

Simple C Expat Wrapper (SCEW)

1.1.3

Generated by Doxygen 1.7.1

Tue Sep 14 2010 19:27:48

Contents

1	Module Index	1
1.1	Modules	1
2	Data Structure Index	3
2.1	Data Structures	3
3	File Index	5
3.1	File List	5
4	Module Documentation	7
4.1	Attributes	7
4.1.1	Detailed Description	8
4.2	Allocation	8
4.2.1	Detailed Description	8
4.2.2	Function Documentation	8
4.2.2.1	scew_attribute_create	8
4.2.2.2	scew_attribute_copy	9
4.2.2.3	scew_attribute_free	9
4.3	Comparison	9
4.3.1	Detailed Description	9
4.3.2	Function Documentation	9
4.3.2.1	scew_attribute_compare	9
4.4	Accessors	10
4.4.1	Detailed Description	10
4.4.2	Function Documentation	10
4.4.2.1	scew_attribute_name	10
4.4.2.2	scew_attribute_value	11
4.4.2.3	scew_attribute_set_name	11
4.4.2.4	scew_attribute_set_value	11

4.5	Hierarchy	11
4.5.1	Detailed Description	12
4.5.2	Function Documentation	12
4.5.2.1	scew_attribute_parent	12
4.6	Elements	12
4.6.1	Detailed Description	13
4.7	Allocation	13
4.7.1	Detailed Description	13
4.7.2	Function Documentation	14
4.7.2.1	scew_element_create	14
4.7.2.2	scew_element_copy	14
4.7.2.3	scew_element_free	14
4.8	Search and iteration	14
4.8.1	Detailed Description	15
4.8.2	Function Documentation	15
4.8.2.1	scew_element_by_name	15
4.8.2.2	scew_element_by_index	15
4.8.2.3	scew_element_list_by_name	15
4.9	Comparison	16
4.9.1	Detailed Description	16
4.9.2	Typedef Documentation	16
4.9.2.1	scew_element_cmp_hook	16
4.9.3	Function Documentation	16
4.9.3.1	scew_element_compare	16
4.10	Accessors	17
4.10.1	Detailed Description	18
4.10.2	Function Documentation	18
4.10.2.1	scew_element_name	18
4.10.2.2	scew_element_contents	18
4.10.2.3	scew_element_set_name	18
4.10.2.4	scew_element_set_contents	19
4.10.2.5	scew_element_free_contents	19
4.11	Hierarchy	19
4.11.1	Detailed Description	20
4.11.2	Function Documentation	20
4.11.2.1	scew_element_count	20

4.11.2.2	scew_element_parent	21
4.11.2.3	scew_element_children	21
4.11.2.4	scew_element_add	21
4.11.2.5	scew_element_add_pair	21
4.11.2.6	scew_element_add_element	22
4.11.2.7	scew_element_delete_all	22
4.11.2.8	scew_element_delete_all_by_name	22
4.11.2.9	scew_element_delete_by_name	22
4.11.2.10	scew_element_delete_by_index	23
4.11.2.11	scew_element_detach	23
4.12	Attributes	23
4.12.1	Detailed Description	24
4.12.2	Function Documentation	24
4.12.2.1	scew_element_attribute_count	24
4.12.2.2	scew_element_attributes	24
4.12.2.3	scew_element_attribute_by_name	25
4.12.2.4	scew_element_attribute_by_index	25
4.12.2.5	scew_element_add_attribute	25
4.12.2.6	scew_element_add_attribute_pair	26
4.12.2.7	scew_element_delete_attribute_all	26
4.12.2.8	scew_element_delete_attribute	26
4.12.2.9	scew_element_delete_attribute_by_name	26
4.12.2.10	scew_element_delete_attribute_by_index	27
4.13	Errors	27
4.13.1	Detailed Description	27
4.14	Codes and descriptions	27
4.14.1	Detailed Description	28
4.14.2	Enumeration Type Documentation	28
4.14.2.1	scew_error	28
4.14.3	Function Documentation	28
4.14.3.1	scew_error_code	28
4.14.3.2	scew_error_string	29
4.15	Expat errors	29
4.15.1	Detailed Description	29
4.15.2	Function Documentation	29
4.15.2.1	scew_error_expat_code	29

4.15.2.2	scew_error_expat_string	30
4.15.2.3	scew_error_expat_line	30
4.15.2.4	scew_error_expat_column	30
4.16	Lists	30
4.16.1	Detailed Description	31
4.16.2	Typedef Documentation	31
4.16.2.1	scew_list_hook	31
4.16.2.2	scew_cmp_hook	31
4.17	Allocation	32
4.17.1	Detailed Description	32
4.17.2	Function Documentation	32
4.17.2.1	scew_list_create	32
4.17.2.2	scew_list_free	32
4.18	Accessors	32
4.18.1	Detailed Description	33
4.18.2	Function Documentation	33
4.18.2.1	scew_list_data	33
4.18.2.2	scew_list_size	33
4.19	Modifiers	33
4.19.1	Detailed Description	34
4.19.2	Function Documentation	34
4.19.2.1	scew_list_append	34
4.19.2.2	scew_list_prepend	34
4.19.2.3	scew_list_delete	35
4.19.2.4	scew_list_delete_item	35
4.20	Traverse	35
4.20.1	Detailed Description	36
4.20.2	Function Documentation	36
4.20.2.1	scew_list_first	36
4.20.2.2	scew_list_last	36
4.20.2.3	scew_list_next	36
4.20.2.4	scew_list_previous	37
4.20.2.5	scew_list_foreach	37
4.21	Search	37
4.21.1	Detailed Description	38
4.21.2	Function Documentation	38

4.21.2.1	scew_list_index	38
4.21.2.2	scew_list_find	38
4.21.2.3	scew_list_find_custom	38
4.22	Parser	39
4.22.1	Detailed Description	39
4.23	Allocation	39
4.23.1	Detailed Description	40
4.23.2	Function Documentation	40
4.23.2.1	scew_parser_create	40
4.23.2.2	scew_parser_namespace_create	40
4.23.2.3	scew_parser_free	40
4.24	Load	41
4.24.1	Detailed Description	41
4.24.2	Typedef Documentation	41
4.24.2.1	scew_parser_load_hook	41
4.24.3	Function Documentation	42
4.24.3.1	scew_parser_load	42
4.24.3.2	scew_parser_load_stream	42
4.24.3.3	scew_parser_reset	43
4.24.3.4	scew_parser_set_element_hook	43
4.24.3.5	scew_parser_set_tree_hook	44
4.24.3.6	scew_parser_ignore_whitespaces	44
4.25	Accessors	44
4.25.1	Detailed Description	45
4.25.2	Function Documentation	45
4.25.2.1	scew_parser_extern	45
4.26	Input/Output	45
4.26.1	Detailed Description	45
4.27	Printer	46
4.27.1	Detailed Description	46
4.28	Allocation	46
4.28.1	Detailed Description	47
4.28.2	Function Documentation	47
4.28.2.1	scew_printer_create	47
4.28.2.2	scew_printer_free	47
4.29	Properties	47

4.29.1	Detailed Description	48
4.29.2	Function Documentation	48
4.29.2.1	scew_printer_set_indented	48
4.29.2.2	scew_printer_set_indentation	48
4.30	Output	48
4.30.1	Detailed Description	49
4.30.2	Function Documentation	49
4.30.2.1	scew_printer_set_writer	49
4.30.2.2	scew_printer_print_tree	50
4.30.2.3	scew_printer_print_element	50
4.30.2.4	scew_printer_print_element_children	50
4.30.2.5	scew_printer_print_element_attributes	51
4.30.2.6	scew_printer_print_attribute	51
4.31	Readers	51
4.31.1	Detailed Description	52
4.31.2	Function Documentation	52
4.31.2.1	scew_reader_create	52
4.31.2.2	scew_reader_data	53
4.31.2.3	scew_reader_read	53
4.31.2.4	scew_reader_end	54
4.31.2.5	scew_reader_error	54
4.31.2.6	scew_reader_close	54
4.31.2.7	scew_reader_free	55
4.32	Memory	55
4.32.1	Detailed Description	55
4.32.2	Function Documentation	55
4.32.2.1	scew_reader_buffer_create	55
4.33	Files	56
4.33.1	Detailed Description	56
4.33.2	Function Documentation	56
4.33.2.1	scew_reader_file_create	56
4.33.2.2	scew_reader_fp_create	57
4.34	Text utilities	57
4.34.1	Detailed Description	59
4.34.2	Define Documentation	60
4.34.2.1	scew_memcpy	60

4.34.2.2	scew_memmove	60
4.34.3	Function Documentation	60
4.34.3.1	scew_strcmp	60
4.34.3.2	scew_strdup	60
4.34.3.3	scew_strtrim	60
4.34.3.4	scew_isempty	61
4.34.3.5	scew_strescape	61
4.35	Trees	61
4.35.1	Detailed Description	62
4.36	Allocation	62
4.36.1	Detailed Description	63
4.36.2	Function Documentation	63
4.36.2.1	scew_tree_create	63
4.36.2.2	scew_tree_copy	63
4.36.2.3	scew_tree_free	63
4.37	Comparison	63
4.37.1	Detailed Description	64
4.37.2	Typedef Documentation	64
4.37.2.1	scew_tree_cmp_hook	64
4.37.3	Function Documentation	64
4.37.3.1	scew_tree_compare	64
4.38	Properties	65
4.38.1	Detailed Description	66
4.38.2	Enumeration Type Documentation	66
4.38.2.1	scew_tree_standalone	66
4.38.3	Function Documentation	66
4.38.3.1	scew_tree_xml_version	66
4.38.3.2	scew_tree_set_xml_version	66
4.38.3.3	scew_tree_xml_encoding	67
4.38.3.4	scew_tree_set_xml_encoding	67
4.38.3.5	scew_tree_xml_standalone	67
4.38.3.6	scew_tree_set_xml_standalone	68
4.39	Contents	68
4.39.1	Detailed Description	69
4.39.2	Function Documentation	69
4.39.2.1	scew_tree_root	69

4.39.2.2	scew_tree_set_root	69
4.39.2.3	scew_tree_set_root_element	69
4.39.2.4	scew_tree_xml_preamble	70
4.39.2.5	scew_tree_set_xml_preamble	70
4.40	Writers	71
4.40.1	Detailed Description	72
4.40.2	Function Documentation	72
4.40.2.1	scew_writer_create	72
4.40.2.2	scew_writer_data	72
4.40.2.3	scew_writer_write	73
4.40.2.4	scew_writer_end	73
4.40.2.5	scew_writer_error	73
4.40.2.6	scew_writer_close	74
4.40.2.7	scew_writer_free	74
4.41	Memory	74
4.41.1	Detailed Description	75
4.41.2	Function Documentation	75
4.41.2.1	scew_writer_buffer_create	75
4.42	Files	75
4.42.1	Detailed Description	76
4.42.2	Function Documentation	76
4.42.2.1	scew_writer_file_create	76
4.42.2.2	scew_writer_fp_create	76
5	Data Structure Documentation	77
5.1	scew_reader_hooks Struct Reference	77
5.1.1	Detailed Description	77
5.1.2	Field Documentation	77
5.1.2.1	read	77
5.1.2.2	end	77
5.1.2.3	error	78
5.1.2.4	close	78
5.1.2.5	free	78
5.2	scew_writer_hooks Struct Reference	78
5.2.1	Detailed Description	78
5.2.2	Field Documentation	78
5.2.2.1	write	78

5.2.2.2	end	79
5.2.2.3	error	79
5.2.2.4	close	79
5.2.2.5	free	79
6	File Documentation	81
6.1	attribute.h File Reference	81
6.1.1	Detailed Description	82
6.2	bool.h File Reference	82
6.2.1	Detailed Description	82
6.3	element.h File Reference	83
6.3.1	Detailed Description	85
6.4	error.h File Reference	86
6.4.1	Detailed Description	86
6.5	export.h File Reference	87
6.5.1	Detailed Description	87
6.6	list.h File Reference	87
6.6.1	Detailed Description	88
6.7	parser.h File Reference	89
6.7.1	Detailed Description	90
6.8	printer.h File Reference	90
6.8.1	Detailed Description	91
6.9	reader.h File Reference	91
6.9.1	Detailed Description	92
6.10	reader_buffer.h File Reference	92
6.10.1	Detailed Description	93
6.11	reader_file.h File Reference	93
6.11.1	Detailed Description	93
6.12	scew.h File Reference	94
6.12.1	Detailed Description	94
6.13	str.h File Reference	94
6.13.1	Detailed Description	97
6.14	tree.h File Reference	97
6.14.1	Detailed Description	98
6.15	writer.h File Reference	99
6.15.1	Detailed Description	100
6.16	writer_buffer.h File Reference	100

6.16.1 Detailed Description	100
6.17 writer_file.h File Reference	100
6.17.1 Detailed Description	101

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Attributes	7
Allocation	8
Comparison	9
Accessors	10
Hierarchy	11
Elements	12
Allocation	13
Search and iteration	14
Comparison	16
Accessors	17
Hierarchy	19
Attributes	23
Errors	27
Codes and descriptions	27
Expat errors	29
Lists	30
Allocation	32
Accessors	32
Modifiers	33
Traverse	35
Search	37
Parser	39
Allocation	39
Load	41
Accessors	44
Input/Output	45
Printer	46
Allocation	46
Properties	47
Output	48
Readers	51

Memory	55
Files	56
Writers	71
Memory	74
Files	75
Text utilities	57
Trees	61
Allocation	62
Comparison	63
Properties	65
Contents	68

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

[scew_reader_hooks](#) (This is the set of functions that are implemented by all SCEW reader sources) [77](#)
[scew_writer_hooks](#) (This is the set of functions that are implemented by all SCEW writers) . . . [78](#)

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

attribute.h (SCEW attribute's handling routines)	81
bool.h (SCEW boolean type declaration)	82
element.h (SCEW element's handling routines)	83
error.h (SCEW error handling functions)	86
export.h (SCEW shared library support)	87
list.h (SCEW general list implementation)	87
parser.h (SCEW parser handling routines)	89
printer.h (SCEW printer routines for XML output)	90
reader.h (SCEW reader common functions)	91
reader_buffer.h (SCEW reader functions for memory buffers)	92
reader_file.h (SCEW reader functions for files)	93
scew.h (SCEW main header file)	94
str.h (SCEW string functions)	94
tree.h (SCEW tree handling routines)	97
writer.h (SCEW writer common functions)	99
writer_buffer.h (SCEW writer functions for memory buffers)	100
writer_file.h (SCEW writer functions for files)	100

Chapter 4

Module Documentation

4.1 Attributes

SCEW provides functions to access and manipulate the attributes of an element.

Modules

- [Allocation](#)
Allocate and free attributes.
- [Comparison](#)
Attribute comparison routines.
- [Accessors](#)
Access attributes' data, such as name and value.
- [Hierarchy](#)
Handle attribute's hierarchy.

Files

- file [attribute.h](#)
SCEW attribute's handling routines.

Typedefs

- typedef struct [scew_attribute](#) [scew_attribute](#)
This is the type declaration for SCEW attributes.

4.1.1 Detailed Description

SCEW provides functions to access and manipulate the attributes of an element. XML element attributes are basically a name-value pair.

4.2 Allocation

Allocate and free attributes.

Files

- file [attribute.h](#)
SCEW attribute's handling routines.

Functions

- SCEW_API [scew_attribute](#) * [scew_attribute_create](#) (XML_Char const *name, XML_Char const *value)
Creates a new attribute with the given pair (name, value).
- SCEW_API [scew_attribute](#) * [scew_attribute_copy](#) ([scew_attribute](#) const *attribute)
Makes a copy of the given attribute.
- SCEW_API void [scew_attribute_free](#) ([scew_attribute](#) *attribute)
Frees the given attribute.

4.2.1 Detailed Description

Allocate and free attributes.

4.2.2 Function Documentation

4.2.2.1 SCEW_API [scew_attribute](#)* [scew_attribute_create](#) (XML_Char const * *name*, XML_Char const * *value*)

Creates a new attribute with the given pair (*name*, *value*).

Precondition

name != NULL
value != NULL

Returns

the created attribute, or NULL if an error is found.

4.2.2.2 SCEW_API scew_attribute* scew_attribute_copy (scew_attribute const * *attribute*)

Makes a copy of the given *attribute*.

Note that the new copy does not belong to any element.

Precondition

`attribute != NULL`

Returns

a new attribute, or NULL if the copy failed.

4.2.2.3 SCEW_API void scew_attribute_free (scew_attribute * *attribute*)

Frees the given *attribute*.

That is, its name and value. You should not call this function with an attribute obtained from an element, use [scew_element_delete_attribute](#) instead. If a NULL *attribute* is given, this function does not have any effect.

4.3 Comparison

Attribute comparison routines.

Files

- file [attribute.h](#)
SCEW attribute's handling routines.

Functions

- SCEW_API [scew_bool scew_attribute_compare](#) ([scew_attribute](#) const *a, [scew_attribute](#) const *b)
Performs a comparison between the two given attributes.

4.3.1 Detailed Description

Attribute comparison routines.

4.3.2 Function Documentation

4.3.2.1 SCEW_API scew_bool scew_attribute_compare (scew_attribute const * *a*, scew_attribute const * *b*)

Performs a comparison between the two given attributes.

That is, name and value must be equal in both attributes. Attribute's elements are not compared.

Remember that XML is case-sensitive.

Precondition

a != NULL
b != NULL

Returns

true if attributes are equal, false otherwise.

4.4 Accessors

Access attributes' data, such as name and value.

Files

- file [attribute.h](#)
SCEW attribute's handling routines.

Functions

- SCEW_API XML_Char const * [scew_attribute_name](#) ([scew_attribute](#) const *attribute)
Returns the given attribute's name.
- SCEW_API XML_Char const * [scew_attribute_value](#) ([scew_attribute](#) const *attribute)
Returns the given attribute's value.
- SCEW_API XML_Char const * [scew_attribute_set_name](#) ([scew_attribute](#) *attribute, XML_Char const *name)
Sets a new name to the given attribute and frees the old one.
- SCEW_API XML_Char const * [scew_attribute_set_value](#) ([scew_attribute](#) *attribute, XML_Char const *value)
Sets a new value to the given attribute and frees the old one.

4.4.1 Detailed Description

Access attributes' data, such as name and value.

4.4.2 Function Documentation

4.4.2.1 SCEW_API XML_Char const* scew_attribute_name (scew_attribute const * attribute)

Returns the given *attribute*'s name.

Precondition

attribute != NULL

4.4.2.2 SCEW_API XML_Char const* scew_attribute_value (scew_attribute const * *attribute*)

Returns the given *attribute*'s value.

Precondition

attribute != NULL

4.4.2.3 SCEW_API XML_Char const* scew_attribute_set_name (scew_attribute * *attribute*, XML_Char const * *name*)

Sets a new *name* to the given *attribute* and frees the old one.

If an error is found, the old name is not freed.

Precondition

attribute != NULL
name != NULL

Returns

the new *attribute*'s name, or NULL if the new name can not be set.

4.4.2.4 SCEW_API XML_Char const* scew_attribute_set_value (scew_attribute * *attribute*, XML_Char const * *value*)

Sets a new *value* to the given *attribute* and frees the old one.

If an error is found, the old value is not freed.

Precondition

attribute != NULL
value != NULL

Returns

the new *attribute*'s value, or NULL if the new value could not be set.

4.5 Hierarchy

Handle attribute's hierarchy.

Files

- file [attribute.h](#)

SCEW attribute's handling routines.

Functions

- SCEW_API [scew_element](#) * [scew_attribute_parent](#) ([scew_attribute](#) const *attribute)
Returns the element that the given attribute belongs to.

4.5.1 Detailed Description

Handle attribute's hierarchy.

4.5.2 Function Documentation

4.5.2.1 SCEW_API [scew_element](#)* [scew_attribute_parent](#) ([scew_attribute](#) const * *attribute*)

Returns the element that the given *attribute* belongs to.

Precondition

`attribute != NULL`

Returns

the given *attribute*'s element, or NULL if the *attribute* is an standalone attribute.

4.6 Elements

Element related functions.

Modules

- [Allocation](#)
Allocate and free elements.
- [Search and iteration](#)
Iterate and search for elements.
- [Comparison](#)
Element comparison routines.
- [Accessors](#)
Access elements' data, such as name and contents.
- [Hierarchy](#)
Handle element's hierarchy.
- [Attributes](#)
Handle element's attributes.

Files

- file [element.h](#)
SCEW element's handling routines.

Typedefs

- typedef struct [scew_element](#) [scew_element](#)
This is the type declaration for SCEW elements.

4.6.1 Detailed Description

Element related functions. SCEW provides functions to access and manipulate the elements of an XML tree.

4.7 Allocation

Allocate and free elements.

Files

- file [element.h](#)
SCEW element's handling routines.

Functions

- SCEW_API [scew_element](#) * [scew_element_create](#) (XML_Char const *name)
Creates a new element with the given name.
- SCEW_API [scew_element](#) * [scew_element_copy](#) ([scew_element](#) const *element)
Makes a deep copy of the given element.
- SCEW_API void [scew_element_free](#) ([scew_element](#) *element)
Frees the given element recursively.

4.7.1 Detailed Description

Allocate and free elements.

4.7.2 Function Documentation

4.7.2.1 SCEW_API `scew_element*` `scew_element_create` (`XML_Char` const * *name*)

Creates a new element with the given *name*.

This element is not yet related to any XML tree.

Precondition

`name != NULL`

Returns

the created element, or NULL if an error is found.

4.7.2.2 SCEW_API `scew_element*` `scew_element_copy` (`scew_element` const * *element*)

Makes a deep copy of the given *element*.

Attributes and children elements will be copied. The new element will not belong to any XML tree.

Precondition

`element != NULL`

Returns

a new element, or NULL if the copy failed.

4.7.2.3 SCEW_API void `scew_element_free` (`scew_element` * *element*)

Frees the given *element* recursively.

That is, it frees all its children and attributes. If the *element* has a parent, it is also detached from it. If a NULL *element* is given, this function does not have any effect.

4.8 Search and iteration

Iterate and search for elements.

Files

- file [element.h](#)
SCEW element's handling routines.

Functions

- SCEW_API `scew_element` * `scew_element_by_name` (`scew_element` const **element*, `XML_Char` const **name*)

Returns the first child from the specified element that matches the given name.

- SCEW_API `scew_element` * `scew_element_by_index` (`scew_element` const *`element`, unsigned int `index`)

Returns the child of the given element at the specified zero-based index.

- SCEW_API `scew_list` * `scew_element_list_by_name` (`scew_element` const *`element`, XML_Char const *`name`)

Returns a list of children from the specified element that matches the given name.

4.8.1 Detailed Description

Iterate and search for elements.

4.8.2 Function Documentation

4.8.2.1 SCEW_API `scew_element`* `scew_element_by_name` (`scew_element` const * `element`, XML_Char const * `name`)

Returns the first child from the specified `element` that matches the given `name`.

Remember that XML names are case-sensitive.

Precondition

`element` != NULL
`name` != NULL

Returns

the first child that matches the given `name`, or NULL if not found.

4.8.2.2 SCEW_API `scew_element`* `scew_element_by_index` (`scew_element` const * `element`, unsigned int `index`)

Returns the child of the given `element` at the specified zero-based `index`.

Precondition

`element` != NULL
`index` < `scew_element_count`

Returns

the child at the specified position, or NULL if there are no children elements.

4.8.2.3 SCEW_API `scew_list`* `scew_element_list_by_name` (`scew_element` const * `element`, XML_Char const * `name`)

Returns a list of children from the specified `element` that matches the given `name`.

This list must be freed after using it via `scew_list_free` (the elements will not be freed, only the list pointing to them).

Precondition

element != NULL
name != NULL

Returns

a list of elements that matches the *name* specified, or NULL if no element is found.

4.9 Comparison

Element comparison routines.

Typedefs

- typedef `scew_bool(* scew_element_cmp_hook)(scew_element const *, scew_element const *)`
SCEW element compare hooks might be used to define new user XML element comparisons.

Functions

- SCEW_API `scew_bool scew_element_compare (scew_element const *a, scew_element const *b, scew_element_cmp_hook hook)`
Performs a deep comparison of the two given elements.

4.9.1 Detailed Description

Element comparison routines.

4.9.2 Typedef Documentation

4.9.2.1 typedef `scew_bool(* scew_element_cmp_hook)(scew_element const *, scew_element const *)`

SCEW element compare hooks might be used to define new user XML element comparisons.

This hook must only compare the element's name and contents and the list of attributes. The new hook is to be used by `scew_element_compare`.

Returns

true if the given elements are considered equal, false otherwise.

4.9.3 Function Documentation

4.9.3.1 SCEW_API `scew_bool scew_element_compare (scew_element const * a, scew_element const * b, scew_element_cmp_hook hook)`

Performs a deep comparison of the two given elements.

The comparison is done via the comparison *hook*. If *hook* is NULL, the default comparison is done:

- Name and contents are equal (case-sensitive).
- Number of attributes match.
- Attribute names and values match (case-sensitive).

It is important to note that, for any given *hook* (or if NULL), the children are automatically traversed recursively using the given *hook*. Therefore, the *hook* must only provide comparisons for element's name and contents and the list of attributes.

There is no restriction on the provided comparison *hook* (if any), thus the user is responsible to define how the comparison is to be done.

Precondition

a != NULL
b != NULL

Parameters

- a* one of the elements to compare.
- b* one of the elements to compare.
- hook* the user defined comparison function. If NULL, the default comparison is used.

Returns

true if both elements are considered equal, false otherwise.

4.10 Accessors

Access elements' data, such as name and contents.

Files

- file [element.h](#)
SCEW element's handling routines.

Functions

- SCEW_API XML_Char const * [scew_element_name](#) ([scew_element](#) const *element)
Returns the given element's name.
- SCEW_API XML_Char const * [scew_element_contents](#) ([scew_element](#) const *element)
Returns the given element's contents.
- SCEW_API XML_Char const * [scew_element_set_name](#) ([scew_element](#) *element, XML_Char const *name)
Sets a new name to the given element and frees the old one.

- SCEW_API XML_Char const * `scew_element_set_contents` (`scew_element` *element, XML_Char const *contents)
Sets a new contents to the given element and frees the old one.
- SCEW_API void `scew_element_free_contents` (`scew_element` *element)
Frees the current contents of the given element.

4.10.1 Detailed Description

Access elements' data, such as name and contents.

4.10.2 Function Documentation

4.10.2.1 SCEW_API XML_Char const* `scew_element_name` (`scew_element` const * *element*)

Returns the given *element's* name.

Precondition

`element != NULL`

Returns

the *element's* name. It is not possible to get a NULL value, as element names are mandatory.

4.10.2.2 SCEW_API XML_Char const* `scew_element_contents` (`scew_element` const * *element*)

Returns the given *element's* contents.

That is, the text between the start and end element tags.

Precondition

`element != NULL`

Returns

the *element's* contents, or NULL if the element has no contents.

4.10.2.3 SCEW_API XML_Char const* `scew_element_set_name` (`scew_element` * *element*, XML_Char const * *name*)

Sets a new *name* to the given *element* and frees the old one.

If the new name can not be set, the old one is not freed.

Precondition

`element != NULL`
`name != NULL`

Returns

the new *element's* name, or NULL if the name can not be set.

4.10.2.4 SCEW_API XML_Char const* scew_element_set_contents (scew_element * element, XML_Char const * contents)

Sets a new *contents* to the given element and frees the old one.

If the new contents can not be set, the old one is not freed.

Precondition

element != NULL

Returns

the new *element's* contents, or NULL if the contents can not be set.

4.10.2.5 SCEW_API void scew_element_free_contents (scew_element * element)

Frees the current contents of the given *element*.

If the *element* has no contents, this functions does not have any effect.

Precondition

element != NULL

4.11 Hierarchy

Handle element's hierarchy.

Files

- file [element.h](#)
SCEW element's handling routines.

Functions

- SCEW_API unsigned int [scew_element_count](#) ([scew_element](#) const *element)
Returns the number of children of the specified element.
- SCEW_API [scew_element](#) * [scew_element_parent](#) ([scew_element](#) const *element)
Returns the parent of the given element.
- SCEW_API [scew_list](#) * [scew_element_children](#) ([scew_element](#) const *element)
Returns the list of all the element's children.

- SCEW_API `scew_element` * `scew_element_add` (`scew_element` *element, XML_Char const *name)
Creates and adds, as a child of element, a new element with the given name.
- SCEW_API `scew_element` * `scew_element_add_pair` (`scew_element` *element, XML_Char const *name, XML_Char const *contents)
Creates and adds, as a child of element, a new element with the given name and contents.
- SCEW_API `scew_element` * `scew_element_add_element` (`scew_element` *element, `scew_element` *child)
Adds a child to the given element.
- SCEW_API void `scew_element_delete_all` (`scew_element` *element)
Deletes all the children for the given element.
- SCEW_API void `scew_element_delete_all_by_name` (`scew_element` *element, XML_Char const *name)
Deletes all the children of the given element that matches name.
- SCEW_API void `scew_element_delete_by_name` (`scew_element` *element, XML_Char const *name)
Deletes the first child of the given element that matches name.
- SCEW_API void `scew_element_delete_by_index` (`scew_element` *element, unsigned int index)
Deletes the child of the given element at the specified zero-based index.
- SCEW_API void `scew_element_detach` (`scew_element` *element)
Detaches the given element from its parent, if any.

4.11.1 Detailed Description

Handle element's hierarchy.

4.11.2 Function Documentation

4.11.2.1 SCEW_API unsigned int scew_element_count (scew_element const * element)

Returns the number of children of the specified *element*.

An element can have zero or more children.

Precondition

`element != NULL`

Returns

the number of children, or 0 if the *element* has no children.

4.11.2.2 SCEW_API scew_element* scew_element_parent (scew_element const * *element*)

Returns the parent of the given *element*.

Precondition

element != NULL

Returns

the *element*'s parent, or NULL if the given *element* has no parent (e.g. root element).

4.11.2.3 SCEW_API scew_list* scew_element_children (scew_element const * *element*)

Returns the list of all the *element*'s children.

This is the internal list where *element*'s children are stored, so no modifications or deletions should be performed on this list.

Precondition

element != NULL

Returns

the list of the given *element*'s children, or NULL if the *element* has no children.

4.11.2.4 SCEW_API scew_element* scew_element_add (scew_element * *element*, XML_Char const * *name*)

Creates and adds, as a child of *element*, a new element with the given *name*.

Precondition

element != NULL

name != NULL

Returns

the new created element, or NULL if an error is found.

4.11.2.5 SCEW_API scew_element* scew_element_add_pair (scew_element * *element*, XML_Char const * *name*, XML_Char const * *contents*)

Creates and adds, as a child of *element*, a new element with the given *name* and *contents*.

Precondition

element != NULL

name != NULL

contents != NULL

Returns

the new created element, or NULL if an error is found.

4.11.2.6 SCEW_API `scew_element* scew_element_add_element (scew_element * element, scew_element * child)`

Adds a *child* to the given *element*.

Note that the element being added should be a clean element, that is, an element created with `scew_element_create` or an element detached from another tree after being detached (via `scew_element_detach`).

Precondition

```
element != NULL
child != NULL
scew_element_parent (child) == NULL
```

Returns

the element being added, or NULL if the element could not be added.

4.11.2.7 SCEW_API `void scew_element_delete_all (scew_element * element)`

Deletes all the children for the given *element*.

This function deletes all subchildren recursively. This will automatically free the elements.

Precondition

```
element != NULL
```

4.11.2.8 SCEW_API `void scew_element_delete_all_by_name (scew_element * element, XML_Char const * name)`

Deletes all the children of the given *element* that matches *name*.

This will automatically free the element.

Precondition

```
element != NULL
name != NULL
```

4.11.2.9 SCEW_API `void scew_element_delete_by_name (scew_element * element, XML_Char const * name)`

Deletes the first child of the given *element* that matches *name*.

This will automatically free the element.

Precondition

```
element != NULL
name != NULL
```

4.11.2.10 SCEW_API void `scew_element_delete_by_index` (`scew_element * element`, unsigned int `index`)

Deletes the child of the given *element* at the specified zero-based *index*.

This will automatically free the element.

Precondition

`element != NULL`
`index < scew_element_count`

4.11.2.11 SCEW_API void `scew_element_detach` (`scew_element * element`)

Detaches the given *element* from its parent, if any.

This function only detaches the element, but does not free it. If the *element* has no parent, this function does not have any effect.

Precondition

`element != NULL`

4.12 Attributes

Handle element's attributes.

Files

- file [element.h](#)
SCEW element's handling routines.

Functions

- SCEW_API unsigned int [scew_element_attribute_count](#) (`scew_element` const *`element`)
Returns the number of attributes of the given element.
- SCEW_API `scew_list * scew_element_attributes` (`scew_element` const *`element`)
Returns the list of all the element's attributes.
- SCEW_API `scew_attribute * scew_element_attribute_by_name` (`scew_element` const *`element`, XML_Char const *`name`)
Returns the first attribute from the specified element that matches the given name.
- SCEW_API `scew_attribute * scew_element_attribute_by_index` (`scew_element` const *`element`, unsigned int `index`)
Returns the attribute of the given element at the specified zero-based index.

- SCEW_API `scew_attribute * scew_element_add_attribute (scew_element *element, scew_attribute *attribute)`
Adds an existent attribute to the given element.
- SCEW_API `scew_attribute * scew_element_add_attribute_pair (scew_element *element, XML_Char const *name, XML_Char const *value)`
Creates and adds a new attribute to the given element.
- SCEW_API `void scew_element_delete_attribute_all (scew_element *element)`
Deletes all the attributes of the given element.
- SCEW_API `void scew_element_delete_attribute (scew_element *element, scew_attribute *attribute)`
Deletes the given attribute from the specified element.
- SCEW_API `void scew_element_delete_attribute_by_name (scew_element *element, XML_Char const *name)`
Deletes the first attribute of the given element that matches name.
- SCEW_API `void scew_element_delete_attribute_by_index (scew_element *element, unsigned int index)`
Deletes the attribute of the given element at the specified zero-based index.

4.12.1 Detailed Description

Handle element's attributes.

4.12.2 Function Documentation

4.12.2.1 SCEW_API unsigned int scew_element_attribute_count (scew_element const * element)

Returns the number of attributes of the given *element*.

An element can have zero or more attributes.

Precondition

`element != NULL`

4.12.2.2 SCEW_API scew_list* scew_element_attributes (scew_element const * element)

Returns the list of all the *element's* attributes.

This is the internal list where *element's* attributes are stored, so no modifications or deletions should be performed on this list.

Precondition

`element != NULL`

Returns

the list of the given *element*'s attributes.

4.12.2.3 SCEW_API scew_attribute* scew_element_attribute_by_name (scew_element const * element, XML_Char const * name)

Returns the first attribute from the specified *element* that matches the given *name*.

Remember that XML attributes are case-sensitive.

Precondition

element != NULL
name != NULL

Returns

the attribute with the given name, or NULL if not found.

4.12.2.4 SCEW_API scew_attribute* scew_element_attribute_by_index (scew_element const * element, unsigned int index)

Returns the attribute of the given *element* at the specified zero-based *index*.

Precondition

element != NULL
index < [scew_element_attribute_count](#)

Returns

the attribute at the specified position, or NULL if the *element* has not attributes.

4.12.2.5 SCEW_API scew_attribute* scew_element_add_attribute (scew_element * element, scew_attribute * attribute)

Adds an existent *attribute* to the given *element*.

It is important to note that the given *attribute* will be part of the element's attributes (ownership is lost), so it should not be later freed, and it should not be part of another attribute element list.

Also note that, if the *attribute* already existed, the old value will be overwritten and the given attribute will not become part of the element's attribute list (only the old value is updated).

Precondition

element != NULL
attribute != NULL

Returns

the new attribute added to the element, or NULL if the *attribute* could not be added or updated.

4.12.2.6 SCEW_API *scew_attribute** *scew_element_add_attribute_pair* (*scew_element* * *element*, XML_Char const * *name*, XML_Char const * *value*)

Creates and adds a new attribute to the given *element*.

An attribute is formed by a pair (name, value).

If the attribute already existed, the old value will be overwritten, thus the new attribute will not be created (only the old value is updated).

Precondition

element != NULL
name != NULL
value != NULL

Returns

the new attribute added to the element, or NULL if the attribute could not be added or updated.

4.12.2.7 SCEW_API void *scew_element_delete_attribute_all* (*scew_element* * *element*)

Deletes all the attributes of the given *element*.

This will also automatically free all the attributes.

Precondition

element != NULL

4.12.2.8 SCEW_API void *scew_element_delete_attribute* (*scew_element* * *element*, *scew_attribute* * *attribute*)

Deletes the given *attribute* from the specified *element*.

This will also automatically free the given *attribute*.

Precondition

element != NULL
attribute != NULL

4.12.2.9 SCEW_API void *scew_element_delete_attribute_by_name* (*scew_element* * *element*, XML_Char const * *name*)

Deletes the first attribute of the given *element* that matches *name*.

This will also automatically free the attribute.

Precondition

element != NULL
name != NULL

4.12.2.10 SCEW_API void `scew_element_delete_attribute_by_index` (`scew_element * element`, `unsigned int index`)

Deletes the attribute of the given *element* at the specified zero-based *index*.

This will also automatically free the attribute.

Precondition

`element != NULL`
`index < scew_element_attribute_count`

4.13 Errors

These are SCEW error functions which return error codes and strings.

Modules

- [Codes and descriptions](#)
SCEW internal error codes and associated descriptions.
- [Expat errors](#)
Routines to access Expat internal error information.

Files

- file [error.h](#)
SCEW error handling functions.

4.13.1 Detailed Description

These are SCEW error functions which return error codes and strings. Expat related errors can also be obtained.

4.14 Codes and descriptions

SCEW internal error codes and associated descriptions.

Files

- file [error.h](#)
SCEW error handling functions.

Enumerations

- enum `scew_error` {
`scew_error_none`, `scew_error_no_memory`, `scew_error_io`, `scew_error_hook`,
`scew_error_expat`, `scew_error_internal`, `scew_error_unknown` }

This is the type declaration of the SCEW error.

Functions

- SCEW_API `scew_error scew_error_code` (void)
Returns the SCEW internal error code.
- SCEW_API XML_Char const * `scew_error_string` (`scew_error` code)
Returns a string describing the given internal SCEW error code.

4.14.1 Detailed Description

SCEW internal error codes and associated descriptions.

4.14.2 Enumeration Type Documentation

4.14.2.1 enum `scew_error`

This is the type declaration of the SCEW error.

That is, an enumeration of all SCEW possible errors.

Enumerator:

`scew_error_none` No error has occurred.
`scew_error_no_memory` No more memory available.
`scew_error_io` General Input/Output error.
`scew_error_hook` Hook returned error.
`scew_error_expat` Expat parser error.
`scew_error_internal` Internal SCEW error.
`scew_error_unknown` end of list marker

4.14.3 Function Documentation

4.14.3.1 SCEW_API `scew_error scew_error_code` (void)

Returns the SCEW internal error code.

If the error code returned is `scew_error_expat` it means that an internal Expat error has occurred, so you will probably want to check Expat error using `scew_error_expat_code` and `scew_error_expat_string`.

Returns

the current internal SCEW error code, if any.

4.14.3.2 SCEW_API XML_Char const* scew_error_string (scew_error code)

Returns a string describing the given internal SCEW error *code*.

Returns

the associated string for the given error *code*.

4.15 Expat errors

Routines to access Expat internal error information.

Files

- file [error.h](#)
SCEW error handling functions.

Functions

- SCEW_API enum XML_Error [scew_error_expat_code](#) (scew_parser *parser)
Returns the Expat internal error code.
- SCEW_API XML_Char const * [scew_error_expat_string](#) (enum XML_Error code)
Returns a string describing the internal Expat error for the given error code.
- SCEW_API int [scew_error_expat_line](#) (scew_parser *parser)
Returns the current line at which the error was detected.
- SCEW_API int [scew_error_expat_column](#) (scew_parser *parser)
Returns the current column at which the error was detected.

4.15.1 Detailed Description

Routines to access Expat internal error information.

4.15.2 Function Documentation

4.15.2.1 SCEW_API enum XML_Error scew_error_expat_code (scew_parser * parser)

Returns the Expat internal error code.

Returns

the internal Expat error code.

4.15.2.2 SCEW_API XML_Char const* scew_error_expat_string (enum XML_Error code)

Returns a string describing the internal Expat error for the given error *code*.

Returns

the internal Expat error string for the given error *code*.

4.15.2.3 SCEW_API int scew_error_expat_line (scew_parser * parser)

Returns the current line at which the error was detected.

Returns

the line where Expat detected the error.

4.15.2.4 SCEW_API int scew_error_expat_column (scew_parser * parser)

Returns the current column at which the error was detected.

Returns

the column where Expat detected the error.

4.16 Lists

This is a generic list implementation currently used by element's children and attributes, though, as a generic list, it can be used with any other type of data.

Modules

- [Allocation](#)
Allocate and free new lists.
- [Accessors](#)
Access lists' data and information.
- [Modifiers](#)
Add and remove items from lists.
- [Traverse](#)
Traverse list items.
- [Search](#)
Search for list items.

Files

- file [list.h](#)
SCEW general list implementation.
- file [list.h](#)
SCEW general list implementation.

Typedefs

- typedef struct [scew_list](#) [scew_list](#)
This is the type declaration for SCEW lists.
- typedef void(* [scew_list_hook](#))([scew_list](#) *, void *)
SCEW lists hooks (functions) are used by [scew_list_foreach](#).
- typedef [scew_bool](#)(* [scew_cmp_hook](#))(void const *, void const *)
SCEW lists comparison hooks are used by [scew_list_find_custom](#).

4.16.1 Detailed Description

This is a generic list implementation currently used by element's children and attributes, though, as a generic list, it can be used with any other type of data.

4.16.2 Typedef Documentation

4.16.2.1 typedef void(* [scew_list_hook](#))([scew_list](#) *, void *)

SCEW lists hooks (functions) are used by [scew_list_foreach](#).

The hook will be used to perform some custom action, defined by the hook, to every list item. These functions take two arguments, the list item where the action should be performed and an additional argument for any data that could be of use to the action.

Parameters

- item* the list item currently being traversed.
- user_data* an optional user data pointer to be used by the hook (might be NULL).

4.16.2.2 typedef [scew_bool](#)(* [scew_cmp_hook](#))(void const *, void const *)

SCEW lists comparison hooks are used by [scew_list_find_custom](#).

The hook takes the two arguments to be compared (of the same type).

Returns

- true if the given arguments are considered (by the provided comparison hook) equal, false otherwise.

4.17 Allocation

Allocate and free new lists.

Files

- file [list.h](#)
SCEW general list implementation.

Functions

- SCEW_API [scew_list](#) * [scew_list_create](#) (void *data)
Creates a new list item with the given data.
- SCEW_API void [scew_list_free](#) ([scew_list](#) *list)
Frees all the items from the given list.

4.17.1 Detailed Description

Allocate and free new lists.

4.17.2 Function Documentation

4.17.2.1 SCEW_API [scew_list](#)* [scew_list_create](#) (void * *data*)

Creates a new list item with the given *data*.

Note that there is no difference between list items and lists, that is, a list item is a list itself.

Precondition

`data != NULL`

Returns

a new list, or NULL if the list could not be created.

4.17.2.2 SCEW_API void [scew_list_free](#) ([scew_list](#) * *list*)

Frees all the items from the given *list*.

The data pointers are not freed, thus they need to be freed separately.

4.18 Accessors

Access lists' data and information.

Files

- file [list.h](#)
SCEW general list implementation.

Functions

- SCEW_API void * [scew_list_data](#) (scew_list *list)
Returns the data pointer of the given list item.
- SCEW_API unsigned int [scew_list_size](#) (scew_list *list)
Returns the number of items in the given list.

4.18.1 Detailed Description

Access lists' data and information.

4.18.2 Function Documentation

4.18.2.1 SCEW_API void* scew_list_data (scew_list * list)

Returns the data pointer of the given *list* item.

Note that this routine does not know if the data pointed by the *list* item has been freed, so it might return a valid address without useful content.

Precondition

`list != NULL`

Returns

the data pointer for the given *list*.

4.18.2.2 SCEW_API unsigned int scew_list_size (scew_list * list)

Returns the number of items in the given *list*.

Returns

the number of items in the given *list* or 0 if *list* is NULL.

4.19 Modifiers

Add and remove items from lists.

Files

- file [list.h](#)
SCEW general list implementation.

Functions

- SCEW_API [scew_list](#) * [scew_list_append](#) ([scew_list](#) *list, void *data)
Creates a new list item with the given data and appends it to list.
- SCEW_API [scew_list](#) * [scew_list_prepend](#) ([scew_list](#) *list, void *data)
Creates a new list item with the given data and prepends it to list.
- SCEW_API [scew_list](#) * [scew_list_delete](#) ([scew_list](#) *list, void *data)
Deletes the first item pointing to data from the given list.
- SCEW_API [scew_list](#) * [scew_list_delete_item](#) ([scew_list](#) *list, [scew_list](#) *item)
Deletes the given list item from list.

4.19.1 Detailed Description

Add and remove items from lists.

4.19.2 Function Documentation

4.19.2.1 SCEW_API [scew_list](#)* [scew_list_append](#) ([scew_list](#) * *list*, void * *data*)

Creates a new list item with the given *data* and appends it to *list*.

If the given *list* is NULL this function acts like [scew_list_create](#).

Precondition

`data != NULL`

Returns

the item appended to *list* or NULL if an item could not be created.

4.19.2.2 SCEW_API [scew_list](#)* [scew_list_prepend](#) ([scew_list](#) * *list*, void * *data*)

Creates a new list item with the given *data* and prepends it to *list*.

If the given *list* is NULL this function acts like [scew_list_create](#).

Precondition

`data != NULL`

Returns

the item prepended to *list* or NULL if an item could not be created.

4.19.2.3 SCEW_API scew_list* scew_list_delete (scew_list * list, void * data)

Deletes the first item pointing to *data* from the given *list*.

This function will search from the given item list, not from the beginning.

Precondition

list != NULL
data != NULL

Returns

list if the item found was not the first list item, or the new first item otherwise.

4.19.2.4 SCEW_API scew_list* scew_list_delete_item (scew_list * list, scew_list * item)

Deletes the given list *item* from *list*.

Precondition

list != NULL

Returns

list if *item* was not the first list item, or the new first item otherwise.

4.20 Traverse

Traverse list items.

Files

- file [list.h](#)
SCEW general list implementation.

Functions

- SCEW_API [scew_list](#) * [scew_list_first](#) ([scew_list](#) *list)
Finds the first item of the given list.
- SCEW_API [scew_list](#) * [scew_list_last](#) ([scew_list](#) *list)
Finds the last item of the given list.
- SCEW_API [scew_list](#) * [scew_list_next](#) ([scew_list](#) *list)
Obtains the next item of the given list item.
- SCEW_API [scew_list](#) * [scew_list_previous](#) ([scew_list](#) *list)
Obtains the previous item of the given list item.

- SCEW_API void `scew_list_foreach` (`scew_list` *list, `scew_list_hook` hook, void *user_data)
Traverses all list items and executes the given hook for each item found.

4.20.1 Detailed Description

Traverse list items.

4.20.2 Function Documentation

4.20.2.1 SCEW_API `scew_list*` `scew_list_first` (`scew_list` * list)

Finds the first item of the given *list*.

This function traverses all the *list* backwards until it finds an item whose previous item is NULL.

Precondition

list != NULL

Returns

the first item of the given *list* or NULL if *list* is NULL.

4.20.2.2 SCEW_API `scew_list*` `scew_list_last` (`scew_list` * list)

Finds the last item of the given *list*.

This function traverses all the *list* forwards until it finds an item whose next item is NULL.

Precondition

list != NULL

Returns

the last item of the given *list* or NULL if *list* is NULL.

4.20.2.3 SCEW_API `scew_list*` `scew_list_next` (`scew_list` * list)

Obtains the next item of the given *list* item.

Precondition

list != NULL

Returns

the next *list* item or NULL if *list* is the last item.

4.20.2.4 SCEW_API scew_list* scew_list_previous (scew_list * list)

Obtains the previous item of the given *list* item.

Precondition

list != NULL

Returns

the previous *list* item or NULL if *list* is the first item.

4.20.2.5 SCEW_API void scew_list_foreach (scew_list * list, scew_list_hook hook, void * user_data)

Traverses all *list* items and executes the given *hook* for each item found.

The hook takes an extra paramter, *user_data*, which might be NULL.

Precondition

list != NULL

func != NULL

Parameters

list the list to traverse.

hook the action to be executed for every traversed item.

user_data an optional user data pointer (might be NULL).

4.21 Search

Search for list items.

Files

- file [list.h](#)
SCEW general list implementation.

Functions

- SCEW_API [scew_list * scew_list_index](#) ([scew_list *list](#), unsigned int index)
Gets the list item at the given index.
- SCEW_API [scew_list * scew_list_find](#) ([scew_list *list](#), void *data)
Finds the first list item that contains data.
- SCEW_API [scew_list * scew_list_find_custom](#) ([scew_list *list](#), void const *data, [scew_cmp_hook](#) hook)
Finds the first list item that matches the given predicate, hook.

4.21.1 Detailed Description

Search for list items.

4.21.2 Function Documentation

4.21.2.1 SCEW_API scew_list* scew_list_index (scew_list * *list*, unsigned int *index*)

Gets the *list* item at the given *index*.

Precondition

list != NULL

Returns

the *list* item at *index*, or NULL if *list* does not contain sufficient items.

4.21.2.2 SCEW_API scew_list* scew_list_find (scew_list * *list*, void * *data*)

Finds the first *list* item that contains *data*.

Precondition

list != NULL
data != NULL

Returns

the first *list* item that contains *data*, or NULL if *data* is not found.

4.21.2.3 SCEW_API scew_list* scew_list_find_custom (scew_list * *list*, void const * *data*, scew_cmp_hook *hook*)

Finds the first *list* item that matches the given predicate, *hook*.

That is, all the *list* will be traversed calling the comparison hook for every *list* item. The comparison hook takes two parameters, the first one is the data of current traversed item, the second is *data*.

Precondition

list != NULL
data != NULL
func != NULL

Parameters

list the list to traverse.

data the user data to be used as one of the arguments for the comparison.

hook the comparison function.

Returns

the first *list* item that matches the predicate *func*, or NULL if the predicate is never true.

4.22 Parser

These are the parser functions that allow reading XML documents from a given SCEW writer (file, memory...).

Modules

- [Allocation](#)

Allocate and free a parser.

- [Load](#)

Load XML documents from different sources.

- [Accessors](#)

Obtain information from parser.

Files

- file [parser.h](#)

SCEW parser handling routines.

Typedefs

- typedef struct [scew_parser](#) [scew_parser](#)

This is the type declaration of the SCEW parser.

4.22.1 Detailed Description

These are the parser functions that allow reading XML documents from a given SCEW writer (file, memory...).

4.23 Allocation

Allocate and free a parser.

Files

- file [parser.h](#)

SCEW parser handling routines.

Functions

- SCEW_API [scew_parser](#) * [scew_parser_create](#) (void)
Creates a new parser.
- SCEW_API [scew_parser](#) * [scew_parser_namespace_create](#) (XML_Char separator)
Creates a new parser with namespaces support.
- SCEW_API void [scew_parser_free](#) ([scew_parser](#) *parser)
Frees a parser memory structure.

4.23.1 Detailed Description

Allocate and free a parser.

4.23.2 Function Documentation

4.23.2.1 SCEW_API scew_parser* scew_parser_create (void)

Creates a new parser.

A parser is necessary to load XML documents. Note that a parser might be re-used to load multiple XML documents, thus it is not necessary to create a parser for each XML document, but to call [scew_parser_load](#).

Returns

a new parser, or NULL if parser is not successfully created.

4.23.2.2 SCEW_API scew_parser* scew_parser_namespace_create (XML_Char separator)

Creates a new parser with namespaces support.

Note that Expat expands the resulting elements and attributes, that is, they are formed by the namespace URI, the given namespace *separator* and the local part of the name.

Parameters

separator the character between namespace URI and identifier. If 0 is given, no separation is performed.

Returns

a new parser with namespace support, or NULL if parser is not successfully created.

4.23.2.3 SCEW_API void scew_parser_free (scew_parser * parser)

Frees a *parser* memory structure.

If a NULL *parser* is given, this function takes no action.

4.24 Load

Load XML documents from different sources.

Files

- file [parser.h](#)
SCEW parser handling routines.

Typedefs

- typedef [scew_bool](#)(* [scew_parser_load_hook](#))([scew_parser](#) *, void *, void *)
SCEW parser hooks might be used as notifications to know when XML elements or trees are parsed.

Functions

- SCEW_API [scew_tree](#) * [scew_parser_load](#) ([scew_parser](#) *parser, [scew_reader](#) *reader)
Loads an XML tree from the specified reader.
- SCEW_API [scew_bool](#) [scew_parser_load_stream](#) ([scew_parser](#) *parser, [scew_reader](#) *reader)
Loads multiple XML trees from the specified stream reader.
- SCEW_API void [scew_parser_reset](#) ([scew_parser](#) *parser)
Resets the given parser for further uses.
- SCEW_API void [scew_parser_set_element_hook](#) ([scew_parser](#) *parser, [scew_parser_load_hook](#) hook, void *user_data)
Registers a hook to be called once an XML element is successfully parsed.
- SCEW_API void [scew_parser_set_tree_hook](#) ([scew_parser](#) *parser, [scew_parser_load_hook](#) hook, void *user_data)
Registers a hook to be called once an XML tree is successfully parsed.
- SCEW_API void [scew_parser_ignore_whitespaces](#) ([scew_parser](#) *parser, [scew_bool](#) ignore)
Tells the parser how to treat white spaces.

4.24.1 Detailed Description

Load XML documents from different sources.

4.24.2 Typedef Documentation

4.24.2.1 typedef [scew_bool](#)(* [scew_parser_load_hook](#))([scew_parser](#) *, void *, void *)

SCEW parser hooks might be used as notifications to know when XML elements or trees are parsed.

Two types of hooks might be registered, one for elements ([scew_parser_set_element_hook](#)) and one for trees ([scew_parser_set_tree_hook](#)). Whenever the parser loads a complete element (when the end of tag is found) the user will be notified via the registered hook, and the same for XML trees.

Parameters

parser the parser that is loading the XML contents.

data this is the pointer to an SCEW element or tree.

user_data an optional user data pointer to be used by the hook (might be NULL).

Returns

true if the hook call had no errors, false otherwise.

4.24.3 Function Documentation

4.24.3.1 SCEW_API scew_tree* scew_parser_load (scew_parser * *parser*, scew_reader * *reader*)

Loads an XML tree from the specified *reader*.

This will get data from the reader and it will try to parse it. The reader might be of any type. Once the parser loads elements or the complete XML tree, the appropriate registered hooks will be called.

Note that this function can only load one XML tree. Concatenated XML documents might be loaded via [scew_parser_load_stream](#).

XML declarations are not mandatory, and if none is found, the SCEW tree will still be created with a default one.

At startup, the *parser* is reset (via [scew_parser_reset](#)).

Precondition

parser != NULL

reader != NULL

Parameters

parser the SCEW *parser* that parses the *reader* contents.

reader the reader from where to load the XML.

Returns

the XML parsed tree or NULL if an error was found.

4.24.3.2 SCEW_API scew_bool scew_parser_load_stream (scew_parser * *parser*, scew_reader * *reader*)

Loads multiple XML trees from the specified stream *reader*.

This will get data from the reader and it will try to parse it. The difference between [scew_parser_load](#) and this function is that, here, at some point the reader might not have any more data to be read, so the function will return. Once more data becomes available subsequent calls to this function are needed to continue parsing.

Another important difference is that concatenated XML documents are allowed. Once the parser loads elements or complete XML trees, the appropriate registered hooks will be called.

It is necessary to register an XML tree hook, otherwise it will not be possible to get a reference to parsed XML trees, causing a memory leak.

Precondition

parser != NULL
reader != NULL
tree hook registered ([scew_parser_set_tree_hook](#))

Parameters

parser the SCEW *parser* that parses the *reader* contents.
reader the stream *reader* from where to load XML information.

Returns

true if the parsing is being successful, false if an error is found.

4.24.3.3 SCEW_API void scew_parser_reset (scew_parser * parser)

Resets the given *parser* for further uses.

Resetting a parser allows the parser to be re-used. This function is automatically called in [scew_parser_load](#), but needs to be called when loading streams, as [scew_parser_load_stream](#) does not reset the parser.

Precondition

parser != NULL

Parameters

parser the parser to reset.

4.24.3.4 SCEW_API void scew_parser_set_element_hook (scew_parser * parser, scew_parser_load_hook hook, void * user_data)

Registers a *hook* to be called once an XML element is successfully parsed.

The hook will only be called once the complete element is parsed, that is, when the end tag is found.

This hook might be useful as a notification mechanism when parsing big XML documents.

Note that no modification or deletion should be performed on the elements as they might still be needed by the parser.

Precondition

parser != NULL
hook != NULL

Parameters

parser the parser that is loading the XML contents.
hook this is the hook to be called once an XML element is parsed.
user_data an optional user data pointer to be used by the hook (might be NULL).

4.24.3.5 SCEW_API void `scew_parser_set_tree_hook` (`scew_parser * parser`, `scew_parser_load_hook hook`, `void * user_data`)

Registers a *hook* to be called once an XML tree is successfully parsed.

The hook will only be called once the complete XML tree is parsed.

This hook is necessary when loading streams (via `scew_parser_load_stream`), as no XML tree is returned there.

Precondition

`parser != NULL`
`hook != NULL`

Parameters

parser the parser that is loading the XML contents.

hook this is the hook to be called once an XML tree is parsed.

user_data an optional user data pointer to be used by the hook (might be NULL).

4.24.3.6 SCEW_API void `scew_parser_ignore_whitespaces` (`scew_parser * parser`, `scew_bool ignore`)

Tells the *parser* how to treat white spaces.

The default is to ignore heading and trailing white spaces.

There is a new section in XML specification which talks about how to handle white spaces in XML. One can set an optional attribute to an element which is called *xml:space*, and it can be set to *default* or *preserve*, and it inherits its value from parent elements.

- **preserve:** leave white spaces as their are found.
- **default:** white spaces are handled by the XML processor (Expat in our case) the way it wants to.

This function gives the possibility to change the XML processor behaviour.

Parameters

parser the parser to set the option to.

ignore whether the *parser* should ignore white spaces, false otherwise.

4.25 Accessors

Obtain information from parser.

Files

- file [parser.h](#)
SCEW parser handling routines.

Functions

- SCEW_API XML_Parser [scew_parser_expat](#) ([scew_parser](#) *parser)
Returns the internal Expat parser being used by the given SCEW parser.

4.25.1 Detailed Description

Obtain information from parser.

4.25.2 Function Documentation

4.25.2.1 SCEW_API XML_Parser scew_parser_expat (scew_parser * parser)

Returns the internal Expat parser being used by the given SCEW *parser*.

Probably some extra low-level Expat functions need to be called by the user. This function gives access to the Expat parser so it is possible to call these functions.

Note that if the Expat parser event handling routines are modified, SCEW will not be able to load XML documents.

4.26 Input/Output

The SCEW I/O system is based on SCEW [Readers](#), [Writers](#) and [Printer](#).

Modules

- [Printer](#)
A SCEW printer provides a set of routines to send XML data to a given SCEW writer.
- [Readers](#)
Read data from different sources: files, memory, etc.
- [Writers](#)
Write data to different destinations: files, memory, etc.

4.26.1 Detailed Description

The SCEW I/O system is based on SCEW [Readers](#), [Writers](#) and [Printer](#). A reader is a common mechanism to load data from different sources (files, memory, ...). A common mechanism means that the functions to read data, for example, from a file or from a memory buffer, are the same. SCEW writers follows the same idea behind the readers, that is, common routines are used to write data to any kind of sources (files, memory, ...).

It is worth mentioning that a user might implement its own SCEW readers and writers.

The SCEW printer provides routines to write SCEW XML data (trees, elements and attributes) to a SCEW writer.

4.27 Printer

A SCEW printer provides a set of routines to send XML data to a given SCEW writer.

Modules

- [Allocation](#)
Allocate and free printers.
- [Properties](#)
Set printer properties.
- [Output](#)
A set of routines to print XML data.

Files

- file [printer.h](#)
SCEW printer routines for XML output.

Typedefs

- typedef struct [scew_printer](#) [scew_printer](#)
This is the type declaration for the SCEW printer.

4.27.1 Detailed Description

A SCEW printer provides a set of routines to send XML data to a given SCEW writer.

4.28 Allocation

Allocate and free printers.

Files

- file [printer.h](#)
SCEW printer routines for XML output.

Functions

- SCEW_API [scew_printer](#) * [scew_printer_create](#) ([scew_writer](#) *writer)
Creates a new SCEW printer that will use the given writer by default.
- SCEW_API void [scew_printer_free](#) ([scew_printer](#) *printer)
Frees the given SCEW printer.

4.28.1 Detailed Description

Allocate and free printers.

4.28.2 Function Documentation

4.28.2.1 SCEW_API [scew_printer](#)* [scew_printer_create](#) ([scew_writer](#) * *writer*)

Creates a new SCEW printer that will use the given *writer* by default.

The SCEW writer will be used by the [Output](#) calls. It is possible to re-use a SCEW printer by setting a new writer via [scew_printer_set_writer](#).

Precondition

writer != NULL

Parameters

writer the SCEW writer to be used by the putput functions.

Returns

a new SCEW printer or NULL if the printer could not be created.

4.28.2.2 SCEW_API void [scew_printer_free](#) ([scew_printer](#) * *printer*)

Frees the given SCEW *printer*.

This will not free the writer being used by the printer.

Parameters

printer the SCEW printer to free.

4.29 Properties

Set printer properties.

Files

- file [printer.h](#)
SCEW printer routines for XML output.

Functions

- SCEW_API void `scew_printer_set_indented` (`scew_printer *printer`, `scew_bool indented`)
Tells whether the output sent to the given SCEW printer should be indented or not.
- SCEW_API void `scew_printer_set_indentation` (`scew_printer *printer`, unsigned int spaces)
Sets the number of spaces to use when indenting output for the given SCEW printer.

4.29.1 Detailed Description

Set printer properties.

4.29.2 Function Documentation

4.29.2.1 SCEW_API void `scew_printer_set_indented` (`scew_printer * printer`, `scew_bool indented`)

Tells whether the output sent to the given SCEW *printer* should be *indented* or not.

Precondition

`printer != NULL`

Parameters

printer the SCEW printer to change its indentation for.

indented true if the output should be indented, false otherwise.

4.29.2.2 SCEW_API void `scew_printer_set_indentation` (`scew_printer * printer`, unsigned int *spaces*)

Sets the number of *spaces* to use when indenting output for the given SCEW *printer*.

Precondition

`printer != NULL`

Parameters

printer the SCEW printer to change its indentation spaces for.

spaces the number of spaces to use for indentation.

4.30 Output

A set of routines to print XML data.

Files

- file [printer.h](#)
SCEW printer routines for XML output.

Functions

- SCEW_API [scew_writer](#) * [scew_printer_set_writer](#) ([scew_printer](#) *printer, [scew_writer](#) *writer)
Sets the given SCEW writer to the specified printer.
- SCEW_API [scew_bool](#) [scew_printer_print_tree](#) ([scew_printer](#) *printer, [scew_tree](#) const *tree)
Prints the given SCEW tree to the specified printer.
- SCEW_API [scew_bool](#) [scew_printer_print_element](#) ([scew_printer](#) *printer, [scew_element](#) const *element)
Prints the given SCEW element to the specified printer.
- SCEW_API [scew_bool](#) [scew_printer_print_element_children](#) ([scew_printer](#) *printer, [scew_element](#) const *element)
Prints the given SCEW element children to the specified printer.
- SCEW_API [scew_bool](#) [scew_printer_print_element_attributes](#) ([scew_printer](#) *printer, [scew_element](#) const *element)
Prints the given SCEW element attributes to the specified printer.
- SCEW_API [scew_bool](#) [scew_printer_print_attribute](#) ([scew_printer](#) *printer, [scew_attribute](#) const *attribute)
Prints the given SCEW attribute to the specified printer.

4.30.1 Detailed Description

A set of routines to print XML data.

4.30.2 Function Documentation

4.30.2.1 SCEW_API [scew_writer](#)* [scew_printer_set_writer](#) ([scew_printer](#) * *printer*, [scew_writer](#) * *writer*)

Sets the given SCEW *writer* to the specified *printer*.

After this call, subsequent calls to output functions will use the given writer internally. This means that the printer can be used to writer to a file or memory buffer indistinctly.

Parameters

- printer* the SCEW printer to change its writer for.
- writer* the SCEW writer to be used in next output calls.

Returns

the old SCEW writer.

4.30.2.2 SCEW_API scew_bool scew_printer_print_tree (scew_printer * *printer*, scew_tree const * *tree*)

Prints the given SCEW *tree* to the specified *printer*.

This will print the XML declaration, the preamble and the root element with all its children.

Precondition

printer != NULL
tree != NULL

Parameters

printer the printer to be used for printing data.
tree the SCEW tree to print.

4.30.2.3 SCEW_API scew_bool scew_printer_print_element (scew_printer * *printer*, scew_element const * *element*)

Prints the given SCEW *element* to the specified *printer*.

This will print the element (with its attributes) and all its children recursively.

Precondition

printer != NULL
element != NULL

Parameters

printer the printer to be used for printing data.
element the SCEW element to print.

4.30.2.4 SCEW_API scew_bool scew_printer_print_element_children (scew_printer * *printer*, scew_element const * *element*)

Prints the given SCEW *element* children to the specified *printer*.

This will print the element children recursively.

Precondition

printer != NULL
element != NULL

Parameters

printer the printer to be used for printing data.
element the SCEW element to print its children for. The element itself is not printed.

4.30.2.5 SCEW_API scew_bool scew_printer_print_element_attributes (scew_printer * *printer*, scew_element const * *element*)

Prints the given SCEW *element* attributes to the specified *printer*.

This will print the list of the element attributes. Note that this will not generate any valid XML data, but might be useful in some cases.

Precondition

printer != NULL
element != NULL

Parameters

printer the printer to be used for printing data.
element the SCEW element to print its attributes for.

4.30.2.6 SCEW_API scew_bool scew_printer_print_attribute (scew_printer * *printer*, scew_attribute const * *attribute*)

Prints the given SCEW *attribute* to the specified *printer*.

Note that this will not generate any valid XML data, but might be useful in some cases.

Precondition

printer != NULL
attribute != NULL

Parameters

printer the printer to be used for printing data.
attribute the SCEW attribute to print.

4.31 Readers

Read data from different sources: files, memory, etc.

Data Structures

- struct [scew_reader_hooks](#)

This is the set of functions that are implemented by all SCEW reader sources.

Modules

- [Memory](#)

Read data from memory buffers.

- [Files](#)

Read data from files.

Files

- file [reader.h](#)
SCEW reader common functions.

Typedefs

- typedef struct [scew_reader](#) [scew_reader](#)
This is the type delcaration for SCEW readers.

Functions

- SCEW_API [scew_reader](#) * [scew_reader_create](#) ([scew_reader_hooks](#) const *hooks, void *data)
Creates a new SCEW reader with the given [scew_reader_hooks](#) implementation.
- SCEW_API void * [scew_reader_data](#) ([scew_reader](#) *reader)
Returns the reference to the internal data structure being used by the given reader.
- SCEW_API size_t [scew_reader_read](#) ([scew_reader](#) *reader, XML_Char *buffer, size_t char_no)
Reads data from the given reader in store it in the specified buffer.
- SCEW_API [scew_bool](#) [scew_reader_end](#) ([scew_reader](#) *reader)
Tells whether the given reader has reached its end.
- SCEW_API [scew_bool](#) [scew_reader_error](#) ([scew_reader](#) *reader)
Tells whether an error was found while reading from the given reader.
- SCEW_API [scew_bool](#) [scew_reader_close](#) ([scew_reader](#) *reader)
Closes the given reader.
- SCEW_API void [scew_reader_free](#) ([scew_reader](#) *reader)
Frees the memory allocated by the given reader.

4.31.1 Detailed Description

Read data from different sources: files, memory, etc. SCEW readers provide a common mechanism to read data from different sources. This is done by implementing the set of functions declared in [scew_reader_hooks](#). A user might create new SCEW readers by implementing those functions.

Once a SCEW reader is created, functions in this section should be used no matter the reader type.

4.31.2 Function Documentation

4.31.2.1 SCEW_API [scew_reader](#)* [scew_reader_create](#) ([scew_reader_hooks](#) const * *hooks*, void * *data*)

Creates a new SCEW reader with the given [scew_reader_hooks](#) implementation.

This function should be called internally when implementing a new SCEW reader source. The *data* argument is a reference to some internal data used by the SCEW reader (file stream pointer, current memory buffer pointer, etc.). This data might be later obtained, by the SCEW reader implementation, via [scew_reader_data](#).

Precondition

hooks != NULL

Parameters

hooks the implementation of the new SCEW reader source.

data data to be used by the new SCEW reader. This is usually a reference to a file stream (in case of files) or a memory buffer pointer, etc.

Returns

a new SCEW reader, or NULL if the reader could not be created.

4.31.2.2 SCEW_API void* scew_reader_data (scew_reader * reader)

Returns the reference to the internal data structure being used by the given *reader*.

Precondition

reader != NULL

Parameters

reader the reader to obtain its internal data for.

Returns

a reference to the reader's internal data, or NULL if no data was set at creation time.

4.31.2.3 SCEW_API size_t scew_reader_read (scew_reader * reader, XML_Char * buffer, size_t char_no)

Reads data from the given *reader* in store it in the specified *buffer*.

This function will read as many characters (of size XML_Char) as specified by *char_no*. [scew_reader_error](#) and [scew_reader_end](#) need to be consulted to check whether an error is found or the end of the reader is reached, respectively.

This function will call the actual *read* function provided by the SCEW reader hooks ([scew_reader_hooks](#)).

Precondition

reader != NULL

buffer != NULL

Parameters

reader the reader from where to read data from.

buffer the memory buffer where to store data.

char_no the number of characters to read.

Returns

the number of characters successfully read.

4.31.2.4 SCEW_API scew_bool scew_reader_end (scew_reader * reader)

Tells whether the given *reader* has reached its end.

That is, no more data is available for reading.

This function will call the actual *end* function provided by the SCEW reader hooks ([scew_reader_hooks](#)).

Precondition

reader != NULL

Parameters

reader the reader to check its end status for.

Returns

true if we are at the end of the reader, false otherwise.

4.31.2.5 SCEW_API scew_bool scew_reader_error (scew_reader * reader)

Tells whether an error was found while reading from the given *reader*.

This function will call the actual *error* function provided by the SCEW reader hooks ([scew_reader_hooks](#)).

Precondition

reader != NULL

Parameters

reader the reader to check its status for.

Returns

true if we an error was found while reading data from the reader, false otherwise.

4.31.2.6 SCEW_API scew_bool scew_reader_close (scew_reader * reader)

Closes the given *reader*.

This function will have different effects depending on the SCEW reader type (e.g. it will close the file for file streams). After calling this function, none of the SCEW reader functions should be used, otherwise undefined behavior is expected.

This function will call the actual *close* function provided by the SCEW reader hooks ([scew_reader_hooks](#)).

Precondition

reader != NULL

Parameters

reader the reader to close.

Returns

true if the reader was successfully closed, false otherwise.

4.31.2.7 SCEW_API void scew_reader_free (scew_reader * reader)

Frees the memory allocated by the given *reader*.

This function will call the actual *free* function provided by the SCEW reader hooks ([scew_reader_hooks](#)).

Parameters

reader the reader to free.

4.32 Memory

Read data from memory buffers.

Files

- file [reader_buffer.h](#)
SCEW reader functions for memory buffers.

Functions

- SCEW_API [scew_reader](#) * [scew_reader_buffer_create](#) (XML_Char const *buffer, size_t size)
Creates a new SCEW reader for the given memory buffer of the specified size.

4.32.1 Detailed Description

Read data from memory buffers.

4.32.2 Function Documentation

4.32.2.1 SCEW_API scew_reader* scew_reader_buffer_create (XML_Char const * buffer, size_t size)

Creates a new SCEW reader for the given memory *buffer* of the specified *size*.

Once the writer is created, any of the [Readers](#) functions might be called in order to read data from the buffer.

Precondition

buffer != NULL
size > 0

Parameters

buffer the memory area where the new SCEW reader should read data from.
size the size of the memory area.

Returns

a new SCEW reader for the given buffer or NULL if the reader could not be created.

4.33 Files

Read data from files.

Files

- file [reader_file.h](#)
SCEW reader functions for files.

Functions

- SCEW_API [scew_reader](#) * [scew_reader_file_create](#) (char const *file_name)
Creates a new SCEW reader for the given file name.
- SCEW_API [scew_reader](#) * [scew_reader_fp_create](#) (FILE *file)
Creates a new SCEW reader for the given file stream.

4.33.1 Detailed Description

Read data from files.

4.33.2 Function Documentation

4.33.2.1 SCEW_API [scew_reader](#)* [scew_reader_file_create](#) (char const * *file_name*)

Creates a new SCEW reader for the given file name.

This routine will open the given file in text mode. Once the reader is created, the [Readers](#) routines must be called in order to read data from the file or to know the file status.

For UTF-16 encoding (only in Windows platforms) the BOM (Byte Order Mask) is automatically handled by the Windows API.

Precondition

file_name != NULL

Parameters

file_name the file name to open for the new SCEW reader.

Returns

a new SCEW reader for the given file name or NULL if the reader could not be created (e.g. memory allocation, the file does not exist, etc.).

4.33.2.2 SCEW_API scew_reader* scew_reader_fp_create (FILE * *file*)

Creates a new SCEW reader for the given *file* stream.

The file stream is opened in text mode. Once the reader is created, any of the [Readers](#) routines must be called in order to read data from the file or to know the file status.

For UTF-16 encoding (only in Windows platforms) the BOM (Byte Order Mask) is automatically handled by the Windows API.

Precondition

file != NULL

Parameters

file the file where the new SCEW reader should read data from.

Returns

a new SCEW reader for the given file stream or NULL if the reader could not be created (e.g. memory allocation, the file does not exist, etc.).

4.34 Text utilities

This module defines a set of functions to work with text strings.

Files

- file [str.h](#)
SCEW string functions.

Defines

- #define [scew_memcpy](#)(dst, src, n) memcpy (dst, src, sizeof (XML_Char) * (n))
Copy the number of given characters from src to dst.
- #define [scew_memmove](#)(dst, src, n) memmove (dst, src, sizeof (XML_Char) * (n))
Move the number of given characters from src to dst.
- #define [_XT](#)(str) str
Creates a regular string or a wide character string.

- #define [scew_printf](#) printf
See standard printf documentation.
- #define [scew_fprintf](#) fprintf
See standard fprintf documentation.
- #define [scew_vfprintf](#) vfprintf
See standard vfprintf documentation.
- #define [scew_fputs](#) fputs
See standard fputs documentation.
- #define [scew_fgets](#) fgets
See standard fgets documentation.
- #define [scew_fputc](#) fputc
See standard fputc documentation.
- #define [scew_fgetc](#) fgetc
See standard fgetc documentation.
- #define [scew_strspn](#)(s, accept) strspn (s, accept)
See standard strspn documentation.
- #define [scew_strcpy](#)(dest, src) strcpy (dest, src)
See standard strcpy documentation.
- #define [scew_strcat](#)(dest, src) strcat (dest, src)
See standard strcat documentation.
- #define [scew_strncpy](#)(dest, src, n) strncpy (dest, src, (n))
See standard strncpy documentation.
- #define [scew_strncat](#)(dest, src, n) strncat (dest, src, (n))
See standard strncat documentation.
- #define [scew_strlen](#)(s) strlen (s)
See standard strlen documentation.
- #define [scew_isalnum](#)(c) isalnum ((unsigned char)(c))
See standard isalnum documentation.
- #define [scew_isalpha](#)(c) isalpha ((unsigned char)(c))
See standard isalpha documentation.
- #define [scew_iscntrl](#)(c) iscntrl ((unsigned char)(c))
See standard iscntrl documentation.
- #define [scew_isdigit](#)(c) isdigit ((unsigned char)(c))

See standard isdigit documentation.

- #define [scew_isxdigit\(c\)](#) isxdigit ((unsigned char)(c))
See standard isxdigit documentation.
- #define [scew_isgraph\(c\)](#) isgraph ((unsigned char)(c))
See standard isgraph documentation.
- #define [scew_islower\(c\)](#) islower ((unsigned char)(c))
See standard islower documentation.
- #define [scew_isupper\(c\)](#) isupper ((unsigned char)(c))
See standard isupper documentation.
- #define [scew_isprint\(c\)](#) isprint ((unsigned char)(c))
See standard isprint documentation.
- #define [scew_ispunct\(c\)](#) ispunct ((unsigned char)(c))
See standard ispunct documentation.
- #define [scew_isspace\(c\)](#) isspace ((unsigned char)(c))
See standard isspace documentation.

Functions

- SCEW_API int [scew_strcmp](#) (XML_Char const *a, XML_Char const *b)
Compares the two given strings s1 and s2.
- SCEW_API XML_Char * [scew_strdup](#) (XML_Char const *src)
Creates a new copy of the given string.
- SCEW_API void [scew_strtrim](#) (XML_Char *src)
Trims off extra spaces from the beginning and end of a string.
- SCEW_API [scew_bool](#) [scew_isempty](#) (XML_Char const *src)
Tells whether the given string is empty.
- XML_Char * [scew_strescape](#) (XML_Char const *src)
Escapes the given string for XML.

4.34.1 Detailed Description

This module defines a set of functions to work with text strings. SCEW has defined wrappers for standard C routines in order to work with regular and wide character strings (wchar_t). The wrappers are simple macros to call the appropriate functions in both cases.

Right now, wide character strings are only available in Windows platforms to provide UTF-16 support (XML_UNICODE_WCHAR_T needs to be defined at compile time).

4.34.2 Define Documentation

4.34.2.1 `#define scew_memcpy(dst, src, n) memcpy (dst, src, sizeof (XML_Char) * (n))`

Copy the number of given characters from *src* to *dst*.

See standard *memcpy* documentation.

4.34.2.2 `#define scew_memmove(dst, src, n) memmove (dst, src, sizeof (XML_Char) * (n))`

Move the number of given characters from *src* to *dst*.

See standard *memmove* documentation.

4.34.3 Function Documentation

4.34.3.1 `SCEW_API int scew_strcmp (XML_Char const * a, XML_Char const * b)`

Compares the two given strings *s1* and *s2*.

Returns

0 if the two strings are identical or NULL, less than zero if *s1* is less than *s2* or greater than zero otherwise.

4.34.3.2 `SCEW_API XML_Char* scew_strdup (XML_Char const * src)`

Creates a new copy of the given string.

Parameters

src the string to be duplicated (might be NULL).

Returns

the duplicated string, or NULL if the given string is NULL.

4.34.3.3 `SCEW_API void scew_strtrim (XML_Char * src)`

Trims off extra spaces from the beginning and end of a string.

The trimming is done in place.

Precondition

src != NULL

Parameters

src the string to be trimmed off.

4.34.3.4 SCEW_API scew_bool scew_isempty (XML_Char const * src)

Tells whether the given string is empty.

That is, all characters are spaces, form-feed, newlines, etc. See *isspace* documentation to see the list of characters considered space.

Precondition

src != NULL

Parameters

src the string to tell if its empty or not.

Returns

true if the given string is empty, false otherwise.

4.34.3.5 XML_Char* scew_strescape (XML_Char const * src)

Escapes the given string for XML.

This will substitute the general XML delimiters:

< > & ' "

to the pre-defined XML entities, respectively:

< > & ' "

A new escaped string will be allocated. Thus, the user is responsible of freeing the new string.

Precondition

src != NULL

Parameters

src the string to be escaped.

Returns

a new allocated string with the XML delimiters (if any) escaped.

4.35 Trees

Tree related functions.

Modules

- [Allocation](#)

Allocate and free XML trees.

- [Comparison](#)
Tree comparison routines.
- [Properties](#)
Handle XML trees properties.
- [Contents](#)
Accessors for XML root elements and preambles.

Files

- file [tree.h](#)
SCEW tree handling routines.

Typedefs

- typedef struct [scew_tree](#) [scew_tree](#)
This is the type delcaration for XML trees.

4.35.1 Detailed Description

Tree related functions. SCEW provides functions to create new XML trees. Trees are SCEW internal XML document representation. A tree contains basic information, such as XML version and encoding, and contains a root element which is the first XML node.

4.36 Allocation

Allocate and free XML trees.

Files

- file [tree.h](#)
SCEW tree handling routines.

Functions

- SCEW_API [scew_tree](#) * [scew_tree_create](#) (void)
Creates a new empty XML tree in memory.
- SCEW_API [scew_tree](#) * [scew_tree_copy](#) ([scew_tree](#) const *tree)
Makes a deep copy of the given tree.

- SCEW_API void `scew_tree_free` (`scew_tree` *tree)

Frees a tree memory structure.

4.36.1 Detailed Description

Allocate and free XML trees.

4.36.2 Function Documentation

4.36.2.1 SCEW_API `scew_tree*` `scew_tree_create` (void)

Creates a new empty XML tree in memory.

By default, the XML version is set to 1.0, and the encoding to UTF-8, also a standalone document is considered.

4.36.2.2 SCEW_API `scew_tree*` `scew_tree_copy` (`scew_tree const *` *tree*)

Makes a deep copy of the given *tree*.

A deep copy means that the root element and its children will be copied recursively. XML encoding, version and standalone attributes are also copied.

Precondition

`tree != NULL`

Parameters

tree the tree to be duplicated.

Returns

a new tree, or NULL if the copy failed.

4.36.2.3 SCEW_API void `scew_tree_free` (`scew_tree *` *tree*)

Frees a tree memory structure.

Call this function when you are done with your XML document. This will also free the root element recursively.

Parameters

tree the tree to delete.

4.37 Comparison

Tree comparison routines.

Typedefs

- typedef [scew_bool](#)(* [scew_tree_cmp_hook](#))(scew_tree const *, [scew_tree](#) const *)
SCEW tree compare hooks might be used to define new user XML tree comparisons.

Functions

- SCEW_API [scew_bool](#) [scew_tree_compare](#) ([scew_tree](#) const *a, [scew_tree](#) const *b, [scew_tree_cmp_hook](#) hook)
Performs a deep comparison for the given trees.

4.37.1 Detailed Description

Tree comparison routines.

4.37.2 Typedef Documentation

4.37.2.1 typedef [scew_bool](#)(* [scew_tree_cmp_hook](#))([scew_tree](#) const *, [scew_tree](#) const *)

SCEW tree compare hooks might be used to define new user XML tree comparisons.

The hooks are used by [scew_tree_compare](#).

Returns

true if the given XML trees are considered equal, false otherwise.

4.37.3 Function Documentation

4.37.3.1 SCEW_API [scew_bool](#) [scew_tree_compare](#) ([scew_tree](#) const * a, [scew_tree](#) const * b, [scew_tree_cmp_hook](#) hook)

Performs a deep comparison for the given trees.

The comparison is done via the comparison *hook*. If *hook* is NULL, the default comparison is done:

- XML declaration: version, encoding and standalone attribute (encoding is considered case-sensitive).
- Preamble is considered case-sensitive as well.
- The root element comparison uses [scew_element_compare](#) with a NULL element comparison hook.

There is no restriction on the provided comparison hook (if any), thus the user is responsible to define how the comparison is to be done.

Precondition

a != NULL
b != NULL

Parameters

- a* one of the trees to compare.
- b* one of the trees to compare.
- hook* the user defined comparison function. If NULL, the default comparison is used.

Returns

true if trees are considered equal, false otherwise.

4.38 Properties

Handle XML trees properties.

Files

- file [tree.h](#)
SCEW tree handling routines.

Enumerations

- enum [scew_tree_standalone](#) { [scew_tree_standalone_unknown](#), [scew_tree_standalone_no](#), [scew_tree_standalone_yes](#) }
List of possible values for the standalone attribute.

Functions

- SCEW_API XML_Char const * [scew_tree_xml_version](#) ([scew_tree](#) const *tree)
Returns the current XML version for the given tree.
- SCEW_API void [scew_tree_set_xml_version](#) ([scew_tree](#) *tree, XML_Char const *version)
Sets the XML version in the XML declaration to the given tree.
- SCEW_API XML_Char const * [scew_tree_xml_encoding](#) ([scew_tree](#) const *tree)
Returns the current XML character encoding for the given tree.
- SCEW_API void [scew_tree_set_xml_encoding](#) ([scew_tree](#) *tree, XML_Char const *encoding)
Sets the character encoding used in the given XML tree.
- SCEW_API [scew_tree_standalone](#) [scew_tree_xml_standalone](#) ([scew_tree](#) const *tree)
Returns whether the given tree is an standalone document.
- SCEW_API void [scew_tree_set_xml_standalone](#) ([scew_tree](#) *tree, [scew_tree_standalone](#) standalone)
The standalone property tells the XML processor whether there are any other extra files to load, such as external entities or DTDs.

4.38.1 Detailed Description

Handle XML trees properties.

4.38.2 Enumeration Type Documentation

4.38.2.1 enum `scew_tree_standalone`

List of possible values for the standalone attribute.

The standalone attribute in an XML declaration defines whether the XML document is self consistent or not, that is, whether it needs to load any extra files.

Enumerator:

`scew_tree_standalone_unknown` Standalone attribute not defined.

`scew_tree_standalone_no` Extra files are necessary.

`scew_tree_standalone_yes` Document stands on its own.

4.38.3 Function Documentation

4.38.3.1 SCEW_API XML_Char const* `scew_tree_xml_version` (`scew_tree` const * *tree*)

Returns the current XML version for the given *tree*.

This is the version specified in the "version" attribute in the XML declaration.

Precondition

`tree` != NULL

Parameters

tree the tree to return its version for.

Returns

a string representing the XML version.

4.38.3.2 SCEW_API void `scew_tree_set_xml_version` (`scew_tree` * *tree*, XML_Char const * *version*)

Sets the XML *version* in the XML declaration to the given *tree*.

Currently there is one XML version, so the value is always 1.0. If there were more XML versions, this property tells to the XML processor which one to use.

Precondition

`tree` != NULL

`version` != NULL

Parameters

tree the XML tree to set the new XML version to.

version the new XML version for the given tree.

4.38.3.3 SCEW_API XML_Char const* scew_tree_xml_encoding (scew_tree const * tree)

Returns the current XML character encoding for the given *tree*.

The default, when creating new SCEW trees, is UTF-8.

Expat supports the following encodings:

- UTF-8, ASCII and ISO-8859-1.
- UTF-16.

As SCEW is based on Expat the same encodings are supported when parsing XML documents. However, SCEW only supports UTF-16 in Windows platforms.

Note that these encodings are only supported when parsing files, but not when creating new ones. So, it is the responsibility of the user to provide the correct characters.

Precondition

tree != NULL

Parameters

tree the XML tree to obtain its character encoding for.

Returns

the character encoding for the given tree.

4.38.3.4 SCEW_API void scew_tree_set_xml_encoding (scew_tree * tree, XML_Char const * encoding)

Sets the character encoding used in the given XML *tree*.

Note that a user might want to use another encoding, different than the ones supported by Expat. And, as SCEW does not provide, or force, any encoding, the user is allowed to do so.

Precondition

tree != NULL
encoding != NULL

Parameters

tree the XML tree to set the new encoding to.

encoding the new character encoding for the given tree.

4.38.3.5 SCEW_API scew_tree_standalone scew_tree_xml_standalone (scew_tree const * tree)

Returns whether the given *tree* is an standalone document.

The standalone property tells the XML processor whether there are any other extra files to load, such as external entities or DTDs.

Precondition

`tree != NULL`

Parameters

tree the tree to check its standalone property for.

Returns

the XML tree standalone property.

4.38.3.6 SCEW_API void `scew_tree_set_xml_standalone` (`scew_tree * tree`, `scew_tree_standalone standalone`)

The standalone property tells the XML processor whether there are any other extra files to load, such as external entities or DTDs.

If the XML document can stand on its own, set it to [`scew_tree_standalone_yes`](#).

Precondition

`tree != NULL`

Parameters

tree the XML tree to set the option to.

standalone the new XML tree standalone property.

4.39 Contents

Accessors for XML root elements and preambles.

Files

- file [tree.h](#)
SCEW tree handling routines.

Functions

- SCEW_API `scew_element * scew_tree_root` (`scew_tree` const *tree)
Returns the root element of the given tree.
- SCEW_API `scew_element * scew_tree_set_root` (`scew_tree` *tree, XML_Char const *name)
Creates the root element of an XML tree with the given name.
- SCEW_API `scew_element * scew_tree_set_root_element` (`scew_tree` *tree, `scew_element` *root)
Sets the root element of an XML tree with the given element.
- SCEW_API XML_Char const * `scew_tree_xml_preamble` (`scew_tree` const *tree)

Return the XML preamble for the given tree.

- SCEW_API void `scew_tree_set_xml_preamble` (`scew_tree` *tree, XML_Char const *preamble)
Sets the preamble string for the XML document.

4.39.1 Detailed Description

Accessors for XML root elements and preambles.

4.39.2 Function Documentation

4.39.2.1 SCEW_API scew_element* scew_tree_root (scew_tree const * tree)

Returns the root element of the given *tree*.

Precondition

`tree != NULL`

Returns

the tree's root element, or NULL if the tree does not have a root element yet.

4.39.2.2 SCEW_API scew_element* scew_tree_set_root (scew_tree * tree, XML_Char const * name)

Creates the root element of an XML *tree* with the given *name*.

Note that if the tree already had a root element, it will be overwritten, possibly causing a memory leak, as the old root element is *not* automatically freed. So, if you plan to set a new root element, remember to free the old one first.

Precondition

`tree != NULL`
`name != NULL`

Parameters

tree the XML tree to set a new root element to.
name the name of the new XML root element.

Returns

the tree's root element, or NULL if the element could not be created.

4.39.2.3 SCEW_API scew_element* scew_tree_set_root_element (scew_tree * tree, scew_element * root)

Sets the root element of an XML *tree* with the given element.

Note that if the tree already had a root element, it will be overwritten, possibly causing a memory leak, as the old root element is *not* automatically freed. So, if you plan to set a new root element, remember to free the old one first.

Precondition

```
tree != NULL
root != NULL
```

Parameters

tree the XML tree to set a new root element to.
root the new XML root element.

Returns

the tree's root element, or NULL if the element could not be created.

4.39.2.4 SCEW_API XML_Char const* scew_tree_xml_preamble (scew_tree const * *tree*)

Return the XML preamble for the given *tree*.

The XML preamble is the text between the XML declaration and the first element. It typically contains DOCTYPE declarations or processing instructions.

SCEW does not provide specific functions for DOCTYPEs or processing instructions, but they are treated as a whole.

Precondition

```
tree != NULL
```

Parameters

tree the XML tree to obtain the preamble for.

Returns

the XML preamble, or NULL if no preamble is found.

4.39.2.5 SCEW_API void scew_tree_set_xml_preamble (scew_tree * *tree*, XML_Char const * *preamble*)

Sets the preamble string for the XML document.

Typically this contains DOCTYPE declarations or processing instructions. The old XML tree preamble will be freed, if any.

SCEW does not provide specific functions for DOCTYPEs or processing instructions, but they can be added as a whole.

Precondition

```
tree != NULL
preamble != NULL
```

Parameters

tree the XML tree to set the preamble to.
preamble the XML preamble text for the given tree.

4.40 Writers

Write data to different destinations: files, memory, etc.

Data Structures

- struct [scew_writer_hooks](#)

This is the set of functions that are implemented by all SCEW writers.

Modules

- [Memory](#)

Write data to memory buffers.

- [Files](#)

Write data to files.

Files

- file [writer.h](#)

SCEW writer common functions.

Typedefs

- typedef struct [scew_writer](#) [scew_writer](#)

This is the type declaration for SCEW writers.

Functions

- SCEW_API [scew_writer](#) * [scew_writer_create](#) ([scew_writer_hooks](#) const *hooks, void *data)

Creates a new SCEW writer with the given [scew_writer_hooks](#) implementation.

- SCEW_API void * [scew_writer_data](#) ([scew_writer](#) *writer)

Returns the reference to the internal data structure being used by the given writer.

- SCEW_API size_t [scew_writer_write](#) ([scew_writer](#) *writer, XML_Char const *buffer, size_t char_no)

Writes data from the given memory buffer to the specified writer.

- SCEW_API [scew_bool](#) [scew_writer_end](#) ([scew_writer](#) *writer)

Tells whether the given writer has reached its end.

- SCEW_API [scew_bool](#) [scew_writer_error](#) ([scew_writer](#) *writer)

Tells whether an error was found while sending data to the given writer.

- SCEW_API `scew_bool scew_writer_close (scew_writer *writer)`
Closes the given writer.
- SCEW_API `void scew_writer_free (scew_writer *writer)`
Frees the memory allocated by the given writer.

4.40.1 Detailed Description

Write data to different destinations: files, memory, etc. SCEW writers provide a common mechanism to write data to different destinations. This is done by implementing the set of functions declared in `scew_writer_hooks`. A user might create new SCEW writers by implementing those functions.

Once a SCEW writer is created, functions in this section should be used no matter the writer type.

4.40.2 Function Documentation

4.40.2.1 SCEW_API `scew_writer* scew_writer_create (scew_writer_hooks const * hooks, void * data)`

Creates a new SCEW writer with the given `scew_writer_hooks` implementation.

This function should be called internally when implementing a new SCEW writer destination. The `data` argument is a reference to some internal data used by the SCEW writer (file stream pointer, current memory buffer pointer, etc.). This data might be later obtained, by the SCEW writer implementation, via `scew_writer_data`.

Precondition

`hooks != NULL`

Parameters

hooks the implementation of the new SCEW writer.

data data to be used by the new SCEW writer. This is usually a reference to a file stream (in case of files) or a memory buffer pointer, etc.

Returns

a new SCEW writer, or NULL if the writer could not be created.

4.40.2.2 SCEW_API `void* scew_writer_data (scew_writer * writer)`

Returns the reference to the internal data structure being used by the given *writer*.

Precondition

`writer != NULL`

Parameters

writer the writer to obtain its internal data for.

Returns

a reference to the writer's internal data, or NULL if no data was set at creation time.

4.40.2.3 SCEW_API size_t scew_writer_write (scew_writer * *writer*, XML_Char const * *buffer*, size_t *char_no*)

Writes data from the given memory *buffer* to the specified *writer*.

This function will write as many characters (of size XML_Char) as specified by *char_no*. [scew_writer_error](#) and [scew_writer_end](#) need to be consulted to check whether an error is found or the end of the writer is reached, respectively.

This function will call the actual *write* function provided by the SCEW writer hooks ([scew_writer_hooks](#)).

Precondition

writer != NULL
buffer != NULL

Parameters

writer the writer where to send the data.
buffer the memory buffer from where to read data from.
char_no the number of characters to write.

Returns

the number of characters successfully written.

4.40.2.4 SCEW_API scew_bool scew_writer_end (scew_writer * *writer*)

Tells whether the given *writer* has reached its end.

That is, no more data can be written to the .

This function will call the actual *end* function provided by the SCEW writer hooks ([scew_writer_hooks](#)).

Precondition

writer != NULL

Parameters

writer the writer to check its end status for.

Returns

true if we are at the end of the writer, false otherwise.

4.40.2.5 SCEW_API scew_bool scew_writer_error (scew_writer * *writer*)

Tells whether an error was found while sending data to the given *writer*.

This function will call the actual *error* function provided by the SCEW writer hooks ([scew_writer_hooks](#)).

Precondition

`writer != NULL`

Parameters

writer the writer to check its status for.

Returns

true if we an error was found while sending data to the writer, false otherwise.

4.40.2.6 SCEW_API scew_bool scew_writer_close (scew_writer * writer)

Closes the given *writer*.

This function will have different effects depending on the SCEW writer type (e.g. it will close the file for file streams). After calling this function, none of the SCEW writer functions should be used, otherwise undefined behavior is expected.

This function will call the actual *close* function provided by the SCEW writer hooks ([scew_writer_hooks](#)).

Precondition

`writer != NULL`

Parameters

writer the writer to close.

Returns

true if the writer was successfully closed, false otherwise.

4.40.2.7 SCEW_API void scew_writer_free (scew_writer * writer)

Frees the memory allocated by the given *writer*.

This function will call the actual *free* function provided by the SCEW writer hooks ([scew_writer_hooks](#)).

Precondition

`writer != NULL`

Parameters

writer the writer to free.

4.41 Memory

Write data to memory buffers.

Files

- file [writer_buffer.h](#)
SCEW writer functions for memory buffers.

Functions

- SCEW_API [scew_writer](#) * [scew_writer_buffer_create](#) (XML_Char *buffer, size_t size)
Creates a new SCEW writer for the given memory buffer of the specified size.

4.41.1 Detailed Description

Write data to memory buffers.

4.41.2 Function Documentation

4.41.2.1 SCEW_API scew_writer* scew_writer_buffer_create (XML_Char * *buffer*, size_t *size*)

Creates a new SCEW writer for the given memory *buffer* of the specified *size*.

The buffer should exist before calling this function and the *size* of the buffer should be large enough to store the desired information (e.g. an XML tree, an element...). Once the writer is created, any of the [Writers](#) functions might be called in order to store data to the buffer.

Precondition

buffer != NULL
size > 0

Parameters

buffer the memory area where the new SCEW writer will write to.
size the size of the memory area.

Returns

a new SCEW writer for the given buffer or NULL if the writer could not be created.

4.42 Files

Write data to files.

Files

- file [writer_file.h](#)
SCEW writer functions for files.

Functions

- SCEW_API [scew_writer](#) * [scew_writer_file_create](#) (char const *file_name)
Creates a new SCEW writer for the given file name.

- SCEW_API `scew_writer` * `scew_writer_fp_create` (FILE *file)
Creates a new SCEW writer for the given file stream.

4.42.1 Detailed Description

Write data to files.

4.42.2 Function Documentation

4.42.2.1 SCEW_API `scew_writer`* `scew_writer_file_create` (`char const` * *file_name*)

Creates a new SCEW writer for the given file name.

This routine will create a new file if the file does not exist or it will overwrite the existing one. The file will be created in text mode. Once the writer is created, the [Writers](#) routines must be called in order to store data to the file or to know the file status.

For UTF-16 encoding (only in Windows platforms) the BOM (Byte Order Mask) is automatically handled by the Windows API.

Precondition

`file_name` != NULL

Parameters

file_name the file name to create for the new SCEW writer.

Returns

a new SCEW writer for the given file name or NULL if the writer could not be created (e.g. memory allocation, file permissions, etc.).

4.42.2.2 SCEW_API `scew_writer`* `scew_writer_fp_create` (`FILE` * *file*)

Creates a new SCEW writer for the given *file* stream.

The file stream is created in text mode. Once the writer is created, any of the [Writers](#) routines must be called in order to store data to the file or to know the file status.

For UTF-16 encoding (only in Windows platforms) the BOM (Byte Order Mask) is automatically handled by the Windows API.

Precondition

`file` != NULL

Parameters

file the file where the new SCEW writer will write to.

Returns

a new SCEW writer for the given file stream or NULL if the writer could not be created (e.g. memory allocation, file permissions, etc.).

Chapter 5

Data Structure Documentation

5.1 `scew_reader_hooks` Struct Reference

This is the set of functions that are implemented by all SCEW reader sources.

```
#include <reader.h>
```

Data Fields

- `size_t(* read)(scew_reader *, XML_Char *, size_t)`
- `scew_bool(* end)(scew_reader *)`
- `scew_bool(* error)(scew_reader *)`
- `scew_bool(* close)(scew_reader *)`
- `void(* free)(scew_reader *)`

5.1.1 Detailed Description

This is the set of functions that are implemented by all SCEW reader sources. They must not be used directly, but through the common routines to be used with any type of SCEW reader.

5.1.2 Field Documentation

5.1.2.1 `size_t(* scew_reader_hooks::read)(scew_reader *, XML_Char *, size_t)`

See also

[scew_reader_read](#)

5.1.2.2 `scew_bool(* scew_reader_hooks::end)(scew_reader *)`

See also

[scew_reader_end](#)

5.1.2.3 `scew_bool(* scew_reader_hooks::error)(scew_reader *)`

See also

[scew_reader_error](#)

5.1.2.4 `scew_bool(* scew_reader_hooks::close)(scew_reader *)`

See also

[scew_reader_close](#)

5.1.2.5 `void(* scew_reader_hooks::free)(scew_reader *)`

See also

[scew_reader_free](#)

The documentation for this struct was generated from the following file:

- [reader.h](#)

5.2 `scew_writer_hooks` Struct Reference

This is the set of functions that are implemented by all SCEW writers.

```
#include <writer.h>
```

Data Fields

- `size_t(* write)(scew_writer *, XML_Char const *, size_t)`
- `scew_bool(* end)(scew_writer *)`
- `scew_bool(* error)(scew_writer *)`
- `scew_bool(* close)(scew_writer *)`
- `void(* free)(scew_writer *)`

5.2.1 Detailed Description

This is the set of functions that are implemented by all SCEW writers. They must not be used directly, but through the common routines to be used with any type of SCEW writer.

5.2.2 Field Documentation

5.2.2.1 `size_t(* scew_writer_hooks::write)(scew_writer *, XML_Char const *, size_t)`

See also

[scew_writer_write](#)

5.2.2.2 `scew_bool(* scew_writer_hooks::end)(scew_writer *)`

See also

[scew_writer_end](#)

5.2.2.3 `scew_bool(* scew_writer_hooks::error)(scew_writer *)`

See also

[scew_writer_error](#)

5.2.2.4 `scew_bool(* scew_writer_hooks::close)(scew_writer *)`

See also

[scew_writer_close](#)

5.2.2.5 `void(* scew_writer_hooks::free)(scew_writer *)`

See also

[scew_writer_free](#)

The documentation for this struct was generated from the following file:

- [writer.h](#)

Chapter 6

File Documentation

6.1 attribute.h File Reference

SCEW attribute's handling routines.

```
#include "element.h"  
#include "bool.h"  
#include <expat.h>
```

Functions

- SCEW_API [scew_attribute](#) * [scew_attribute_create](#) (XML_Char const *name, XML_Char const *value)
Creates a new attribute with the given pair (name, value).
- SCEW_API [scew_attribute](#) * [scew_attribute_copy](#) ([scew_attribute](#) const *attribute)
Makes a copy of the given attribute.
- SCEW_API void [scew_attribute_free](#) ([scew_attribute](#) *attribute)
Frees the given attribute.
- SCEW_API [scew_bool](#) [scew_attribute_compare](#) ([scew_attribute](#) const *a, [scew_attribute](#) const *b)
Performs a comparison between the two given attributes.
- SCEW_API XML_Char const * [scew_attribute_name](#) ([scew_attribute](#) const *attribute)
Returns the given attribute's name.
- SCEW_API XML_Char const * [scew_attribute_value](#) ([scew_attribute](#) const *attribute)
Returns the given attribute's value.
- SCEW_API XML_Char const * [scew_attribute_set_name](#) ([scew_attribute](#) *attribute, XML_Char const *name)
Sets a new name to the given attribute and frees the old one.

- SCEW_API XML_Char const * [scew_attribute_set_value](#) ([scew_attribute](#) *attribute, XML_Char const *value)
Sets a new value to the given attribute and frees the old one.
- SCEW_API [scew_element](#) * [scew_attribute_parent](#) ([scew_attribute](#) const *attribute)
Returns the element that the given attribute belongs to.

6.1.1 Detailed Description

SCEW attribute's handling routines.

Author

Aleix Conchillo Flaque <aleix@member.fsf.org>

Date

Mon Nov 25, 2002 00:39 , , ,

6.2 bool.h File Reference

SCEW boolean type declaration.

Defines

- #define [SCEW_TRUE](#) (([scew_bool](#)) 1)
True.
- #define [SCEW_FALSE](#) (([scew_bool](#)) 0)
False.

Typedefs

- typedef unsigned char [scew_bool](#)
This should be defined using `stdbool.h` when C99 is available.

6.2.1 Detailed Description

SCEW boolean type declaration.

Author

Aleix Conchillo Flaque <aleix@member.fsf.org>

Date

Thu Sep 04, 2008 11:42

6.3 element.h File Reference

SCEW element's handling routines.

```
#include "export.h"
#include "list.h"
#include <expat.h>
```

Typedefs

- typedef struct [scew_element](#) [scew_element](#)
This is the type declaration for SCEW elements.
- typedef struct [scew_attribute](#) [scew_attribute](#)
This is the type declaration for SCEW attributes.
- typedef [scew_bool](#)(* [scew_element_cmp_hook](#))([scew_element](#) const *, [scew_element](#) const *)
SCEW element compare hooks might be used to define new user XML element comparisons.

Functions

- SCEW_API [scew_element](#) * [scew_element_create](#) (XML_Char const *name)
Creates a new element with the given name.
- SCEW_API [scew_element](#) * [scew_element_copy](#) ([scew_element](#) const *element)
Makes a deep copy of the given element.
- SCEW_API void [scew_element_free](#) ([scew_element](#) *element)
Frees the given element recursively.
- SCEW_API [scew_element](#) * [scew_element_by_name](#) ([scew_element](#) const *element, XML_Char const *name)
Returns the first child from the specified element that matches the given name.
- SCEW_API [scew_element](#) * [scew_element_by_index](#) ([scew_element](#) const *element, unsigned int index)
Returns the child of the given element at the specified zero-based index.
- SCEW_API [scew_list](#) * [scew_element_list_by_name](#) ([scew_element](#) const *element, XML_Char const *name)
Returns a list of children from the specified element that matches the given name.
- SCEW_API [scew_bool](#) [scew_element_compare](#) ([scew_element](#) const *a, [scew_element](#) const *b, [scew_element_cmp_hook](#) hook)
Performs a deep comparison of the two given elements.
- SCEW_API XML_Char const * [scew_element_name](#) ([scew_element](#) const *element)
Returns the given element's name.

- SCEW_API XML_Char const * [scew_element_contents](#) ([scew_element](#) const *element)
Returns the given element's contents.
- SCEW_API XML_Char const * [scew_element_set_name](#) ([scew_element](#) *element, XML_Char const *name)
Sets a new name to the given element and frees the old one.
- SCEW_API XML_Char const * [scew_element_set_contents](#) ([scew_element](#) *element, XML_Char const *contents)
Sets a new contents to the given element and frees the old one.
- SCEW_API void [scew_element_free_contents](#) ([scew_element](#) *element)
Frees the current contents of the given element.
- SCEW_API unsigned int [scew_element_count](#) ([scew_element](#) const *element)
Returns the number of children of the specified element.
- SCEW_API [scew_element](#) * [scew_element_parent](#) ([scew_element](#) const *element)
Returns the parent of the given element.
- SCEW_API [scew_list](#) * [scew_element_children](#) ([scew_element](#) const *element)
Returns the list of all the element's children.
- SCEW_API [scew_element](#) * [scew_element_add](#) ([scew_element](#) *element, XML_Char const *name)
Creates and adds, as a child of element, a new element with the given name.
- SCEW_API [scew_element](#) * [scew_element_add_pair](#) ([scew_element](#) *element, XML_Char const *name, XML_Char const *contents)
Creates and adds, as a child of element, a new element with the given name and contents.
- SCEW_API [scew_element](#) * [scew_element_add_element](#) ([scew_element](#) *element, [scew_element](#) *child)
Adds a child to the given element.
- SCEW_API void [scew_element_delete_all](#) ([scew_element](#) *element)
Deletes all the children for the given element.
- SCEW_API void [scew_element_delete_all_by_name](#) ([scew_element](#) *element, XML_Char const *name)
Deletes all the children of the given element that matches name.
- SCEW_API void [scew_element_delete_by_name](#) ([scew_element](#) *element, XML_Char const *name)
Deletes the first child of the given element that matches name.
- SCEW_API void [scew_element_delete_by_index](#) ([scew_element](#) *element, unsigned int index)
Deletes the child of the given element at the specified zero-based index.

- SCEW_API void `scew_element_detach` (`scew_element *element`)
Detaches the given element from its parent, if any.
- SCEW_API unsigned int `scew_element_attribute_count` (`scew_element const *element`)
Returns the number of attributes of the given element.
- SCEW_API `scew_list * scew_element_attributes` (`scew_element const *element`)
Returns the list of all the element's attributes.
- SCEW_API `scew_attribute * scew_element_attribute_by_name` (`scew_element const *element`, `XML_Char const *name`)
Returns the first attribute from the specified element that matches the given name.
- SCEW_API `scew_attribute * scew_element_attribute_by_index` (`scew_element const *element`, unsigned int `index`)
Returns the attribute of the given element at the specified zero-based index.
- SCEW_API `scew_attribute * scew_element_add_attribute` (`scew_element *element`, `scew_attribute *attribute`)
Adds an existent attribute to the given element.
- SCEW_API `scew_attribute * scew_element_add_attribute_pair` (`scew_element *element`, `XML_Char const *name`, `XML_Char const *value`)
Creates and adds a new attribute to the given element.
- SCEW_API void `scew_element_delete_attribute_all` (`scew_element *element`)
Deletes all the attributes of the given element.
- SCEW_API void `scew_element_delete_attribute` (`scew_element *element`, `scew_attribute *attribute`)
Deletes the given attribute from the specified element.
- SCEW_API void `scew_element_delete_attribute_by_name` (`scew_element *element`, `XML_Char const *name`)
Deletes the first attribute of the given element that matches name.
- SCEW_API void `scew_element_delete_attribute_by_index` (`scew_element *element`, unsigned int `index`)
Deletes the attribute of the given element at the specified zero-based index.

6.3.1 Detailed Description

SCEW element's handling routines.

Author

Aleix Conchillo Flaque <aleix@member.fsf.org>

Date

Mon Nov 25, 2002 00:48 , , , ,

6.4 error.h File Reference

SCEW error handling functions.

```
#include "export.h"
#include "parser.h"
#include <expat.h>
```

Enumerations

- enum `scew_error` {
`scew_error_none`, `scew_error_no_memory`, `scew_error_io`, `scew_error_hook`,
`scew_error_expat`, `scew_error_internal`, `scew_error_unknown` }

This is the type declaration of the SCEW error.

Functions

- SCEW_API `scew_error scew_error_code` (void)
Returns the SCEW internal error code.
- SCEW_API XML_Char const * `scew_error_string` (scew_error code)
Returns a string describing the given internal SCEW error code.
- SCEW_API enum XML_Error `scew_error_expat_code` (scew_parser *parser)
Returns the Expat internal error code.
- SCEW_API XML_Char const * `scew_error_expat_string` (enum XML_Error code)
Returns a string describing the internal Expat error for the given error code.
- SCEW_API int `scew_error_expat_line` (scew_parser *parser)
Returns the current line at which the error was detected.
- SCEW_API int `scew_error_expat_column` (scew_parser *parser)
Returns the current column at which the error was detected.

6.4.1 Detailed Description

SCEW error handling functions.

Author

Aleix Conchillo Flaque <aleix@member.fsf.org>

Date

Mon May 05, 2003 10:29 , ,

6.5 export.h File Reference

SCEW shared library support.

6.5.1 Detailed Description

SCEW shared library support.

Author

Aleix Conchillo Flaque <aleix@member.fsf.org>

Date

Fri Sep 04, 2009 00:14

6.6 list.h File Reference

SCEW general list implementation.

```
#include "export.h"
#include "bool.h"
```

Typedefs

- typedef struct [scew_list](#) [scew_list](#)
This is the type delcaration for SCEW lists.
- typedef void(* [scew_list_hook](#))([scew_list](#) *, void *)
SCEW lists hooks (functions) are used by [scew_list_foreach](#).
- typedef [scew_bool](#)(* [scew_cmp_hook](#))(void const *, void const *)
SCEW lists comparison hooks are used by [scew_list_find_custom](#).

Functions

- SCEW_API [scew_list](#) * [scew_list_create](#) (void *data)
Creates a new list item with the given data.
- SCEW_API void [scew_list_free](#) ([scew_list](#) *list)
Frees all the items from the given list.
- SCEW_API void * [scew_list_data](#) ([scew_list](#) *list)
Returns the data pointer of the given list item.
- SCEW_API unsigned int [scew_list_size](#) ([scew_list](#) *list)
Returns the number of items in the given list.

- SCEW_API `scew_list * scew_list_append (scew_list *list, void *data)`
Creates a new list item with the given data and appends it to list.
- SCEW_API `scew_list * scew_list_prepend (scew_list *list, void *data)`
Creates a new list item with the given data and prepends it to list.
- SCEW_API `scew_list * scew_list_delete (scew_list *list, void *data)`
Deletes the first item pointing to data from the given list.
- SCEW_API `scew_list * scew_list_delete_item (scew_list *list, scew_list *item)`
Deletes the given list item from list.
- SCEW_API `scew_list * scew_list_first (scew_list *list)`
Finds the first item of the given list.
- SCEW_API `scew_list * scew_list_last (scew_list *list)`
Finds the last item of the given list.
- SCEW_API `scew_list * scew_list_next (scew_list *list)`
Obtains the next item of the given list item.
- SCEW_API `scew_list * scew_list_previous (scew_list *list)`
Obtains the previous item of the given list item.
- SCEW_API `scew_list * scew_list_index (scew_list *list, unsigned int index)`
Gets the list item at the given index.
- SCEW_API `void scew_list_foreach (scew_list *list, scew_list_hook hook, void *user_data)`
Traverses all list items and executes the given hook for each item found.
- SCEW_API `scew_list * scew_list_find (scew_list *list, void *data)`
Finds the first list item that contains data.
- SCEW_API `scew_list * scew_list_find_custom (scew_list *list, void const *data, scew_cmp_hook hook)`
Finds the first list item that matches the given predicate, hook.

6.6.1 Detailed Description

SCEW general list implementation.

Author

Aleix Conchillo Flaque <aleix@member.fsf.org>

Date

Thu Jul 12, 2007 20:09 , , , ,

6.7 parser.h File Reference

SCEW parser handling routines.

```
#include "export.h"
#include "bool.h"
#include "reader.h"
#include "tree.h"
#include <expat.h>
#include <stdio.h>
```

Typedefs

- typedef struct [scew_parser](#) [scew_parser](#)
This is the type declaration of the SCEW parser.
- typedef [scew_bool](#)(* [scew_parser_load_hook](#))([scew_parser](#) *, void *, void *)
SCEW parser hooks might be used as notifications to know when XML elements or trees are parsed.

Functions

- SCEW_API [scew_parser](#) * [scew_parser_create](#) (void)
Creates a new parser.
- SCEW_API [scew_parser](#) * [scew_parser_namespace_create](#) (XML_Char separator)
Creates a new parser with namespaces support.
- SCEW_API void [scew_parser_free](#) ([scew_parser](#) *parser)
Frees a parser memory structure.
- SCEW_API [scew_tree](#) * [scew_parser_load](#) ([scew_parser](#) *parser, [scew_reader](#) *reader)
Loads an XML tree from the specified reader.
- SCEW_API [scew_bool](#) [scew_parser_load_stream](#) ([scew_parser](#) *parser, [scew_reader](#) *reader)
Loads multiple XML trees from the specified stream reader.
- SCEW_API void [scew_parser_reset](#) ([scew_parser](#) *parser)
Resets the given parser for further uses.
- SCEW_API void [scew_parser_set_element_hook](#) ([scew_parser](#) *parser, [scew_parser_load_hook](#) hook, void *user_data)
Registers a hook to be called once an XML element is successfully parsed.
- SCEW_API void [scew_parser_set_tree_hook](#) ([scew_parser](#) *parser, [scew_parser_load_hook](#) hook, void *user_data)
Registers a hook to be called once an XML tree is successfully parsed.

- SCEW_API void `scew_parser_ignore_whitespaces` (`scew_parser` *parser, `scew_bool` ignore)
Tells the parser how to treat white spaces.
- SCEW_API XML_Parser `scew_parser_extern` (`scew_parser` *parser)
Returns the internal Expat parser being used by the given SCEW parser.

6.7.1 Detailed Description

SCEW parser handling routines.

Author

Aleix Conchillo Flaque <aleix@member.fsf.org>

Date

Mon Nov 25, 2002 00:57 , , ,

6.8 printer.h File Reference

SCEW printer routines for XML output.

```
#include "export.h"  
#include "writer.h"
```

Typedefs

- typedef struct `scew_printer` `scew_printer`
This is the type declaration for the SCEW printer.

Functions

- SCEW_API `scew_printer` * `scew_printer_create` (`scew_writer` *writer)
Creates a new SCEW printer that will use the given writer by default.
- SCEW_API void `scew_printer_free` (`scew_printer` *printer)
Frees the given SCEW printer.
- SCEW_API void `scew_printer_set_indented` (`scew_printer` *printer, `scew_bool` indented)
Tells whether the output sent to the given SCEW printer should be indented or not.
- SCEW_API void `scew_printer_set_indentation` (`scew_printer` *printer, unsigned int spaces)
Sets the number of spaces to use when indenting output for the given SCEW printer.
- SCEW_API `scew_writer` * `scew_printer_set_writer` (`scew_printer` *printer, `scew_writer` *writer)
Sets the given SCEW writer to the specified printer.

- SCEW_API `scew_bool scew_printer_print_tree` (`scew_printer *printer`, `scew_tree const *tree`)
Prints the given SCEW tree to the specified printer.
- SCEW_API `scew_bool scew_printer_print_element` (`scew_printer *printer`, `scew_element const *element`)
Prints the given SCEW element to the specified printer.
- SCEW_API `scew_bool scew_printer_print_element_children` (`scew_printer *printer`, `scew_element const *element`)
Prints the given SCEW element children to the specified printer.
- SCEW_API `scew_bool scew_printer_print_element_attributes` (`scew_printer *printer`, `scew_element const *element`)
Prints the given SCEW element attributes to the specified printer.
- SCEW_API `scew_bool scew_printer_print_attribute` (`scew_printer *printer`, `scew_attribute const *attribute`)
Prints the given SCEW attribute to the specified printer.

6.8.1 Detailed Description

SCEW printer routines for XML output.

Author

Aleix Conchillo Flaque <aleix@member.fsf.org>

Date

Fri Jan 16, 2009 22:34 , , ,

6.9 reader.h File Reference

SCEW reader common functions.

```
#include "export.h"  
#include "bool.h"  
#include <expat.h>  
#include <stddef.h>
```

Data Structures

- struct `scew_reader_hooks`

This is the set of functions that are implemented by all SCEW reader sources.

Typedefs

- typedef struct `scew_reader` `scew_reader`
This is the type declaration for SCEW readers.

Functions

- SCEW_API `scew_reader *` `scew_reader_create` (`scew_reader_hooks` const `*hooks`, void `*data`)
Creates a new SCEW reader with the given `scew_reader_hooks` implementation.
- SCEW_API void `*` `scew_reader_data` (`scew_reader *``reader`)
Returns the reference to the internal data structure being used by the given reader.
- SCEW_API `size_t` `scew_reader_read` (`scew_reader *``reader`, XML_Char `*buffer`, `size_t` `char_no`)
Reads data from the given reader in store it in the specified buffer.
- SCEW_API `scew_bool` `scew_reader_end` (`scew_reader *``reader`)
Tells whether the given reader has reached its end.
- SCEW_API `scew_bool` `scew_reader_error` (`scew_reader *``reader`)
Tells whether an error was found while reading from the given reader.
- SCEW_API `scew_bool` `scew_reader_close` (`scew_reader *``reader`)
Closes the given reader.
- SCEW_API void `scew_reader_free` (`scew_reader *``reader`)
Frees the memory allocated by the given reader.

6.9.1 Detailed Description

SCEW reader common functions.

Author

Aleix Conchillo Flaque <aleix@member.fsf.org>

Date

Sun Nov 23, 2008 13:36

6.10 reader_buffer.h File Reference

SCEW reader functions for memory buffers.

```
#include "export.h"
#include "reader.h"
#include <expat.h>
```

Functions

- SCEW_API `scew_reader` * `scew_reader_buffer_create` (XML_Char const *buffer, size_t size)
Creates a new SCEW reader for the given memory buffer of the specified size.

6.10.1 Detailed Description

SCEW reader functions for memory buffers.

Author

Aleix Conchillo Flaque <aleix@member.fsf.org>

Date

Tue Aug 25, 2009 02:02

6.11 reader_file.h File Reference

SCEW reader functions for files.

```
#include "export.h"  
#include "reader.h"  
#include <stdio.h>
```

Functions

- SCEW_API `scew_reader` * `scew_reader_file_create` (char const *file_name)
Creates a new SCEW reader for the given file name.
- SCEW_API `scew_reader` * `scew_reader_fp_create` (FILE *file)
Creates a new SCEW reader for the given file stream.

6.11.1 Detailed Description

SCEW reader functions for files.

Author

Aleix Conchillo Flaque <aleix@member.fsf.org>

Date

Sun Nov 23, 2008 13:51

6.12 scew.h File Reference

SCEW main header file.

```
#include "export.h"
#include "attribute.h"
#include "bool.h"
#include "element.h"
#include "error.h"
#include "list.h"
#include "parser.h"
#include "printer.h"
#include "reader.h"
#include "reader_buffer.h"
#include "reader_file.h"
#include "str.h"
#include "tree.h"
#include "writer.h"
#include "writer_buffer.h"
#include "writer_file.h"
```

6.12.1 Detailed Description

SCEW main header file.

Author

Aleix Conchillo Flaque <aleix@member.fsf.org>

Date

Mon Nov 25, 2002 01:34

6.13 str.h File Reference

SCEW string functions.

```
#include "bool.h"
#include "export.h"
#include <expat.h>
#include <string.h>
#include <ctype.h>
```

Defines

- #define [scew_memcpy](#)(dst, src, n) memcpy (dst, src, sizeof (XML_Char) * (n))
Copy the number of given characters from src to dst.
- #define [scew_memmove](#)(dst, src, n) memmove (dst, src, sizeof (XML_Char) * (n))
Move the number of given characters from src to dst.
- #define [_XT](#)(str) str
Creates a regular string or a wide character string.
- #define [scew_printf](#) printf
See standard printf documentation.
- #define [scew_fprintf](#) fprintf
See standard fprintf documentation.
- #define [scew_vfprintf](#) vfprintf
See standard vfprintf documentation.
- #define [scew_fputs](#) fputs
See standard fputs documentation.
- #define [scew_fgets](#) fgets
See standard fgets documentation.
- #define [scew_fputc](#) fputc
See standard fputc documentation.
- #define [scew_fgetc](#) fgetc
See standard fgetc documentation.
- #define [scew_strspn](#)(s, accept) strspn (s, accept)
See standard strspn documentation.
- #define [scew_strcpy](#)(dest, src) strcpy (dest, src)
See standard strcpy documentation.
- #define [scew_strcat](#)(dest, src) strcat (dest, src)
See standard strcat documentation.
- #define [scew_strncpy](#)(dest, src, n) strncpy (dest, src, (n))
See standard strncpy documentation.
- #define [scew_strncat](#)(dest, src, n) strncat (dest, src, (n))
See standard strncat documentation.
- #define [scew_strlen](#)(s) strlen (s)
See standard strlen documentation.

- #define `scew_isalnum(c)` `isalnum ((unsigned char)(c))`
See standard `isalnum` documentation.
- #define `scew_isalpha(c)` `isalpha ((unsigned char)(c))`
See standard `isalpha` documentation.
- #define `scew_iscntrl(c)` `iscntrl ((unsigned char)(c))`
See standard `iscntrl` documentation.
- #define `scew_isdigit(c)` `isdigit ((unsigned char)(c))`
See standard `isdigit` documentation.
- #define `scew_isxdigit(c)` `isxdigit ((unsigned char)(c))`
See standard `isxdigit` documentation.
- #define `scew_isgraph(c)` `isgraph ((unsigned char)(c))`
See standard `isgraph` documentation.
- #define `scew_islower(c)` `islower ((unsigned char)(c))`
See standard `islower` documentation.
- #define `scew_isupper(c)` `isupper ((unsigned char)(c))`
See standard `isupper` documentation.
- #define `scew_isprint(c)` `isprint ((unsigned char)(c))`
See standard `isprint` documentation.
- #define `scew_ispunct(c)` `ispunct ((unsigned char)(c))`
See standard `ispunct` documentation.
- #define `scew_isspace(c)` `isspace ((unsigned char)(c))`
See standard `isspace` documentation.

Functions

- SCEW_API int `scew_strcmp` (XML_Char const *a, XML_Char const *b)
Compares the two given strings `s1` and `s2`.
- SCEW_API XML_Char * `scew_strdup` (XML_Char const *src)
Creates a new copy of the given string.
- SCEW_API void `scew_strtrim` (XML_Char *src)
Trims off extra spaces from the beginning and end of a string.
- SCEW_API `scew_bool scew_isempty` (XML_Char const *src)
Tells whether the given string is empty.
- XML_Char * `scew_strescape` (XML_Char const *src)
Escapes the given string for XML.

6.13.1 Detailed Description

SCEW string functions.

Author

Aleix Conchillo Flaque <aleix@member.fsf.org>

Date

Sun Dec 01, 2002 13:05

6.14 tree.h File Reference

SCEW tree handling routines.

```
#include "export.h"
#include "element.h"
#include <expat.h>
```

Typedefs

- typedef struct [scew_tree](#) [scew_tree](#)
This is the type declaration for XML trees.
- typedef [scew_bool](#)(* [scew_tree_cmp_hook](#))([scew_tree](#) const *, [scew_tree](#) const *)
SCEW tree compare hooks might be used to define new user XML tree comparisons.

Enumerations

- enum [scew_tree_standalone](#) { [scew_tree_standalone_unknown](#), [scew_tree_standalone_no](#), [scew_tree_standalone_yes](#) }
List of possible values for the standalone attribute.

Functions

- SCEW_API [scew_tree](#) * [scew_tree_create](#) (void)
Creates a new empty XML tree in memory.
- SCEW_API [scew_tree](#) * [scew_tree_copy](#) ([scew_tree](#) const *tree)
Makes a deep copy of the given tree.
- SCEW_API void [scew_tree_free](#) ([scew_tree](#) *tree)
Frees a tree memory structure.
- SCEW_API [scew_bool](#) [scew_tree_compare](#) ([scew_tree](#) const *a, [scew_tree](#) const *b, [scew_tree_cmp_hook](#) hook)

Performs a deep comparison for the given trees.

- SCEW_API XML_Char const * [scew_tree_xml_version](#) ([scew_tree](#) const *tree)
Returns the current XML version for the given tree.
- SCEW_API void [scew_tree_set_xml_version](#) ([scew_tree](#) *tree, XML_Char const *version)
Sets the XML version in the XML declaration to the given tree.
- SCEW_API XML_Char const * [scew_tree_xml_encoding](#) ([scew_tree](#) const *tree)
Returns the current XML character encoding for the given tree.
- SCEW_API void [scew_tree_set_xml_encoding](#) ([scew_tree](#) *tree, XML_Char const *encoding)
Sets the character encoding used in the given XML tree.
- SCEW_API [scew_tree_standalone](#) [scew_tree_xml_standalone](#) ([scew_tree](#) const *tree)
Returns whether the given tree is an standalone document.
- SCEW_API void [scew_tree_set_xml_standalone](#) ([scew_tree](#) *tree, [scew_tree_standalone](#) standalone)
The standalone property tells the XML processor whether there are any other extra files to load, such as external entities or DTDs.
- SCEW_API [scew_element](#) * [scew_tree_root](#) ([scew_tree](#) const *tree)
Returns the root element of the given tree.
- SCEW_API [scew_element](#) * [scew_tree_set_root](#) ([scew_tree](#) *tree, XML_Char const *name)
Creates the root element of an XML tree with the given name.
- SCEW_API [scew_element](#) * [scew_tree_set_root_element](#) ([scew_tree](#) *tree, [scew_element](#) *root)
Sets the root element of an XML tree with the given element.
- SCEW_API XML_Char const * [scew_tree_xml_preamble](#) ([scew_tree](#) const *tree)
Return the XML preamble for the given tree.
- SCEW_API void [scew_tree_set_xml_preamble](#) ([scew_tree](#) *tree, XML_Char const *preamble)
Sets the preamble string for the XML document.

6.14.1 Detailed Description

SCEW tree handling routines.

Author

Aleix Conchillo Flaque <aleix@member.fsf.org>

Date

Thu Feb 20, 2003 23:32 , , ,

6.15 writer.h File Reference

SCEW writer common functions.

```
#include "export.h"
#include "tree.h"
#include "attribute.h"
#include "bool.h"
#include <expat.h>
#include <stddef.h>
```

Data Structures

- struct [scew_writer_hooks](#)

This is the set of functions that are implemented by all SCEW writers.

Typedefs

- typedef struct [scew_writer](#) [scew_writer](#)

This is the type declaration for SCEW writers.

Functions

- SCEW_API [scew_writer](#) * [scew_writer_create](#) ([scew_writer_hooks](#) const *hooks, void *data)
Creates a new SCEW writer with the given [scew_writer_hooks](#) implementation.
- SCEW_API void * [scew_writer_data](#) ([scew_writer](#) *writer)
Returns the reference to the internal data structure being used by the given writer.
- SCEW_API size_t [scew_writer_write](#) ([scew_writer](#) *writer, XML_Char const *buffer, size_t char_no)
Writes data from the given memory buffer to the specified writer.
- SCEW_API [scew_bool](#) [scew_writer_end](#) ([scew_writer](#) *writer)
Tells whether the given writer has reached its end.
- SCEW_API [scew_bool](#) [scew_writer_error](#) ([scew_writer](#) *writer)
Tells whether an error was found while sending data to the given writer.
- SCEW_API [scew_bool](#) [scew_writer_close](#) ([scew_writer](#) *writer)
Closes the given writer.
- SCEW_API void [scew_writer_free](#) ([scew_writer](#) *writer)
Frees the memory allocated by the given writer.

6.15.1 Detailed Description

SCEW writer common functions.

Author

Aleix Conchillo Flaque <aleix@member.fsf.org>

Date

Thu Sep 11, 2003 00:36

6.16 writer_buffer.h File Reference

SCEW writer functions for memory buffers.

```
#include "export.h"
#include "writer.h"
```

Functions

- SCEW_API [scew_writer](#) * [scew_writer_buffer_create](#) (XML_Char *buffer, size_t size)
Creates a new SCEW writer for the given memory buffer of the specified size.

6.16.1 Detailed Description

SCEW writer functions for memory buffers.

Author

Aleix Conchillo Flaque <aleix@member.fsf.org>

Date

Thu Nov 13, 2008 11:03

6.17 writer_file.h File Reference

SCEW writer functions for files.

```
#include "export.h"
#include "writer.h"
#include <stdio.h>
```

Functions

- SCEW_API [scew_writer](#) * [scew_writer_file_create](#) (char const *file_name)
Creates a new SCEW writer for the given file name.

- SCEW_API `scew_writer` * `scew_writer_fp_create` (FILE *file)
Creates a new SCEW writer for the given file stream.

6.17.1 Detailed Description

SCEW writer functions for files.

Author

Aleix Conchillo Flaque <aleix@member.fsf.org>

Date

Thu Nov 13, 2008 11:01

Index

- Accessors, [10](#), [17](#), [32](#), [44](#)
- Allocation, [8](#), [13](#), [32](#), [39](#), [46](#), [62](#)
- attribute.h, [81](#)
- Attributes, [7](#), [23](#)

- bool.h, [82](#)

- close
 - scew_reader_hooks, [78](#)
 - scew_writer_hooks, [79](#)
- Codes and descriptions, [27](#)
- Comparison, [9](#), [16](#), [63](#)
- Contents, [68](#)

- element.h, [83](#)
- Elements, [12](#)
- end
 - scew_reader_hooks, [77](#)
 - scew_writer_hooks, [78](#)
- error
 - scew_reader_hooks, [77](#)
 - scew_writer_hooks, [79](#)
- error.h, [86](#)
- Errors, [27](#)
- Expat errors, [29](#)
- export.h, [87](#)

- Files, [56](#), [75](#)
- free
 - scew_reader_hooks, [78](#)
 - scew_writer_hooks, [79](#)

- Hierarchy, [11](#), [19](#)

- Input/Output, [45](#)

- list.h, [87](#)
- Lists, [30](#)
- Load, [41](#)

- Memory, [55](#), [74](#)
- Modifiers, [33](#)

- Output, [48](#)

- Parser, [39](#)
- parser.h, [89](#)

- Printer, [46](#)
- printer.h, [90](#)
- Properties, [47](#), [65](#)

- read
 - scew_reader_hooks, [77](#)
- reader.h, [91](#)
- reader_buffer.h, [92](#)
- reader_file.h, [93](#)
- Readers, [51](#)

- scew.h, [94](#)
- scew_error_expat
 - SCEWErrorCodes, [28](#)
- scew_error_hook
 - SCEWErrorCodes, [28](#)
- scew_error_internal
 - SCEWErrorCodes, [28](#)
- scew_error_io
 - SCEWErrorCodes, [28](#)
- scew_error_no_memory
 - SCEWErrorCodes, [28](#)
- scew_error_none
 - SCEWErrorCodes, [28](#)
- scew_error_unknown
 - SCEWErrorCodes, [28](#)
- scew_tree_standalone_no
 - SCEWTreeProp, [66](#)
- scew_tree_standalone_unknown
 - SCEWTreeProp, [66](#)
- scew_tree_standalone_yes
 - SCEWTreeProp, [66](#)
- scew_attribute_compare
 - SCEWAttributeCompare, [9](#)
- scew_attribute_copy
 - SCEWAttributeAlloc, [8](#)
- scew_attribute_create
 - SCEWAttributeAlloc, [8](#)
- scew_attribute_free
 - SCEWAttributeAlloc, [9](#)
- scew_attribute_name
 - SCEWAttributeAcc, [10](#)
- scew_attribute_parent
 - SCEWAttributeHier, [12](#)
- scew_attribute_set_name

- SCEWAttributeAcc, 11
- scew_attribute_set_value
 - SCEWAttributeAcc, 11
- scew_attribute_value
 - SCEWAttributeAcc, 10
- scew_cmp_hook
 - SCEWList, 31
- scew_element_add
 - SCEWElementHier, 21
- scew_element_add_attribute
 - SCEWElementAttr, 25
- scew_element_add_attribute_pair
 - SCEWElementAttr, 25
- scew_element_add_element
 - SCEWElementHier, 21
- scew_element_add_pair
 - SCEWElementHier, 21
- scew_element_attribute_by_index
 - SCEWElementAttr, 25
- scew_element_attribute_by_name
 - SCEWElementAttr, 25
- scew_element_attribute_count
 - SCEWElementAttr, 24
- scew_element_attributes
 - SCEWElementAttr, 24
- scew_element_by_index
 - SCEWElementSearch, 15
- scew_element_by_name
 - SCEWElementSearch, 15
- scew_element_children
 - SCEWElementHier, 21
- scew_element_cmp_hook
 - SCEWElementCompare, 16
- scew_element_compare
 - SCEWElementCompare, 16
- scew_element_contents
 - SCEWElementAcc, 18
- scew_element_copy
 - SCEWElementAlloc, 14
- scew_element_count
 - SCEWElementHier, 20
- scew_element_create
 - SCEWElementAlloc, 14
- scew_element_delete_all
 - SCEWElementHier, 22
- scew_element_delete_all_by_name
 - SCEWElementHier, 22
- scew_element_delete_attribute
 - SCEWElementAttr, 26
- scew_element_delete_attribute_all
 - SCEWElementAttr, 26
- scew_element_delete_attribute_by_index
 - SCEWElementAttr, 26
- scew_element_delete_attribute_by_name
 - SCEWElementAttr, 26
- scew_element_delete_by_index
 - SCEWElementHier, 22
- scew_element_delete_by_name
 - SCEWElementHier, 22
- scew_element_detach
 - SCEWElementHier, 23
- scew_element_free
 - SCEWElementAlloc, 14
- scew_element_free_contents
 - SCEWElementAcc, 19
- scew_element_list_by_name
 - SCEWElementSearch, 15
- scew_element_name
 - SCEWElementAcc, 18
- scew_element_parent
 - SCEWElementHier, 20
- scew_element_set_contents
 - SCEWElementAcc, 19
- scew_element_set_name
 - SCEWElementAcc, 18
- scew_error
 - SCEWErrorCodes, 28
- scew_error_code
 - SCEWErrorCodes, 28
- scew_error_extern_code
 - SCEWErrorExtern, 29
- scew_error_extern_column
 - SCEWErrorExtern, 30
- scew_error_extern_line
 - SCEWErrorExtern, 30
- scew_error_extern_string
 - SCEWErrorExtern, 29
- scew_error_string
 - SCEWErrorCodes, 28
- scew_isempty
 - SCEWString, 60
- scew_list_append
 - SCEWListMod, 34
- scew_list_create
 - SCEWListAlloc, 32
- scew_list_data
 - SCEWListAcc, 33
- scew_list_delete
 - SCEWListMod, 34
- scew_list_delete_item
 - SCEWListMod, 35
- scew_list_find
 - SCEWListSearch, 38
- scew_list_find_custom
 - SCEWListSearch, 38
- scew_list_first
 - SCEWListTrav, 36
- scew_list_foreach

- SCEWListTrav, 37
- scew_list_free
 - SCEWListAlloc, 32
- scew_list_hook
 - SCEWList, 31
- scew_list_index
 - SCEWListSearch, 38
- scew_list_last
 - SCEWListTrav, 36
- scew_list_next
 - SCEWListTrav, 36
- scew_list_prepend
 - SCEWListMod, 34
- scew_list_previous
 - SCEWListTrav, 36
- scew_list_size
 - SCEWListAcc, 33
- scew_memcpy
 - SCEWString, 60
- scew_memmove
 - SCEWString, 60
- scew_parser_create
 - SCEWParserAlloc, 40
- scew_parser_extern
 - SCEWParserAcc, 45
- scew_parser_free
 - SCEWParserAlloc, 40
- scew_parser_ignore_whitespace
 - SCEWParserLoad, 44
- scew_parser_load
 - SCEWParserLoad, 42
- scew_parser_load_hook
 - SCEWParserLoad, 41
- scew_parser_load_stream
 - SCEWParserLoad, 42
- scew_parser_namespace_create
 - SCEWParserAlloc, 40
- scew_parser_reset
 - SCEWParserLoad, 43
- scew_parser_set_element_hook
 - SCEWParserLoad, 43
- scew_parser_set_tree_hook
 - SCEWParserLoad, 43
- scew_printer_create
 - SCEWPrinterAlloc, 47
- scew_printer_free
 - SCEWPrinterAlloc, 47
- scew_printer_print_attribute
 - SCEWPrinterOutput, 51
- scew_printer_print_element
 - SCEWPrinterOutput, 50
- scew_printer_print_element_attributes
 - SCEWPrinterOutput, 50
- scew_printer_print_element_children
 - SCEWPrinterOutput, 50
- scew_printer_print_tree
 - SCEWPrinterOutput, 50
- scew_printer_set_indentation
 - SCEWPrinterProp, 48
- scew_printer_set_indented
 - SCEWPrinterProp, 48
- scew_printer_set_writer
 - SCEWPrinterOutput, 49
- scew_reader_buffer_create
 - SCEWReaderMemory, 55
- scew_reader_close
 - SCEWReader, 54
- scew_reader_create
 - SCEWReader, 52
- scew_reader_data
 - SCEWReader, 53
- scew_reader_end
 - SCEWReader, 54
- scew_reader_error
 - SCEWReader, 54
- scew_reader_file_create
 - SCEWReaderFile, 56
- scew_reader_fp_create
 - SCEWReaderFile, 57
- scew_reader_free
 - SCEWReader, 55
- scew_reader_hooks, 77
 - close, 78
 - end, 77
 - error, 77
 - free, 78
 - read, 77
- scew_reader_read
 - SCEWReader, 53
- scew_strcmp
 - SCEWString, 60
- scew_strdup
 - SCEWString, 60
- scew_strescape
 - SCEWString, 61
- scew_strtrim
 - SCEWString, 60
- scew_tree_cmp_hook
 - SCEWTreeCompare, 64
- scew_tree_compare
 - SCEWTreeCompare, 64
- scew_tree_copy
 - SCEWTreeAlloc, 63
- scew_tree_create
 - SCEWTreeAlloc, 63
- scew_tree_free
 - SCEWTreeAlloc, 63
- scew_tree_root

- SCEWTreeContent, 69
- scew_tree_set_root
 - SCEWTreeContent, 69
- scew_tree_set_root_element
 - SCEWTreeContent, 69
- scew_tree_set_xml_encoding
 - SCEWTreeProp, 67
- scew_tree_set_xml_preamble
 - SCEWTreeContent, 70
- scew_tree_set_xml_standalone
 - SCEWTreeProp, 68
- scew_tree_set_xml_version
 - SCEWTreeProp, 66
- scew_tree_standalone
 - SCEWTreeProp, 66
- scew_tree_xml_encoding
 - SCEWTreeProp, 66
- scew_tree_xml_preamble
 - SCEWTreeContent, 70
- scew_tree_xml_standalone
 - SCEWTreeProp, 67
- scew_tree_xml_version
 - SCEWTreeProp, 66
- scew_writer_buffer_create
 - SCEWWriterMemory, 75
- scew_writer_close
 - SCEWWriter, 74
- scew_writer_create
 - SCEWWriter, 72
- scew_writer_data
 - SCEWWriter, 72
- scew_writer_end
 - SCEWWriter, 73
- scew_writer_error
 - SCEWWriter, 73
- scew_writer_file_create
 - SCEWWriterFile, 76
- scew_writer_fp_create
 - SCEWWriterFile, 76
- scew_writer_free
 - SCEWWriter, 74
- scew_writer_hooks, 78
 - close, 79
 - end, 78
 - error, 79
 - free, 79
 - write, 78
- scew_writer_write
 - SCEWWriter, 73
- SCEWAttributeAcc
 - scew_attribute_name, 10
 - scew_attribute_set_name, 11
 - scew_attribute_set_value, 11
 - scew_attribute_value, 10
- SCEWAttributeAlloc
 - scew_attribute_copy, 8
 - scew_attribute_create, 8
 - scew_attribute_free, 9
- SCEWAttributeCompare
 - scew_attribute_compare, 9
- SCEWAttributeHier
 - scew_attribute_parent, 12
- SCEWElementAcc
 - scew_element_contents, 18
 - scew_element_free_contents, 19
 - scew_element_name, 18
 - scew_element_set_contents, 19
 - scew_element_set_name, 18
- SCEWElementAlloc
 - scew_element_copy, 14
 - scew_element_create, 14
 - scew_element_free, 14
- SCEWElementAttr
 - scew_element_add_attribute, 25
 - scew_element_add_attribute_pair, 25
 - scew_element_attribute_by_index, 25
 - scew_element_attribute_by_name, 25
 - scew_element_attribute_count, 24
 - scew_element_attributes, 24
 - scew_element_delete_attribute, 26
 - scew_element_delete_attribute_all, 26
 - scew_element_delete_attribute_by_index, 26
 - scew_element_delete_attribute_by_name, 26
- SCEWElementCompare
 - scew_element_cmp_hook, 16
 - scew_element_compare, 16
- SCEWElementHier
 - scew_element_add, 21
 - scew_element_add_element, 21
 - scew_element_add_pair, 21
 - scew_element_children, 21
 - scew_element_count, 20
 - scew_element_delete_all, 22
 - scew_element_delete_all_by_name, 22
 - scew_element_delete_by_index, 22
 - scew_element_delete_by_name, 22
 - scew_element_detach, 23
 - scew_element_parent, 20
- SCEWElementSearch
 - scew_element_by_index, 15
 - scew_element_by_name, 15
 - scew_element_list_by_name, 15
- SCEWErrorCodes
 - scew_error_expat, 28
 - scew_error_hook, 28
 - scew_error_internal, 28
 - scew_error_io, 28
 - scew_error_no_memory, 28

- scew_error_none, 28
- scew_error_unknown, 28
- scew_error, 28
- scew_error_code, 28
- scew_error_string, 28
- SCEWErrorExpat
 - scew_error_expat_code, 29
 - scew_error_expat_column, 30
 - scew_error_expat_line, 30
 - scew_error_expat_string, 29
- SCEWList
 - scew_cmp_hook, 31
 - scew_list_hook, 31
- SCEWListAcc
 - scew_list_data, 33
 - scew_list_size, 33
- SCEWListAlloc
 - scew_list_create, 32
 - scew_list_free, 32
- SCEWListMod
 - scew_list_append, 34
 - scew_list_delete, 34
 - scew_list_delete_item, 35
 - scew_list_prepend, 34
- SCEWListSearch
 - scew_list_find, 38
 - scew_list_find_custom, 38
 - scew_list_index, 38
- SCEWListTrav
 - scew_list_first, 36
 - scew_list_foreach, 37
 - scew_list_last, 36
 - scew_list_next, 36
 - scew_list_previous, 36
- SCEWParserAcc
 - scew_parser_expat, 45
- SCEWParserAlloc
 - scew_parser_create, 40
 - scew_parser_free, 40
 - scew_parser_namespace_create, 40
- SCEWParserLoad
 - scew_parser_ignore_whitespace, 44
 - scew_parser_load, 42
 - scew_parser_load_hook, 41
 - scew_parser_load_stream, 42
 - scew_parser_reset, 43
 - scew_parser_set_element_hook, 43
 - scew_parser_set_tree_hook, 43
- SCEWPrinterAlloc
 - scew_printer_create, 47
 - scew_printer_free, 47
- SCEWPrinterOutput
 - scew_printer_print_attribute, 51
 - scew_printer_print_element, 50
- scew_printer_print_element_attributes, 50
- scew_printer_print_element_children, 50
- scew_printer_print_tree, 50
- scew_printer_set_writer, 49
- SCEWPrinterProp
 - scew_printer_set_indentation, 48
 - scew_printer_set_indented, 48
- SCEWReader
 - scew_reader_close, 54
 - scew_reader_create, 52
 - scew_reader_data, 53
 - scew_reader_end, 54
 - scew_reader_error, 54
 - scew_reader_free, 55
 - scew_reader_read, 53
- SCEWReaderFile
 - scew_reader_file_create, 56
 - scew_reader_fp_create, 57
- SCEWReaderMemory
 - scew_reader_buffer_create, 55
- SCEWString
 - scew_isempty, 60
 - scew_memcpy, 60
 - scew_memmove, 60
 - scew_strcmp, 60
 - scew_strdup, 60
 - scew_strescape, 61
 - scew_strtrim, 60
- SCEWTreeAlloc
 - scew_tree_copy, 63
 - scew_tree_create, 63
 - scew_tree_free, 63
- SCEWTreeCompare
 - scew_tree_cmp_hook, 64
 - scew_tree_compare, 64
- SCEWTreeContent
 - scew_tree_root, 69
 - scew_tree_set_root, 69
 - scew_tree_set_root_element, 69
 - scew_tree_set_xml_preamble, 70
 - scew_tree_xml_preamble, 70
- SCEWTreeProp
 - scew_tree_standalone_no, 66
 - scew_tree_standalone_unknown, 66
 - scew_tree_standalone_yes, 66
 - scew_tree_set_xml_encoding, 67
 - scew_tree_set_xml_standalone, 68
 - scew_tree_set_xml_version, 66
 - scew_tree_standalone, 66
 - scew_tree_xml_encoding, 66
 - scew_tree_xml_standalone, 67
 - scew_tree_xml_version, 66
- SCEWWriter
 - scew_writer_close, 74

- scew_writer_create, [72](#)
- scew_writer_data, [72](#)
- scew_writer_end, [73](#)
- scew_writer_error, [73](#)
- scew_writer_free, [74](#)
- scew_writer_write, [73](#)
- SCEWWriterFile
 - scew_writer_file_create, [76](#)
 - scew_writer_fp_create, [76](#)
- SCEWWriterMemory
 - scew_writer_buffer_create, [75](#)
- Search, [37](#)
- Search and iteration, [14](#)
- str.h, [94](#)

- Text utilities, [57](#)
- Traverse, [35](#)
- tree.h, [97](#)
- Trees, [61](#)

- write
 - scew_writer_hooks, [78](#)
- writer.h, [99](#)
- writer_buffer.h, [100](#)
- writer_file.h, [100](#)
- Writers, [71](#)