list function.

See Also
23.4 (__pop_args), 23.9 (__push_list), 23.5 (__pop_list), 12.17 (typeof), 23.6 (__pop_n)
23.9 _push_list

Synopsis
Push the elements of a list to the stack

Usage
__push_list (List_Type list)

Description
This function pushes the elements of a list to the stack.

Example
private define list_to_array (list)
{
    return [...push_list (list)];
}

See Also
23.5 (_pop_list)

23.10 _stkdepth

Usage
Get the number of objects currently on the stack

Synopsis
Integer_Type _stkdepth ()

Description
The _stkdepth function returns number of items on the stack.

See Also
23.7 (_print_stack), 23.11 (_stk_reverse), 23.12 (_stk_roll)

23.11 _stk_reverse

Synopsis
Reverse the order of the objects on the stack

Usage
_stk_reverse (Integer_Type n)

Description
The _stk_reverse function reverses the order of the top n items on the stack.

See Also
23.10 (_stkdepth), 23.12 (_stk_roll)
23.12 _stk_roll

Synopsis
Roll items on the stack

Usage
_stk_roll (Integer_Type n)

Description
This function may be used to alter the arrangement of objects on the stack. Specifically, if the integer n is positive, the top n items on the stack are rotated up. If n is negative, the top \( \text{abs}(n) \) items on the stack are rotated down.

Example
If the stack looks like:

```
item-0
item-1
item-2
item-3
```

where item-0 is at the top of the stack, then \( \text{stk\_roll}(-3) \) will change the stack to:

```
item-2
item-0
item-1
item-3
```

Notes
This function only has an effect if \( \text{abs}(n) > 1 \).

See Also
23.10 (_stk\_depth), 23.11 (_stk\_reverse), 23.6 (_pop\_n), 23.7 (_print\_stack)
Chapter 24

Functions that deal with the S-Lang readline interface

24.1 rline_bolp

Synopsis
   Test of the editing point is at the beginning of the line

Usage
   Int_Type rline_bolp()

Description
   The rline_bolp function returns a non-zero value if the current editing position is at the
   beginning of the line.

Notes
   This function is part of the S-Lang readline interface.

See Also
   24.4 (rline_eolp), 24.9 (rline_get_point), 24.8 (rline_get_line)

24.2 rline_call

Synopsis
   Invoke an internal readline function

Usage
   rline_call (String_Type func)

Description
   Not all of the readline functions are available directly from the S-Lang interpreter. For
   example, the "deleol" function, which deletes through the end of the line may be executed using
Chapter 24. Functions that deal with the S-Lang readline interface

rline_call("deleol");

See the documentation for the rline_setkey function for a list of internal functions that may be invoked by rline_call.

Notes
This function is part of the S-Lang readline interface.

See Also
24.12 (rline_setkey), 24.3 (rline_del), 24.11 (rline_ins)

24.3 rline_del

Synopsis
Delete a specified number of characters at the current position

Usage
rline_del(Int_Type n)

Description
This function delete a specified number of characters at the current editing position. If the number n is less than zero, then the previous n characters will be deleted. Otherwise, the next n characters will be deleted.

Notes
This function is part of the S-Lang readline interface.

See Also
24.11 (rline_ins), 24.12 (rline_setkey)

24.4 rline_eolp

Synopsis
Test of the editing point is at the end of the line

Usage
Int_Type rline_eolp()

Description
The rline_bolp function returns a non-zero value if the current editing position is at the end of the line.

Notes
This function is part of the S-Lang readline interface.

See Also
24.1 (rline_bolp), 24.9 (rline_get_point), 24.8 (rline_get_line)
24.5  rline_getkey

Synopsis
Obtain the next byte in the readline input stream

Usage
Int_Type rline_getkey ()

Description
This function returns the next byte in the readline input stream. If no byte is available, the function will wait until one is.

Notes
This function is part of the S-Lang readline interface.

See Also
24.10 (rline_input_pending), 24.12 (rline_setkey)

24.6  rline_get_edit_width

Synopsis
Get the width of the readline edit window

Usage
Int_Type rline_get_edit_width ()

Description
This function returns the width of the edit window. For slsh, this number corresponds to the width of the terminal window.

Notes
This function is part of the S-Lang readline interface.

See Also
24.11 (rline_ins)

24.7  rline_get_history

Synopsis
Retrieve the readline history

Usage
Array_Type rline_get_history ()

Description
This function returns the readline edit history as an array of strings.
Notes
This function is part of the S-Lang readline interface.

See Also
24.15 (rline_set_line)

24.8 rline_get_line

Synopsis
Get a copy of the line being edited

Usage
String_Type rline_get_line ()

Description
This function returns the current edit line.

Notes
This function is part of the S-Lang readline interface.

See Also
24.15 (rline_set_line), 24.7 (rline_get_history)

24.9 rline_get_point

Synopsis
Get the current editing position

Usage
Int_Type rline_get_point ()

Description
The rline_get_point function returns the byte-offset of the current editing position.

Notes
This function is part of the S-Lang readline interface.

See Also
24.17 (rline_set_point)
24.10  rline_input_pending

Synopsis
Test to see if readline input is available for reading

Usage
Int_Type rline_input_pending (Int_Type tsecs)

Description
This function returns a non-zero value if readline input is available to be read. If none is immediately available, it will wait for up to tsecs tenths of a second for input before returning.

Notes
This function is part of the S-Lang readline interface.

See Also
24.5 (rline_getkey)

24.11  rline_ins

Synopsis
Insert a string at the current editing point

Usage
rline_ins (String_Type text)

Description
This function inserts the specified string into the line being edited.

Notes
This function is part of the S-Lang readline interface.

See Also
24.15 (rline_set_line), 24.3 (rline_del)

24.12  rline_setkey

Synopsis
Bind a key in the readline keymap to a function

Usage
rline_setkey (func, keyseq)
**Description**

The `rline_setkey` function binds the function `func` to the specified key sequence `keyseq`. The value of `func` may be either a reference to a S-Lang function, or a string giving the name of an internal readline function.

Functions that are internal to the readline interface include:

- `bdel` Delete the previous character
- `bol` Move to the beginning of the line
- `complete` The command line completion function
- `del` Delete the character at the current position
- `delbol` Delete to the beginning of the line
- `deleol` Delete through the end of the line
- `down` Goto the next line in the history
- `enter` Return to the caller of the readline function
- `eol` Move to the end of the line
- `kbd_quit` Abort editing of the current line
- `left` Move left one character
- `quoted_insert` Insert the next byte into the line
- `redraw` Redraw the line
- `right` Move right one character
- `self_insert` Insert the byte that invoked the function
- `trim` Remove whitespace about the current position
- `up` Goto the previous line in the history

**Notes**

This function is part of the S-Lang readline interface.

**See Also**

24.18 (rline_unsetkey)

24.13 `rline_set_completion_callback`

**Synopsis**

Set the function to be used for completion at the readline prompt

**Usage**

```c
rline_set_completion_callback (Ref_Type func)
```

**Description**

This function sets the callback function to be used for completion at the readline prompt. The callback function must be defined to accept two values, the first being a string containing the text of the line being edited, and an integer giving the position of the byte-offset into the string where completion was requested.

The callback function must return two values: an array giving the list of possible completion strings, and an integer giving the byte offset into the string of the start of the text to be completed.
24.14. rline_set_history

Example

See completion-callback function defined in the slsh library file rline/complete.sl.

Notes

This function is part of the S-Lang readline interface.

See Also

24.16 (rline_set_list_completions_callback)

24.14 rline_set_history

Synopsis

Replace the current history list with a new one

Usage

rline_set_history (Array_Type lines)

Description

The rline_set_history function replaces the current history by the specified array of strings.

Notes

This function is part of the S-Lang readline interface.

See Also

24.7 (rline_get_history)

24.15 rline_set_line

Synopsis

Replace the current line with a new one

Usage

rline_set_line (String_Type line)

Description

The rline_set_line function replaces the line being edited by the specified one.

Notes

This function is part of the S-Lang readline interface.

See Also

24.8 (rline_get_line)
24.16  rline_set_list_completions_callback

Synopsis

Set a callback function to display the list of completions

Usage

rline_set_list_completions_callback (Ref_Type func)

Description

This function sets the S-Lang function that is to be used to display the list of possible completions for current word at the readline prompt. The callback function must be defined to accept a single parameter representing an array of completion strings.

Example

This callback function writes the completions using the message functions:

private define display_completions (strings)
{
    variable str;
    vmessage ("There are %d completions:\n", length(strings));
    foreach str (strings) vmessage ("%s\n", str);
}
rline_set_list_completions_callback (&display_completions);

See Also

24.13 (rline_set_completion_callback)

24.17  rline_set_point

Synopsis

Move the current editing position to another

Usage

rline_set_point (Int_Type ofs)

Description

The rline_set_point function sets the editing point to the specified byte-offset from the beginning of the line.

Notes

This function is part of the S-Lang readline interface.

See Also

24.9 (rline_get_point)
24.18  rline_unsetkey

Synopsis
   Unset a key binding from the readline keymap

Usage
   rline_unsetkey (String_Type keyseq)

Description
   The rline_unsetkey function unbinds the specified key sequence from the readline keymap.

Notes
   This function is part of the S-Lang readline interface.

See Also
   24.12 (rline_setkey)
Chapter 24. Functions that deal with the S-Lang readline interface
Chapter 25

Miscellaneous Functions

25.1 _auto_declare

Synopsis

Set automatic variable declaration mode

Usage

Integer_Type _auto_declare

Description

The _auto_declare variable may be used to have undefined variable implicitly declared. If set to zero, any variable must be declared with a variable declaration before it can be used. If set to one, then any undeclared variable will be declared as a static variable.

The _auto_declare variable is local to each compilation unit and setting its value in one unit has no effect upon its value in other units. The value of this variable has no effect upon the variables in a function.

Example

The following code will not compile if X not been declared:

\[
X = 1;
\]

However,

\[
_auto_declare = 1; \quad \% \text{declare variables as static.}
X = 1;
\]

is equivalent to

\[
\text{static variable } X = 1;
\]

Notes

This variable should be used sparingly and is intended primarily for interactive applications where one types S-Lang commands at a prompt.
25.2 __class_id

Synopsis
Return the class-id of a specified type

Usage
Int_Type __class_id (DataType_Type type)

Description
This function returns the internal class-id of a specified data type.

See Also
12.17 (typeof), 12.16 (_typeof), 25.3 (__class_type), 25.5 (__datatype)

25.3 __class_type

Synopsis
Return the class-type of a specified type

Usage
Int_Type __class_type (DataType_Type type))

Description
Internally **S-Lang** objects are classified according to four types: scalar, vector, pointer, and memory managed types. For example, an integer is implemented as a scalar, a complex number as a vector, and a string is represented as a pointer. The __class_type function returns an integer representing the class-type associated with the specified data type. Specifically, it returns:

0  memory-managed
1  scalar
2  vector
3  pointer

See Also
12.17 (typeof), 12.16 (_typeof), 25.2 (__class_id), 25.5 (__datatype)

25.4 current_namespace

Synopsis
Get the name of the current namespace

Usage
String_Type current_namespace ()
Description

The `current_namespace` function returns the name of the static namespace associated with the compilation unit. If there is no such namespace associated with the compilation unit, then the empty string "" will be returned.

See Also

25.10 (implements), 25.21 (use_namespace), 21.2 (import), 19.5 (evalfile)

25.5 __datatype

Synopsis

Get the DataType_Type for a specified internal class-id

Usage

`DataType_Type __datatype (Int_Type id)`

Description

This function is the inverse of `__class_type` in the sense that it returns the `DataType_Type` for the specified class-id. If no such class exists, the function will return NULL.

Notes

One should not expect distinct interpreter instances to always return the same value for a dynamically assigned class-id such as one defined by a module or one stemming from a `typedef` statement.

See Also

25.2 (`__class_id`), 25.3 (`__class_type`), 12.17 (typeof)

25.6 _eqs

Synopsis

Test for equality of two objects

Usage

`Int_Type _eqs (a, b)`

Description

This function tests its two arguments for equality and returns 1 if they are equal or 0 otherwise. What it means to be equal depends upon the data types of the objects being compared. If the types are numeric, they are regarded as equal if their numerical values are equal. If they are arrays, then they are equal if they have the same shape with equal elements. If they are structures, then they are equal if they contain identical fields, and the corresponding values are equal.

Example
Chapter 25. Miscellaneous Functions

Notes
For testing sameness, use __is_same.

See Also
12.17 (typeof), 25.14 (__is_same), 25.9 (__get_reference), 25.11 (__is_callable)

25.7 get_environ

Synopsis
Get all environment variables

Usage
String_Type[] = get_environ()

Description
The get_environ function returns an array of strings representing the environment variables defined for the current process. Each element of the array will be of the form NAME=VALUE.
This function will return NULL if the system does not support this feature.

See Also
25.8 (getenv), 25.15 (putenv), 8.9 (is_defined)

25.8 getenv

Synopsis
Get the value of an environment variable

Usage
String_Type getenv(String_Type var)

Description
The getenv function returns a string that represents the value of an environment variable var. It will return NULL if there is no environment variable whose name is given by var.

Example

    if (NULL != getenv("USE_COLOR"))
    {
        set_color ("normal", "white", "blue");
        set_color ("status", "black", "gray");
        USE_ANSI_COLORS = 1;
    }
See Also

25.7 (get_environ), 25.15 (putenv), 4.24 (strlen), 8.9 (is_defined)

25.9 __get_reference

Synopsis
Get a reference to a global object

Usage
Ref_Type __get_reference (String_Type nm)

Description
This function returns a reference to a global variable or function whose name is specified by
nm. If no such object exists, it returns NULL, otherwise it returns a reference.

Example
Consider the function:

```s-lang
define runhooks (hook)
{
    variable f;
    f = __get_reference (hook);
    if (f != NULL)
        @f ();
}
```

This function could be called from another S-Lang function to allow customization of that
function, e.g., if the function represents a jed editor mode, the hook could be called to setup
keybindings for the mode.

See Also

8.9 (is_defined), 12.17 (typeof), 19.4 (eval), 19.2 (autoload), 8.10 (__is_initialized), 25.20
(__uninitialize)

25.10 implements

Synopsis
Create a new static namespace

Usage
implements (String_Type name)

Description
The implements function may be used to create a new static namespace and have it associated
with the current compilation unit. If a namespace with the specified name already exists, a
NamespaceError exception will be thrown.
In addition to creating a new static namespace and associating it with the compilation unit, the function will also create a new private namespace. As a result, any symbols in the previous private namespace will be no longer be accessible. For this reason, it is recommended that this function should be used before any private symbols have been created.

Example

Suppose that some file t.sl contains:

```plaintext
implements ("My");
define message (x)
{
    Global->message ("My's message: $x"$);
}
message ("hello");
```

will produce "My's message: hello". This `message` function may be accessed from outside the namespace via:

```plaintext
My->message ("hi");
```

Notes

Since `message` is an intrinsic function, it is public and may not be redefined in the public namespace.

The `implements` function should rarely be used. It is preferable to allow a static namespace to be associated with a compilation unit using, e.g., `evalfile`.

See Also

25.21 (use_namespace), 25.4 (current_namespace), 21.2 (import)

25.11 __is_callable

Synopsis

Determine whether or not an object is callable

Usage

```plaintext
Int_Type __is_callable (obj)
```

Description

This function may be used to determine if an object is callable by dereferencing the object. It returns 1 if the argument is callable, or zero otherwise.

Example

```plaintext
__is_callable (7)  ==> 0
__is_callable (&sin)  ==> 1
a = [&sin];
__is_callable (a[0])  ==> 1
__is_callable (&a[0])  ==> 0
```

See Also

25.13 (__is_numeric), 8.9 (is_defined)
25.12 __is_datatype_numeric

Synopsis
Determine whether or not a type is a numeric type

Usage
Int_Type __is_datatype_numeric (DataType_Type type)

Description
This function may be used to determine if the specified datatype represents a numeric type. It returns 0 if the datatype does not represent a numeric type; otherwise it returns 1 for an integer type, 2 for a floating point type, and 3 for a complex type.

See Also
12.17 (typeof), 25.13 (__is_numeric), 25.11 (__is_callable)

25.13 __is_numeric

Synopsis
Determine whether or not an object is a numeric type

Usage
Int_Type __is_numeric (obj)

Description
This function may be used to determine if an object represents a numeric type. It returns 0 if the argument is non-numeric, 1 if it is an integer, 2 if a floating point number, and 3 if it is complex. If the argument is an array, then the array type will be used for the test.

Example

```
__is_numeric ("foo"); ==> 0
__is_numeric ("0"); ==> 0
__is_numeric (0); ==> 1
__is_numeric (PI); ==> 2
__is_numeric (2j); ==> 3
__is_numeric ([1,2]); ==> 1
__is_numeric ({1,2}); ==> 0
```

See Also
12.17 (typeof), 25.12 (__is_datatype_numeric)

25.14 __is_same

Synopsis
Test for sameness of two objects
Usage

Int_Type __is_same (a, b)

Description
This function tests its two arguments for sameness and returns 1 if they are the same, or 0 otherwise. To be the same, the data type of the arguments must match and the values of the objects must reference the same underlying object.

Example

```
__is_same (1, 1)  ===> 1
__is_same (1, 1.0)  ===> 0
__is_same ("a", 1)  ===> 0
__is_same ([1,2], [1,2])  ===> 0
```

Notes
For testing equality, use _eqs.

See Also
12.17 (typeof), 25.6 (_eqs), 25.9 (__get_reference), 25.11 (__is_callable)

25.15 putenv

Synopsis
Add or change an environment variable

Usage
putenv (String_Type s)

Description
This function adds string s to the environment. Typically, s should of the form "name=value". The function throws an OSError upon failure.

Notes
This function may not be available on all systems.

See Also
25.8 (getenv), 4.10 (sprintf)

25.16 __set_argc_argv

Synopsis
Set the argument list

Usage
__set_argc_argv (Array_Type a)

Description
This function sets the __argc and __argv intrinsic variables.
25.17  _slang_install_prefix

Synopsis
S-Lang’s installation prefix

Usage
String_Type  _slang_install_prefix

Description
The value of this variable is set at the S-Lang library’s compilation time. On Unix systems, the value corresponds to the value of the prefix variable in the Makefile. For normal installations, the library itself will be located in the lib subdirectory of the prefix directory.

Notes
The value of this variable may or may not have anything to do with where the slang library is located. As such, it should be regarded as a hint. A standard installation will have the slsh library files located in the share/slsh subdirectory of the installation prefix.

See Also
8.13 (_slang_doc_dir)

25.18  _slang_utf8_ok

Synopsis
Test if the interpreter running in UTF-8 mode

Usage
Int_Type  _slang_utf8_ok

Description
If the value of this variable is non-zero, then the interpreter is running in UTF-8 mode. In this mode, characters in strings are interpreted as variable length byte sequences according to the semantics of the UTF-8 encoding.

Notes
When running in UTF-8 mode, one must be careful not to confuse a character with a byte. For example, in this mode the strlen function returns the number of characters in a string which may be different than the number of bytes. The latter information may be obtained by the strbytelen function.

See Also
4.12 (strbytelen), 4.24 (strlen), 4.15 (strcharlen)
25.19  __tmp

Synopsis

Returns the value of a variable and uninitialize the variable

Usage

__tmp (x)

Description

The __tmp function takes a single argument, a variable, returns the value of the variable, and then undefines the variable. The purpose of this pseudo-function is to free any memory associated with a variable if that variable is going to be re-assigned.

Example

\[
x = 3;
y = __tmp(x);
\]

will result in 'y' having a value of '3' and 'x' will be undefined.

Notes

This function is a pseudo-function because a syntax error results if used like

\[
__tmp(sin(x));
\]

See Also

25.20 (__uninitialize), 8.10 (__is_initialized)

25.20  __uninitialize

Synopsis

Uninitialize a variable

Usage

__uninitialize (Ref_Type x)

Description

The __uninitialize function may be used to uninitialize the variable referenced by the parameter x.

Example

The following two lines are equivalent:

\[
() = __tmp(x);
__uninitialize (&x);
\]

See Also

25.19 (__tmp), 8.10 (__is_initialized)
25.21 use_namespace

Synopsis

Change to another namespace

Usage

use_namespace (String_Type name)

Description

The use_namespace function changes the current static namespace to the one specified by the parameter. If the specified namespace does not exist, a NamespaceError exception will be generated.

See Also

25.10 (implements), 25.4 (current_namespace), 21.2 (import)