S-Lang Run-Time Library Reference: Version 2.0.4

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

Jun 18, 2005

Contents

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 object; 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 = Assoc_Type[d];
```

For example,

```
A = Assoc_Type[Int_Type];
```

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

```
A = Assoc_Type[];
```

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

```
A = Assoc_Type[Int_Type, -1];
```

creates an integer-valued associative array with a default element value of -1. Then `A["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:

```
assoc_get_keys
assoc_get_values
assoc_key_exists
assoc_delete_key
```

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 on of the following forms:

```
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

`List_Type, Array_Type, Struct_Type`

1.2 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.,

```
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` objects to loop over its elements:

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

See Also

`Array_Type, Assoc_Type, Struct_Type`

1.3 Struct_Type

Synopsis

A structure datatype

Description

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

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

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

```
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 (@):

```
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

`List_Type`, `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

```
1       2       3       4        5
6       7       8       9       10
```

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

`where, any`

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

```
1       2       3       4        5
6       7       8       9       10
```

generated by

```
a = _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

`where, all`

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:

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

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

See Also

typeof, array_info, array_shape, length, reshape, _reshape

2.4 array_map

Synopsis

Apply a function to each element of an array

Usage

```
Array_Type array_map (type, func, arg0, ...)

        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. The `arg0` parameter should be an array and is used to determine the dimensions of the resulting array. If any subsequent arguments correspond to an array of the same size, then those array elements will be passed in parallel with the first arrays arguments.

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"];
```

Notes

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

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

See Also

`array_info, strlen, strcat, 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

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

```
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

`array_swap, transpose`

2.6 array_shape

Synopsis

Get the shape or dimensions of an array

Usage

`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

`array_info, reshape`

2.7 array_sort

Synopsis

Sort an array

Usage

```
Array_Type array_sort (Array_Type a [, String_Type or Ref_Type f])
```

Description

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

If `f` is present, then it must be either a string representing the name of the comparison function, or a reference to the function. The sort function represented by `f` must be a **S-Lang** function that takes two arguments. The function must return an integer that is less than zero if the first parameter is considered to be less than the second, zero if they are equal, and a value greater than zero if the first is greater than the second.

If the comparison function is not specified, then a built-in comparison function appropriate for the data type will be used. For example, if `a` is an array of character strings, then the sort will be performed using the `strcmp` function.

The integer array returned by this function is simply an index array that indicates the order of the sorted array. The input array `a` is not changed.

Example

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

```
A = ["gamma", "alpha", "beta"];
I = array_sort (A, &strcmp);
```

Alternatively, one may use

```
variable I = array_sort (A);
```

to use the built-in comparison function.

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

```
A = A[array_sort(A)];
```

See Also

strcmp

2.8 array_swap

Synopsis

Swap elements of an array

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]);
```

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

See Also

array_reverse, 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

sum

2.10 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 character array `a` by setting the elements of the array `a` to the corresponding characters 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 characters in the string `"HelloWorld"`.

Notes

The character array must be large enough to hold all the characters of the initialization string.

See Also

`bstring_to_array, strlen, strcat`

2.11 _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

`where, array_map`

2.12 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

`array_info, array_shape, typeof, strlen`

2.13 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

```
1       2       3       4        5
6       7       8       9       10
```

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

```
6       7       8       9       10
```

Notes

This function ignores NaNs in the input array.

See Also

`min, sum, reshape`

2.14 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

```
1       2       3       4       5
6       7       8       9       10
```

generated by

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

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

```
1       2       3       4       5
```

Notes

This function ignores NaNs in the input array.

See Also

`max, sum, reshape`

2.15 _reshape

Synopsis

Copy an array to a new shape

Usage

```
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:

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

See Also

reshape, array_shape, array_info

2.16 reshape

Synopsis

Reshape an array

Usage

```
reshape (Array_Type A, Array_Type I)
```

Description

The reshape function changes the shaper 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

```
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

_reshape, array_info, array_shape

2.17 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

```
sum(a)/length(a)
```

See Also

cumsum, transpose, reshape

2.18 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

_reshape, reshape, sum, array_info, array_shape

2.19 where

Synopsis

Get indices where a numeric array is non-zero

Usage

```
Array_Type where (Array_Type a)
```

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.

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`.

See Also

`array_info, array_shape, _isnull`

Chapter 3

Associative Array Functions

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

`assoc_key_exists, 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

`assoc_get_values, assoc_key_exists, assoc_delete_key, 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

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

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

```
static define int_sort_fun (x, y)
{
   return sign (x - y);
}
define sort_and_print_values (a)
{
   variable v = assoc_get_values (a);
   variable i = array_sort (v, &int_sort_fun);
   v = v[i];
   foreach (v)
     {
        variable vi = ();
        () = fprintf (stdout, "%d\n", vi);
     }
}
```

See Also

`assoc_get_values`, `assoc_key_exists`, `assoc_delete_key`, `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

`assoc_get_keys`, `assoc_get_values`, `assoc_delete_key`

Chapter 4

Functions that Operate on Strings

4.1 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`, ...,`s_n`, using the string `delim` as a delimiter. The resulting string is equivalent to one obtained via

```
  s_1 + delim + s_2 + delim + ... + 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

`strjoin, is_list_element, extract_element, strchop, strcat`

4.2 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)
{
   variable nth = 0;
   while (NULL != extract_element (list, nth, delim))
     nth++;
   return nth;
}
```

Alternatively, the `strchop` function may be more useful. In fact, `extract_element` may be expressed in terms of the function `strchop` as

```
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:

```
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

`is_list_element`, `is_substr`, `strtok`, `strchop`, `create_delimited_string`

4.3 glob_to_regexp

Synopsis

Convert a globbing expression to a regular expression

Usage

```
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

`string_match, listdir`

4.4 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

`extract_element, is_substr, create_delimited_string`

4.5 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.

See Also

`substr, string_match, strreplace`

4.6 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

`eval, str_quote_string`

4.7 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

`sprintf, string, sscanf, vmessage`

4.8 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 according to the format specification string `fmt`.

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

The `%` 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 `%` character may be one or more of the following flag characters:

```
-         Use left-justification
#         Use alternate form for formatting.
0         Use 0 padding
+         Preceed 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 `-` 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 g, depending upon its value
c    as a character
%    a literal percent character
d    as a signed decimal integer
u    as an unsigned decimal integer
o    as an octal integer
X    as hexadecimal
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)                    ===> "3.14159"
sprintf("%g",PI)                    ===> "3.14159"
sprintf("%.2g",PI)                  ===> "3.1"
sprintf("%.2e",PI)                  ===> "3.14e+00"
sprintf("%.2f",PI)                  ===> "3.14"
sprintf("|% 8.2f|",PI)              ===> "|    3.14|"
sprintf("|%-8.2f|",PI)              ===> "|3.14    |"
sprintf("|%+8.2f|",PI)              ===> "|   +3.14|"
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

`string`, `sscanf`, `message`

4.9 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 `l`, 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'.
x     Matches a hexadecimal integer.
X     Matches a long hexadecimal integer (same as 'lx').
e,f,g Matches a decimal floating point number (Float_Type).
E,F,G Matches a double precision floating point number, same as 'lf'.
s     Matches a string of non-whitespace characters (String_Type).
c     Matches one character.  If width is given, width
      characters are matched.
n     Assigns the number of characters scanned so far.
[...] Matches zero or more characters from the set of characters
```

```
enclosed by the square brackets.  If '^' is given as the
first character, then the complement set is matched.
```

Example

Suppose that `s` is `"Coffee:  (3,4,12.4)"`. Then

```
n = sscanf (s, "%[a-zA-Z]: (%d,%d,%lf)", &item, &x, &y, &z);
```

will set `n` to `4`, `item` to `"Coffee"`, `x` to `3`, `y` to `4`, and `z` to the double precision number `12.4`. However,

```
n = sscanf (s, "%s: (%d,%d,%lf)", &item, &x, &y, &z);
```

will set `n` to `1`, `item` to `"Coffee:"` and the remaining variables will not be assigned.

See Also

`sprintf, unpack, string, atof, int, integer, string_match`

4.10 strbytelen

Synopsis

Get the number of bytes in a string

Usage

```
Int_Type strbytelen (String_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 `strlen` or `strcharlen` functions should be used.

See Also

`strlen, strcharlen, length`

4.11 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 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

`strsub, is_substr, strreplace, strbytelen`

4.12 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

`sprintf, strjoin`

4.13 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.

See Also

`strlen, strbytelen`

4.14 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, ",");
}
```

See Also

`strchopr, strjoin, strtok`

4.15 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

`strchop, strtok, strjoin`

4.16 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

The `strup` 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`.

See Also

`strup, strncmp`

4.17 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"`.

See Also

`strtrim, strtrans, str_delete_chars`

4.18 string_match

Synopsis

Match a string against a regular expression

Usage

```
Int_Type string_match(String_Type str, String_Type pat, Int_Type nth)
```

Description

The `string_match` function returns zero if `str` does not match regular expression specified by `pat`. This function performs the match starting at the `nth` byte in the string `str` (numbered from 1). This function returns the position in bytes (numbered from 1) of the start of the match in `str`. If the match fails, the function will return -1. 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.

See Also

`string_match_nth, strcmp, strncmp`

4.19 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:

```
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.

See Also

`string_match`

4.20 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 a delimiter `delim`. If `delim` is the empty string `""`, then the result will simply be the 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

`strchop, strcat`

4.21 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`.

See Also

`strbytelen, strcharlen, bstrlen, length, substr`

4.22 strlow

Synopsis

Convert a string to lowercase

Usage

`String_Type strlow (String_Type s)`

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 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.

See Also

`strup, tolower, strcmp, strtrim, define_case`

4.23 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.

See Also

`strncmp, strncharcmp, strcmp`

4.24 strncharcmp

Synopsis

Compare the first n characters of two strings

Usage

```
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.

See Also

`strncmp`, `strnbytecmp`, `strcmp`

4.25 strncmp

Synopsis

Compare the first few characters of two strings

Usage

```
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

```
strcmp ("apple", "appliance", 3);
```

will return zero since the first three characters match.

Notes

This function uses character semantics.

See Also

`strcmp`, `strlen`, `strncharcmp`, `strnbytecmp`

4.26 strreplace

Synopsis

Replace one or more substrings

Usage

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

       String_Type a, b, c, rep;
       Int_Type n, max_n;
```

Description

The `strreplace` function may be used to replace one or more occurrences of `b` in `a` with `c`. If the integer `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.

Example

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

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

See Also

`is_substr, strsub, strtrim, strtrans, str_delete_chars`

4.27 strsub

Synopsis

Replace a character with another in a string.

Usage

```
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

```
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

`is_substr, strreplace, strlen`

4.28 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 backslash
```

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

`strchop, strcompress, strjoin`

4.29 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:

```
\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
```

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
```

See Also

strreplace, strtrim, strup, strlow

4.30 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
```

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

See Also

strtrim_beg, strtrim_end, strcompress

4.31 strtrim_beg

Synopsis

Remove leading whitespace from a string

Usage

```
String_Type strtrim_beg (String_Type s [,String_Type w])
```

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 form more information about the whitespace parameter.

See Also

strtrim, strtrim_end, strcompress

4.32 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 form more information about the whitespace parameter.

See Also

`strtrim, strtrim_beg, strcompress`

4.33 strup

Synopsis

Convert a string to uppercase

Usage

```
String_Type strup (String_Type s)
```

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

```
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.

See Also

`strlow, toupper, strcmp, strtrim, define_case, strtrans`

4.34 str_delete_chars

Synopsis

Delete characters from a string

Usage

```
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 `del_set` from the string `str`. The result 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
```

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.

See Also

`strtrans, strreplace, strcompress`

4.35 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

`str_uncomment_string, make_printable_string`

4.36 str_replace

Synopsis

Replace a substring of a string (deprecated)

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 superceded by `strreplace`. It should no longer be used.

See Also

`strreplace`

4.37 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 multicharacter comment delimiters and it assumes that comments are not nested.

See Also

`str_quote_string`, `str_delete_chars`, `strtrans`

4.38 substr

Synopsis

Extract a substring from a string

Usage

```
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

```
substr ("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 `substr` function. In fact, if UTF-8 mode is not in effect, `substr(s,i+1,strlen(s))` is equivalent to `s[[i:]]`. Array indexing uses byte-semantics, not character semantics assumed by the `substr` function.

See Also

`is_substr`, `substrbytes`, `strlen`

4.39 substrbytes

Synopsis

Extract a byte sequence from a string

Usage

```
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

```
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 `substr` 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

`substr, 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

`bstring_to_array, init_char_array`

5.2 bstring_to_array

Synopsis

Convert a binary string to an array of characters

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 characters in the binary string.

See Also

`array_to_bstring, init_char_array`

5.3 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 chracters.

Example

```
s = "hello\0";
len = bstrlen (s);      % ==> len = 6
len = strlen (s);       % ==> len = 5
```

See Also

`strlen, length`

5.4 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:

```
c     char
C     unsigned char
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

```
a = pack ("cc", 'A', 'B');         % ==> a = "AB";
a = pack ("c2", 'A', 'B');         % ==> a = "AB";
a = pack ("xxcxxc", 'A', 'B');     % ==> a = "\0\0A\0\0B";
a = pack ("h2", 'A', 'B');         % ==> a = "\0A\0B" or "\0B\0A"
a = pack (">h2", 'A', 'B');        % ==> a = "\0\xA\0\xB"
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  "
a = pack ("S4S2", "AB", "CD");     % ==> a = "AB  CD"
a = pack ("z4", "AB");             % ==> a = "AB\0\0"
a = pack ("s4", "ABCDEFG");        % ==> a = "ABCD"
a = pack ("z4", "ABCDEFG");        % ==> a = "ABC\0"
```

See Also

`unpack, sizeof_pack, pad_pack_format, sprintf`

5.5 pad_pack_format

Synopsis

Add padding to a pack format

Usage

```
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

`pack, unpack, sizeof_pack`

5.6 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

`pack, unpack, pad_pack_format`

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

See Also

pack, sizeof_pack, 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

__add_unary, __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);
```

will generate the message:

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

See Also

__add_unary, __add_binary, __add_destroy

6.3 __add_unary

Synopsis

Extend a unary operator to a user-defined type

Usage

__add_unary (op, return_type, unary_func, user_type)

```
String_Type op;
Ref_Type unary_func;
DataType_Type return_type, user_type;
```

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:

```
"++", "--",
"abs", "sign", "sqr", "mul2", "_ispos", "_isneg", "_isnonneg",
```

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:

```
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

__add_binary, __add_string, __add_destroy

6.4 get_struct_field

Synopsis

Get the value associated with a structure field

Usage

```
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`.

Example

The following example illustrates how this function may be used to to print the value of a structure.

```
define print_struct (s)
{
   variable name;

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

See Also

set_struct_field, get_struct_field_names, array_info

6.5 get_struct_field_names

Synopsis

Retrieve the field names associated with a structure

Usage

```
String_Type[] = get_struct_field_names (Struct_Type s)
```

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 to print the value of a structure.

```
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

`_push_struct_field_values, get_struct_field`

6.6 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

`typeof, _typeof, _is_struct_type`

6.7 _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

get_struct_field_names, get_struct_field

6.8 set_struct_field

Synopsis

Set the value associated with a structure field

Usage

```
set_struct_field (s, field_name, field_value)

       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

get_struct_field, get_struct_field_names, set_struct_fields, array_info

6.9 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

set_struct_field, 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 object, Int_Type nth)
```

Description

The `list_append` function is like `list_insert` except this function inserts the object into the list after the `nth` item. See the documentation on `list_insert` for more details.

See Also

`list_insert, list_delete, list_pop, list_new, list_reverse`

7.2 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

`list_insert, list_append, list_pop, list_new, list_reverse`

7.3 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 before the `nth` position in a 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 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

`list_append`, `list_pop`, `list_delete`, `list_new`, `list_reverse`

7.4 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

`list_delete`, `list_insert`, `list_append`, `list_reverse`, `list_pop`

7.5 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 a 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

`list_delete, list_insert, list_append, list_reverse, list_new`

7.6 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

`list_new, list_insert, list_append, list_delete, list_pop`

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

`set_doc_files, get_doc_files, 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 second parameter `flags` is a bit mapped value whose bits are defined according to the following table

```
 1          Intrinsic Function
 2          User-defined Function
 4          Intrinsic Variable
 8          User-defined Variable
```

Example

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

  vmessage ("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

`is_defined, sprintf, _get_namespaces`

8.3 _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

`_trace_function, is_defined`

8.4 __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

is_defined, _apropos, _get_namespaces

8.5 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

set_doc_files, add_doc_file, get_doc_string_from_file

8.6 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

add_doc_file, set_doc_files, get_doc_files, _slang_doc_dir

8.7 _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

_apropos, use_namespace, implements, __get_defined_symbols

8.8 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:

```
 +1     intrinsic function
 +2     slang function
 -1     intrinsic variable
 -2     slang variable
  0     undefined
```

Example

Consider the function:

```
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

typeof, eval, autoload, __get_reference, __is_initialized

8.9 __is_initialized

Synopsis

Determine whether or not a variable has a value

Usage

```
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:

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

will always return zero, but

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

will return one.

See Also

`__get_reference, __uninitialize, is_defined, typeof, eval`

8.10 _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:

```
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

__pop_args, __push_args, typeof

8.11 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**.

```
files = glob ("/path/to/doc/files/*.sld");
set_doc_files ([files, get_doc_files ()]);
```

See Also

get_doc_files, add_doc_file, get_doc_string_from_file

8.12 _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

get_doc_string_from_file

8.13 _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

_slang_version_string

8.14 _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

_slang_version

Chapter 9

Mathematical Functions

9.1 abs

Synopsis

Compute the absolute value of a number

Usage

```
y = 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

`sign, sqr`

9.2 acos

Synopsis

Compute the arc-cosine of a number

Usage

```
y = 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

`cos, atan, acosh, cosh`

9.3 acosh

Synopsis

Compute the inverse cosh of a number

Usage

```
y = acosh (x)
```

Description

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

See Also

`cos, atan, acosh, cosh`

9.4 asin

Synopsis

Compute the arc-sine of a number

Usage

```
y = asin (x)
```

Description

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

See Also

`cos, atan, acosh, cosh`

9.5 asinh

Synopsis

Compute the inverse-sinh of a number

Usage

```
y = asinh (x)
```

Description

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

See Also

`cos, atan, acosh, cosh`

9.6 atan

Synopsis

Compute the arc-tangent of a number

Usage

```
y = atan (x)
```

Description

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

See Also

`atan2, cos, acosh, cosh`

9.7 atan2

Synopsis

Compute the arc-tangent of the ratio of two variables

Usage

```
z = atan2 (y, x)
```

Description

The `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

`hypot, cos, atan, acosh, cosh`

9.8 atanh

Synopsis

Compute the inverse-tanh of a number

Usage

```
y = atanh (x)
```

Description

The `atanh` function computes the inverse hyperbolic tangent of a number and returns the result. If its argument is an array, the `atanh` function will be applied to each element and the result returned as an array.

See Also

`cos, atan, acosh, cosh`

9.9 ceil

Synopsis

Round x up to the nearest integral value

Usage

```
y = 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

`floor, round`

9.10 Conj

Synopsis

Compute the complex conjugate of a number

Usage

```
z1 = Conj (z)
```

Description

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

See Also

`Real, Imag, abs`

9.11 cos

Synopsis

Compute the cosine of a number

Usage

```
y = cos (x)
```

Description

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

See Also

`cos, atan, acosh, cosh`

9.12 cosh

Synopsis

Compute the hyperbolic cosine of a number

Usage

```
y = cosh (x)
```

Description

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

See Also

`cos, atan, acosh, cosh`

9.13 _diff

Synopsis

Compute the absolute difference between two values

Usage

```
y = _diff (x, y)
```

Description

The `_diff` function returns a floating point number equal to the absolute value of the difference between its two arguments. If either argument is an array, an array of the corresponding values will be returned.

See Also

`abs`

9.14 exp

Synopsis

Compute the exponential of a number

Usage

```
y = exp (x)
```

Description

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

See Also

`cos, atan, acosh, cosh`

9.15 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

`ceil, round`

9.16 hypot

Synopsis

Compute sqrt(x^2+y^2)

Usage

```
r = hypot (x, y)
```

Description

The `hypot` function computes the quantity `sqrt(x^2+y^2)` except that it employs an algorithm that tries to avoid arithmetic overflow when `x` or `y` are large. If either argument is an array, an array of the corresponding values will be returned.

See Also

`atan2, cos, atan, acosh, cosh`

9.17 Imag

Synopsis

Compute the imaginary part of a number

Usage

```
i = Imag (z)
```

Description

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

See Also

`Real, Conj, abs`

9.18 isinf

Synopsis

Test for infinity

Usage

```
y = 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

`isnan, _Inf`

9.19 isnan

Synopsis

isnan

Usage

```
y = 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

`isinf, _NaN`

9.20 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

`cos, atan, acosh, cosh`

9.21 log10

Synopsis

Compute the base-10 logarithm of a number

Usage

```
y = log10 (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

`cos, atan, acosh, cosh`

9.22 _max

Synopsis

Compute the maximum of two values

Usage

```
z = _max (x,y)
```

Description

The `_max` function returns a floating point number equal to the maximum value of its two arguments. If either argument is an array, an array of the corresponding values will be returned.

Notes

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

See Also

max, _min, min

9.23 _min

Synopsis

Compute the minimum of two values

Usage

```
z = _min (x,y)
```

Description

The _min function returns a floating point number equal to the minimum value of its two arguments. If either argument is an array, an array of the corresponding values will be returned.

Notes

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

See Also

min, _max, max

9.24 mul2

Synopsis

Multiply a number by 2

Usage

```
y = 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

sqr, abs

9.25 polynom

Synopsis

Evaluate a polynomial

Usage

```
Double_Type polynom(Double_Type a, b, ...c, Integer_Type n, Double_Type x)
```

Description

The `polynom` function returns the value of the polynomial expression:

```
ax^n + bx^(n - 1) + ... c
```

Notes

The `polynom` function should be extended to work with complex and array data types. The current implementation is limited to `Double_Type` quantities.

See Also

`exp`

9.26 Real

Synopsis

Compute the real part of a number

Usage

```
r = Real (z)
```

Description

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

See Also

`Imag, Conj, abs`

9.27 round

Synopsis

Round to the nearest integral value

Usage

```
y = 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

`floor, ceil`

9.28 set_float_format

Synopsis

Set the format for printing floating point values.

Usage

```
set_float_format (String_Type fmt)
```

Description

The `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 `string` function, any anything based upon the `string` function. The default value is `"%g"`

Example

```
s = string (PI);                % --> s = "3.14159"
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

`string, sprintf, double`

9.29 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

`abs`

9.30 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

`cos, atan, acosh, cosh`

9.31 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

`cos, atan, acosh, cosh`

9.32 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 `sqr(x)` will result in faster executing code. However, if `x` is an array, then `sqr(x)` will execute faster.

See Also

`abs, mul2`

9.33 sqrt

Synopsis

Compute the square root of a number

Usage

```
y = sqrt (x)
```

Description

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

See Also

`sqr, cos, atan, acosh, cosh`

9.34 tan

Synopsis

Compute the tangent of a number

Usage

```
y = tan (x)
```

Description

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

See Also

`cos, atan, acosh, cosh`

9.35 tanh

Synopsis

Compute the hyperbolic tangent of a number

Usage

```
y = tanh (x)
```

Description

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

See Also

`cos, atan, acosh, 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:

```
EPERM           "Not owner"
ENOENT          "No such file or directory"
ESRCH           "No such process"
ENXIO           "No such device or address"
ENOEXEC         "Exec format error"
EBADF           "Bad file number"
ECHILD          "No children"
ENOMEM          "Not enough core"
EACCES          "Permission denied"
EFAULT          "Bad address"
ENOTBLK         "Block device required"
EBUSY           "Mount device busy"
EEXIST          "File exists"
EXDEV           "Cross-device link"
ENODEV          "No such device"
ENOTDIR         "Not a directory"
EISDIR          "Is a directory"
```

```
EINVAL          "Invalid argument"
ENFILE          "File table overflow"
EMFILE          "Too many open files"
ENOTTY          "Not a typewriter"
ETXTBSY         "Text file busy"
EFBIG           "File too large"
ENOSPC          "No space left on device"
ESPIPE          "Illegal seek"
EROFS           "Read-only file system"
EMLINK          "Too many links"
EPIPE           "Broken pipe"
ELOOP           "Too many levels of symbolic links"
ENAMETOOLONG    "File name too long"
```

Example

The `mkdir` function will attempt to create a directory. If that directory already exists, the function will fail and set `errno` to `EEXIST`.

```
define create_dir (dir)
{
   if (0 == mkdir (dir)) return;
   if (errno != EEXIST)
     throw IOError, sprintf ("mkdir %s failed: %s",
                             dir, errno_string (errno));
}
```

See Also

`errno_string`, `error`, `mkdir`

10.2 errno_string

Synopsis

Return a string describing an errno.

Usage

```
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:

```
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

errno, 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

verror, message

10.4 message

Synopsis

Print a string onto the message device

Usage

message (String_Type s

Description

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

Example

```
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

`vmessage, sprintf, error`

10.5 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

`error, verror`

10.6 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

`error, message`

10.7 verror

Synopsis

Generate an error condition (deprecated)

Usage

```
verror (String_Type fmt, ...)
```

Description

This function has been deprecated in favor or `throw`.

The `verror` function performs the same role as the `error` function. The only difference is that instead of a single string argument, `verror` takes a sprintf style argument list.

Example

```
define open_file (file)
{
   variable fp;

   fp = fopen (file, "r");
   if (fp == NULL) verror ("Unable to open %s", file);
   return fp;
}
```

Notes

In the current implementation, the `verror` function is not an intrinsic function. Rather it is a predefined **S-Lang** function using a combination of `sprintf` and `error`.

To generate a specific exception, a `throw` statement should be used. In fact, a `throw` statement such as:

```
if (fp == NULL)
  throw OpenError, "Unable to open $file"$;
```

is preferable to the use of `verror` in the above example.

See Also

`error, Sprintf, vmessage`

10.8 vmessage

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

`message, sprintf, Sprintf, 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

`time, _time, localtime, 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`.

See Also

`localtime, _time, mktime`

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

`gmtime, _time, ctime, 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.

See Also

`localtime, gmtime, _time`

11.5 _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.

Example

See Also

`_toc, times, tic, toc`

11.6 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

`toc, times`

11.7 _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

`ctime, time, localtime, gmtime`

11.8 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

`ctime, message, substr`

11.9 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

`_tic`, `_toc`, `_time`

11.10 _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.

Example

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

`_tic`, `_tic`, `_toc`, `times`, `_time`

11.11 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

`tic`, `_tic`, `_toc`, `times`, `_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 (x)))
     return atof (s);
   throw InvalidParmError, "$s is not a double"$;
}
```

See Also

`typecast, double, _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

`integer, atol, atoll, atof, sscanf`

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

`integer, atoi, atoll, atof, sscanf`

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

`integer, atoi, atol, atof, sscanf`

12.5 char

Synopsis

Convert an ascii value into a string

Usage

```
String_Type char (Integer_Type c)
```

Description

The `char` function converts an integer 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"`.

Notes

If UTF-8 mode is in effect (`_slang_utf8_ok` is non-zero), the resulting single character may be represented by several bytes.

See Also

`integer, string, typedef`

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

`strlow, 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

`typecast, atof, 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

`typecast, double, integer, char, isdigit`

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 type-mismatch error will be generated.

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

`typecast, _slang_guess_type, string, sprintf, char`

12.10 isdigit

Synopsis

Tests for a decimal digit character

Usage

```
Integer_Type isdigit (s)
```

Description

This function returns a non-zero value if the character represented by `s` is a digit; otherwise, it returns zero. If `s` is a string, the first character of `s` will be used for the test.

Example

A simple, user defined implementation of `isdigit` is

```
define isdigit (x)
{
   return ((x <= '9') and (x >= '0'));
}
```

However, the intrinsic function `isdigit` executes many times faster than the representation defined above, and works properly when `x` is a Unicode character.

See Also

`int, integer`

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

`integer, string, double, atof, __is_datatype_numeric`

12.12 string

Synopsis

Convert an object to a string representation.

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

```
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

`typecast, sprintf, integer, char`

12.13 tolower

Synopsis

Convert a character to lowercase.

Usage

```
Integer_Type lower (Integer_Type ch)
```

Description

This function takes an integer `ch` and returns its lowercase equivalent.

See Also

`toupper, strup, strlow, int, char, define_case`

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

`tolower, strup, strlow, int, char, 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

`int, double, 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

`typeof, array_info, _slang_guess_type, typecast`

12.17 typeof

Synopsis

Get the data type of an object

Usage

```
DataType_Type typeof (x)
```

Description

This function returns the data type of `x`.

Example

```
if (Integer_Type == typeof (x)) message ("x is an integer");
```

See Also

_typeof, is_struct_type, array_info, _slang_guess_type, 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

`ferror, feof, 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 a 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

`fopen, fgets, fflush, pclose, 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

The `fclose` function does not close the `File_Type` object returned from this function. The underlying file object must be closed by the `close` function.

See Also

`fileno, fopen, open, close, fclose`

13.4 feof

Synopsis

Get the end-of-file status

Usage

```
Integer_Type feof (File_Type fp)
```

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 `clearerr` function.

See Also

`ferror, clearerr, fopen`

13.5 ferror

Synopsis

Determine the error status of an open file descriptor

Usage

```
Integer_Type ferror (File_Type fp)
```

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 `clearerr` function.

See Also

`feof, clearerr, fopen`

13.6 fflush

Synopsis

Flush an output stream

Usage

```
Integer_Type fflush (File_Type fp)
```

Description

The `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 `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:

```
() = fputs ("Enter value> ", stdout);
() = fflush (stdout);
```

See Also

`fopen, fclose`

13.7 fgets

Synopsis

Read a line from a file

Usage

```
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:

```
define read_file (file)
{
   variable buf, fp, root, tail;
   variable list_type = struct { text, next };

   root = NULL;

   fp = fopen(file, "r");
   if (fp == NULL)
     error("fopen %s failed." file);
   while (-1 != fgets (&buf, fp))
     {
        if (root == NULL)
          {
             root = @list_type;
             tail = root;
          }
        else
          {
             tail.next = @list_type;
             tail = tail.next;
          }
        tail.text = buf;
        tail.next = NULL;
     }
   () = fclose (fp);
   return root;
}
```

See Also

`fgetslines, fopen, fclose, fputs, fread, error`

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 lines 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:

```
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

`fgets, fread, fopen, 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 (file, str)
{
   variable fp = fopen (file, "a");
   if (fp == NULL)
     throw OpenError, "$file could not be opened"$;
   () = fputs (string, 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

fclose, fgets, fputs, 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

fputs, printf, fwrite, 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 (s, fp))
       or (-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

`fclose, fopen, fgets, 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 (fp, lines))
  throw WriteError;
```

See Also

`fputs, fgetslines, 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;
   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`).

See Also

`fread_bytes, fwrite, fgets, fopen, pack, 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

`fread, fwrite, fgets, fopen, pack, unpack`

13.15 fseek

Synopsis

Reposition a stdio stream

Usage

```
Integer_Type fseek (File_Type fp, LLong_Type ofs, Integer_Type whence
```

Description

The `fseek` function may be used to reposition the file position pointer associated with the open file stream `fp`. Specifically, it moves the pointer `ofs` bytes relative to the position indicated by `whence`. If `whence` is set to one of the symbolic constants `SEEK_SET`, `SEEK_CUR`, or `SEEK_END`, the offset is relative to the start of the file, the current position indicator, or end-of-file, respectively.

The function returns 0 upon success, or -1 upon failure and sets `errno` accordingly.

Example

define rewind (fp) { if (0 == fseek (fp, 0, SEEK_SET)) return; vmessage (”rewind failed, reason: %s”, errno_string (errno)); }

See Also

`ftell, fopen`

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

`fseek, 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:

```
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:

```
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

`fread, fputs, fopen, pack, unpack`

13.18 pclose

Synopsis

Close a process pipe

Usage

```
Integer_Type pclose (File_Type fp)
```

Description

The `pclose` function waits for the process associated with `fp` to exit and the returns the exit status of the command.

See Also

`pclose, fclose`

13.19 popen

Synopsis

Open a pipe to a process

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.

See Also

`pclose, fopen`

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:

```
() = printf ("%s laid %d eggs\n", chicken_name, num_egg);
```

See Also

`fputs, printf, fwrite, message`

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 and open file descriptor created by the `open` function. Upon success `0` is returned, otherwise the function returns `-1` and sets `errno` accordingly.

See Also

`open, fclose, read, write`

14.2 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

`open, close`

14.3 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.

See Also

`fopen, open, fclose, close, dup_fd`

14.4 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

`fopen, fclose, fileno`

14.5 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

`fseek, ftell, open, close`

14.6 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:

```
O_RDONLY   (read-only)
O_WRONLY   (write-only)
O_RDWR     (read/write)
```

In addition, `flags` may also be bitwise-or'd with any of the following:

```
O_BINARY   (open the file in binary mode)
O_TEXT     (open the file in text mode)
O_CREAT    (create the file if it does not exist)
O_EXCL     (fail if the file already exists)
O_NOCTTY   (do not make the device the controlling terminal)
O_TRUNC    (truncate the file if it exists)
O_APPEND   (open the file in append mode)
O_NONBLOCK (open the file in non-blocking mode)
```

Some of these flags make sense only when combined with other flags. For example, if O_EXCL is used, then O_CREAT must also be specified, otherwise unpredictable behavior may result.

If `O_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

`fopen, close, read, write, stat_file`

14.7 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

`fread, open, close, write`

14.8 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

`read, fwrite, open, close`

Chapter 15

Directory Functions

15.1 chdir

Synopsis

Change the current working directory

Usage

```
Int_Type chdir (String_Type dir)
```

Description

The `chdir` function may be used to change the current working directory to the directory specified by `dir`. Upon success it returns zero. Upon failure it returns `-1` and sets `errno` accordingly.

See Also

`mkdir, stat_file`

15.2 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

`chown, stat_file`

15.3 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

`chmod, stat_file`

15.4 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

`mkdir, chdir, errno`

15.5 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

`stat_file, stat_is, length`

15.6 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

`stat_file, readlink`

15.7 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

```
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

`rmdir, getcwd, chdir, fopen, errno`

15.8 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

`symlink, lstat_file, stat_file, stat_is`

15.9 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

`rename, rmdir`

15.10 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

`remove, errno`

15.11 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

`rename, remove, mkdir`

15.12 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

`lstat_file, stat_is`

15.13 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

`stat_file, lstat_file`

15.14 symlink

Synopsis

Create a symbolic link

Usage

```
status = 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

`readlink`

Chapter 16

Functions that Parse Filenames

16.1 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

`path_dirname, path_extname, path_concat, path_is_absolute`

16.2 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

`path_dirname, path_basename, path_extname, path_concat, path_is_absolute`

16.3 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

`path_dirname, path_basename, path_extname, path_is_absolute`

16.4 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

`path_basename, path_extname, path_concat, path_is_absolute`

16.5 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 `""`.

Notes

Under VMS, the file version number is not returned as part of the extension.

See Also

`path_sans_extname, path_dirname, path_basename, path_concat, path_is_absolute`

16.6 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

`set_slang_load_path, get_slang_load_path`

16.7 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 is `filename` refers to an absolute filename, otherwise it returns zero.

See Also

`path_dirname, path_basename, path_extname, path_concat`

16.8 path_sans_extname

Synopsis

Strip the extension from a filename

Usage

```
String_Type path_sans_extname (String_Type filename)
```

Description

The `path_sans_extname` function removes the file name extension (including the dot) from the filename and returns the result.

See Also

`path_basename_sans_extname`, `path_extname`, `path_basename`, `path_dirname`, `path_concat`

Chapter 17

System Call Functions

17.1 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

`getgid, geteuid, setgid`

17.2 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

`getuid, setuid, setgid`

17.3 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

`getpid, getppid`

17.4 getpid

Synopsis

Get the current process id

Usage

```
Integer_Type getpid ()
```

Description

The `getpid` function returns the current process identification number.

See Also

`getppid, getgid`

17.5 getppid

Synopsis

Get the parent process id

Usage

```
Integer_Type getppid ()
```

Description

The `getpid` function returns the process identification number of the parent process.

Notes

This function is not supported by all systems.

See Also

`getpid, getgid`

17.6 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

`getuid, getegid`

17.7 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:

```
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

`getpid`

17.8 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

`stat_file`

17.9 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

`getgid, setuid`

17.10 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

`setgid, setuid`

17.11 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

`setgid, setpgid, getuid, geteuid`

17.12 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.

17.13 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

`popen, listdir`

17.14 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

`stat_file`

17.15 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

`getenv`

Chapter 18

Eval Functions

18.1 autoload

Synopsis

Load a function from a file

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 `func` 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

`evalfile, import`

18.2 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

`evalfile`

18.3 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

`is_defined, autoload, evalfile`

18.4 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.

See Also

`eval, autoload, set_slang_load_path, get_slang_load_path`

18.5 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

`set_slang_load_path, path_get_delimiter`

18.6 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

```
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;

   variable d = path_get_delimiter ();
   set_slang_load_path (strcat (p, d, get_slang_load_path ()));
}
```

Notes

Some applications may not support the built-in load-path searching facility provided by the underlying library.

See Also

`get_slang_load_path`, `path_get_delimiter`, `evalfile`, `autoload`

Chapter 19

Module Functions

19.1 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

`import, set_import_module_path`

19.2 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

`set_import_module_path, use_namespace, current_namespace, getenv, evalfile`

19.3 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

`import, get_import_module_path, path_get_delimiter`

Chapter 20

Debugging Functions

20.1 _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** interpeter 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 value of `_boseos_info` controls the generation of code for callbacks as follows:

```
Value      Description
-----------------------------------------------------------------
  0        No code for making callbacks will be produced.
  1        Callback generation will take place for all non-branching
           statements.
  2        Same as for 1 with the addition that code will also be
           generated for branching statements.
```

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`.

Example

Consider the following:

```
_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.sl` make use of this facility.

See Also

`_set_bos_handler, _set_eos_handler, _debug_info`

20.2 _clear_error

Synopsis

Clear an error condition (deprecated)

Usage

```
_clear_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:

```
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

_trace_function, _slangtrace, _traceback

20.3 _debug_info

Synopsis

Configure debugging information

Usage

```
Integer_Type _debug_info
```

Description

The _debug_info variable controls whether or not extra code should be generated for additional debugging and traceback information. Currently, if _debug_info is zero, no extra code will be generated; otherwise extra code will be inserted into the compiled bytecode for additional debugging data.

The value of this variable is local to each compilation unit and setting its value in one unit has no effect upon its value in other units.

Example

```
_debug_info = 1;   % Enable debugging information
```

Notes

Setting this variable to a non-zero value may slow down the interpreter somewhat.

The value of this variable is not currently used.

See Also

_traceback, _slangtrace

20.4 _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

```
static define bos_handler (file, line)
{
  () = fputs ("About to execute $file:$line\n"$, 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

`_set_eos_handler`, `_boseos_info`

20.5 _set_eos_handler

Synopsis

Set the beginning 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

```
static define eos_handler ()
{
  () = fputs ("Done executing the statement\n", stdout);
}
_set_eos_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

`_set_eos_handler`, `_boseos_info`

20.6 _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

_trace_function, _traceback, _print_stack

20.7 _traceback

Synopsis

Generate a traceback upon error

Usage

```
Integer_Type _traceback
```

Description

_traceback is an intrinsic integer variable whose value controls whether or not a traceback of the call stack is to be generated upon error. If _traceback is greater than zero, a full traceback will be generated, which includes the values of local variables. If the value is less than zero, a traceback will be generated without local variable information, and if _traceback is zero the traceback will not be generated.

Running **slsh** with the -g option causes this variable to be set to 1.

See Also

_boseos_info

20.8 _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

_slangtrace, _traceback

Chapter 21

Stack Functions

21.1 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

`pop, typeof`

21.2 exch

Synopsis

Exchange two items on the stack

Usage

```
exch ()
```

Description

The `exch` swaps the two top items on the stack.

See Also

`pop, _stk_reverse, _stk_roll`

21.3 pop

Synopsis

Discard an item from the stack

Usage

```
pop ()
```

Description

The `pop` function removes the top item from the stack.

See Also

`_pop_n`, `__pop_args`

21.4 __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:

```
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.

```
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`.

See Also

`__push_args, typeof, _pop_n`

21.5 _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

`_stkdepth, pop`

21.6 _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

`_stkdepth, string, message`

21.7 __push_args

Synopsis

Remove 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.

See Also

`__pop_args, typeof, _pop_n`

21.8 _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

`_print_stack, _stk_reverse, _stk_roll`

21.9 _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

`_stkdepth, _stk_roll`

21.10 _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 `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 `_stk_roll(-3)` will change the stack to:

```
item-2
item-0
item-1
item-3
```

Notes

This function only has an effect if `abs(n) > 1`.

See Also

`_stkdepth`, `_stk_reverse`, `_pop_n`, `_print_stack`

Chapter 22

Miscellaneous Functions

22.1 _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;   % declare variables as static.
X = 1;
```

is equivalent to

```
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.

22.2 __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

`typeof`, `_typeof`, `__class_type`, `__datatype`

22.3 __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

`typeof`, `_typeof`, `__class_id`, `__datatype`

22.4 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

`implements, use_namespace, import, evalfile`

22.5 _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

_eqs (1, 1) ===¿ 1 _eqs (1, 1.0) ===¿ 1 _eqs (”a”, 1) ===¿ 0 _eqs ([1,2], [1.0,2.0]) ===¿ 1

See Also

`typeof, _eqs, __get_reference, __is_callable`

Notes

For testing sameness, use `__is_same`.

22.6 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

putenv, strlen, is_defined

22.7 __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:

```
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

is_defined, typeof, eval, autoload, __is_initialized, __uninitialize

22.8 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 accessable. 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:

```
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:

```
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

`use_namespace, current_namespace, import`

22.9 __is_callable

Synopsis

Determine whether or not an object is callable

Usage

```
Int_Type __is_callable (obj)
```

Description

This function may be used to determine if an object is callable. It returns 1 if the argument is callable, or zero otherwise.

Example

__is_callable (7) ==¿ 0 __is_callable (&sin) ==¿ 1

See Also

`__is_numeric, is_defined`

22.10 __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 `1` if the argument is numeric, or zero otherwise. 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); ==¿ 1 __is_numeric ([1,2]); ==¿ 1 __is_numeric ({1,2}); ==¿ 0

See Also

`typeof`

22.11 __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

See Also

`typeof, _eqs, __get_reference, __is_callable`

Notes

For testing equality, use `_eqs`.

22.12 putenv

Synopsis

Add or change an environment variable

Usage

```
putenv (String_Type s)
```

Description

This functions 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

`getenv, sprintf`

22.13 _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

`_slang_doc_dir`

22.14 _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

`strbytelen, strlen, strcharlen`

22.15 __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(z);
__uninitialize (&z);
```

See Also

`__tmp, __is_initialized`

22.16 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

`implements, current_namespace, import`