



SolarCapture C Bindings User Guide

SF-115721-CD , Issue 3

2019/06/25 17:01:26

Solarflare Communications Inc

SolarCapture C Bindings User Guide

Copyright © 2019 SOLARFLARE Communications, Inc. All rights reserved.

The software and hardware as applicable (the “Product”) described in this document, and this document, are protected by copyright laws, patents and other intellectual property laws and international treaties. The Product described in this document is provided pursuant to a license agreement, evaluation agreement and/or non-disclosure agreement. The Product may be used only in accordance with the terms of such agreement. The software as applicable may be copied only in accordance with the terms of such agreement.

The furnishing of this document to you does not give you any rights or licenses, express or implied, by estoppel or otherwise, with respect to any such Product, or any copyrights, patents or other intellectual property rights covering such Product, and this document does not contain or represent any commitment of any kind on the part of SOLARFLARE Communications, Inc. or its affiliates.

The only warranties granted by SOLARFLARE Communications, Inc. or its affiliates in connection with the Product described in this document are those expressly set forth in the license agreement, evaluation agreement and/or non-disclosure agreement pursuant to which the Product is provided. EXCEPT AS EXPRESSLY SET FORTH IN SUCH AGREEMENT, NEITHER SOLARFLARE COMMUNICATIONS, INC. NOR ITS AFFILIATES MAKE ANY REPRESENTATIONS OR WARRANTIES OF ANY KIND (EXPRESS OR IMPLIED) REGARDING THE PRODUCT OR THIS DOCUMENTATION AND HEREBY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT, AND ANY WARRANTIES THAT MAY ARISE FROM COURSE OF DEALING, COURSE OF PERFORMANCE OR USAGE OF TRADE. Unless otherwise expressly set forth in such agreement, to the extent allowed by applicable law (a) in no event shall SOLARFLARE Communications, Inc. or its affiliates have any liability under any legal theory for any loss of revenues or profits, loss of use or data, or business interruptions, or for any indirect, special, incidental or consequential damages, even if advised of the possibility of such damages; and (b) the total liability of SOLARFLARE Communications, Inc. or its affiliates arising from or relating to such agreement or the use of this document shall not exceed the amount received by SOLARFLARE Communications, Inc. or its affiliates for that copy of the Product or this document which is the subject of such liability.

The Product is not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

SF-115721-CD

Last Revised: 62019

Issue 3

Contents

1	Introduction	1
2	Embedding SolarCapture	3
2.1	Sessions — struct <code>sc_session</code>	3
2.2	Attributes — struct <code>sc_attr</code>	3
2.3	Threads — struct <code>sc_thread</code>	4
2.4	Virtual Interfaces — struct <code>sc_vi</code>	4
2.5	Nodes — struct <code>sc_node</code>	4
2.6	Mailboxes — struct <code>sc_mailbox</code>	5
2.7	Built-In Nodes	6
3	Extending SolarCapture	7
3.1	Node factories — struct <code>sc_node_factory</code>	7
3.2	Node types — struct <code>sc_node_type</code>	8
3.3	Node libraries	8
3.4	Insert a user-defined node between capture and <code>sc_writer</code>	9
4	Examples	11
4.1	<code>c_api</code>	11
4.2	<code>c_api_export</code>	11
4.3	<code>extensions_api</code>	12
4.4	<code>forwarding</code>	12
4.4.1	<code>trivial_bridge.py</code>	12
4.4.2	<code>bpf_firewall.py</code>	12
4.4.3	<code>reflect.py</code>	12
4.5	<code>py_api</code>	13
4.6	<code>unmanaged</code>	13
4.6.1	<code>unmanaged_mailbox.c</code>	13
4.6.2	<code>unmanaged_thread.c</code>	13

5	Nodes	15
5.1	sc_append_to_list Node Reference	16
5.2	sc_arista_ts Node Reference	17
5.2.1	sc_arista_ts, switch_model=7150	17
5.2.2	sc_arista_ts, switch_model=7280, ts_format=64bit	19
5.2.3	sc_arista_ts, switch_model=7280, ts_format=48bit	21
5.3	sc_batch_limiter Node Reference	22
5.4	sc_cpacket_encap Node Reference	22
5.5	sc_cpacket_ts Node Reference	23
5.6	sc_delay_line Node Reference	23
5.7	sc_exit Node Reference	24
5.8	sc_fd_reader Node Reference	25
5.9	sc_fd_writer Node Reference	26
5.10	sc_filter Node Reference	26
5.11	sc_flow_balancer Node Reference	27
5.12	sc_injector Node Reference	29
5.13	sc_line_reader Node Reference	29
5.14	sc_merge_sorter Node Reference	30
5.15	sc_no_op Node Reference	30
5.16	sc_pacer Node Reference	31
5.17	sc_pass_n Node Reference	31
5.18	sc_pcap_packer Node Reference	32
5.19	sc_pool_forwarder Node Reference	33
5.20	sc_ps_packer Node Reference	33
5.21	sc_ps_unpacker Node Reference	34
5.22	sc_range_filter Node Reference	34
5.23	sc_rate_monitor Node Reference	35
5.24	sc_reader Node Reference	36
5.25	sc_repeater Node Reference	37

5.26	sc_rr_gather Node Reference	37
5.27	sc_rr_spreader Node Reference	38
5.28	sc_rt_pacer Node Reference	38
5.29	sc_shm_broadcast Node Reference	39
5.30	sc_shm_export Node Reference	41
5.31	sc_shm_import Node Reference	42
5.32	sc_sim_work Node Reference	42
5.33	sc_snap Node Reference	43
5.34	sc_subnode_helper Node Reference	43
5.35	sc_tap Node Reference	44
5.36	sc_timestamp_filter Node Reference	45
5.37	sc_token_bucket_shaper Node Reference	46
5.38	sc_tracer Node Reference	47
5.39	sc_ts_adjust Node Reference	47
5.40	sc_tunnel Node Reference	48
5.41	sc_tuntap Node Reference	49
5.42	sc_vi_node Node Reference	49
5.43	sc_vss Node Reference	50
5.44	sc_writer Node Reference	50
6	Statistics	53
6.1	sc_arista_ts Statistics Reference	53
6.1.1	sc_arista_ts, switch_model=7150	53
6.1.2	sc_arista_ts, switch_model=7280, ts_format=64bit	54
6.1.3	sc_arista_ts, switch_model=7280, ts_format=48bit	54
6.2	sc_batch_limiter Statistics Reference	55
6.3	sc_filter Statistics Reference	55
6.4	sc_flow_balancer Statistics Reference	55
6.5	sc_pcap_packer Statistics Reference	55
6.6	sc_rate_monitor Statistics Reference	56
6.7	sc_shm Statistics Reference	56
6.8	sc_subnode_helper Statistics Reference	56
6.9	sc_writer Statistics Reference	56

7	Data Structure Index	57
7.1	Data Structures	57
8	File Index	59
8.1	File List	59
9	Data Structure Documentation	61
9.1	sc_append_to_list Struct Reference	61
9.1.1	Detailed Description	61
9.1.2	Field Documentation	61
9.1.2.1	append_to	61
9.1.2.2	free_link	62
9.1.2.3	links	62
9.1.2.4	n_links	62
9.2	sc_arg Struct Reference	62
9.2.1	Detailed Description	62
9.2.2	Field Documentation	63
9.2.2.1	name	63
9.2.2.2	type	63
9.2.2.3	val	63
9.3	sc_attr Struct Reference	63
9.3.1	Detailed Description	63
9.4	sc_callback Struct Reference	64
9.4.1	Detailed Description	64
9.4.2	Field Documentation	64
9.4.2.1	cb_handler_fn	64
9.4.2.2	cb_link	64
9.4.2.3	cb_private	64
9.5	sc_dlist Struct Reference	65
9.5.1	Detailed Description	65

9.5.2	Field Documentation	65
9.5.2.1	next	65
9.5.2.2	prev	65
9.6	sc_hash_table Struct Reference	65
9.6.1	Detailed Description	66
9.7	sc_iovec_ptr Struct Reference	66
9.7.1	Detailed Description	66
9.7.2	Field Documentation	66
9.7.2.1	io	66
9.7.2.2	iov	66
9.7.2.3	iovlen	67
9.8	sc_node Struct Reference	67
9.8.1	Detailed Description	67
9.8.2	Field Documentation	67
9.8.2.1	nd_name	67
9.8.2.2	nd_private	67
9.8.2.3	nd_type	68
9.9	sc_node_factory Struct Reference	68
9.9.1	Detailed Description	68
9.9.2	Field Documentation	68
9.9.2.1	nf_init_fn	68
9.9.2.2	nf_name	68
9.9.2.3	nf_node_api_ver	69
9.9.2.4	nf_private	69
9.9.2.5	nf_reserved	69
9.9.2.6	nf_source_file	69
9.10	sc_node_link Struct Reference	69
9.10.1	Detailed Description	69
9.10.2	Field Documentation	70

9.10.2.1	name	70
9.11	sc_node_type Struct Reference	70
9.11.1	Detailed Description	70
9.11.2	Field Documentation	70
9.11.2.1	nt_add_link_fn	70
9.11.2.2	nt_end_of_stream_fn	71
9.11.2.3	nt_name	71
9.11.2.4	nt_pkts_fn	71
9.11.2.5	nt_prep_fn	71
9.11.2.6	nt_private	71
9.11.2.7	nt_select_subnode_fn	71
9.12	sc_object Struct Reference	71
9.12.1	Detailed Description	72
9.13	sc_packed_packet Struct Reference	72
9.13.1	Detailed Description	72
9.13.2	Field Documentation	73
9.13.2.1	ps_cap_len	73
9.13.2.2	ps_flags	73
9.13.2.3	ps_next_offset	73
9.13.2.4	ps_orig_len	73
9.13.2.5	ps_pkt_start_offset	73
9.13.2.6	ps_ts_nsec	73
9.13.2.7	ps_ts_sec	74
9.14	sc_packet Struct Reference	74
9.14.1	Detailed Description	74
9.14.2	Field Documentation	75
9.14.2.1	flags	75
9.14.2.2	frags	75
9.14.2.3	frags_n	75

9.14.2.4	frags_tail	75
9.14.2.5	frame_len	75
9.14.2.6	iov	75
9.14.2.7	iovlen	75
9.14.2.8	metadata	76
9.14.2.9	next	76
9.14.2.10	reserved1	76
9.14.2.11	reserved2	76
9.14.2.12	ts_nsec	76
9.14.2.13	ts_sec	76
9.15	sc_packet_list Struct Reference	76
9.15.1	Detailed Description	77
9.15.2	Field Documentation	77
9.15.2.1	head	77
9.15.2.2	num_frags	77
9.15.2.3	num_pkts	77
9.15.2.4	tail	77
9.16	sc_pkt_predicate Struct Reference	78
9.16.1	Detailed Description	78
9.16.2	Field Documentation	78
9.16.2.1	pred_private	78
9.16.2.2	pred_test_fn	78
9.17	sc_session_error Struct Reference	78
9.17.1	Detailed Description	79
9.17.2	Field Documentation	79
9.17.2.1	err_errno	79
9.17.2.2	err_file	79
9.17.2.3	err_func	79
9.17.2.4	err_line	79

9.17.2.5	err_msg	80
9.18	sc_stream Struct Reference	80
9.18.1	Detailed Description	80
9.19	sc_subnode_helper Struct Reference	80
9.19.1	Detailed Description	81
9.19.2	Friends And Related Function Documentation	81
9.19.2.1	sc_sh_handle_backlog_fn	82
9.19.2.2	sc_sh_handle_end_of_stream_fn	82
9.19.2.3	sc_subnode_helper_from_node()	82
9.19.2.4	sc_subnode_helper_request_callback()	83
9.19.3	Field Documentation	83
9.19.3.1	sh_backlog	83
9.19.3.2	sh_free_link	83
9.19.3.3	sh_handle_backlog_fn	83
9.19.3.4	sh_handle_end_of_stream_fn	84
9.19.3.5	sh_links	84
9.19.3.6	sh_n_links	84
9.19.3.7	sh_node	84
9.19.3.8	sh_poll_backlog_ns	84
9.19.3.9	sh_pool	84
9.19.3.10	sh_pool_threshold	84
9.19.3.11	sh_private	85
9.20	sc_vi Struct Reference	85
9.20.1	Detailed Description	85

10 File Documentation	87
10.1 <code>append_to_list.h</code> File Reference	87
10.1.1 Detailed Description	87
10.2 <code>args.h</code> File Reference	87
10.2.1 Detailed Description	88
10.2.2 Enumeration Type Documentation	88
10.2.2.1 <code>sc_param_type</code>	88
10.2.3 Function Documentation	88
10.2.3.1 <code>SC_ARG_DBL()</code>	88
10.2.3.2 <code>SC_ARG_INT()</code>	89
10.2.3.3 <code>SC_ARG_OBJ()</code>	89
10.2.3.4 <code>SC_ARG_STR()</code>	89
10.3 <code>attr.h</code> File Reference	90
10.3.1 Detailed Description	90
10.3.2 Function Documentation	91
10.3.2.1 <code>sc_attr_alloc()</code>	91
10.3.2.2 <code>sc_attr_doc()</code>	91
10.3.2.3 <code>sc_attr_dup()</code>	91
10.3.2.4 <code>sc_attr_free()</code>	92
10.3.2.5 <code>sc_attr_from_object()</code>	92
10.3.2.6 <code>sc_attr_reset()</code>	92
10.3.2.7 <code>sc_attr_set_from_fmt()</code>	93
10.3.2.8 <code>sc_attr_set_from_str()</code>	93
10.3.2.9 <code>sc_attr_set_int()</code>	94
10.3.2.10 <code>sc_attr_set_str()</code>	94
10.3.2.11 <code>sc_attr_to_object()</code>	94
10.4 <code>declare_types.h</code> File Reference	95
10.4.1 Detailed Description	95
10.4.2 Macro Definition Documentation	96

10.4.2.1	ST_CONSTANT	96
10.4.2.2	ST_FIELD	96
10.4.2.3	ST_FIELD_STR	97
10.4.2.4	ST_STRUCT	97
10.5	dlist.h File Reference	97
10.5.1	Detailed Description	98
10.5.2	Macro Definition Documentation	99
10.5.2.1	SC_CONTAINER	99
10.5.2.2	SC_DLIST_FOR_EACH_OBJ	100
10.5.2.3	SC_DLIST_FOR_EACH_OBJ_SAFE	100
10.5.3	Function Documentation	101
10.5.3.1	sc_dlist_init()	101
10.5.3.2	sc_dlist_pop_head()	102
10.5.3.3	sc_dlist_pop_tail()	102
10.5.3.4	sc_dlist_push_head()	102
10.5.3.5	sc_dlist_push_tail()	103
10.5.3.6	sc_dlist_rehome()	103
10.5.3.7	sc_dlist_remove()	103
10.6	ethernet.h File Reference	104
10.6.1	Detailed Description	104
10.6.2	Macro Definition Documentation	104
10.6.2.1	SC_8021Q_VID_MASK	104
10.6.2.2	SC_ETHERTYPE_8021Q	104
10.6.2.3	SC_ETHERTYPE_8021QinQ	104
10.7	event.h File Reference	104
10.7.1	Detailed Description	105
10.7.2	Typedef Documentation	105
10.7.2.1	sc_callback_handler_fn	105
10.7.3	Function Documentation	106

10.7.3.1	sc_callback_alloc()	106
10.7.3.2	sc_callback_alloc2()	106
10.7.3.3	sc_callback_free()	107
10.7.3.4	sc_callback_is_active()	107
10.7.3.5	sc_callback_on_idle()	107
10.7.3.6	sc_callback_remove()	108
10.7.3.7	sc_callback_set_description()	108
10.7.3.8	sc_epoll_ctl()	108
10.8	ext_node.h File Reference	109
10.8.1	Detailed Description	111
10.8.2	Macro Definition Documentation	111
10.8.2.1	sc_node_fwd_error	111
10.8.2.2	sc_node_set_error	111
10.8.2.3	sc_node_set_errorv	112
10.8.3	Typedef Documentation	112
10.8.3.1	sc_node_add_link_fn	112
10.8.3.2	sc_node_end_of_stream_fn	113
10.8.3.3	sc_node_init_fn	113
10.8.3.4	sc_node_pkts_fn	114
10.8.3.5	sc_node_prep_fn	114
10.8.3.6	sc_node_select_subnode_fn	114
10.8.4	Function Documentation	115
10.8.4.1	sc_forward()	115
10.8.4.2	sc_forward2()	115
10.8.4.3	sc_forward_list()	116
10.8.4.4	sc_forward_list2()	116
10.8.4.5	sc_node_export_state()	116
10.8.4.6	sc_node_init_get_arg_dbl()	117
10.8.4.7	sc_node_init_get_arg_int()	117

10.8.4.8	<code>sc_node_init_get_arg_int64()</code>	118
10.8.4.9	<code>sc_node_init_get_arg_obj()</code>	118
10.8.4.10	<code>sc_node_init_get_arg_str()</code>	119
10.8.4.11	<code>sc_node_link_end_of_stream()</code>	119
10.8.4.12	<code>sc_node_link_end_of_stream2()</code>	120
10.8.4.13	<code>sc_node_prep_check_links()</code>	120
10.8.4.14	<code>sc_node_prep_does_not_forward()</code>	120
10.8.4.15	<code>sc_node_prep_get_link()</code>	121
10.8.4.16	<code>sc_node_prep_get_link_or_free()</code>	121
10.8.4.17	<code>sc_node_prep_get_pool()</code>	122
10.8.4.18	<code>sc_node_prep_link_forwards_from_node()</code>	122
10.8.4.19	<code>sc_node_type_alloc()</code>	123
10.9	<code>ext_packet.h</code> File Reference	123
10.9.1	Detailed Description	124
10.9.2	Macro Definition Documentation	124
10.9.2.1	<code>SC_MEMBER_OFFSET</code>	124
10.9.2.2	<code>SC_MEMBER_SIZE</code>	124
10.9.3	Function Documentation	125
10.9.3.1	<code>sc_packet_bytes()</code>	125
10.9.3.2	<code>sc_packet_frags_tail()</code>	125
10.9.3.3	<code>sc_packet_prefetch_r()</code>	125
10.9.3.4	<code>sc_packet_prefetch_rw()</code>	126
10.9.3.5	<code>sc_packet_timespec()</code>	126
10.10	<code>ext_packet_list.h</code> File Reference	126
10.10.1	Detailed Description	127
10.10.2	Function Documentation	127
10.10.2.1	<code>sc_packet_list_append()</code>	127
10.10.2.2	<code>sc_packet_list_append_list()</code>	127
10.10.2.3	<code>sc_packet_list_finalise()</code>	128

10.10.2.4	sc_packet_list_init()	128
10.10.2.5	sc_packet_list_is_empty()	128
10.10.2.6	sc_packet_list_pop_head()	128
10.10.2.7	sc_packet_list_push_head()	129
10.10.2.8	sc_packet_list_tail()	129
10.11	hash_table.h File Reference	129
10.11.1	Detailed Description	130
10.11.2	Function Documentation	130
10.11.2.1	sc_hash_table_alloc()	130
10.11.2.2	sc_hash_table_clear()	131
10.11.2.3	sc_hash_table_del()	131
10.11.2.4	sc_hash_table_del_val()	131
10.11.2.5	sc_hash_table_free()	132
10.11.2.6	sc_hash_table_get()	132
10.11.2.7	sc_hash_table_get_next_entry()	133
10.11.2.8	sc_hash_table_grow()	133
10.11.2.9	sc_hash_table_key_size()	133
10.11.2.10	sc_hash_table_num_entries()	134
10.11.2.11	sc_hash_table_val_size()	134
10.11.2.12	sc_hash_table_val_to_key()	134
10.12	iovec.h File Reference	135
10.12.1	Detailed Description	136
10.12.2	Function Documentation	136
10.12.2.1	sc_iovec_copy_from_end()	136
10.12.2.2	sc_iovec_copy_from_end_offset()	136
10.12.2.3	sc_iovec_copy_to_end_offset()	137
10.12.2.4	sc_iovec_ptr_bytes()	137
10.12.2.5	sc_iovec_ptr_copy_out()	137
10.12.2.6	sc_iovec_ptr_find_chr()	138

10.12.2.7	sc_iovec_ptr_init()	138
10.12.2.8	sc_iovec_ptr_init_buf()	138
10.12.2.9	sc_iovec_ptr_init_packet()	139
10.12.2.10	sc_iovec_ptr_skip()	139
10.12.2.11	sc_iovec_trim_end()	139
10.13	ip.h File Reference	140
10.13.1	Detailed Description	140
10.13.2	Macro Definition Documentation	140
10.13.2.1	SC_IP4_FRAG_DONT	140
10.13.2.2	SC_IP4_FRAG_MORE	140
10.13.2.3	SC_IP4_OFFSET_MASK	141
10.13.2.4	SC_TCP_ACK	141
10.13.2.5	SC_TCP_FIN	141
10.13.2.6	SC_TCP_PSH	141
10.13.2.7	SC_TCP_RST	141
10.13.2.8	SC_TCP_SYN	141
10.13.2.9	SC_TCP_URG	141
10.14	mailbox.h File Reference	142
10.14.1	Detailed Description	142
10.14.2	Function Documentation	142
10.14.2.1	sc_mailbox_alloc()	142
10.14.2.2	sc_mailbox_connect()	143
10.14.2.3	sc_mailbox_get_send_node()	143
10.14.2.4	sc_mailbox_poll()	143
10.14.2.5	sc_mailbox_send()	144
10.14.2.6	sc_mailbox_send_list()	144
10.14.2.7	sc_mailbox_set_rcv()	144
10.15	misc.h File Reference	145
10.15.1	Detailed Description	145

10.15.2 Function Documentation	145
10.15.2.1 sc_join_mcast_group()	145
10.16node.h File Reference	145
10.16.1 Detailed Description	146
10.16.2 Function Documentation	146
10.16.2.1 sc_node_add_info_int()	146
10.16.2.2 sc_node_add_info_str()	147
10.16.2.3 sc_node_add_link()	147
10.16.2.4 sc_node_alloc()	148
10.16.2.5 sc_node_alloc_from_str()	149
10.16.2.6 sc_node_alloc_named()	150
10.16.2.7 sc_node_factory_lookup()	151
10.16.2.8 sc_node_from_object()	152
10.16.2.9 sc_node_get_thread()	152
10.16.2.10sc_node_to_object()	153
10.17object.h File Reference	153
10.17.1 Detailed Description	154
10.17.2 Enumeration Type Documentation	154
10.17.2.1 sc_object_type	154
10.17.3 Function Documentation	154
10.17.3.1 sc_object_type()	154
10.17.3.2 sc_opaque_alloc()	154
10.17.3.3 sc_opaque_free()	155
10.17.3.4 sc_opaque_get_ptr()	155
10.17.3.5 sc_opaque_set_ptr()	155
10.18packed_stream.h File Reference	156
10.18.1 Detailed Description	157
10.18.2 Macro Definition Documentation	157
10.18.2.1 SC_PS_FLAG_BAD_FCS	157

10.18.2.2	SC_PS_FLAG_BAD_L3_CSUM	157
10.18.2.3	SC_PS_FLAG_BAD_L4_CSUM	157
10.18.2.4	SC_PS_FLAG_CLOCK_IN_SYNC	157
10.18.2.5	SC_PS_FLAG_CLOCK_SET	157
10.18.3	Function Documentation	157
10.18.3.1	sc_packed_packet_next()	157
10.18.3.2	sc_packed_packet_payload()	158
10.18.3.3	sc_packet_packed_end()	158
10.18.3.4	sc_packet_packed_first()	158
10.18.4	Variable Documentation	159
10.18.4.1	ps_cap_len	159
10.18.4.2	ps_flags	159
10.18.4.3	ps_next_offset	159
10.18.4.4	ps_orig_len	159
10.18.4.5	ps_pkt_start_offset	159
10.18.4.6	ps_ts_nsec	160
10.18.4.7	ps_ts_sec	160
10.19	pkt_pool.h File Reference	160
10.19.1	Detailed Description	160
10.19.2	Function Documentation	161
10.19.2.1	sc_packet_append_iovec_ptr()	161
10.19.2.2	sc_pool_duplicate_packed_packet()	161
10.19.2.3	sc_pool_duplicate_packet()	161
10.19.2.4	sc_pool_from_object()	162
10.19.2.5	sc_pool_get_buffer_size()	162
10.19.2.6	sc_pool_get_packets()	162
10.19.2.7	sc_pool_on_threshold()	163
10.19.2.8	sc_pool_return_packets()	163
10.19.2.9	sc_pool_set_refill_node()	164

10.19.2.10	sc_pool_to_object()	164
10.19.2.11	sc_pool_wraps_node()	164
10.20	predicate.h File Reference	165
10.20.1	Detailed Description	166
10.20.2	Function Documentation	166
10.20.2.1	sc_pkt_predicate_alloc()	166
10.20.2.2	sc_pkt_predicate_from_object()	166
10.20.2.3	sc_pkt_predicate_to_object()	166
10.21	session.h File Reference	167
10.21.1	Detailed Description	167
10.21.2	Function Documentation	168
10.21.2.1	sc_session_alloc()	168
10.21.2.2	sc_session_destroy()	168
10.21.2.3	sc_session_error_free()	168
10.21.2.4	sc_session_error_get()	169
10.21.2.5	sc_session_go()	169
10.21.2.6	sc_session_pause()	169
10.21.2.7	sc_session_prepare()	170
10.21.2.8	sc_session_run()	170
10.21.2.9	sc_session_stop()	171
10.22	stream.h File Reference	171
10.22.1	Detailed Description	172
10.22.2	Function Documentation	172
10.22.2.1	sc_stream_all()	172
10.22.2.2	sc_stream_alloc()	172
10.22.2.3	sc_stream_eth_dhost()	173
10.22.2.4	sc_stream_eth_shost()	173
10.22.2.5	sc_stream_eth_type()	173
10.22.2.6	sc_stream_eth_vlan_id()	174

10.22.2.7	sc_stream_free()	174
10.22.2.8	sc_stream_ip_dest_host()	174
10.22.2.9	sc_stream_ip_dest_hostport()	175
10.22.2.10	sc_stream_ip_dest_port()	175
10.22.2.11	sc_stream_ip_protocol()	176
10.22.2.12	sc_stream_ip_source_host()	176
10.22.2.13	sc_stream_ip_source_hostport()	176
10.22.2.14	sc_stream_ip_source_port()	177
10.22.2.15	sc_stream_mismatch()	177
10.22.2.16	sc_stream_reset()	177
10.22.2.17	sc_stream_set_str()	178
10.23	subnode_helper.h File Reference	179
10.23.1	Detailed Description	180
10.24	thread.h File Reference	180
10.24.1	Detailed Description	180
10.24.2	Function Documentation	180
10.24.2.1	sc_thread_alloc()	180
10.24.2.2	sc_thread_calloc()	181
10.24.2.3	sc_thread_calloc_aligned()	181
10.24.2.4	sc_thread_get_time()	182
10.24.2.5	sc_thread_mfree()	182
10.24.2.6	sc_thread_poll()	182
10.24.2.7	sc_thread_poll_timers()	183
10.24.2.8	sc_thread_waitable_fd_get()	183
10.24.2.9	sc_thread_waitable_fd_prime()	183
10.25	time.h File Reference	184
10.25.1	Detailed Description	184
10.25.2	Function Documentation	184
10.25.2.1	sc_ns_from_ms()	184

10.25.2.2	sc_ns_from_ts()	184
10.25.2.3	sc_ns_from_tv()	185
10.25.2.4	sc_ns_from_us()	185
10.25.2.5	sc_timer_expire_after_ns()	185
10.25.2.6	sc_timer_expire_at()	186
10.25.2.7	sc_timer_get_expiry_time()	186
10.25.2.8	sc_timer_push_back_ns()	186
10.26	vi.h File Reference	187
10.26.1	Detailed Description	187
10.26.2	Function Documentation	188
10.26.2.1	sc_vi_add_stream()	188
10.26.2.2	sc_vi_alloc()	188
10.26.2.3	sc_vi_alloc_from_group()	188
10.26.2.4	sc_vi_get_interface_name()	189
10.26.2.5	sc_vi_get_thread()	189
10.26.2.6	sc_vi_group_add_stream()	189
10.26.2.7	sc_vi_group_alloc()	190
10.26.2.8	sc_vi_group_get_session()	190
10.26.2.9	sc_vi_set_recv_node()	191
Index		193

Chapter 1

Introduction

SolarCapture is a toolkit for high performance packet capture and other packet processing applications. SolarCapture includes command line applications, utilities, and various API bindings. For more detail see the SolarCapture User Guide.

This document describes the C bindings which are used to extend SolarCapture and to embed it into applications.

The SolarCapture C bindings are included in the `solar_capture-core` package. You should also install the `solar_capture-python` package which includes tools and documentation as well as the Python language bindings.

Developers can use the SolarCapture C bindings to build high performance packet processing applications on Linux. SolarCapture can receive and send packets with the minimum number of CPU cycles for packet capture, network security, NFV or other packet processing (C, C++, python) applications:

- This library can be embedded in the user's own applications. See [Embedding SolarCapture](#).
- Users can also extend SolarCapture by providing processing nodes which can be integrated into SolarCapture's packet processing pipeline. See [Extending SolarCapture](#).

The SolarCapture C bindings can be used with any network adapter that supports SolarCapture. No AppFlex licenses are required to use the SolarCapture C bindings.

Chapter 2

Embedding SolarCapture

The SolarCapture distribution includes C bindings, allowing SolarCapture to be embedded in applications. Example code can be found at:

```
/usr/share/doc/solar_capture-<version>/examples
```

Applications that embed SolarCapture should include the `<solar_capture.h>` header, and should link to the `solarcapture1` library, as shown in the sample code. The header files can be found at:

```
/usr/include/solar_capture/
```

The following sections describe the main objects and concepts used in SolarCapture. For more information please refer to the example code, and the other documentation in this Guide.

2.1 Sessions — struct `sc_session`

All applications embedding SolarCapture must first instantiate a session object. A session provides an association between SolarCapture components.

All objects in a SolarCapture session are allocated up front, and packet processing is then initiated by calling [sc_session_go\(\)](#). Once packet processing has started it is not possible add new objects to the session.

2.2 Attributes — struct `sc_attr`

Attributes provide a convenient way to specify options, such as the size of buffers. For detailed information concerning SolarCapture attributed, refer to SolarCapture Attributes on page 108.

2.3 Threads — struct `sc_thread`

A session includes one or more threads that work together. Threads can be used for packet capture, packet processing and other tasks. The threads in a particular session are started and stopped as a group.

Objects that are part of the data-path are associated with a particular thread and are only accessed by that thread. This allows SolarCapture to operate without locks or expensive atomic operations, and helps to avoid sharing state between CPU caches.

A thread can be bound to a particular CPU core by setting the `affinity_core` attribute.

By default SolarCapture threads use busy-waiting. That is, they consume CPU cycles even when they do not have any work to do. Threads can be configured to sleep when idle by setting the `busy_wait` attribute to 0.

2.4 Virtual Interfaces — struct `sc_vi`

A virtual interface (VI) receives packets from a network interface, and passes them on to a node. The `sc_stream` interface is used to indicate which packets should be steered to a particular VI.

2.5 Nodes — struct `sc_node`

Nodes perform processing on packets. SolarCapture includes many node types that can be used in applications, and new node types can be implemented using the [Extending SolarCapture API](#).

VIs and nodes are connected in a directed acyclic graph, with node links passing packet buffers from one node to another. The buffers that are passed between nodes don't have to contain packets: They can contain any sort of data or messages. Nodes can be used to inspect or modify the packet buffers, generate new packet buffers, perform custom processing or interact with other parts of the system.

Connections can be made between nodes in the same thread or in different threads, provided that the threads are in the same SolarCapture session. Connections between threads use mailboxes, which are created automatically.

Packet buffers are allocated by packet pools. Many nodes receive packet buffers from VIs or other nodes via incoming links. It is also possible for nodes to allocate buffers from a pool. Buffers are freed back to their pool by forwarding them through a node link that is not connected.

A user application can consist of one or more nodes which may co-operate in order to progressively process the received network packets.

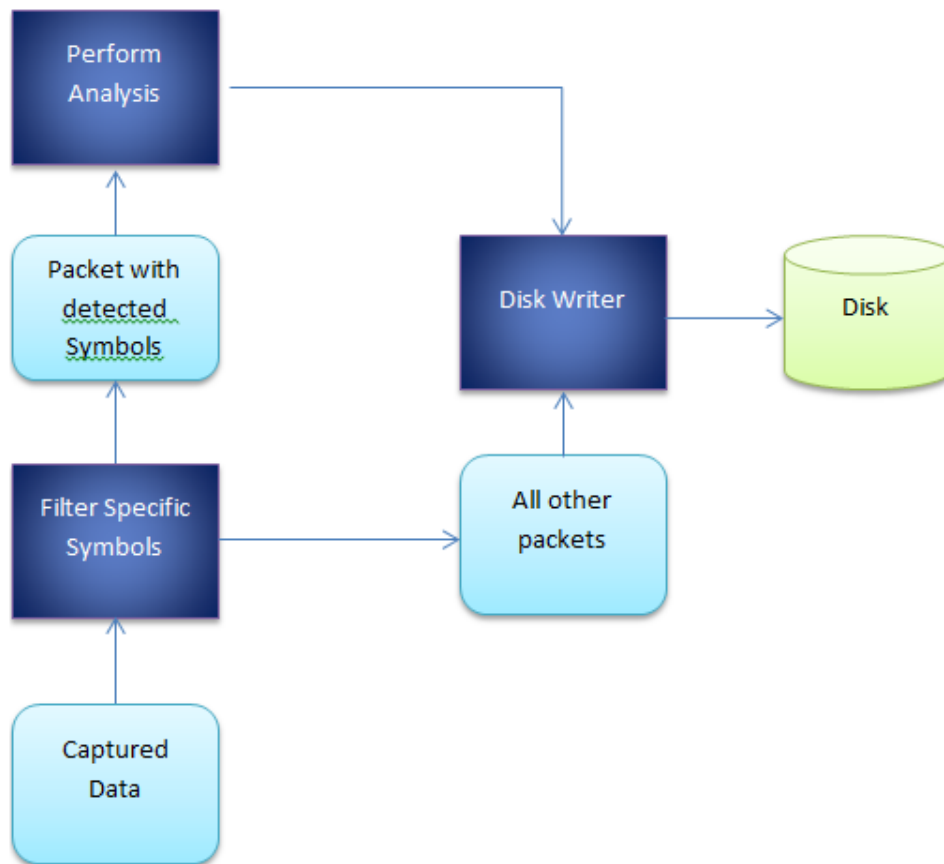


Figure 2.1 Example Application Nodes

The figure above is an example of co-operating nodes in a stock trading environment. Captured packets are fed to a filtering node which selects packets of interest to be forwarded to a second node for further analysis. All other packets a fed to the disk writer node. The analysis node will conduct further analysis on the packets such as statistics collection before passing packets to the disk writer node.

2.6 Mailboxes — struct `sc_mailbox`

Mailboxes are used to pass packet buffers between nodes in different threads, using an efficient lock-free mechanism. Each mailbox is paired with another in a different thread, and packets can be passed through a pair of mailboxes in both directions.

Mailboxes are created automatically when nodes in different threads are connected. Applications can create mailboxes explicitly if they need more fine-grained control.

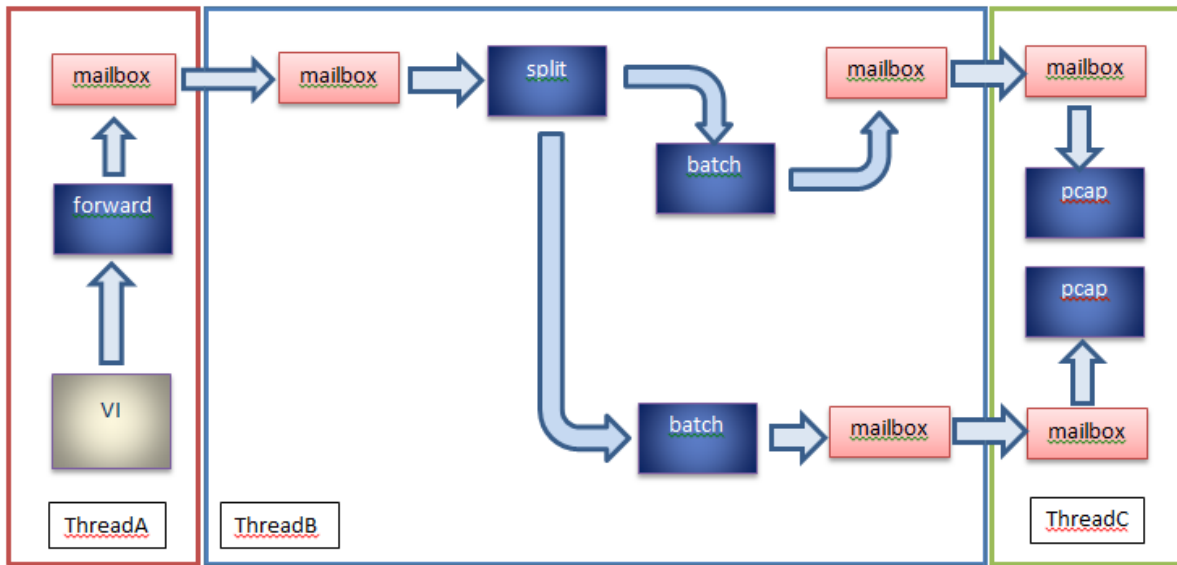


Figure 2.2 Mailboxes

2.7 Built-In Nodes

Many built-in nodes are available. These are documented in [Nodes](#).

Chapter 3

Extending SolarCapture

SolarCapture defines a coherent API allowing applications to be constructed from reusable components known as nodes. The core SolarCapture functionality can be extended by implementing new types of nodes in C. An example of how to define a new node type can be found at:

```
/usr/share/doc/solar_capture-<version>/examples/extensions_api
```

Implementations of new node types should include the `<solar_capture.h>` header, and should link to the `solarcapture1` library, as shown in the sample code. The header files can be found at:

```
/usr/include/solar_capture/
```

This chapter describes the objects and concepts needed to create new nodes. For more information please refer to the example code, and the other documentation in this Guide.

3.1 Node factories — struct `sc_node_factory`

A node factory provides an interface for instantiating new nodes. When a node is allocated with `sc_node_alloc()` or similar, the `nf_init_fn()` is invoked which should initialize the implementation and set the node type. Private state for the node implementation can be stored in the `nd_private` field.

The `nf_init_fn()` can retrieve arguments passed when allocating a node by invoking the following functions:

- `sc_node_init_get_arg_int()`
- `sc_node_init_get_arg_int64()`
- `sc_node_init_get_arg_dbl()`
- `sc_node_init_get_arg_str()`
- `sc_node_init_get_arg_obj()`

3.2 Node types — struct `sc_node_type`

This object defines the behavior of a node via a set of callbacks. Implementations must only instantiate objects of this type by calling `sc_node_type_alloc()`. A single node type instance can be shared by multiple node instances.

The `nt_prep_fn()` callback is invoked once per node just before the threads in a session are started. The outgoing links configured by the application are passed to this function. For nodes where the names of links can be chosen by the application, the links array should be inspected directly. Nodes that support links with fixed names can use the following functions to find their links:

- `sc_node_prep_get_link()`
- `sc_node_prep_get_link_or_free()`

The `nt_pkts_fn()` callback is invoked when packets arrive at a node. This callback provides the core functionality of the node. Packets provided to this callback should be forwarded via one of the node's outgoing links with `sc_forward()` or `sc_forward_list()`. (Packets do not have to be forwarded immediately).

The `nt_end_of_stream_fn()` callback is invoked when a node has received the last packet. That is, `nt_pkts_fn()` is never invoked after `nt_end_of_stream_fn()`.

3.3 Node libraries

A node library is a shared object file that contains one or more `sc_node_factory` instances. Each factory instance must be named `<something>_sc_node_factory` so that it can be found by `sc_node`.

If a node library contains a single factory, it is conventional to give the factory and the file matching names so that it is not necessary to name the library in the call to `sc_node_factory_lookup()`. For example, in the “reflect” example, the factory instance is `reflect_sc_node_factory`, and the library is `reflect.so`. If a node library is placed in one of the directories on the node library lookup path, then it will be found by a call to `sc_node_factory_lookup()`, `sc_node_alloc_named()` or `sc_node_alloc_from_str()`.

The node library lookup path includes the following directories:

- `.` (The current working directory)
- Directories identified by the environment variable `SC_NODE_PATH`
- `/usr/lib64/solar_capture/site-nodes`
- `/usr/lib/x86_64-linux-gnu/solar_capture/site-nodes`
- `/usr/lib64/solar_capture/nodes`
- `/usr/lib/x86_64-linux-gnu/solar_capture/nodes`

Note

Node factories do not have to be placed in node libraries. They can simply be instantiated within an application that embeds SolarCapture and be passed directly to `sc_node_alloc()`. Node libraries are useful when nodes are reusable.

3.4 Insert a user-defined node between capture and sc_writer

User-defined nodes can be inserted between the capture node and sc_writer node. See the extensions_api sample code for examples included in the solar_capture-python RPM.

The following example demonstrates how to insert a user-defined node called 'header_strip' into the solar_capture pipeline:

```
# SC_NODE_PATH must include directory containing header_strip.so
export SC_NODE_PATH=/path/to/nodes
solar_capture eth4=/captures/eth4.pcap header_strip:
```

The following example demonstrates how to pass arguments to the 'header_strip' node:

```
solar_capture eth4=/captures/eth4.pcap "header_strip:arg1=foo;arg2=bar"
```


Chapter 4

Examples

Solarflare SolarCapture comes with a range of example applications - including source code and make files.

Application	Description
c_api	Illustrates how to embed SolarCapture in a C application
c_api_export	Illustrates how to export content of packets for external processing
extensions_api	Illustrates the SolarCapture extensions API
forwarding	Illustrates how to forward packets between network ports
py_api	Illustrates basic use of the SolarCapture python bindings
unmanaged	Shows ways to pass packet buffers between threads managed by SolarCapture and threads managed by the application

4.1 c_api

This example illustrates how to embed SolarCapture in a C application. Please see the source for usage instructions and for further details.

4.2 c_api_export

This example illustrates how to export content of packets captured with SolarCapture C API for external processing.

In the scenario there are n threads created and the incoming traffic is split so that each of the threads receives subset of the traffic based on source and destination IP addresses' hash.

A sample implementation of SolarCapture custom node is provided that perform exporting and splitting of the traffic.

Please see the source for usage instructions and for further details.

4.3 extensions_api

These examples illustrate the SolarCapture extensions API. The extensions API lets you write custom packet processing nodes, which can then be used in a SolarCapture application.

Nodes typically receive packets on their inputs, do something useful with the packets and forward them to their outputs. Nodes may also generate new packets using buffers from a packet pool.

The 'reflect' examples all do roughly the same job: They switch the source and destination Ethernet MAC addresses. The different versions show different features of the extensions API.

Please see the source for usage instructions and for further details.

4.4 forwarding

The examples in this directory illustrate how to forward packets between network ports.

4.4.1 trivial_bridge.py

This example simply connects ports together in pairs. Each command line argument is a pair of interfaces that are connected together with a uni-directional channel.

Please see the source for usage instructions and for further details.

4.4.2 bpf_firewall.py

This example implements a very simple firewall. It forwards packets from one network interface to another, discarding packets that match a filter specified with BPF syntax.

Please see the source for usage instructions and for further details.

4.4.3 reflect.py

This example shows how packets can be modified before they are forwarded. It uses the 'reflect' node from the 'extensions_api' sample to swap the MAC addresses of received packets, and sends them back out of the interface they were received on.

Please see the source for usage instructions and for further details.

4.5 py_api

This example illustrates basic use of the SolarCapture python bindings. The python bindings can be used to embed SolarCapture into a python application. The python bindings are used to setup and control a packet processing pipeline.

Note that custom packet processing nodes can only be written in C (not python), but they can then be incorporated into a SolarCapture session using the python bindings.

The example itself is an extremely simple topology, consisting of just a VI to capture packets and a writer node to output captured packets to a pcap file.

Please see the source for usage instructions and for further details.

Note

You will find further examples using the python bindings, such as in the [forwarding](#) subdirectory.

4.6 unmanaged

The examples in this directory show ways to pass packet buffers between threads managed by SolarCapture and threads managed by the application.

Every component in SolarCapture has to be associated by an `sc_thread` object. By default `sc_threads` are managed by SolarCapture, which means that SolarCapture will create the underlying OS thread and manage its lifetime etc. An unmanaged `sc_thread` is used when you want to use SolarCapture components in a thread created by your application. Unmanaged threads are created by setting the thread attribute 'managed' to 0.

4.6.1 unmanaged_mailbox.c

This is the easiest way to pass packets out of SolarCapture. Mailboxes are the mechanism SolarCapture uses to pass packets between threads. To pass packets to an unmanaged thread you just need an unmanaged mailbox.

Please see the source for usage instructions and for further details.

4.6.2 unmanaged_thread.c

This example shows how an unmanaged thread can work with other SolarCapture components, including nodes and VIs. In this example a custom node is used to place packets onto a list, which is drained by the foreground application thread. Note that the `deliver_pkts` node shown in this example is essentially the same as the built-in `sc_append_to_list` node.

Please see the source for usage instructions and for further details.

Chapter 5

Nodes

This section describes the built-in nodes included in SolarCapture.

Node	Description
sc_append_to_list	Append incoming packets to an sc_packet_list .
sc_arista_ts	Replace SolarCapture timestamp with timestamp from an Arista switch.
sc_batch_limiter	Node to limit the batch size sent to downstream nodes.
sc_cpacket_encap	This node adds cPacket timestamps to packets.
sc_cpacket_ts	Node to replace card arrival timestamp with CPacket footer timestamp.
sc_delay_line	Node to delay upstream packets by a random time within a given time range.
sc_exit	Node that causes the process to exit when a condition is met.
sc_fd_reader	Reads data from a file or file descriptor.
sc_fd_writer	Write data to a file descriptor.
sc_filter	Node to filter packets, directing all matched packets to one output, and all other packets to another output.
sc_flow_balancer	This node distributes load by spreading packets over its output links while preserving flow affinity.
sc_injector	Packets sent to an injector node are transmitted on the network.
sc_line_reader	This node parses out lines from a data stream.
sc_merge_sorter	Merges inputs to output, sorting in timestamp order.
sc_no_op	Forward inputs to output.
sc_pacer	Emits packets at the time indicated by their associated timestamp.
sc_pass_n	A node which forwards a fixed number of packets.
sc_pcap_packer	A node that packs incoming packets into buffers that are ready to be written to a pcap file.
sc_pool_forwarder	Node that forwards packets from a packet pool.
sc_ps_packer	Takes individual packets as input and packs them into packed-stream format.
sc_ps_unpacker	Takes packed-stream buffers as input and unpacks them.
sc_range_filter	Node that forwards one or more ranges of packets.
sc_rate_monitor	Node that measures and exports packet rate and bandwidth to <code>solar_capture_monitor</code> .
sc_reader	Converts PCAP file format to SolarCapture packets on output.
sc_repeater	Replay packets in a loop.
sc_rr_gather	This node receives packets from multiple inputs, and forwards one packet from each input in turn in round-robin order.

Node	Description
sc_rr_spreader	This node spreads received packets over its set of outgoing links in round-robin order.
sc_rt_pacer	Emits packets at a variable rate determined by a control input.
sc_shm_broadcast	Export packets or messages to a shared memory channel with multiple consumers.
sc_shm_export	Export packets or messages to a shared memory channel.
sc_shm_import	Import packets or messages from a shared memory channel.
sc_sim_work	Simulate doing CPU intensive work on each packet.
sc_snap	Node that limits the length of a packet buffer.
sc_subnode_helper	Node used as a sub-node to manage inputs and/or pools.
sc_tap	Forward input to output, and a copy of input to the 'tap' output with optional filtering.
sc_timestamp_filter	Filter packets, accepting only those in a given range of timestamps.
sc_token_bucket_shaper	This node performs traffic shaping using the token bucket algorithm.
sc_tracer	Write debug trace to standard error.
sc_ts_adjust	Adjust packet buffer timestamps.
sc_tunnel	A node used to pass sc_packet objects between two SolarCapture sessions via a TCP socket.
sc_tuntap	Pass packets between SolarCapture and the kernel stack via a tun/tap interface.
sc_vi_node	A node which passes packets to and/or from a network interface.
sc_vss	Replace packet buffer timestamp with timestamp generated by VSS packet broker, and demultiplex by VSS port.
sc_writer	Node that writes packets to a file in pcap format.

5.1 [sc_append_to_list](#) Node Reference

Append incoming packets to an [sc_packet_list](#).

Detailed Description

This node provides a simple way to get packet buffers out of the SolarCapture node graph, and is typically used with an unmanaged thread. It is often used when writing code to adapt the SolarCapture API to another API, or embed SolarCapture in an application.

After allocating an instance of this node, the application must initialise [sc_append_to_list::append_to](#) so that it points to an initialised [sc_packet_list](#). Here is an example:

```
sc_node_alloc_named(&node, attr, thread, "sc_append_to_list", NULL, NULL, 0);
struct sc_append_to_list* atl = node->nd_private;
struct sc_packet_list my_packet_list;
sc_packet_list_init(&my_packet_list);
atl->append_to = &my_packet_list;
```

Packet buffers delivered in this way should eventually be returned to SolarCapture by forwarding them through one of this node's output links, or through its `free_link`:

```
sc_forward_list2(atl->free_link, &my_packet_list);
sc_packet_list_init(&my_packet_list);
```


5.2 `sc_arista_ts` Node Reference

Replace SolarCapture timestamp with timestamp from an Arista switch.

Detailed Description

This node is used with new Arista switches that are configured to add hardware timestamps to packets. It replaces the SolarCapture timestamp with the timestamp generated by the switch. The node has several modes for decoding timestamps from different models of arista switches and different settings of timestamping on Arista switches. The node arguments and statistics depend on the value of the `switch_model` and `ts_format` arguments.

Arguments

Argument	Optional?	Default	Type	Description
<code>switch_model</code>	Yes	NULL	SC_PARAM_STR	Which switch timestamp protocol to decode. Will assume "7150" if unspecified.
<code>ts_format</code>	Yes	NULL	SC_PARAM_STR	Which timestamp format to decode. Applicable for "7280" switch model only, will assume "48bit" if unspecified. Note: "64bit" timestamps are DEPRECATED.

Modes

[sc_arista_ts, switch_model=7150](#) | Replace SolarCapture timestamp with timestamp from an Arista 7150 switch.

[sc_arista_ts, switch_model=7280, ts_format=64bit](#) | Replace SolarCapture timestamp with 64bit timestamp from an Arista 7280 switch.

[sc_arista_ts, switch_model=7280, ts_format=48bit](#) | Replace SolarCapture timestamp with timestamp decoded from 48bit timestamp from an Arista 7280 switch.

5.2.1 `sc_arista_ts, switch_model=7150`

Replace SolarCapture timestamp with timestamp from an Arista 7150 switch.

Detailed Description

This mode is used to decode timestamps added by Arista 7150 series switches.

Arguments

Argument	Optional?	Default	Type	Description
kf_ip_proto	Yes	253	SC_PARAM_INT	The IP protocol used to send key frames.
log_level	Yes	"sync"	SC_PARAM_STR	The logging level of the node, must be set to one of "silent", "errors", "setup", "sync" or "verbose".
filter_oui	Yes		SC_PARAM_STR	Filter out timestamps with this OUI.
kf_device	Yes		SC_PARAM_STR	Filter keyframes by device field.
kf_eth_dhost	No		SC_PARAM_STR	Destination MAC address for the keyframes.
kf_ip_dest	No		SC_PARAM_STR	Destination IP address for the keyframes.
tick_freq	Yes	350000000	SC_PARAM_INT	Expected frequency in Hz of the switch tick.
max_freq_error_ppm	Yes	20000	SC_PARAM_INT	Max ppm between expected and observed frequency before entering no sync state.
lost_sync_ms	Yes	10000	SC_PARAM_INT	Time after last keyframe to enter lost sync state.
no_sync_ms	Yes	60000	SC_PARAM_INT	Time after last keyframe to enter no sync state.
no_sync_drop	Yes	0	SC_PARAM_INT	Toggle sync drop, set to 1 for on 0 for off.
strip_ticks	Yes	1	SC_PARAM_INT	Toggle the option for the node to strip switch timestamps. Set to 0 for off and 1 for on.
has_fcs	Yes	0	SC_PARAM_INT	The incoming packets have a trailing FCS, after the ticks.
drop_sync_on_skew	Yes	0	SC_PARAM_INT	If set then sync is dropped when packets are received with a bad or absent ticks field.
switch_model	Yes		SC_PARAM_STR	Passed through from sc_arista_ts, must either '7150' or unspecified.

Named Input Links

None

Output Links

Link	Default	Description
""	free	Packets with corrected timestamps
lost_sync	default	Packets with corrected timestamps when no keyframes have been seen for a while
no_sync	lost_sync or free*	Used when no recent keyframes have been seen
no_timestamp	no_sync	Packets with no arista timestamp
keyframes	no_sync	Used for keyframes
lldp	no_timestamp	Used for LLDP packets

Keyframes and LLDP packets are treated specially because they are not timestamped by the switch, and so it is not possible to give them timestamps with the same clock as other packets.

(*) no_sync packets go to the same place as lost_sync packets by default. If no_sync_drop=1, then they are freed by default.

Exposed Statistics

Arista timestamp statistics that are exposed by the `sc_arista_ts` node when `switch_model=7150`.

Name	Type	Data Type	Description
max_host_t_delta	double	config	Max delta in seconds the node can compute a tick-delta over.
max_freq_error	double	config	Max ppm allowed between measured and expected tick frequency before entering no sync state.
lost_sync_ms	int	config	Time in milliseconds spent in lost sync state.
no_sync_ms	int	config	Time in milliseconds spent in no sync state.
exp_tick_freq	int	config	The expected tick frequency in Hz.
strip_ticks	int	config	1 if the node is stripping ticks 0 otherwise.
log_level	int	config	The log level.
tick_freq	double	magnitude	The measured tick frequency in Hz.
n_keyframes	uint64_t	pkt_count	Number of keyframes processed by the node.
n_filtered_oui	uint64_t	pkt_count	Number of packets filtered out by OUI.
n_filtered_other	uint64_t	pkt_count	Number of packets filtered out for other reasons.
n_skew_zero_ticks	uint64_t	pkt_count	Number of packets without a timestamp (ticks is zero).
n_lost_sync	uint64_t	pkt_count	Number of packets processed whilst in lost sync state.
n_no_sync	uint64_t	pkt_count	Number of packets processed whilst in no sync state.
n_kf_len_mismatch	uint64_t	pkt_count	Number of packets received where the keyframe length did not match.
n_kf_dev_mismatch	uint64_t	pkt_count	Number of packets received where the device field did not match.
n_kf_bad_fcs_type	uint64_t	pkt_count	Number of keyframes with a bad FCS.
kf_switch_drops	uint64_t	pkt_count	Number of keyframes dropped by the switch.
n_kf_big_gap	uint64_t	pkt_count	Number of large gaps between keyframes.
n_skew	uint64_t	pkt_count	Number of skews.
n_host_ts_misorder	uint64_t	ev_count	Host timestamp detected out-of-order.
n_kf_host_ts_misorder	uint64_t	ev_count	Host timestamp of keyframe detected out-of-order.
enter_no_sync	uint64_t	ev_count	Number of times the node has entered no sync state.
enter_sync1	uint64_t	ev_count	Number of times the node has entered sync1 state.
enter_sync2	uint64_t	ev_count	Number of times the node has entered sync2 state.
enter_lost_sync	uint64_t	ev_count	Number of times the node has entered lost sync state.

5.2.2 sc_arista_ts, switch_model=7280, ts_format=64bit

Replace SolarCapture timestamp with 64bit timestamp from an Arista 7280 switch.

Detailed Description

This mode is used to decode timestamps added by Arista 7280 series switches.

Arguments

Argument	Optional?	Default	Type	Description
filter_oui	Yes		SC_PARAM_STR	Assume packets with this OUI in the Ethernet source field do not have a switch timestamp.
strip_ticks	Yes	1	SC_PARAM_INT	Toggle the option for the node to strip switch timestamps. Set to 0 for off and 1 for on.
rollover_window_ms	Yes	1000	SC_PARAM_INT	Window before lower bit rollover in which to check packets for the rollover bug.
switch_model	Yes		SC_PARAM_STR	Passed through from <code>sc_arista_ts</code> , must be either '7280' or unspecified.
ts_format	Yes		SC_PARAM_STR	Passed through from <code>sc_arista_ts</code> , must be either '64bit' or unspecified.

Named Input Links

None

Output Links

Link	Default	Description
""	free	Packets with corrected timestamps
no_timestamp	default	Packets with no arista timestamp
lldp	no_timestamp	Used for LLDP packets

LLDP packets are treated specially because they are not timestamped by the switch, and so it is not possible to give them timestamps with the same clock as other packets.

Exposed Statistics

Arista timestamp statistics that are exposed by the `sc_arista_ts` node when `switch_model=7280`, `ts_format=64bit`.

Name	Type	Data Type	Description
strip_ticks	int	config	1 if the node is stripping ticks 0 otherwise.
rollover_window_ns	uint64_t	config	The window over which the node is checking for the rollover bug.
last_good_delta_ns	int64_t	time delta	The last measured time delta between arista and NIC times from packets outside the rollover window.

5.2.3 `sc_arista_ts, switch_model=7280, ts_format=48bit`

Replace SolarCapture timestamp with timestamp decoded from 48bit timestamp from an Arista 7280 switch.

Detailed Description

This mode is used to decode 48 bit timestamps added by Arista 7280 series switches.

Arguments

Argument	Optional?	Default	Type	Description
<code>filter_oui</code>	Yes		SC_PARAM_STR	Assume packets with this OUI in the Ethernet source field do not have a switch timestamp.
<code>strip_ticks</code>	Yes	1	SC_PARAM_INT	Toggle the option for the node to strip switch timestamps. Set to 0 for off and 1 for on.
<code>ts_src_mac</code>	Yes	0	SC_PARAM_INT	If set then timestamp is retrieved from source mac address field instead of ethertype layer.
<code>replacement_src_mac</code>	Yes		SC_PARAM_STR	Replace timestamp located in source mac address field by given mac address. Applicable for <code>ts_src_mac=1</code> only.
<code>switch_model</code>	Yes		SC_PARAM_STR	Passed through from <code>sc_arista_ts</code> , must be either '7280' or unspecified.
<code>ts_format</code>	Yes		SC_PARAM_STR	Passed through from <code>sc_arista_ts</code> , must be either '48bit' or unspecified.

Named Input Links

None

Output Links

Link	Default	Description
<code>""</code>	free	Packets with corrected timestamps
<code>no_timestamp</code>	default	Packets with no arista 48bit timestamp
<code>lldp</code>	<code>no_timestamp</code>	Used for LLDP packets

LLDP packets are treated specially because they are not timestamped by the switch, and so it is not possible to give them timestamps with the same clock as other packets.

Exposed Statistics

Arista timestamp statistics that are exposed by the `sc_arista_ts` node when `switch_model=7280, ts_format=48bit`.

Name	Type	Data Type	Description
strip_ticks	int	config	1 if the node is stripping ticks 0 otherwise.
replace_src_mac	int	config	1 if the node is replacing source mac 0 otherwise.
n_filtered_oui	uint64_t	pkt_count	Number of packets filtered out by OUI.
n_filtered_arista	uint64_t	pkt_count	Number of packets filtered out because of invalid Arista field.
n_filtered_other	uint64_t	pkt_count	Number of packets filtered out for some other reasons.
n_rollover	uint64_t	pkt_count	Number of packets with seconds rollover.

5.3 sc_batch_limiter Node Reference

Node to limit the batch size sent to downstream nodes.

Detailed Description

This node forwards packets from its input to its output, emitting at most 'max_packets' in each batch.

By default a batch of packets is emitted in each polling loop. If mode="on_idle", then packets are only emitted when the sc_thread is idle (via an idle callback).

Arguments

Argument	Optional?	Default	Type	Description
max_packets	Yes	64	SC_PARAM_INT	The maximum number of packets in each batch.
mode	Yes	NULL	SC_PARAM_STR	Set mode="on_idle" to only emit packets when thread is idle.

Exposed Statistics

Statistics exposed by the [sc_batch_limiter](#) node.

Name	Type	Data Type	Description
max_packets	int	config	The maximum number of packets sent per batch.
fwd_on_idle	int	config	Set to 1 if mode is on_idle else 0.
backlog	int	pkt_count	The current number of packets waiting to be forwarded.

5.4 sc_cpacket_encap Node Reference

This node adds cPacket timestamps to packets.

Detailed Description

This node adds cPacket timestamps to packets.

Arguments

Argument	Optional?	Default	Type	Description
fcs_present	Yes	detect	SC_PARAM_INT	Set to 0 if FCS not present in input, 1 if FCS is present. Leave at default to auto-detect.

Named Input Links

None

Output Links

Link	Description
""	Receives a copy of the input packets with cPacket encapsulation added.
"input"	Input packets are forwarded here unmodified.

5.5 sc_cpacket_ts Node Reference

Node to replace card arrival timestamp with CPacket footer timestamp.

Detailed Description

Node to replace card arrival timestamp with CPacket footer timestamp.

Arguments

Argument	Optional?	Default	Type	Description
has_fcs	No	N/A	SC_PARAM_INT	Whether the input packets still have their trailing frame checksums
keep_cpacket_footer	Yes	0(false)	SC_PARAM_INT	Whether the CPacket footer information should be kept
is_metamako	Yes	0(false)	SC_PARAM_INT	Whether the CPacket footer has Arista-Metamako TLV extensions

Named Input Links

None

Output Links

Link	Description
""	Packets with timestamps set from CPacket footer, (and optionally footer stripped off).

5.6 sc_delay_line Node Reference

Node to delay upstream packets by a random time within a given time range.

Detailed Description

Node to delay upstream packets by a random time within a given time range. Randomness is achieved by performing a hash on the destination IP address and can be controlled using the `num_lines` argument.

If `num_lines = 1`:

- `usec/msec` must be a single value.
- All packets will be delayed by this amount.

If `num_lines > 1`:

- `usec/msec` must be a range of values `<min_delay>-<max_delay>`.
- Non-IP packets are delayed by exactly `<min_delay>`.
- IP packets are assigned a line by hashing the destination IP address.
- For a given line in (0, ..., `num_lines-1`) the delay is `<min_delay> + (<max_delay> - <min_delay>) * (line / num_lines)`

Arguments

Argument	Optional?	Default	Type	Description
<code>num_lines</code>	Yes	1	SC_PARAM_INT	Number of lines used in the hash.
<code>usec</code>	Yes	NULL	SC_PARAM_STR	Set this to a string of the form " <code>\<min_delay\>[-\<max_delay\>]</code> " to set the delay time of the node in microseconds.
<code>msec</code>	Yes	NULL	SC_PARAM_STR	Set this to a string of the form " <code>\<min_delay\>[-\<max_delay\>]</code> " to set the delay time of the node in milliseconds.

Note: One and only one of `usec` and `msec` must be set.

Named Input Links

None

Output Links

Link	Description
""	All packets are sent down this link.

5.7 sc_exit Node Reference

Node that causes the process to exit when a condition is met.

Detailed Description

By default this node causes the process to exit when all `sc_exit` nodes in the process have received the end-of-stream signal on their inputs. Typically `sc_exit` nodes are placed at the end of a pipeline so that the process exits after all packet processing is complete.

Each `sc_exit` node has one or more "exit conditions", set by the `end_of_stream` and `predicate` arguments. Each `sc_exit` node also has a scope. When all of the `sc_exit` nodes in a scope detect their exit condition, the "exit action" is invoked. The scope argument may take the following values:

- `process`: Includes all `sc_exit` nodes in the same process.
- `session`: Includes all `sc_exit` nodes in the same session.
- `none`: Each `sc_exit` node has its own scope.

If the session was started with `sc_session_run()`, then the default action is to call `sc_session_stop()` so that the `sc_session_run()` call returns. Otherwise the default action is to exit the process by calling `exit()`. (NB. It is important to ensure that the application cannot also call `exit()`, because `exit()` is not thread safe).

Input packets are forwarded to the output unmodified.

Arguments

Argument	Optional?	Default	Type	Description
<code>action</code>	Yes	(1)	<code>SC_PARAM_STR</code>	Action to take when exit condition met. One of: <code>exit</code> , <code>stop</code> .
<code>end_of_stream</code>	Yes	1	<code>SC_PARAM_INT</code>	Exit condition is met when end-of-stream signal is received.
<code>scope</code>	Yes	<code>process</code>	<code>SC_PARAM_STR</code>	See description. May be <code>process</code> , <code>session</code> or <code>none</code> .
<code>predicate</code>	Yes		<code>SC_PARAM_OBJ</code>	Predicate is invoked on each input packet. Exit condition is met when it returns true.

(1) See above for a description of the default action.

5.8 `sc_fd_reader` Node Reference

Reads data from a file or file descriptor.

Detailed Description

This node reads data from a file in the filesystem, or from a file descriptor, and passes the data to its output link.

By default each output buffer contains data from a single `read()` call. This may be less than a full buffers worth if the file descriptor is a socket or pipe. Set `fill_buffers=1` to ensure that each buffer is filled completely before releasing it to the output.

If the input file descriptor is a datagram socket or similar (and `fill_buffers=0`) then each output packet will contain a single datagram.

Arguments

Argument	Optional?	Default	Type	Description
filename	Yes		SC_PARAM_STR	The name of a file to read data from. (If fd is also set then this name is just informational).
fd	Yes		SC_PARAM_INT	File descriptor to read data from.
signal_eof	Yes	1	SC_PARAM_INT	Set to 0 to prevent this node from signalling end-of-stream when the whole file has been read.
close_on_eof	Yes	1	SC_PARAM_INT	Whether to close the file descriptor when the whole file has been read.
fill_buffers	Yes	0	SC_PARAM_INT	Whether or not to completely fill output packets.
repeat	Yes	0	SC_PARAM_INT	If set to true, when we reach the end of the file, we seek to the beginning again and keep reading.
repeat_offset	Yes	0	SC_PARAM_INT	Offset to seek to if repeating. (This can be used to skip a per-file header).

5.9 sc_fd_writer Node Reference

Write data to a file descriptor.

Detailed Description

This node writes the raw contents of each incoming packet to a file descriptor. It can be used to write data into a file, socket, pipe etc.

The contents of each [sc_packet](#) arriving at this node is written with a single `writew()` call (or equivalent). If the file descriptor is a datagram socket then each [sc_packet](#) generates a single datagram.

If the file descriptor supports non-blocking writes then this node uses `epoll` to avoid blocking the thread.

Arguments

Argument	Optional?	Default	Type	Description
fd	No		SC_PARAM_INT	File descriptor to write data to.
close_on_eos	Yes	0	SC_PARAM_INT	Whether to close the file descriptor when the end-of-stream signal is received.

Output Links

Link	Description
""	Input packets are forwarded to this output once written.

5.10 sc_filter Node Reference

Node to filter packets, directing all matched packets to one output, and all other packets to another output.

Detailed Description

This node directs all matched packets to one output and all other packets to another output. The filter can be provided via a BPF string, or via a [sc_pkt_predicate](#) object.

Arguments

Argument	Optional?	Default	Type	Description
bpf	Yes	NULL	SC_PARAM_STR	Filter string in Berkeley Packet Filter format.
predicate	Yes	NULL	SC_PARAM_OBJ	An SC_OBJ_PKT_PREDICATE to use as a filter.

Note: Exactly one of bpf and predicate must be set.

Named Input Links

None

Output Links

Link	Description
""	Packets matched by the filter.
"not_matched"	Packets not matched by the filter.

Exposed Statistics

Statistics exposed by the [sc_filter](#), [sc_range_filter](#) and [sc_timestamp_filter](#) nodes.

Name	Type	Data Type	Description
pkts_rejected	uint64_t	pkt_count	The number of packets not matched by the filter.

5.11 sc_flow_balancer Node Reference

This node distributes load by spreading packets over its output links while preserving flow affinity.

Detailed Description

This node either forwards or copies packets from its input to its outputs (see `copy_mode`). It attempts to distribute load evenly over the outputs, whilst also preserving flow affinity. That is, packets from the same flow are directed to the same output, and both directions in a conversation are directed to the same output.

NB. Both directions in a TCP conversation are directed to the same output only if the communicating hosts have different IP addresses. That should always be true unless you are analysing packets on a loopback interface.

The flow key always includes VLAN ID and `ether_type`. For IPv4 packets it also includes the IP addresses and protocol, and for TCP it includes the port numbers.

The input can be in normal or packed-stream format. When `copy_mode=copy`, the output is in packed-stream format. When `copy_mode=zc` the output is normal format.

When `mode=round-robin` new flows are assigned to each output in round-robin order. When `mode=estimate` an estimate of the current load experienced by each output is maintained, and new flows are directed to the output with the lowest estimated load.

Arguments

Argument	Optional?	Default	Type	Description
<code>copy_mode</code>	No		SC_PARAM_STR	Copy from input to output ('copy') or use zero-copy ('zc').
<code>mode</code>	No		SC_PARAM_STR	Balancing mode; 'estimate' or 'round-robin'.
<code>flow_table_capacity</code>	Yes	1024	SC_PARAM_INT	Initial capacity of the flow table.
<code>flush_ns</code>	Yes	10000000	SC_PARAM_INT	Flush timeout when <code>copy_mode=copy</code> .
<code>max_grow_attempts</code>	Yes	3	SC_PARAM_INT	The maximum number of attempts the flow balancer will make when trying to grow the hash table.

Output Links

An outgoing link named "input" is treated specially: It receives the input packets when `copy_mode=copy`.

Exposed Statistics

Statistics exposed by the [sc_flow_balancer](#) node.

Name	Type	Data Type	Description
<code>flow_table_capacity</code>	<code>uint64_t</code>	magnitude	Capacity of the flow table.
<code>avg_flow_load</code>	<code>uint64_t</code>	bandwidth	Moving average of the load per flow.
<code>n_flows</code>	<code>int</code>	magnitude	Current number of flows directed to this output.
<code>total_flows</code>	<code>uint64_t</code>	magnitude	Total number of flows directed to this output.
<code>total_work</code>	<code>uint64_t</code>	magnitude	Estimate of total work directed to this output.
<code>load_est_short</code>	<code>uint64_t</code>	bandwidth	Short-term load estimate for this output.
<code>load_est_long</code>	<code>uint64_t</code>	bandwidth	Long-term load estimate for this output.
<code>drops</code>	<code>uint64_t</code>	<code>pkt_count</code>	Number of packets dropped at this output due to running out of buffering.

5.12 sc_injector Node Reference

Packets sent to an injector node are transmitted on the network.

Detailed Description

An `sc_injector` node is used to transmit packets out of a Solarflare network interface. Packets are forwarded to the output link after they have been transmitted.

Arguments

Argument	Optional?	Default	Type	Description
<code>interface</code>	No		SC_PARAM_STR	The name of the network interface to use.
<code>csum_ip</code>	Yes	0	SC_PARAM_INT	Set to 1 to enable offload of the IPv4 header checksum.
<code>csum_tcpudp</code>	Yes	0	SC_PARAM_INT	Set to 1 to enable offload of TCP/UDP checksums.

5.13 sc_line_reader Node Reference

This node parses out lines from a data stream.

Detailed Description

This node parses out lines from a data stream. Input is interpreted as a stream of text data. Output is a single contiguous packet buffer per line of input.

This is useful for parsing [sc_packet](#) objects created by an [sc_fd_reader](#) node, and converting them into one [sc_packet](#) object per line.

Arguments

Argument	Optional?	Default	Type	Description
<code>forward_truncated</code>	Yes	0	SC_PARAM_INT	Specifies whether lines too large to fit in an sc_packet object should be sent down stream. If set to true such packets will have the SC_TRUNCATED flag set.
<code>lstrip</code>	Yes	1	SC_PARAM_INT	Specifies whether whitespace should be stripped from the start of a line.
<code>rstrip</code>	Yes	1	SC_PARAM_INT	Specifies whether whitespace should be stripped from the end of a line.
<code>strip_comments</code>	Yes	1	SC_PARAM_INT	Specifies whether lines starting with '#' should be forwarded.

strip_blank	Yes	1	SC_PARAM_INT	Specifies whether blank lines should be forwarded.
add_nul	Yes	1	SC_PARAM_INT	Specifies whether a nul ('\0') character should be appended to each line sent downstream.
add_new_line	Yes	0	SC_PARAM_INT	Specifies whether a new line ('\n') character should be appended to each line sent downstream.

Named Input Links

None

Output Links

Link	Description
""	One sc_packet object per line in the input data stream.
"input"	The sc_packet objects sent on the "" input link.

5.14 sc_merge_sorter Node Reference

Merges inputs to output, sorting in timestamp order.

Detailed Description

This node merges its inputs and forwards them to its output in timestamp order. It is assumed that within each input the packets are already sorted in timestamp order.

Arguments

None

5.15 sc_no_op Node Reference

Forward inputs to output.

Detailed Description

This node forwards its inputs to its output. It is sometimes useful as a convenience when setting up node graphs because it doesn't care what its inputs and output are named.

Arguments

None

5.16 sc_pacer Node Reference

Emits packets at the time indicated by their associated timestamp.

Detailed Description

This node forwards packets to the output, but only emits them once the timestamp in the packet is current or in the past. Packets are emitted in FIFO order.

Arguments

None

Output Links

Link	Description
""	Input packets are forwarded to this output.

5.17 sc_pass_n Node Reference

A node which forwards a fixed number of packets.

Detailed Description

This node forwards the indicated number of packets to its default output link. Any further packets that arrive at this node are either leaked (default) or forwarded to an output named "the_rest" (if it exists).

Arguments

Argument	Optional?	Default	Type	Description
n	No		SC_PARAM_INT	Number of packets to forward.

Output Links

Link	Description
""	The first n packets are forwarded here.
"the_rest"	Subsequent packets are forwarded to this output if it exists.

5.18 sc_pcap_packer Node Reference

A node that packs incoming packets into buffers that are ready to be written to a pcap file.

Detailed Description

A node that packs incoming packets into buffers that are ready to be written to a pcap file.

Arguments

Argument	Optional?	Default	Type	Description
snap	Yes		SC_PARAM_INT	Bytes of frame data to store. If unset or zero, use the "snap" attribute, else at least 16KiB if the attribute is not set.
rotate_seconds	Yes	0	SC_PARAM_INT	If nonzero, a new capture file is created after the given number of seconds.
rotate_file_size	Yes	0	SC_PARAM_INT	If nonzero, a new capture file is created whenever the previous file exceeds the given size in bytes.
format	Yes	"pcap"	SC_PARAM_STR	File format. Set to "pcap-ns" for nano-second PCAP format or "pcap" for the default format that uses microseconds.
on_error	Yes	"exit"	SC_PARAM_STR	Set behaviour for errors. Can be one of "exit", "abort", "message" and "silent".
discard_mask	Yes	0	SC_PARAM_INT	Mask with packed stream packets to discard. Bits in the mask that take effect are SC_CSUM_ERROR and SC_CRC_ERROR. Not that this argument will have no effect on packets not in packed stream format.
filename	No		SC_PARAM_STR	Template for filename. This is used to generate filenames for the initial and post-rotation files. The filename may include a time format defined by strftime(3). If the filename includes the string '\$i' then it is replaced by an incrementing index.

Named Input Links

Input links may be named, in which case the packets are forwarded to a matching named output link.

Output Links

Link	Description
""	Packed buffers in pcap format are sent out on this link.
"#input"	Packets from all inputs are forwarded to this link.
NAME	If NAME matches the name of an input link, then input packets are forwarded to the corresponding output link.

Exposed Statistics

Statistics exposed by the [sc_pcap_packer](#) node.

Name	Type	Data Type	Description
pcap_bytes	uint64_t	byte_count	Sum of bytes of encapsulated data send to output.
buffer_low	uint64_t	ev_count	Number of times the pool of buffers has run out.

5.19 sc_pool_forwarder Node Reference

Node that forwards packets from a packet pool.

Detailed Description

This node allocates a pool and forwards buffers from the pool to its output link. As buffers are recycled back to the pool, they are collected by this node and forwarded on again.

Buffers are initialised as described in [sc_pool_get_packets\(\)](#).

If the `batch_num_pkts` attribute is set it determines the minimum number of buffers that this node will emit in each polling loop. If it is not set then the minimum is one quarter of the pool size (or the maximum if smaller). Exported in `solar_capture_monitor` as `'batch_min'`.

If the `batch_max_pkts` attribute is set it determines the maximum number of buffers that this node will emit in each polling loop. If it is not set then the maximum is one quarter of the pool size (or the minimum if larger). Exported in `solar_capture_monitor` as `'batch_max'`.

Arguments

None

5.20 sc_ps_packer Node Reference

Takes individual packets as input and packs them into packed-stream format.

Detailed Description

Takes individual packets as input and packs them into packed-stream format, outputting larger packed buffers.

Control the size of the packed buffers via the 'buf_size' attribute of the attributes you pass in when instantiating the node.

Named Input Links

None

Output Links

Link	Description
""	Packed-stream buffers are forwarded on this link

5.21 sc_ps_unpacker Node Reference

Takes packed-stream buffers as input and unpacks them.

Detailed Description

Takes packed-stream buffers as input and unpacks them, allocating new buffers and copying the individual packets into them.

Named Input Links

None

Output Links

Link	Description
""	Unpacked packets are forwarded on this link
"input"	The input buffers are forwarded on this link

5.22 sc_range_filter Node Reference

Node that forwards one or more ranges of packets.

Detailed Description

Node that forwards one or more ranges of packets.

Incoming packets are assigned an index starting at zero. Packets whose index lies within the ranges indicated by the "range" argument are forwarded to the default output, and other packets are forwarded to the "reject" output or freed.

- Ranges must be non-overlapping and in order.
- Ranges are inclusive at both ends.
- Indices are zero-based.

Arguments

Argument	Optional?	Default	Type	Description
range	No		SC_PARAM_STR	Comma-separated list of packet ranges or indices.

Output Links

Link	Description
""	Packets indicated by the "range" argument are forwarded here.
"reject"	Other packets are forwarded here.

Exposed Statistics

Statistics exposed by the [sc_filter](#), [sc_range_filter](#) and [sc_timestamp_filter](#) nodes.

Name	Type	Data Type	Description
pkts_rejected	uint64_t	pkt_count	The number of packets not matched by the filter.

5.23 sc_rate_monitor Node Reference

Node that measures and exports packet rate and bandwidth to solar_capture_monitor.

Detailed Description

This node measures and exports packet rate and bandwidth to solar_capture_monitor.

It passes packets from input to output without modification, and measures packet rate and bandwidth statistics using an exponential moving average.

The statistics can be accessed with the solar_capture_monitor tool.

Note that the total number of packets is also available from the solar_capture_monitor output in the pkts_in field, as for all nodes.

Arguments

Argument	Optional?	Default	Type	Description
alpha	Yes	0.5	SC_PARAM_DBL	Alpha value for the exponential moving average. Higher values give more weight to newer samples.
period	Yes	0.1	SC_PARAM_DBL	Period in seconds over which samples are measured.

Exposed Statistics

Statistics exposed by the `sc_rate_monitor` node.

Name	Type	Data Type	Description
pkt_rate	int	pkt_rate	Packet rate (packets/second).
cap_bytes	uint64_t	byte_count	Sum of payload bytes.
link_bytes	uint64_t	byte_count	Sum of frame_len (bytes on wire before snapping).
cap_bw	uint64_t	bandwidth	Payload bandwidth (bits/second).
link_bw	uint64_t	bandwidth	Bandwidth before snap (bits/second) (from frame_len field).

5.24 sc_reader Node Reference

Converts PCAP file format to SolarCapture packets on output.

Detailed Description

This node converts PCAP file format to SolarCapture packets on output.

The input can either be a file on disk (by setting the "filename" arg) or a file descriptor (by setting the "fd" arg). Alternatively if neither are given then the input packets are interpreted as a binary stream of PCAP formatted packets and de-encapsulated.

By default the input is streamed to the output. If `prefill=all-input` then the node only starts emitting packets when it has read in the whole input file. Note that if the packet pool is not large enough to buffer the whole input then an error message will be emitted and the process will exit.

If `prefill=all-buffers` then the node starts emitting packets when it has read in the whole input file, or when the packet pool is exhausted, whichever happens first.

Arguments

Argument	Optional?	Default	Type	Description
filename	Yes		SC_PARAM_STR	The name of a PCAP file to read packet data from. (If fd is also set then this name is just informational).
fd	Yes		SC_PARAM_INT	File descriptor to read PCAP formatted packet data from.
prefill	Yes	"none"	SC_PARAM_STR	Whether to stream input to output or buffer. One of: "none", "all-input" or "all-buffers".
signal_eof	Yes	1	SC_PARAM_INT	Set to 0 to prevent this node from signalling end-of-stream at the end of the file.

Output Links

Link	Description
""	The unpacked stream of packets with one <code>sc_packet</code> per packet in the PCAP.
"input"	The PCAP format stream.

5.25 `sc_repeater` Node Reference

Replay packets in a loop.

Detailed Description

This node plays its input to its output multiple times. In all cases the input is buffered until the end-of-stream indication is seen, and then replaying to the output starts.

After the first play-out the packet timestamps are adjusted by a constant amount each time around so that the timestamp of the first packet comes after the previous. (This ensures that timestamps on the output are monotonically increasing, provided that the timestamps in the input are also monotonically increasing).

If the node has an incoming link named "recycle" then it is expected that this link receives packets from the output. In this mode the input is forwarded to the output without any copying. Otherwise the input is buffered and copied to the output.

Arguments

Argument	Optional?	Default	Type	Description
<code>n_repeats</code>	Yes	infinite	<code>SC_PARAM_INT</code>	Number of times to repeat the input to output

5.26 `sc_rr_gather` Node Reference

This node receives packets from multiple inputs, and forwards one packet from each input in turn in round-robin order.

Detailed Description

This node receives packets from multiple inputs, and forwards one packet from each input in turn in round-robin order. See `sc_rr_spreader` for more details.

Arguments

None

Named Input Links

None

5.27 `sc_rr_spreader` Node Reference

This node spreads received packets over its set of outgoing links in round-robin order.

Detailed Description

This node spreads received packets over its set of outgoing links in round-robin order. It is usually used together with `sc_rr_gather` to spread load over multiple worker threads.

Packets are emitted by `sc_rr_gather` in the same order that they are received by `sc_rr_spreader`. (To ensure this, corresponding links must be added to `sc_rr_spreader` and `sc_rr_gather` in the same order). There is no guarantee as to the order in which packets will be handled by worker threads.

This mechanism of spreading load is suitable when the packet processing is stateless, and the work done per packet is either independent of the packet length, or the packet lengths are distributed randomly.

Arguments

None

Named Input Links

None

5.28 `sc_rt_pacer` Node Reference

Emits packets at a variable rate determined by a control input.

Detailed Description

This node is used to control packet rate in real-time under the control of an interactive input. That is, this node emits packets at a rate determined by the control input, and the rate can be changed immediately in response to new control inputs.

This node expects two inputs: A control input named "controller" and a data-path input named "" or NULL. The data-path input is forwarded to the output under the control of commands read from the control input.

Each buffer on the control input should contain a single command formatted as a nul terminated string. The node is initially stopped. The commands are:

Command	Description
speedup MUL	Start forwarding with speedup (MUL > 1.0) or slow down (MUL < 1.0) relative to real-time.
pps PPS	Start forwarding. PPS gives the target packet rate in packets-per-second.
bw BPS	Start forwarding with constant bandwidth BPS (in bits per second).
stop	Stop forwarding now.
n N	Stop forwarding after N packets.
pause TIME	Pause processing of commands for give time. TIME must have suffix "s", "ms", "us", or "ns".
sleep TIME	Synonym for "pause TIME".

5.29 sc_shm_broadcast Node Reference

Export packets or messages to a shared memory channel with multiple consumers.

Detailed Description

This node is used in conjunction with [sc_shm_import](#) to pass packets one or more consumer processes. Packets delivered to `sc_shm_broadcast` are forwarded over the channel to one or more [sc_shm_import](#) nodes in consumer processes.

See also [sc_shm_export](#), which is more suitable when there is only a single consumer, or a reliable channel is needed.

Arguments

Argument	Optional?	Default	Type	Description
path	No		SC_PARAM_STR	Prefix of a path in the filesystem used for creating a socket and shared memory files.
max_channels	Yes	1	SC_PARAM_INT	The maximum number of consumers that can connect to this channel.
max_in_flight	Yes	"100%"	SC_PARAM_STR	Maximum total amount of buffering that can be in flight at any one time. Specified as a percentage of the incoming pool (" suffix), or in bytes ('B', 'KiB', 'MiB' or 'GiB' suffix).

Argument	Optional?	Default	Type	Description
max_in_flight_per_channel	Yes	"100%"	SC_PARAM_STR	Maximum amount of buffering that can be in flight per consumer. Specified as a percentage of the incoming pool (" suffix), or in bytes ('B', 'KiB' or 'GiB' suffix). max_in_flight_per_channel cannot exceed max_in_flight.
in_flight_reserved_per_channel	Yes	50% / max_channels	SC_PARAM_STR	Proportion of buffering that is dedicated to each channel. The remainder is shared and can be used by any channel. This can be specified as a percentage of max_in_flight (with a " suffix), or in bytes (with 'B', 'KiB', 'MiB' or 'GiB' suffix). max_channels * in_flight_reserved cannot exceed max_in_flight.
min_connected_reliable_channels	Yes	0	SC_PARAM_INT	Packets reaching this node are buffered until at least this many reliable channels are connected.
send_retry_ns	Yes	10000	SC_PARAM_INT	Period for retrying sending packets if ring is full.
drop_notification_retry_ns	Yes	10000000	SC_PARAM_INT	Period for retrying drop notifications if ring is full.
exit_on_disconnect	Yes	0	SC_PARAM_INT	Exit as soon as a client disconnects. This can only be set if max_channels is set to 1.
reliable_mode	Yes	0	SC_PARAM_INT	If this is set, all connections are treated as reliable.

Named Input Links

None

Output Links

Link	Description
""	sc_packet objects received over the shared memory interface.

Exposed Statistics

Statistics exposed by the [sc_shm_broadcast](#) and [sc_shm_import](#) nodes.

Name	Type	Data Type	Description
pkts_dropped	uint64_t	pkt_count	The number of packets dropped by the node.
wake_msgs	uint64_t	ev_count	The number of wake messages.
sleep_notifies	uint64_t	ev_count	The number of sleep notifications.
pkts_in_flight	uint64_t	pkt_count	The number of packets in flight.
reliable_pkts_in_flight	uint64_t	pkt_count	The number of packets in flight in reliable mode.

5.30 sc_shm_export Node Reference

Export packets or messages to a shared memory channel.

Detailed Description

This node is used in conjunction with [sc_shm_import](#) to form a unidirectional shared memory channel between two SolarCapture sessions. Packets delivered to [sc_shm_export](#) are forwarded over the channel to the connected [sc_shm_import](#) instance.

By default [sc_shm_export](#) creates a reliable channel. If packets arrive at this node before a consumer is connected, then they are buffered.

See also [sc_shm_broadcast](#), which supports multiple consumers.

Arguments

Argument	Optional?	Default	Type	Description
path	No		SC_PARAM_STR	The path prefix that should be used for creating the sc_shm_export node listening socket and shared memory files
max_in_flight	Yes	100%	SC_PARAM_INT	Maximum amount of buffering that can be in flight at a time. Specified as a percentage of the incoming pool (" suffix), or in bytes ('B', 'KiB', 'MiB' or 'GiB' suffix).

Named Input Links

Packets arriving on an input link named "foo" are forwarded to an output link named "foo" on the other side of the shared memory channel. Note that these named channels do not support high performance.

5.31 `sc_shm_import` Node Reference

Import packets or messages from a shared memory channel.

Detailed Description

This node is used in conjunction with `sc_shm_broadcast` or `sc_shm_export` to form a unidirectional shared memory channel between two SolarCapture sessions. `sc_shm_import` is the receiving end of the channel. Packets pushed into the channel are emitted by this node on its output.

Arguments

Argument	Optional?	Default	Type	Description
<code>path</code>	No		<code>SC_PARAM_STR</code>	Location in the filesystem for the control socket.
<code>reliable</code>	Yes	0	<code>SC_PARAM_INT</code>	Set to 1 to request a reliable connection (which may cause head-of-line blocking).
<code>active_connect</code>	Yes	1	<code>SC_PARAM_INT</code>	If set to 0 then a listening socket is created at the path provided, and the remote side should do an active open.

Named Input Links

Packets arriving on an input link named "foo" are forwarded to an output link named "foo" on the other side of the shared memory channel. Note that these named channels do not support high performance.

Exposed Statistics

Statistics exposed by the `sc_shm_broadcast` and `sc_shm_import` nodes.

Name	Type	Data Type	Description
<code>pkts_dropped</code>	<code>uint64_t</code>	<code>pkt_count</code>	The number of packets dropped by the node.
<code>wake_msgs</code>	<code>uint64_t</code>	<code>ev_count</code>	The number of wake messages.
<code>sleep_notifies</code>	<code>uint64_t</code>	<code>ev_count</code>	The number of sleep notifications.
<code>pkts_in_flight</code>	<code>uint64_t</code>	<code>pkt_count</code>	The number of packets in flight.
<code>reliable_pkts_in_flight</code>	<code>uint64_t</code>	<code>pkt_count</code>	The number of packets in flight in reliable mode.

5.32 `sc_sim_work` Node Reference

Simulate doing CPU intensive work on each packet.

Detailed Description

Simulate the behaviour of a node that performs CPU intensive work on the packets it handles.

Arguments

Argument	Optional?	Default	Type	Description
per_packet_ns	Yes	0	SC_PARAM_INT	Amount of per-packet work
per_batch_ns	Yes	0	SC_PARAM_INT	Amount of per-batch work
touch_wrapper	Yes	0	SC_PARAM_INT	Whether to touch per-packet wrapper
touch_payload	Yes	0	SC_PARAM_INT	Whether to read frame data

5.33 sc_snap Node Reference

Node that limits the length of a packet buffer.

Detailed Description

Node that limits the length of a packet buffer. Packets forwarded by this node are modified as follows: If the length of the payload area exceeds the snap length, then it is reduced to the snap length. The frame_len field is not modified.

Arguments

Argument	Optional?	Default	Type	Description
snap	No		SC_PARAM_INT	Maximum payload length.

5.34 sc_subnode_helper Node Reference

Node used as a sub-node to manage inputs and/or pools.

Detailed Description

This node is used as a subnode to manage buffered input and/or outputs that use a packet pool. It is never instantiated on its own. Here are two example scenarios where this node is helpful:

- 1) Generating output based on the input without modifying the input. The top-level node instantiates a pool and an [sc_subnode_helper](#), and directs its input to the subnode. Incoming buffers are placed in the backlog, and the handler is invoked when the backlog is non-empty and the pool has buffers. ie. When the resources are available to make progress.
- 2) Keeping multiple inputs separate. When a node receives buffers from multiple input links it is not possible to tell which buffers came from which input. A solution is to use an `nt_select_subnode_fn()` handler to instantiate a subnode for each distinct input.

When packets are received by an [sc_subnode_helper](#) they are appended to a backlog list ([sc_subnode_helper::sh_backlog](#)). The backlog handler is invoked repeatedly until either the backlog is emptied or the packet handler leaves the backlog unmodified. If [sc_subnode_helper::sh_pool_threshold](#) is set, then the backlog handler is only invoked so long as the pool has at least the requested number of buffers available.

If `sc_subnode_helper::sh_handle_end_of_stream_fn` is set then it is invoked when the end-of-stream signal has been received and the backlog is empty. If `sh_pool_threshold` is also set, the end-of-stream handler is only invoked when the pool has at least the requested number of buffers available.

If `sc_subnode_helper::sh_poll_backlog_ns` is set then the backlog handler is invoked periodically whenever the backlog is non-empty, even if the pool threshold has been set and not yet reached.

When `'with_pool=1'`, a packet pool is allocated and a pointer stored at `sc_subnode_helper::sh_pool`. The attributes of the pool are set by the attributes passed to the node allocation function.

Alternatively `sh_pool` can be set to point at a pool allocated elsewhere (eg. by the parent node). This is useful when implementing nodes that forward information from inputs to outputs, but in new buffers.

By default, a link for freeing packets is allocated and placed in `sc_subnode_helper::sh_free_link`. If `with_free_link=0` then a free link is not allocated. If `with_free_link=2` then a free link is only allocated if the node has no other outgoing links.

Any outgoing links added to the node are made available via `sc_subnode_helper::sh_links` and `sc_subnode_helper::sh_n_links`. If no links are added then a copy of `sh_free_link` (if requested) is placed at `sh_links[0]`. This allows access `sh_links[0]` without having to check whether any links were added.

See also `sc_subnode_helper` for further details of the interface to this node.

Arguments

Argument	Optional?	Default	Type	Description
<code>with_pool</code>	Yes	0	<code>SC_PARAM_INT</code>	Whether to allocate a pool.
<code>with_free_link</code>	Yes	1	<code>SC_PARAM_INT</code>	Whether to allocate a free link.

Output Links

You can add an arbitrary set of outgoing links to this node, and they are made available via `sc_subnode_helper::sh_links`.

Exposed Statistics

Batch limiter statistics are exposed by the `sc_subnode_helper` node.

Name	Type	Data Type	Description
<code>backlog_len</code>	<code>uint64_t</code>	<code>pkt_count</code>	Number of packets in backlog.

5.35 sc_tap Node Reference

Forward input to output, and a copy of input to the 'tap' output with optional filtering.

Detailed Description

Forward input to output and a copy of input to the 'tap' output. If a BPF or predicate filter is specified, only packets matching the filter are duplicated to the 'tap' output.

The node can be placed in one of two modes:

- default:
 - Input packets are always forwarded to "" immediately.
 - If buffers are available, they are copied to "tap" immediately, otherwise they never go to tap.
- Reliable:
 - If buffers are available, all packets are forwarded to "" and "tap" immediately.
 - If not, they are delayed until buffers are available and then forwarded to "" and "tap" at that point.

Note: In reliable mode this node can potentially create a backlog large enough to provoke drops in an upstream node or VI.

Arguments

Argument	Optional?	Default	Type	Description
snap	Yes	0	SC_PARAM_INT	Copy at most n bytes of the duplicated frames, set to 0 to disable.
reliable	Yes	0	SC_PARAM_INT	Set to 1/0 to enable/disable reliable mode.
bpf	Yes	NULL	SC_PARAM_STR	Filter to select packets to duplicate in BPF format.
predicate	Yes	NULL	SC_PARAM_OBJ	Predicate object to select packets to duplicate.

Note: At most one of bpf and predicate may be specified.

Named Input Links

None

Output Links

Link	Description
""	All packets are sent down this link.
"tap"	The copy of the input.

5.36 sc_timestamp_filter Node Reference

Filter packets, accepting only those in a given range of timestamps.

Detailed Description

The range can be specified either as a start and end timestamp given in seconds since 1970, or as a time range given as a string.

Arguments

Argument	Optional?	Default	Type	Description
start_time	Yes	-1.0	SC_PARAM_DBL	Start of time range to accept (seconds since 1970).
end_time	Yes	-1.0	SC_PARAM_DBL	End of time range to accept (seconds since 1970).
range	Yes	NULL	SC_PARAM_STR	Time range over which packets are accepted.

The `range` argument takes the form `START-END`, where either `START` or `END` may be omitted. `START` and `END` should take one of the following forms:

Format	Description
X.Y[smh]	Time in seconds since first packet in input file
+X.Y[smh]	Time in seconds since start of time range (END only)
HH:MM:SS	Time of day
YYYY/MM/DD HH:MM:SS	Absolute time and date

When a time is given without a date, then the date is the date of the start of the range (if given) or otherwise the date of that first packet in the input.

Exposed Statistics

Statistics exposed by the `sc_filter`, `sc_range_filter` and `sc_timestamp_filter` nodes.

Name	Type	Data Type	Description
pkts_rejected	uint64_t	pkt_count	The number of packets not matched by the filter.

5.37 sc_token_bucket_shaper Node Reference

This node performs traffic shaping using the token bucket algorithm.

Detailed Description

This node performs traffic shaping using the token bucket algorithm. It can be used to limit packet rate (`max_pps`), or bandwidth (`max_bps`). It can also be used to limit a blend of packet rate and bandwidth by setting `max_bps` and `overhead`.

Arguments

Argument	Optional?	Default	Type	Description
max_pps	Yes	-	SC_PARAM_DBL	Maximum packet rate in packets-per-second.
max_bps	Yes	-	SC_PARAM_DBL	Maximum bandwidth in bits-per-second.
overhead	Yes	0	SC_PARAM_INT	Per packet overhead in bytes (used with <code>max_bps</code>).
show_config	Yes	0	SC_PARAM_INT	When this is set to 1 the configuration is written to <code>stderr</code> at startup.

Named Input Links

None

Output Links

Link	Description
""	All packets are sent down this link.

5.38 sc_tracer Node Reference

Write debug trace to standard error.

Detailed Description

This node forwards its input to its output, and prints debug traces to the standard error stream. A message is emitted for each packet buffer forwarded, giving various information about the packet buffer.

The following output formats are available:

- `trace`: [sc_packet](#) metadata, in human-readable format
- `hexdump`: payload, as a hexadecimal dump
- `print`: payload, as printed strings.

Arguments

Argument	Optional?	Default	Type	Description
mode	Yes	trace	SC_PARAM_STR	Either "trace", "hexdump", or "print".

5.39 sc_ts_adjust Node Reference

Adjust packet buffer timestamps

Detailed Description

This node adjusts the timestamps on the packet buffers passing through. Timestamps can be adjusted by a constant offset, and inter-packet gaps can also be scaled, or set to a fixed packet rate or bandwidth.

This node is often used together with [sc_pacer](#) to emit packets in real-time (or speeded up or slowed down). The `sc_ts_adjust` node is used to modify the timestamps in the input so that they give the desired transmit time, and `sc_pacer` holds buffers up until their transmit time is reached.

Here is an example using the Python bindings that reads packets from a PCAP file, and transmits them through interface eth4. The packets are transmitted at a rate of 1000 packets per second, and transmitting starts 5 seconds after the process begins:

```
reader = thread.new_node('sc_reader', args=dict(filename="pkts.pcap", prefill="all-input"))
ts_adjust = thread.new_node('sc_ts_adjust', args=dict(start_now=1, offset=5, pps=1000))
pacer = thread.new_node('sc_pacer')
injector = thread.new_node('sc_injector', args=dict(interface="eth4"))
reader.connect(ts_adjust).connect(pacer).connect(injector)
```

Arguments

Argument	Optional?	Default	Type	Description
start_now	Yes	0	SC_PARAM_INT	Make the first packet timestamp be "now"
start_at	Yes	0.0	SC_PARAM_DBL	Make the first packet timestamp be the one given by this argument (in seconds since 1970)
offset	Yes	0.0	SC_PARAM_DBL	Adjust timestamps by the given relative offset (seconds)
speedup	Yes	1.0	SC_PARAM_DBL	Adjust inter-packet gap to speed up or slow down by the given factor
pps	Yes	0.0	SC_PARAM_DBL	Adjust inter-packet gap to give fixed packet rate
fixed_rate	Yes	0.0	SC_PARAM_DBL	For backward compatibility only (same as pps)
bw	Yes	0.0	SC_PARAM_DBL	Adjust inter-packet gap to give fixed bandwidth

NB. The fixed bandwidth mode (bw) uses the actual payload length given by `sc_packet_bytes()` rather than the `frame_len` field.

5.40 sc_tunnel Node Reference

A node used to pass `sc_packet` objects between two SolarCapture sessions via a TCP socket.

Detailed Description

This node establishes a TCP connection between two SolarCapture node graphs so that you can pass packets between them. Packets arriving on an input link are forwarded over the connection to an output link on the other side that has the same name. Each `sc_tunnel` can support multiple input and output links, so that multiple separate channels are created.

Arguments

Argument	Optional?	Default	Type	Description
connect_to	Yes		SC_PARAM_STR	Connect to specified "host:port".
server_name	Yes		SC_PARAM_STR	Hostname or IP address of the server interface to connect to if active, or to bind to if passive.
server_port	Yes		SC_PARAM_STR	The TCP port number of the server to connect to if active, or to bind to if passive.
socket_fd	Yes		SC_PARAM_INT	A file descriptor that is a connected stream socket (in which case <code>server_name</code> , <code>server_port</code> , <code>passive_open</code> and <code>connect_to</code> are not used).
remote_args	Yes		SC_PARAM_STR	Opaque message to send to remote side prior to starting <code>sc_tunnel</code> protocol.
passive_open	Yes		SC_PARAM_INT	Should this node be opened in passive mode? Defaults to passive mode unless <code>connect_to</code> is set.
recv_buf_size	Yes	32k	SC_PARAM_INT	Socket receive buffer size (note: does not constrain message size)

max_msg_size	Yes	> 1514	SC_PARAM_INT	Maximum supported message size; by default is large enough to hold any non-jumbo frame
--------------	-----	--------	--------------	--

To make an active connection: Set `connect_to=host:port` or set `passive_open=0`, `server_name=host` and `server_port=port`.

To make a passive connection: Set `server_port=port` and optionally `server_host=host` if you want to bind to a specific IP interface.

The `remote_args` feature is typically used on the client side when connecting to a server process. The specified message (not including nul) is sent to the server immediately after the connection is established, preceded by its length encoded as a 32-bit integer in network byte order. It can be used to give information to the server about the service required, which the server can then use when setting up the node graph that the client will interact with. Note that `sc_tunnel` does not itself consume this message: It must be consumed by the application prior to passing the socket to an `sc_tunnel` instance using the `socket_fd` argument.

Named Input Links

Packets arriving on an input link named "foo" are forwarded to an output link named "foo" on the other side.

Output Links

Link	Description
"#exit"	Receives an end-of-stream indication once end of stream is signalled on all inputs and outputs and outstanding data is sent.

5.41 sc_tuntap Node Reference

Pass packets between SolarCapture and the kernel stack via a tun/tap interface.

Detailed Description

Packets sent to this node are forwarded to the kernel stack via the tun/tap interface. Packets sent to the tun/tap interface by the kernel stack are delivered through the node's output link.

You can also create an `sc_tuntap` node indirectly by creating an `sc_vi_node` with an interface name such as "tap:tap0".

Arguments

Argument	Optional?	Default	Type	Description
interface	No		SC_PARAM_STR	Name for the tun/tap interface.
up	Yes	1	SC_PARAM_INT	Whether or not to bring the interface up.

5.42 sc_vi_node Node Reference

A node which passes packets to and/or from a network interface.

Detailed Description

This node passes packets to and/or from a network interface.

Packets arriving at this node are passed to the indicated network interface. Packets received from the network interface are passed to this node's output link.

This node creates an [sc_vi_node](#) if an output link is added, and creates an [sc_injector](#) if an incoming link is added.

The 'streams' argument is used to indicate which packets from the interface should be captured on the receive path. This is analogous to calling [sc_vi_add_stream\(\)](#).

If the interface name looks like "tap:name" then an [sc_tuntap](#) node is instantiated.

Arguments

Argument	Optional?	Default	Type	Description
interface	No		SC_PARAM_STR	Name of the network interface.
streams	Yes	"all"	SC_PARAM_STR	',' separated list of streams to be captured on receive.

5.43 sc_vss Node Reference

Replace packet buffer timestamp with timestamp generated by VSS packet broker, and demultiplex by VSS port.

Detailed Description

This node is used with VSS packet brokers. These devices append a record to packets which can include one or both of a timestamp and a port number.

If a timestamp is present, then it replaces the SolarCapture timestamp. If the port field is present it is used to select an outgoing link. If the port field is not present, the port is treated as 0.

Outgoing links whose name is an integer receive packets with the corresponding VSS port. An outgoing link named "" receives any packets whose port does not match a named outgoing link.

Arguments

Argument	Optional?	Default	Type	Description
strip	Yes	1	SC_PARAM_INT	Whether to strip the VSS record
timestamp	Yes	1	SC_PARAM_INT	Whether the timestamp field is present in the VSS record
portstamp	Yes	1	SC_PARAM_INT	Whether the port field is present in the VSS record

5.44 sc_writer Node Reference

Node that writes packets to a file in pcap format.

Detailed Description

The `sc_writer` node writes incoming packets to a file.

By default `O_DIRECT` and asynchronous-I/O are used to maximise performance if the underlying filesystem supports those features.

Arguments

Argument	Optional?	Default	Type	Description
filename	No		SC_PARAM_STR	Name of file to write to, or filename template when using file rotation
format	Yes	pcap	SC_PARAM_STR	File format. One of: pcap (microsecond timestamps) or pcap-ns (nanosecond timestamps).
on_error	Yes	exit	SC_PARAM_STR	What to do if an error is generated. One of: exit, abort, message or silent.
append	Yes	0	SC_PARAM_INT	Set to 1 to append if file exists. (Not compatible with file rotation).
rotate_seconds	Yes	0	SC_PARAM_INT	Rotate to a new file every n seconds.
rotate_file_size	Yes	0	SC_PARAM_INT	Rotate to a new file when file exceeds given size in bytes.
snap	Yes	0	SC_PARAM_INT	Maximum number of bytes of packet data to store. By default whole packets are stored.
sync_on_close	Yes	0	SC_PARAM_INT	Set to 1 to cause an <code>fsync()</code> when a file is closed.

Named Input Links

Input links may be named, in which case the packets are forwarded to a matching named output link.

Output Links

Link	Description
""	Packets from all inputs are forwarded to this link.
"#packed"	Buffers containing the on-disk format are forwarded to this link (if they are generated).
NAME	If NAME matches the name of an input link, then input packets are forwarded to the corresponding output link.

Exposed Statistics

Statistics exposed by the `sc_writer` node.

Name	Type	Data Type	Description
cap_bytes	uint64_t	byte_count	Sum of payload bytes.
link_bytes	uint64_t	byte_count	Sum of frame_len (bytes on wire before snapping).
write_bytes	uint64_t	byte_count	Sum of bytes written to disk.

Chapter 6

Statistics

This part of the documentation describes the statistics available in SolarCapture.

Statistics	Description
sc_arista_ts	Node stats from sc_arista_ts node, stats exposed depend on the value of the switch_model argument.
sc_batch_limiter	Statistics exposed by the sc_batch_limiter node.
sc_filter	Statistics exposed by the sc_filter , sc_range_filter and sc_timestamp_filter nodes.
sc_flow_balancer	Statistics exposed by the sc_flow_balancer node.
sc_pcap_packer	Statistics exposed by the sc_pcap_packer node.
sc_rate_monitor	Statistics exposed by the sc_rate_monitor node.
sc_shm	Statistics exposed by the sc_shm_broadcast and sc_shm_import nodes.
sc_subnode_helper	Batch limiter statistics are exposed by the sc_subnode_helper node.
sc_writer	Statistics exposed by the sc_writer node.

6.1 [sc_arista_ts](#) Statistics Reference

Node stats from [sc_arista_ts](#) node, stats exposed depend on the value of the [switch_model](#) argument.

Modes

[sc_arista_ts, switch_model=7150](#) | Arista timestamp statistics that are exposed by the [sc_arista_ts](#) node when [switch_model=7150](#).

[sc_arista_ts, switch_model=7280, ts_format=64bit](#) | Arista timestamp statistics that are exposed by the [sc_arista_ts](#) node when [switch_model=7280](#), [ts_format=64bit](#).

[sc_arista_ts, switch_model=7280, ts_format=48bit](#) | Arista timestamp statistics that are exposed by the [sc_arista_ts](#) node when [switch_model=7280](#), [ts_format=48bit](#).

6.1.1 [sc_arista_ts, switch_model=7150](#)

Arista timestamp statistics that are exposed by the [sc_arista_ts](#) node when [switch_model=7150](#).

Name	Type	Data Type	Description
max_host_t_delta	double	config	Max delta in seconds the node can compute a tick-delta over.
max_freq_error	double	config	Max ppm allowed between measured and expected tick frequency before entering no sync state.
lost_sync_ms	int	config	Time in milliseconds spent in lost sync state.
no_sync_ms	int	config	Time in milliseconds spent in no sync state.
exp_tick_freq	int	config	The expected tick frequency in Hz.
strip_ticks	int	config	1 if the node is stripping ticks 0 otherwise.
log_level	int	config	The log level.
tick_freq	double	magnitude	The measured tick frequency in Hz.
n_keyframes	uint64_t	pkt_count	Number of keyframes processed by the node.
n_filtered_oui	uint64_t	pkt_count	Number of packets filtered out by OUI.
n_filtered_other	uint64_t	pkt_count	Number of packets filtered out for other reasons.
n_skew_zero_ticks	uint64_t	pkt_count	Number of packets without a timestamp (ticks is zero).
n_lost_sync	uint64_t	pkt_count	Number of packets processed whilst in lost sync state.
n_no_sync	uint64_t	pkt_count	Number of packets processed whilst in no sync state.
n_kf_len_mismatch	uint64_t	pkt_count	Number of packets received where the keyframe length did not match.
n_kf_dev_mismatch	uint64_t	pkt_count	Number of packets received where the device field did not match.
n_kf_bad_fcs_type	uint64_t	pkt_count	Number of keyframes with a bad FCS.
kf_switch_drops	uint64_t	pkt_count	Number of keyframes dropped by the switch.
n_kf_big_gap	uint64_t	pkt_count	Number of large gaps between keyframes.
n_skew	uint64_t	pkt_count	Number of skews.
n_host_ts_misorder	uint64_t	ev_count	Host timestamp detected out-of-order.
n_kf_host_ts_misorder	uint64_t	ev_count	Host timestamp of keyframe detected out-of-order.
enter_no_sync	uint64_t	ev_count	Number of times the node has entered no sync state.
enter_sync1	uint64_t	ev_count	Number of times the node has entered sync1 state.
enter_sync2	uint64_t	ev_count	Number of times the node has entered sync2 state.
enter_lost_sync	uint64_t	ev_count	Number of times the node has entered lost sync state.

6.1.2 sc_arista_ts, switch_model=7280, ts_format=64bit

Arista timestamp statistics that are exposed by the [sc_arista_ts](#) node when switch_model=7280, ts_format=64bit.

Name	Type	Data Type	Description
strip_ticks	int	config	1 if the node is stripping ticks 0 otherwise.
rollover_window_ns	uint64_t	config	The window over which the node is checking for the rollover bug.
last_good_delta_ns	int64_t	time delta	The last measured time delta between arista and NIC times from packets outside the rollover window.

6.1.3 sc_arista_ts, switch_model=7280, ts_format=48bit

Arista timestamp statistics that are exposed by the [sc_arista_ts](#) node when switch_model=7280, ts_format=48bit.

Name	Type	Data Type	Description
strip_ticks	int	config	1 if the node is stripping ticks 0 otherwise.
replace_src_mac	int	config	1 if the node is replacing source mac 0 otherwise.
n_filtered_oui	uint64_t	pkt_count	Number of packets filtered out by OUI.
n_filtered_arista	uint64_t	pkt_count	Number of packets filtered out because of invalid Arista field.
n_filtered_other	uint64_t	pkt_count	Number of packets filtered out for some other reasons.
n_rollover	uint64_t	pkt_count	Number of packets with seconds rollover.

6.2 sc_batch_limiter Statistics Reference

Statistics exposed by the [sc_batch_limiter](#) node.

Name	Type	Data Type	Description
max_packets	int	config	The maximum number of packets sent per batch.
fwd_on_idle	int	config	Set to 1 if mode is on_idle else 0.
backlog	int	pkt_count	The current number of packets waiting to be forwarded.

6.3 sc_filter Statistics Reference

Statistics exposed by the [sc_filter](#), [sc_range_filter](#) and [sc_timestamp_filter](#) nodes.

Name	Type	Data Type	Description
pkts_rejected	uint64_t	pkt_count	The number of packets not matched by the filter.

6.4 sc_flow_balancer Statistics Reference

Statistics exposed by the [sc_flow_balancer](#) node.

Name	Type	Data Type	Description
flow_table_capacity	uint64_t	magnitude	Capacity of the flow table.
avg_flow_load	uint64_t	bandwidth	Moving average of the load per flow.
n_flows	int	magnitude	Current number of flows directed to this output.
total_flows	uint64_t	magnitude	Total number of flows directed to this output.
total_work	uint64_t	magnitude	Estimate of total work directed to this output.
load_est_short	uint64_t	bandwidth	Short-term load estimate for this output.
load_est_long	uint64_t	bandwidth	Long-term load estimate for this output.
drops	uint64_t	pkt_count	Number of packets dropped at this output due to running out of buffering.

6.5 sc_pcap_packer Statistics Reference

Statistics exposed by the [sc_pcap_packer](#) node.

Name	Type	Data Type	Description
pcap_bytes	uint64_t	byte_count	Sum of bytes of encapsulated data send to output.
buffer_low	uint64_t	ev_count	Number of times the pool of buffers has run out.

6.6 sc_rate_monitor Statistics Reference

Statistics exposed by the [sc_rate_monitor](#) node.

Name	Type	Data Type	Description
pkt_rate	int	pkt_rate	Packet rate (packets/second).
cap_bytes	uint64_t	byte_count	Sum of payload bytes.
link_bytes	uint64_t	byte_count	Sum of frame_len (bytes on wire before snapping).
cap_bw	uint64_t	bandwidth	Payload bandwidth (bits/second).
link_bw	uint64_t	bandwidth	Bandwidth before snap (bits/second) (from frame_len field).

6.7 sc_shm Statistics Reference

Statistics exposed by the [sc_shm_broadcast](#) and [sc_shm_import](#) nodes.

Name	Type	Data Type	Description
pkts_dropped	uint64_t	pkt_count	The number of packets dropped by the node.
wake_msgs	uint64_t	ev_count	The number of wake messages.
sleep_notifies	uint64_t	ev_count	The number of sleep notifications.
pkts_in_flight	uint64_t	pkt_count	The number of packets in flight.
reliable_pkts_in_flight	uint64_t	pkt_count	The number of packets in flight in reliable mode.

6.8 sc_subnode_helper Statistics Reference

Batch limiter statistics are exposed by the [sc_subnode_helper](#) node.

Name	Type	Data Type	Description
backlog_len	uint64_t	pkt_count	Number of packets in backlog.

6.9 sc_writer Statistics Reference

Statistics exposed by the [sc_writer](#) node.

Name	Type	Data Type	Description
cap_bytes	uint64_t	byte_count	Sum of payload bytes.
link_bytes	uint64_t	byte_count	Sum of frame_len (bytes on wire before snapping).
write_bytes	uint64_t	byte_count	Sum of bytes written to disk.

Chapter 7

Data Structure Index

7.1 Data Structures

Here are the data structures with brief descriptions:

sc_append_to_list	Private state of sc_append_to_list node	61
sc_arg	Representation of an argument. Used by node init functions	62
sc_attr	Attribute object	63
sc_callback	A callback object	64
sc_dlist	Doubly linked list pointers	65
sc_hash_table	A hash table	65
sc_iovec_ptr	An sc_iovec_ptr provides a convenient way to iterate over an iovec array without modifying it	66
sc_node	Description of a node	67
sc_node_factory	Struct to hold information about how to create an instance of this node	68
sc_node_link	Description of a link the node has	69
sc_node_type	Describes a type of node	70
sc_object	An opaque object. Use this to pass all types of data that are not ints, doubles or char arrays (see SC_PARAM_INT , SC_PARAM_DBL and SC_PARAM_STR respectively for these) to nodes	71
sc_packed_packet	A packed-stream packet	72
sc_packet	Representation of a packet	74
sc_packet_list	A list of packets or packet buffers	76
sc_pkt_predicate	A packet predicate object	78

sc_session_error	A SolarCapture session error object returned by sc_session_error_get	78
sc_stream	A stream object	80
sc_subnode_helper	sc_subnode_helper node private state	80
sc_vi	A VI object	85

Chapter 8

File Index

8.1 File List

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

append_to_list.h	Private state of sc_append_to_list node	87
args.h	Sc_arg: An argument to a node's initialisation function	87
attr.h	Sc_attr: Control optional behaviours and tunables	90
declare_types.h	This header is used to generate C type definitions and corresponding runtime type information for data structures that are shared by SolarCapture with other processes	95
dlist.h	Sc_dlist: A doubly-linked list	97
ethernet.h	Ethernet protocol definitions	104
event.h	Sc_callback: Interface for event notification	104
ext_node.h	Interface for writing custom nodes	109
ext_packet.h	Sc_packet: The representation of a packet or other data	123
ext_packet_list.h	Sc_packet_list: A list of packets	126
hash_table.h	A hash table with open addressing and double hashing	129
iovec.h	Sc_iovec_ptr: Supports iterating over a 'struct iovec'	135
ip.h	IP protocol definitions	140
mailbox.h	Sc_mailbox: A means to pass packets from one thread to another	142
misc.h	Miscellaneous utility functions	145
node.h	Sc_node: An object that processes packets	145

object.h	Sc_object: Opaque object interface. Use this to pass all types of data that are not ints, doubles or char arrays (see SC_PARAM_INT , SC_PARAM_DBL and SC_PARAM_STR respectively for these)	153
packed_stream.h	Sc_packed_packet: The packed-stream encapsulation	156
pkt_pool.h	Sc_pool: A pool of packet buffers	160
predicate.h	Sc_pkt_predicate: Interface for testing properties of packets	165
session.h	Sc_session: A set of threads and other objects	167
stream.h	This header file defines sc_stream objects for directing packets to a sc_vi instance. A packet must match all the stream criteria for it to be directed by the stream to an sc_vi instance	171
subnode_helper.h	sc_subnode_helper node interface	179
thread.h	Sc_thread: Representation of a thread in SolarCapture	180
time.h	Functions for managing time	184
vi.h	Sc_vi: Supports receiving packets from the network	187

Chapter 9

Data Structure Documentation

9.1 `sc_append_to_list` Struct Reference

Private state of `sc_append_to_list` node.

```
#include <append_to_list.h>
```

Data Fields

- const struct `sc_node_link` * `free_link`
- const struct `sc_node_link` ** `links`
- int `n_links`
- struct `sc_packet_list` * `append_to`

9.1.1 Detailed Description

Private state of `sc_append_to_list` node.

See the `sc_append_to_list` node for details of how this is used.

9.1.2 Field Documentation

9.1.2.1 `append_to`

```
struct sc_packet_list* append_to
```

Application must point this at an initialised packet list.

9.1.2.2 free_link

```
const struct sc_node_link* free_link
```

After 'prep' points to a link that can be used to free packets.

9.1.2.3 links

```
const struct sc_node_link** links
```

After 'prep' points to the node's output links.

9.1.2.4 n_links

```
int n_links
```

After 'prep' gives the number of output links.

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

- [append_to_list.h](#)

9.2 sc_arg Struct Reference

Representation of an argument. Used by node init functions.

```
#include <args.h>
```

Data Fields

- const char * **name**
- enum **sc_param_type** type
- union {
 - const char * **str**
 - int64_t **i**
 - struct **sc_object** * **obj**
 - double **dbl**

9.2.1 Detailed Description

Representation of an argument. Used by node init functions.

9.2.2 Field Documentation

9.2.2.1 name

```
const char* name
```

Parameter name

9.2.2.2 type

```
enum sc\_param\_type type
```

Parameter type

9.2.2.3 val

```
union { ... } val
```

Parameter value

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

- [args.h](#)

9.3 [sc_attr](#) Struct Reference

Attribute object.

```
#include <attr.h>
```

9.3.1 Detailed Description

Attribute object.

Attributes are used to specify optional behaviours and parameters, usually when allocating other SolarCapture objects. Each attribute object defines a complete set of the attributes that SolarCapture understands.

For example, the "affinity_core" attribute controls which CPU core an `sc_thread` runs on.

Functions to create and manage attributes are in the file [attr.h](#).

The default values for attributes may be overridden by setting the environment variable `SC_ATTR`. For example:

```
SC_ATTR="log_level=3;snap=2"
```

Each function that takes an attribute argument will only be interested in a subset of the attributes specified by an [sc_attr](#) instance. Other attributes are ignored.

The set of attributes supported by SolarCapture may change between releases, so applications should where possible tolerate failures when setting attributes.

Attribute objects are not associated with `sc_session` objects, so the SolarCapture error status is not set when functions that operate on attributes report an error. Instead, functions that operate on attributes return 0 on success and a negative error code otherwise.

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

- [attr.h](#)

9.4 `sc_callback` Struct Reference

A callback object.

```
#include <event.h>
```

Data Fields

- `void * cb_private`
- `sc_callback_handler_fn * cb_handler_fn`
- `struct sc_dlist cb_link`

9.4.1 Detailed Description

A callback object.

Callback objects provide a way to be notified when an event of interest occurs.

9.4.2 Field Documentation

9.4.2.1 `cb_handler_fn`

```
sc_callback_handler_fn* cb_handler_fn
```

Callback function to be invoked when the event of interest occurs.

9.4.2.2 `cb_link`

```
struct sc_dlist cb_link
```

Internal use only.

9.4.2.3 `cb_private`

```
void* cb_private
```

Private state for the implementation.

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

- [event.h](#)

9.5 `sc_dlist` Struct Reference

Doubly linked list pointers.

```
#include <dlist.h>
```

Data Fields

- struct `sc_dlist` * `prev`
- struct `sc_dlist` * `next`

9.5.1 Detailed Description

Doubly linked list pointers.

9.5.2 Field Documentation

9.5.2.1 `next`

```
struct sc_dlist* next
```

A pointer to next item in list (set to itself if it is at the end of the list).

9.5.2.2 `prev`

```
struct sc_dlist* prev
```

A pointer to previous item in list (set to itself if it is at the start of the list).

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

- [dlist.h](#)

9.6 `sc_hash_table` Struct Reference

A hash table.

```
#include <hash_table.h>
```

9.6.1 Detailed Description

A hash table.

This is an opaque pointer to a hash table created using `sc_hash_table_alloc()`.

NOTE: Hash tables are only supported on x86_64 and x86 CPUs with the SSE 4.2 instruction set. In particular, the CRC32 instruction is required.

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

- [hash_table.h](#)

9.7 `sc_iovec_ptr` Struct Reference

An `sc_iovec_ptr` provides a convenient way to iterate over an iovec array without modifying it.

```
#include <iovec.h>
```

Data Fields

- const struct iovec * `io`
- int `ioflen`
- struct iovec `io`

9.7.1 Detailed Description

An `sc_iovec_ptr` provides a convenient way to iterate over an iovec array without modifying it.

9.7.2 Field Documentation

9.7.2.1 `io`

```
struct iovec io
```

Currently iterated iovec

9.7.2.2 `io`

```
const struct iovec* io
```

Pointer to start of array

9.7.2.3 iovlen

```
int iovlen
```

Length of iovec array

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

- [iovec.h](#)

9.8 sc_node Struct Reference

Description of a node.

```
#include <ext_node.h>
```

Data Fields

- const struct [sc_node_type](#) * [nd_type](#)
- char * [nd_name](#)
- void * [nd_private](#)

9.8.1 Detailed Description

Description of a node.

This is passed to every function used to call into the node.

9.8.2 Field Documentation

9.8.2.1 nd_name

```
char* nd_name
```

Name of the node, set automatically when creating node. [nd_name](#) is set to attribute name if this provided, otherwise [sc_node_factory.nf_name](#) with a unique node instance number appended

9.8.2.2 nd_private

```
void* nd_private
```

Set by node for local state

9.8.2.3 nd_type

```
const struct sc_node_type* nd_type
```

Type of node, set automatically on creation of the node

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

- [ext_node.h](#)

9.9 sc_node_factory Struct Reference

Struct to hold information about how to create an instance of this node.

```
#include <ext_node.h>
```

Data Fields

- int [nf_node_api_ver](#)
- const char * [nf_name](#)
- const char * [nf_source_file](#)
- void * [nf_private](#)
- [sc_node_init_fn](#) * [nf_init_fn](#)
- void * [nf_reserved](#) [8]

9.9.1 Detailed Description

Struct to hold information about how to create an instance of this node.

9.9.2 Field Documentation

9.9.2.1 nf_init_fn

```
sc_node_init_fn* nf_init_fn
```

Function called by SolarCapture core to initialise a new node.

9.9.2.2 nf_name

```
const char* nf_name
```

Name of this node factory.

9.9.2.3 `nf_node_api_ver`

```
int nf_node_api_ver
```

Minimum version of SolarCapture that this node is compatible with.

9.9.2.4 `nf_private`

```
void* nf_private
```

Private state for the implementation.

9.9.2.5 `nf_reserved`

```
void* nf_reserved[8]
```

Reserved.

9.9.2.6 `nf_source_file`

```
const char* nf_source_file
```

Name of the source file for this node. (Use **FILE** if you like).

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

- [ext_node.h](#)

9.10 `sc_node_link` Struct Reference

Description of a link the node has.

```
#include <ext_node.h>
```

Data Fields

- `const char * name`

9.10.1 Detailed Description

Description of a link the node has.

This is passed to the node initialisation function

9.10.2 Field Documentation

9.10.2.1 name

```
const char* name
```

Set when a link is added to the node

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

- [ext_node.h](#)

9.11 sc_node_type Struct Reference

Describes a type of node.

```
#include <ext_node.h>
```

Data Fields

- `const char * nt_name`
- `void * nt_private`
- `sc_node_prep_fn * nt_prep_fn`
- `sc_node_pkts_fn * nt_pkts_fn`
- `sc_node_add_link_fn * nt_add_link_fn`
- `sc_node_select_subnode_fn * nt_select_subnode_fn`
- `sc_node_end_of_stream_fn * nt_end_of_stream_fn`

9.11.1 Detailed Description

Describes a type of node.

This struct describes what functions are responsible for the behaviour of the node.

9.11.2 Field Documentation

9.11.2.1 nt_add_link_fn

```
sc_node_add_link_fn* nt_add_link_fn
```

(Optional) Add an outgoing link.

9.11.2.2 `nt_end_of_stream_fn`

`sc_node_end_of_stream_fn* nt_end_of_stream_fn`

(Optional) Handle end-of-stream signal.

9.11.2.3 `nt_name`

`const char* nt_name`

Name of the node type (set from `sc_node_factory.nf_name`).

9.11.2.4 `nt_pkts_fn`

`sc_node_pkts_fn* nt_pkts_fn`

(Optional) Handle incoming packets.

9.11.2.5 `nt_prep_fn`

`sc_node_prep_fn* nt_prep_fn`

(Optional) Prepare for packet processing.

9.11.2.6 `nt_private`

`void* nt_private`

Private state for the implementation.

9.11.2.7 `nt_select_subnode_fn`

`sc_node_select_subnode_fn* nt_select_subnode_fn`

(Optional) Select target node for an incoming link.

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

- [ext_node.h](#)

9.12 `sc_object` Struct Reference

An opaque object. Use this to pass all types of data that are not ints, doubles or char arrays (see [SC_PARAM_INT](#), [SC_PARAM_DBL](#) and [SC_PARAM_STR](#) respectively for these) to nodes.

```
#include <object.h>
```

9.12.1 Detailed Description

An opaque object. Use this to pass all types of data that are not ints, doubles or char arrays (see [SC_PARAM_INT](#), [SC_PARAM_DBL](#) and [SC_PARAM_STR](#) respectively for these) to nodes.

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

- [object.h](#)

9.13 [sc_packed_packet](#) Struct Reference

A packed-stream packet.

```
#include <packed_stream.h>
```

Data Fields

- [uint16_t ps_next_offset](#)
- [uint8_t ps_pkt_start_offset](#)
- [uint8_t ps_flags](#)
- [uint16_t ps_cap_len](#)
- [uint16_t ps_orig_len](#)
- [uint32_t ps_ts_sec](#)
- [uint32_t ps_ts_nsec](#)

9.13.1 Detailed Description

A packed-stream packet.

Packed-stream is an encapsulation that encodes multiple packets or other data in a buffer. Each packet is represented by an [sc_packed_packet](#) header which gives information about the packet stored and the offset to the next packet in the buffer.

The offset of the last packet in the buffer must generate a pointer that lies beyond the end of the buffer containing packed-stream data.

The following example code shows how to iterate over the set of packets stored in an [sc_packet](#) that contains packed-stream packets:

```
void do_something_to_each(struct sc_packet* pkt)
{
    struct sc_packed_packet* ps_pkt = sc_packet_packed_first(pkt);
    struct sc_packed_packet* ps_end = sc_packet_packed_end(pkt);
    for( ; ps_pkt < ps_end; ps_pkt = sc_packed_packet_next(ps_pkt) )
        do_something(sc_packed_packet_payload(ps_pkt), ps_pkt->
            ps_cap_len);
}
```


9.13.2 Field Documentation

9.13.2.1 ps_cap_len

`uint16_t ps_cap_len`

Number of bytes of packet payload stored.

9.13.2.2 ps_flags

`uint8_t ps_flags`

SC_PS_FLAG_* flags.

9.13.2.3 ps_next_offset

`uint16_t ps_next_offset`

Offset of next packet from start of this struct.

9.13.2.4 ps_orig_len

`uint16_t ps_orig_len`

Original length of the frame.

9.13.2.5 ps_pkt_start_offset

`uint8_t ps_pkt_start_offset`

Offset of packet payload from start of this struct.

9.13.2.6 ps_ts_nsec

`uint32_t ps_ts_nsec`

Timestamp (nanoseconds).

9.13.2.7 ps_ts_sec

```
uint32_t ps_ts_sec
```

Timestamp (seconds).

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

- [packed_stream.h](#)

9.14 sc_packet Struct Reference

Representation of a packet.

```
#include <ext_packet.h>
```

Data Fields

- [uint64_t ts_sec](#)
- [uint32_t ts_nsec](#)
- [uint16_t flags](#)
- [uint16_t frame_len](#)
- [uint8_t frags_n](#)
- [uint8_t iovlen](#)
- [uint16_t reserved1](#)
- [uint32_t reserved2](#)
- [struct iovec * iov](#)
- [struct sc_packet * next](#)
- [struct sc_packet * frags](#)
- [struct sc_packet ** frags_tail](#)
- [uintptr_t * metadata](#)

9.14.1 Detailed Description

Representation of a packet.

This data-structure describes a packet. It includes pointers to the packet contents, meta-data relating to the packet and fields to support creating lists of packets.

Each [sc_packet](#) instance is usually associated with a buffer that holds the packet contents. A packet may span multiple such buffers, in which case the 'head' buffer uses [frags](#) and [frags_tail](#) to identify the remaining buffers (which are linked via the [next](#) field). Nodes should generally not use the [frags](#), [frags_n](#) and [frags_tail](#) fields, because they are sometimes used in special ways. Instead nodes should use [iov](#) and [iovlen](#) to find the buffer(s) underlying an [sc_packet](#).

9.14.2 Field Documentation

9.14.2.1 flags

`uint16_t flags`

flags defined below

9.14.2.2 frags

`struct sc_packet* frags`

list of chained fragments

9.14.2.3 frags_n

`uint8_t frags_n`

number of fragments in `frags` chain

9.14.2.4 frags_tail

`struct sc_packet** frags_tail`

last fragment in chain

9.14.2.5 frame_len

`uint16_t frame_len`

original frame length in bytes

9.14.2.6 iov

`struct iovec* iov`

identifies packet data

9.14.2.7 iovlen

`uint8_t iovlen`

number of entries in `iov` array

9.14.2.8 metadata

`uintptr_t* metadata`

packet metadata

9.14.2.9 next

`struct sc_packet* next`

next packet in a packet list

9.14.2.10 reserved1

`uint16_t reserved1`

reserved

9.14.2.11 reserved2

`uint32_t reserved2`

reserved

9.14.2.12 ts_nsec

`uint32_t ts_nsec`

timestamp (nanoseconds)

9.14.2.13 ts_sec

`uint64_t ts_sec`

timestamp (seconds)

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

- [ext_packet.h](#)

9.15 sc_packet_list Struct Reference

A list of packets or packet buffers.

```
#include <ext_packet_list.h>
```

Data Fields

- struct `sc_packet` * `head`
- struct `sc_packet` ** `tail`
- int `num_pkts`
- int `num_frags`

9.15.1 Detailed Description

A list of packets or packet buffers.

9.15.2 Field Documentation

9.15.2.1 head

```
struct sc_packet* head
```

Head of list

9.15.2.2 num_frags

```
int num_frags
```

Number of pkt frags in the list

9.15.2.3 num_pkts

```
int num_pkts
```

Number of pkts in the list

9.15.2.4 tail

```
struct sc_packet** tail
```

Ptr to next field in tail of list

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

- [ext_packet_list.h](#)

9.16 `sc_pkt_predicate` Struct Reference

A packet predicate object.

```
#include <predicate.h>
```

Data Fields

- `sc_pkt_predicate_test_fn * pred_test_fn`
- `void * pred_private`

9.16.1 Detailed Description

A packet predicate object.

This can be used with an `sc_filter` node to match packets for filtering.

9.16.2 Field Documentation

9.16.2.1 `pred_private`

```
void* pred_private
```

Field to hold state for the predicate function

9.16.2.2 `pred_test_fn`

```
sc_pkt_predicate_test_fn* pred_test_fn
```

The predicate test function. It should return 1 (true) or 0 (false)

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

- [predicate.h](#)

9.17 `sc_session_error` Struct Reference

A SolarCapture session error object returned by `sc_session_error_get`.

```
#include <session.h>
```

Data Fields

- char * [err_msg](#)
- char * [err_func](#)
- char * [err_file](#)
- int [err_line](#)
- int [err_errno](#)

9.17.1 Detailed Description

A SolarCapture session error object returned by [sc_session_error_get](#).

9.17.2 Field Documentation

9.17.2.1 [err_errno](#)

```
int err_errno
```

The errno for the error.

9.17.2.2 [err_file](#)

```
char* err_file
```

The source file the error occurred in.

9.17.2.3 [err_func](#)

```
char* err_func
```

The function the error occurred in.

9.17.2.4 [err_line](#)

```
int err_line
```

The line number the error was issued from.

9.17.2.5 `err_msg`

```
char* err_msg
```

The error message.

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

- [session.h](#)

9.18 `sc_stream` Struct Reference

A stream object.

```
#include <stream.h>
```

9.18.1 Detailed Description

A stream object.

An `sc_stream` object specifies criteria to select packets. The criteria usually refer to fields in packet headers.

Stream objects are used to specify which packets should be steered by an adapter to a SolarCapture application via an `sc_vi` instance.

Fields in this structure are not exposed, and must not be directly accessed. Instead use the functions in [stream.h](#).

Different adapter models, different firmware versions and different firmware modes (or variants) all affect the combinations of header fields and other criteria that can be matched. Attempting to use an unsupported set of criteria may fail when modifying the stream object, or when adding the stream to a VI. For more information, see [sc_stream_set_str\(\)](#).

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

- [stream.h](#)

9.19 `sc_subnode_helper` Struct Reference

`sc_subnode_helper` node private state.

```
#include <subnode_helper.h>
```


Data Fields

- void * [sh_private](#)
- struct [sc_node](#) * [sh_node](#)
- const struct [sc_node_link](#) * [sh_free_link](#)
- const struct [sc_node_link](#) ** [sh_links](#)
- struct [sc_packet_list](#) [sh_backlog](#)
- uint64_t [sh_poll_backlog_ns](#)
- [sc_sh_handle_backlog_fn](#) * [sh_handle_backlog_fn](#)
- struct [sc_pool](#) * [sh_pool](#)
- [sc_sh_handle_end_of_stream_fn](#) * [sh_handle_end_of_stream_fn](#)
- int [sh_pool_threshold](#)
- int [sh_n_links](#)

Related Functions

(Note that these are not member functions.)

- typedef void() [sc_sh_handle_backlog_fn](#)(struct [sc_subnode_helper](#) *sh)
Signature of [sh_handle_backlog_fn](#).
- typedef void() [sc_sh_handle_end_of_stream_fn](#)(struct [sc_subnode_helper](#) *sh)
Signature of [sh_handle_end_of_stream_fn](#).
- static struct [sc_subnode_helper](#) * [sc_subnode_helper_from_node](#) (struct [sc_node](#) *node)
Get [sc_subnode_helper](#) from [sc_node](#)
- void [sc_subnode_helper_request_callback](#) (struct [sc_subnode_helper](#) *sh)
Request that [sc_subnode_helper](#) calls its backlog handler at a safe time.

9.19.1 Detailed Description

[sc_subnode_helper](#) node private state.

9.19.2 Friends And Related Function Documentation

9.19.2.1 `sc_sh_handle_backlog_fn`

```
typedef void() sc_sh_handle_backlog_fn(struct sc\_subnode\_helper *sh) [related]
```

Signature of `sh_handle_backlog_fn`.

Parameters

<code>sh</code>	The sc_subnode_helper instance.
-----------------	---

The backlog handler is responsible for forwarding packets in the backlog to one of the outgoing links. It is invoked when any of the following events occurs:

1. The backlog transitions from empty to non-empty and `sh_pool` (if set) has at least `sh_pool_threshold` buffers available.
2. The backlog is non-empty and the pool fill level increases above `sh_pool_threshold`.
3. Periodically every `sh_backlog_poll_ns` (if non-zero) while the backlog is non-empty.
4. After [sc_subnode_helper_request_callback\(\)](#) is called.

The handler is called repeatedly until either the backlog is empty or the length of the backlog remains unmodified across the callback. Note that when the backlog handler is invoked due to timeout or `request_callback()`, the pool threshold is not considered.

9.19.2.2 `sc_sh_handle_end_of_stream_fn`

```
typedef void() sc_sh_handle_end_of_stream_fn(struct sc\_subnode\_helper *sh) [related]
```

Signature of `sh_handle_end_of_stream_fn`.

Parameters

<code>sh</code>	The sc_subnode_helper instance.
-----------------	---

The end-of-stream handler is invoked when the following conditions are all true:

1. The node has received the end-of-stream signal.
2. The backlog is empty.
3. `sh_pool` (if set) has at least `sh_pool_threshold` buffers available.

If this handler is set, it is responsible for propagating end-of-stream to the outgoing links. If no handler is provided, end-of-stream is automatically propagated to all outputs once the backlog is empty.

9.19.2.3 `sc_subnode_helper_from_node()`

```
static struct sc\_subnode\_helper * sc_subnode_helper_from_node (  
    struct sc\_node * node ) [related]
```

Get [sc_subnode_helper](#) from [sc_node](#)

Parameters

<i>node</i>	Node of type sc_subnode_helper .
-------------	--

Returns

The [sc_subnode_helper](#) from the node

9.19.2.4 [sc_subnode_helper_request_callback\(\)](#)

```
void sc_subnode_helper_request_callback (  
    struct sc\_subnode\_helper * sh ) [related]
```

Request that [sc_subnode_helper](#) calls its backlog handler at a safe time.

Parameters

<i>sh</i>	An sc_subnode_helper instance
-----------	---

9.19.3 Field Documentation

9.19.3.1 [sh_backlog](#)

```
struct sc\_packet\_list sh_backlog
```

Unprocessed incoming packets

9.19.3.2 [sh_free_link](#)

```
const struct sc\_node\_link* sh_free_link
```

A node link for freeing packets (if requested)

9.19.3.3 [sh_handle_backlog_fn](#)

```
sc\_sh\_handle\_backlog\_fn* sh_handle_backlog_fn
```

Handler invoked to process the backlog

9.19.3.4 sh_handle_end_of_stream_fn

```
sc_sh_handle_end_of_stream_fn* sh_handle_end_of_stream_fn
```

Handler invoked when end of stream has been signalled and the backlog is empty

9.19.3.5 sh_links

```
const struct sc_node_link** sh_links
```

Outgoing links

9.19.3.6 sh_n_links

```
int sh_n_links
```

Number of outgoing links

9.19.3.7 sh_node

```
struct sc_node* sh_node
```

The node

9.19.3.8 sh_poll_backlog_ns

```
uint64_t sh_poll_backlog_ns
```

Interval at which to poll backlog handler when backlog is not empty

9.19.3.9 sh_pool

```
struct sc_pool* sh_pool
```

A packet pool (if requested)

9.19.3.10 sh_pool_threshold

```
int sh_pool_threshold
```

Number of buffers that must be available in the pool before calling the backlog handler

9.19.3.11 `sh_private`

```
void* sh_private
```

Private state for the user

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

- [subnode_helper.h](#)

9.20 `sc_vi` Struct Reference

A VI object.

```
#include <vi.h>
```

9.20.1 Detailed Description

A VI object.

Fields in this structure are not exposed, and must not be directly accessed. Instead use the functions in [vi.h](#).

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

- [vi.h](#)

Chapter 10

File Documentation

10.1 `append_to_list.h` File Reference

Private state of `sc_append_to_list` node.

Data Structures

- struct `sc_append_to_list`
Private state of `sc_append_to_list` node.

10.1.1 Detailed Description

Private state of `sc_append_to_list` node.

10.2 `args.h` File Reference

`sc_arg`: An argument to a node's initialisation function.

Data Structures

- struct `sc_arg`
Representation of an argument. Used by node init functions.

Enumerations

- enum `sc_param_type` { `SC_PARAM_STR`, `SC_PARAM_INT`, `SC_PARAM_OBJ`, `SC_PARAM_DBL` }
Possible parameter types that can be used for arguments in a node's init function.

Functions

- static struct `sc_arg SC_ARG_INT` (const char *name, int64_t val)
- static struct `sc_arg SC_ARG_STR` (const char *name, const char *val)
- static struct `sc_arg SC_ARG_OBJ` (const char *name, struct `sc_object` *val)
- static struct `sc_arg SC_ARG_DBL` (const char *name, double val)

10.2.1 Detailed Description

`sc_arg`: An argument to a node's initialisation function.

10.2.2 Enumeration Type Documentation

10.2.2.1 `sc_param_type`

```
enum sc_param_type
```

Possible parameter types that can be used for arguments in a node's init function.

Enumerator

SC_PARAM_STR	const char pointer (nul terminated)
SC_PARAM_INT	signed 64 bit int
SC_PARAM_OBJ	<code>sc_object</code> pointer
SC_PARAM_DBL	native double type

10.2.3 Function Documentation

10.2.3.1 `SC_ARG_DBL()`

```
static struct sc_arg SC_ARG_DBL (
    const char * name,
    double val ) [static]
```

Function to construct a `sc_arg` struct of type `SC_PARAM_DBL`

Parameters

<i>name</i>	Name of argument.
<i>val</i>	Value of argument.

Returns

The constructed `sc_arg` struct

10.2.3.2 SC_ARG_INT()

```
static struct sc_arg SC_ARG_INT (
    const char * name,
    int64_t val ) [static]
```

Function to construct a `sc_arg` struct of type `SC_PARAM_INT`

Parameters

<i>name</i>	Name of argument.
<i>val</i>	Value of argument.

Returns

The constructed `sc_arg` struct

10.2.3.3 SC_ARG_OBJ()

```
static struct sc_arg SC_ARG_OBJ (
    const char * name,
    struct sc_object * val ) [static]
```

Function to construct a `sc_arg` struct of type `SC_PARAM_OBJ`

Parameters

<i>name</i>	Name of argument.
<i>val</i>	Value of argument.

Returns

The constructed `sc_arg` struct

10.2.3.4 SC_ARG_STR()

```
static struct sc_arg SC_ARG_STR (
    const char * name,
    const char * val ) [static]
```

Function to construct a `sc_arg` struct of type `SC_PARAM_STR`

Parameters

<i>name</i>	Name of argument.
<i>val</i>	Value of argument.

Returns

The constructed `sc_arg` struct

10.3 attr.h File Reference

`sc_attr`: Control optional behaviours and tunables.

Functions

- int `sc_attr_alloc` (struct `sc_attr` **attr_out)
Allocate an attribute object.
- void `sc_attr_free` (struct `sc_attr` *attr)
Free an attribute object.
- void `sc_attr_reset` (struct `sc_attr` *attr)
Return attributes to their default values.
- int `sc_attr_set_int` (struct `sc_attr` *attr, const char *name, int64_t val)
Set an attribute to an integer value.
- int `sc_attr_set_str` (struct `sc_attr` *attr, const char *name, const char *val)
Set an attribute to a string value.
- int `sc_attr_set_from_str` (struct `sc_attr` *attr, const char *name, const char *val)
Set an attribute from a string value.
- int `sc_attr_set_from_fmt` (struct `sc_attr` *attr, const char *name, const char *fmt,...)
`__attribute__((format(printf`
Set an attribute to a string value (with formatting).
- int struct `sc_attr` * `sc_attr_dup` (const struct `sc_attr` *attr)
Duplicate an attribute object.
- int `sc_attr_doc` (const char *attr_name_opt, const char ***docs_out, int *docs_len_out)
Returns documentation for attributes. Used by solar_capture_doc.
- struct `sc_object` * `sc_attr_to_object` (const struct `sc_attr` *attr)
Convert an sc_attr to an sc_object.
- const struct `sc_attr` * `sc_attr_from_object` (struct `sc_object` *obj)
Convert an sc_object to an sc_attr.

10.3.1 Detailed Description

`sc_attr`: Control optional behaviours and tunables.

10.3.2 Function Documentation

10.3.2.1 `sc_attr_alloc()`

```
int sc_attr_alloc (
    struct sc_attr ** attr_out )
```

Allocate an attribute object.

Parameters

<code>attr_out</code>	The attribute object is returned here.
-----------------------	--

Returns

- 0 on success, or a negative error code:
- ENOMEM if memory could not be allocated
- EINVAL if the `SC_ATTR` environment variable is malformed.

10.3.2.2 `sc_attr_doc()`

```
int sc_attr_doc (
    const char * attr_name_opt,
    const char *** docs_out,
    int * docs_len_out )
```

Returns documentation for attributes. Used by `solar_capture_doc`.

Parameters

<code>attr_name_opt</code>	The attribute name.
<code>docs_out</code>	On success, the resulting doc string output.
<code>docs_len_out</code>	On success, the length of the doc string output.

Returns

- 0 on success, or a negative error code.

10.3.2.3 `sc_attr_dup()`

```
int struct sc_attr* sc_attr_dup (
    const struct sc_attr * attr )
```

Duplicate an attribute object.

Parameters

<i>attr</i>	The attribute object.
-------------	-----------------------

Returns

A new attribute object.

This function is useful when you want to make non-destructive changes to an existing attribute object.

10.3.2.4 sc_attr_free()

```
void sc_attr_free (
    struct sc_attr * attr )
```

Free an attribute object.

Parameters

<i>attr</i>	The attribute object.
-------------	-----------------------

10.3.2.5 sc_attr_from_object()

```
const struct sc_attr* sc_attr_from_object (
    struct sc_object * obj )
```

Convert an `sc_object` to an `sc_attr`.

Parameters

<i>obj</i>	An <code>sc_object</code> instance or NULL
------------	--

Returns

NULL if `obj` is NULL otherwise the `sc_attr`.

Also returns NULL if `obj` is not of type `SC_OBJ_C_ATTR`.

10.3.2.6 sc_attr_reset()

```
void sc_attr_reset (
    struct sc_attr * attr )
```

Return attributes to their default values.

Parameters

<i>attr</i>	The attribute object.
-------------	-----------------------

10.3.2.7 `sc_attr_set_from_fmt()`

```
int sc_attr_set_from_fmt (
    struct sc_attr * attr,
    const char * name,
    const char * fmt,
    ... )
```

Set an attribute to a string value (with formatting).

Parameters

<i>attr</i>	The attribute object.
<i>name</i>	Name of the attribute.
<i>fmt</i>	Format string for the new attribute value.

Returns

0 on success, or a negative error code:
-ENOENT if *name* is not a valid attribute name
-EINVAL if it is not possible to convert *val* to a valid value for the attribute -EOVERFLOW if *val* is not within the range of values this attribute can take.

This function behaves exactly as `sc_attr_set_from_str()`, except that the string value is generated from a `printf()`-style format string.

10.3.2.8 `sc_attr_set_from_str()`

```
int sc_attr_set_from_str (
    struct sc_attr * attr,
    const char * name,
    const char * val )
```

Set an attribute from a string value.

Parameters

<i>attr</i>	The attribute object.
<i>name</i>	Name of the attribute.
<i>val</i>	New value for the attribute.

Returns

0 on success, or a negative error code:
-ENOENT if *name* is not a valid attribute name
-EINVAL if it is not possible to convert *val* to a valid value for the attribute -EOVERFLOW if *val* is not within the range of values this attribute can take.

10.3.2.9 `sc_attr_set_int()`

```
int sc_attr_set_int (
    struct sc_attr * attr,
    const char * name,
    int64_t val )
```

Set an attribute to an integer value.

Parameters

<i>attr</i>	The attribute object.
<i>name</i>	Name of the attribute.
<i>val</i>	New value for the attribute.

Returns

- 0 on success, or a negative error code:
- ENOENT if `name` is not a valid attribute name
- EOVERFLOW if `val` is not within the range of values this attribute can take.

10.3.2.10 `sc_attr_set_str()`

```
int sc_attr_set_str (
    struct sc_attr * attr,
    const char * name,
    const char * val )
```

Set an attribute to a string value.

Parameters

<i>attr</i>	The attribute object.
<i>name</i>	Name of the attribute.
<i>val</i>	New value for the attribute (may be NULL).

Returns

- 0 on success, or a negative error code:
- ENOENT if `name` is not a valid attribute name
- ENOMSG if the attribute is not a string attribute.

10.3.2.11 `sc_attr_to_object()`

```
struct sc_object* sc_attr_to_object (
    const struct sc_attr * attr )
```

Convert an `sc_attr` to an `sc_object`.

Parameters

<code>attr</code>	An <code>sc_attr</code> instance or NULL
-------------------	--

Returns

NULL if `attr` is NULL otherwise the `sc_object`.

10.4 declare_types.h File Reference

This header is used to generate C type definitions and corresponding runtime type information for data structures that are shared by SolarCapture with other processes.

Macros

- #define `ST_CONSTANT`(name, val) enum { name = val };
A constant value in the template definition.
- #define `ST_STRUCT`(name) struct name {
Start of the template definition.
- #define `ST_FIELD_STR`(name, len, kind) char name[len];
A string field in the template definition.
- #define `ST_FIELD`(type, name, kind) type name;
A C basic type field in the template definition.
- #define `ST_STRUCT_END`};
End of the template definition.

10.4.1 Detailed Description

This header is used to generate C type definitions and corresponding runtime type information for data structures that are shared by SolarCapture with other processes.

In order to create runtime type information a template header file must be created. For example a node called `my_node` could have a template file `my_node_tmpl.h` as follows

```
ST_STRUCT(my_node_stats)
  ST_FIELD(double,    some_stat,    config)
  ST_FIELD(int,      some_other_stat, pkt_count)
  ST_FIELD(double,   another_stat,  magnitude)
  ST_FIELD(double,   yet_one_more_stat, ev_count)
  ...
ST_STRUCT_END
```

In the node source file the node must

1. Define `SC_TYPE_TEMPLATE` to be a header file containing the node's type template definition.
2. Define `SC_DECLARE_TYPES` to be the name of the declaration function to create.

3. Include `declare_types.h`
4. Call the function defined by (2)
5. Call `sc_node_export_state` to allocate a struct of the type defined in the node's type template definition.

Stats can then be updated by changing the values of the fields in the newly created struct from step (5). If stats need to be updated during runtime a means of accessing this struct should be kept in the nodes `nd_private` field.

For example, a node which would like to create a declaration function with name `my_node_declare` using the template `my_node_tmpl.h` would do the following

```
#define SC_TYPE_TEMPLATE <my_node_tmpl.h>
#define SC_DECLARE_TYPES my_node_declare
#include <solar_capture/declare_types.h>

...

static int my_node_init(struct sc_node* node, const struct sc_attr* attr,
                      const struct sc_node_factory* factory)
{
    ...
    my_node_declare(sc_thread_get_session(sc_node_get_thread(node)));
    ...
    struct my_node_stats* stats;
    sc_node_export_state(node, "my_node_stats",
                        sizeof(struct my_node_stats), &stats);
}
```

10.4.2 Macro Definition Documentation

10.4.2.1 ST_CONSTANT

```
#define ST_CONSTANT(
    name,
    val ) enum { name = val };a
```

A constant value in the template definition.

After the node has initialised its shared data structures `name` will be used as the field in the stats struct to update this data.

Parameters

<i>name</i>	The field name.
<i>val</i>	The constant.

10.4.2.2 ST_FIELD

```
#define ST_FIELD(
    type,
    name,
    kind ) type name;
```

A C basic type field in the template definition.

After the node has initialised its shared data structures `name` will be used as the field in the stats struct to update this data.

Parameters

<i>type</i>	The basic data type.
<i>name</i>	The field name.
<i>kind</i>	A string to describe the kind of data. Examples used by SolarCapture nodes are <code>pkt_count</code> , <code>ev_count</code> , <code>config</code> , <code>const</code> , <code>magnitude</code> .

10.4.2.3 ST_FIELD_STR

```
#define ST_FIELD_STR(  
    name,  
    len,  
    kind ) char name[len];
```

A string field in the template definition.

After the node has initialised its shared data structures `name` will be used as the field in the stats struct to update this data.

Parameters

<i>name</i>	The field name.
<i>len</i>	The length of the string.
<i>kind</i>	A string to describe the kind of data. Examples used by SolarCapture nodes are <code>pkt_count</code> , <code>ev_count</code> , <code>config</code> , <code>const</code> , <code>magnitude</code> .

10.4.2.4 ST_STRUCT

```
#define ST_STRUCT(  
    name ) struct name {
```

Start of the template definition.

After the node has initialised its shared data structures the resulting struct type for updating the stats will use `name` for its type.

Parameters

<i>name</i>	The name of the template.
-------------	---------------------------

10.5 dlist.h File Reference

[sc_dlist](#): A doubly-linked list.

Data Structures

- struct `sc_dlist`
Doubly linked list pointers.

Macros

- #define `SC_CONTAINER(c_type, mbr_name, p_mbr)` ((c_type*) ((char*)(p_mbr) - `SC_MEMBER_OFFSET(c_type, mbr_name)`))
Get pointer to container from pointer to member.
- #define `SC_DLIST_FOR_EACH_OBJ(list, iter, mbr)`
Create a for statement that loops over each container item in the list. It is not safe to modify the list using this macro, if list modifications are required see `SC_DLIST_FOR_EACH_OBJ_SAFE`.
- #define `SC_DLIST_FOR_EACH_OBJ_SAFE(list, iter, next_entry, mbr)`
Create a for statement that loops over each container item in the list which can be safely be modified during traversal.

Functions

- static void `sc_dlist_init` (struct `sc_dlist` *list)
Initialise a pre-allocated `sc_dlist` to be an empty doubly linked list.
- static int `sc_dlist_is_empty` (const struct `sc_dlist` *list)
Check if a doubly linked list is empty, returns 1 if true 0 otherwise.
- static void `sc_dlist_push_head` (struct `sc_dlist` *list, struct `sc_dlist` *l)
Prepend an item to the head of a doubly-linked list.
- static void `sc_dlist_push_tail` (struct `sc_dlist` *list, struct `sc_dlist` *l)
Append an item to the tail of a doubly-linked list.
- static void `sc_dlist_remove` (struct `sc_dlist` *l)
Remove an item from the list.
- static struct `sc_dlist` * `sc_dlist_pop_head` (struct `sc_dlist` *list)
Pop off the head of a list.
- static struct `sc_dlist` * `sc_dlist_pop_tail` (struct `sc_dlist` *list)
Pop the tail of a list.
- static void `sc_dlist_rehome` (struct `sc_dlist` *to_list, struct `sc_dlist` *from_list)
Replace an item in a list with another item.

10.5.1 Detailed Description

`sc_dlist`: A doubly-linked list.

A doubly-linked list always has one item with no data at the head. This can be used by embedding `dlist` in a parent struct.

For example:

```
#include <stdio.h>
#include <stdlib.h>
#include <solar_capture.h>

int main()
{
    struct my_struct {
        int my_int;
        double my_double;
        struct sc_dlist list_ptr;
    };

    struct sc_dlist my_list;
    sc_dlist_init(&my_list);
    int i;
    struct my_struct* element;
    // Add some elements to the list
    for( i=0; i < 10; ++i )
    {
        element = malloc(sizeof(struct my_struct));
        element->my_int = i;
        element->my_double = i;
        sc_dlist_push_tail(&my_list, &element->list_ptr);
    }
    // cycle over the list
    SC_DLIST_FOR_EACH_OBJ(&my_list, element, list_ptr)
        printf("element->my_int=%d, element->my_double=%f\n", element->my_int, element->my_double);

    // remove each item from the list
    struct sc_dlist* list_ptr;
    while( !sc_dlist_is_empty(&my_list) ) {
        list_ptr = sc_dlist_pop_tail(&my_list);
        element = SC_CONTAINER(struct my_struct, list_ptr, list_ptr);
        printf("Just popped element with element->my_int=%d", element->my_int);
        free(element);
    }
}
```

10.5.2 Macro Definition Documentation

10.5.2.1 SC_CONTAINER

```
#define SC_CONTAINER(  
    c_type,  
    mbr_name,  
    p_mbr ) ( (c_type*) ((char*) (p_mbr) - SC_MEMBER_OFFSET(c_type, mbr_name)) )
```

Get pointer to container from pointer to member.

Parameters

<i>c_type</i>	The container type.
<i>mbr_name</i>	The name of the member in <i>c_type</i> .
<i>p_mbr</i>	Pointer to the member.

10.5.2.2 SC_DLIST_FOR_EACH_OBJ

```
#define SC_DLIST_FOR_EACH_OBJ(
    list,
    iter,
    mbr )
```

Value:

```
for( (iter) = SC_CONTAINER(sizeof(*(iter)), mbr, (list)->next);
    &(iter)->mbr != (list);
    (iter) = SC_CONTAINER(sizeof(*(iter)), mbr, (iter)->mbr.next) )
```

Create a for statement that loops over each container item in the list. It is not safe to modify the list using this macro, if list modifications are required see [SC_DLIST_FOR_EACH_OBJ_SAFE](#).

Parameters

<i>list</i>	A pointer to the head of the sc_dlist .
<i>iter</i>	A pointer of the same type as the container.
<i>mbr</i>	The name of the field in the container containing the sc_dlist struct.

10.5.2.3 SC_DLIST_FOR_EACH_OBJ_SAFE

```
#define SC_DLIST_FOR_EACH_OBJ_SAFE(
    list,
    iter,
    next_entry,
    mbr )
```

Value:

```
for( (iter) = SC_CONTAINER(sizeof(*(iter)), mbr, (list)->next),
    (next_entry) = SC_CONTAINER(sizeof(*(iter)), mbr, (iter)->mbr.next);
    &(iter)->mbr != (list);
    (iter) = (next_entry),
    (next_entry) = SC_CONTAINER(sizeof(*(iter)), mbr, (iter)->mbr.next) )
```

Create a for statement that loops over each container item in the list which can be safely be modified during traversal.

Parameters

<i>list</i>	A pointer to the head of the sc_dlist .
<i>iter</i>	A pointer of the same type as the container.
<i>next_entry</i>	A pointer of the same type as the container.
<i>mbr</i>	The name of the field in the container containing the sc_dlist struct.

10.5.3 Function Documentation

10.5.3.1 `sc_dlist_init()`

```
static void sc_dlist_init (  
    struct sc_dlist * list ) [inline], [static]
```

Initialise a pre-allocated `sc_dlist` to be an empty doubly linked list.

Parameters

<i>list</i>	A pointer to the pre-allocated <code>sc_dlist</code> to be initialised.
-------------	---

10.5.3.2 `sc_dlist_pop_head()`

```
static struct sc_dlist* sc_dlist_pop_head (  
    struct sc_dlist * list ) [static]
```

Pop off the head of a list.

Parameters

<i>list</i>	The point to pop the head from.
-------------	---------------------------------

Returns

The item popped from `list`.

10.5.3.3 `sc_dlist_pop_tail()`

```
static struct sc_dlist* sc_dlist_pop_tail (  
    struct sc_dlist * list ) [static]
```

Pop the tail of a list.

Parameters

<i>list</i>	The point to pop the tail from.
-------------	---------------------------------

Returns

The item popped from `list`.

10.5.3.4 `sc_dlist_push_head()`

```
static void sc_dlist_push_head (  
    struct sc_dlist * list,  
    struct sc_dlist * l ) [inline], [static]
```

Prepend an item to the head of a doubly-linked list.

Parameters

<i>list</i>	The list to prepend to.
<i>l</i>	The item to prepend to <i>list</i> .

10.5.3.5 `sc_dlist_push_tail()`

```
static void sc_dlist_push_tail (  
    struct sc_dlist * list,  
    struct sc_dlist * l ) [inline], [static]
```

Append an item to the tail of a doubly-linked list.

Parameters

<i>list</i>	The list to append to.
<i>l</i>	The item to append to <i>list</i> .

10.5.3.6 `sc_dlist_rehome()`

```
static void sc_dlist_rehome (  
    struct sc_dlist * to_list,  
    struct sc_dlist * from_list ) [inline], [static]
```

Replace an item in a list with another item.

Parameters

<i>to_list</i>	The item to add to the list, replacing <i>from_list</i> .
<i>from_list</i>	The item to remove from the list.

10.5.3.7 `sc_dlist_remove()`

```
static void sc_dlist_remove (  
    struct sc_dlist * l ) [inline], [static]
```

Remove an item from the list.

Parameters

<i>l</i>	The item to remove.
----------	---------------------

10.6 ethernet.h File Reference

Ethernet protocol definitions.

Macros

- `#define SC_ETHERTYPE_8021Q 0x8100`
- `#define SC_ETHERTYPE_8021QinQ 0x88a8`
- `#define SC_8021Q_VID_MASK 0xfff`

10.6.1 Detailed Description

Ethernet protocol definitions.

10.6.2 Macro Definition Documentation

10.6.2.1 SC_8021Q_VID_MASK

```
#define SC_8021Q_VID_MASK 0xfff
```

Mask for VLAN identifier (VID)

10.6.2.2 SC_ETHERTYPE_8021Q

```
#define SC_ETHERTYPE_8021Q 0x8100
```

EtherType for IEEE 802.1Q

10.6.2.3 SC_ETHERTYPE_8021QinQ

```
#define SC_ETHERTYPE_8021QinQ 0x88a8
```

EtherType for IEEE 802.1QinQ

10.7 event.h File Reference

[sc_callback](#): Interface for event notification.

```
#include <sys/epoll.h>
```


Data Structures

- struct `sc_callback`
A callback object.

Typedefs

- typedef void() `sc_callback_handler_fn`(struct `sc_callback` *, void *event_info)
A callback handler function.

Functions

- int `sc_callback_alloc` (struct `sc_callback` **cb_out, const struct `sc_attr` *attr, struct `sc_thread` *thread)
Allocate a callback object instance.
- int `sc_callback_alloc2` (struct `sc_callback` **cb_out, const struct `sc_attr` *attr, struct `sc_thread` *thread, const char *description)
Allocate a callback object and set description.
- void `sc_callback_free` (struct `sc_callback` *cb)
Free a callback object instance.
- void `sc_callback_set_description` (struct `sc_callback` *cb, const char *fmt,...) `__attribute__((format(printf`
Set description of a callback.
- void static int `sc_callback_is_active` (const struct `sc_callback` *cb)
Returns true if a callback object is active.
- static void `sc_callback_remove` (struct `sc_callback` *cb)
Unregister a callback object from its event source.
- void `sc_callback_on_idle` (struct `sc_callback` *cb)
Request a callback when the thread is idle.
- int `sc_epoll_ctl` (struct `sc_thread` *thread, int op, int fd, unsigned events, struct `sc_callback` *cb)
Request a callback when the thread is idle.

10.7.1 Detailed Description

`sc_callback`: Interface for event notification.

10.7.2 Typedef Documentation

10.7.2.1 `sc_callback_handler_fn`

```
typedef void() sc_callback_handler_fn(struct sc_callback *, void *event_info)
```

A callback handler function.

Parameters

<i>callback</i>	The callback struct registered with this callback.
<i>event_info</i>	If callback was registered using sc_epoll_ctl this will contain the <code>uint32_t</code> <code>epoll_events</code> bitmask (see <code>man 2 epoll_ctl</code>) In all other cases this is not used.

10.7.3 Function Documentation

10.7.3.1 `sc_callback_alloc()`

```
int sc_callback_alloc (
    struct sc\_callback ** cb_out,
    const struct sc\_attr * attr,
    struct sc\_thread * thread )
```

Allocate a callback object instance.

Parameters

<i>cb_out</i>	The allocated callback object is returned here
<i>attr</i>	Attributes
<i>thread</i>	The thread the callback will be used with

Returns

0 on success, or a negative error code.

This function allocates a callback object instance.

Before using the callback object the `sc_callback::cb_handler_fn` field must be initialised. The `sc_callback::cb_private` field may be used to store or point to caller-specific state.

A callback object can only be registered with a single event source at a time. If a callback object is registered with an event source it is "active". If an active callback is registered with an event source, it is automatically removed from the previous event source.

10.7.3.2 `sc_callback_alloc2()`

```
int sc_callback_alloc2 (
    struct sc\_callback ** cb_out,
    const struct sc\_attr * attr,
    struct sc\_thread * thread,
    const char * description )
```

Allocate a callback object and set description.

Parameters

<i>cb_out</i>	The allocated callback object is returned here
<i>attr</i>	Attributes
<i>thread</i>	The thread the callback will be used with
<i>description</i>	Description of callback (used for log traces)

Returns

0 on success, or a negative error code.

This function behaves as [sc_callback_alloc\(\)](#) except that you can also set a custom description.

10.7.3.3 sc_callback_free()

```
void sc_callback_free (
    struct sc_callback * cb )
```

Free a callback object instance.

Parameters

<i>cb</i>	The callback object to free
-----------	-----------------------------

This function frees a callback object instance.

10.7.3.4 sc_callback_is_active()

```
void static int sc_callback_is_active (
    const struct sc_callback * cb ) [inline], [static]
```

Returns true if a callback object is active.

Parameters

<i>cb</i>	The callback object
-----------	---------------------

10.7.3.5 sc_callback_on_idle()

```
void sc_callback_on_idle (
    struct sc_callback * cb )
```

Request a callback when the thread is idle.

Parameters

<i>cb</i>	The callback object
-----------	---------------------

The callback will be invoked from the associated thread's polling loop if there is no work done in that loop iteration. ie. The when thread is idle.

The callback is only invoked once. If further callbacks are wanted the callback must be reregistered explicitly.

10.7.3.6 `sc_callback_remove()`

```
static void sc_callback_remove (
    struct sc_callback * cb ) [inline], [static]
```

Unregister a callback object from its event source.

Parameters

<i>cb</i>	The callback object
-----------	---------------------

This function has no effect if the callback object is not active.

10.7.3.7 `sc_callback_set_description()`

```
void sc_callback_set_description (
    struct sc_callback * cb,
    const char * fmt,
    ... )
```

Set description of a callback.

Parameters

<i>cb</i>	The callback object
<i>fmt</i>	Printf-style format string

This function sets the description for a callback object. The description is currently only used in log traces.

If *fmt* is NULL, then log tracing is suppressed for callback *cb*.

10.7.3.8 `sc_epoll_ctl()`

```
int sc_epoll_ctl (
    struct sc_thread * thread,
    int op,
    int fd,
    unsigned events,
    struct sc_callback * cb )
```

Request a callback when the thread is idle.

Parameters

<i>thread</i>	The thread managing <code>fd</code>
<i>op</i>	EPOLL_CTL_ADD, EPOLL_CTL_MOD or EPOLL_CTL_DEL
<i>fd</i>	The file descriptor
<i>events</i>	Event flags (EPOLLIN, EPOLLOUT etc.)
<i>cb</i>	The callback object

Returns

0 on success, or a negative error code.

Request a callback when a file descriptor is readable or writable, or if `op` is EPOLL_CTL_DEL then cancel a callback.

This function uses `epoll` as the underlying mechanism to manage file descriptors, so please refer to the documentation of `epoll` for detailed semantics.

`events` and `cb` are ignored when `op` is EPOLL_CTL_DEL.

A callback registered via this interface cannot be removed with `sc_callback_remove`, and must not be re-registered with another event source without first calling `sc_epoll_ctl` with `op` set to EPOLL_CTL_DEL.

10.8 ext_node.h File Reference

Interface for writing custom nodes.

```
#include <stdarg.h>
```

Data Structures

- struct `sc_node`
Description of a node.
- struct `sc_node_link`
Description of a link the node has.
- struct `sc_node_factory`
Struct to hold information about how to create an instance of this node.
- struct `sc_node_type`
Describes a type of node.

Macros

- #define `sc_node_set_error(node, errno_code, ...)`
Set error from within the implementation of a node.
- #define `sc_node_set_errorv(node, errno_code, fmt, args)`
Set error from within the implementation of a node.
- #define `sc_node_fwd_error(node, rc) __sc_node_fwd_error((node), __FILE__, __LINE__, __func__, (rc))`
Forward error from a failed `sc` call.

Typedefs

- typedef int() `sc_node_init_fn`(struct `sc_node` *node, const struct `sc_attr` *attr, const struct `sc_node_factory` *factory)
Signature of function to initialise a node.
- typedef int() `sc_node_prep_fn`(struct `sc_node` *node, const struct `sc_node_link` *const *links, int n_links)
Signature of nt_prep_fn function.
- typedef void() `sc_node_pkts_fn`(struct `sc_node` *node, struct `sc_packet_list` *packet_list)
Signature of nt_pkts_fn function.
- typedef int() `sc_node_add_link_fn`(struct `sc_node` *from_node, const char *link_name, struct `sc_node` *to_node, const char *to_name_opt)
Signature of nt_add_link_fn function.
- typedef struct `sc_node` *() `sc_node_select_subnode_fn`(struct `sc_node` *node, const char *name_opt, char **new_name_out)
Signature of nt_select_subnode_fn function.
- typedef void() `sc_node_end_of_stream_fn`(struct `sc_node` *node)
Signature of nt_end_of_stream_fn function.

Functions

- int `sc_node_type_alloc` (struct `sc_node_type` **nt_out, const struct `sc_attr` *attr_opt, const struct `sc_node_factory` *factory)
Allocate an sc_node_type instance.
- void `sc_forward_list` (struct `sc_node` *node, const struct `sc_node_link` *link, struct `sc_packet_list` *pl)
Forward a list of packets.
- void `sc_forward_list2` (const struct `sc_node_link` *link, struct `sc_packet_list` *pl)
Forward a list of packets.
- void `sc_forward` (struct `sc_node` *node, const struct `sc_node_link` *link, struct `sc_packet` *packet)
Forward a single packet.
- void `sc_forward2` (const struct `sc_node_link` *link, struct `sc_packet` *packet)
Forward a single packet.
- int `sc_node_init_get_arg_int` (int *v_out, struct `sc_node` *node, const char *name, int v_default)
Get an integer argument.
- int `sc_node_init_get_arg_int64` (int64_t *v_out, struct `sc_node` *node, const char *name, int64_t v_default)
Get a 64 bit integer argument.
- int `sc_node_init_get_arg_str` (const char **v_out, struct `sc_node` *node, const char *name, const char *v_default)
Get an string argument.
- int `sc_node_init_get_arg_obj` (struct `sc_object` **obj_out, struct `sc_node` *node, const char *name, enum `sc_object_type` obj_type)
Get an sc_object argument.
- int `sc_node_init_get_arg_dbl` (double *v_out, struct `sc_node` *node, const char *name, double v_default)
Get a double argument.
- const struct `sc_node_link` * `sc_node_prep_get_link` (struct `sc_node` *node, const char *link_name)
Find a named outgoing link.
- const struct `sc_node_link` * `sc_node_prep_get_link_or_free` (struct `sc_node` *node, const char *link_name)
Find a named outgoing link or return a link for freeing.
- int `sc_node_prep_check_links` (struct `sc_node` *node)
Check the node's links for any unused links.

- int `sc_node_prep_get_pool` (struct `sc_pool` **`pool_out`, const struct `sc_attr` *`attr`, struct `sc_node` *`node`, const struct `sc_node_link` *const *`links`, int `n_links`)
Get a packet pool that can be used to obtain empty packet buffers that can be passed to any of the given set of links.
- void `sc_node_prep_does_not_forward` (struct `sc_node` *`node`)
Indicate that this node does not forward to all of its links.
- void `sc_node_prep_link_forwards_from_node` (struct `sc_node` *`node`, const struct `sc_node_link` *`link`, struct `sc_node` *`from_node`)
Indicate that packets arriving at a node pass through a link.
- void `sc_node_link_end_of_stream` (struct `sc_node` *`node`, const struct `sc_node_link` *`link`)
Indicate end-of-stream on a link.
- void `sc_node_link_end_of_stream2` (const struct `sc_node_link` *`link`)
Indicate end-of-stream on a link.
- int `sc_node_export_state` (struct `sc_node` *`node`, const char *`type_name`, int `type_size`, void *`pp_area`)
Export dynamic state to solar_capture_monitor.

10.8.1 Detailed Description

Interface for writing custom nodes.

10.8.2 Macro Definition Documentation

10.8.2.1 `sc_node_fwd_error`

```
#define sc_node_fwd_error(  
    node,  
    rc ) __sc_node_fwd_error((node), __FILE__, __LINE__, __func__, (rc))
```

Forward error from a failed sc call.

Parameters

<code>node</code>	The node that forwards the error
<code>rc</code>	The error code returned by the sc call

Call this function to propagate an error generated by SolarCapture.

10.8.2.2 `sc_node_set_error`

```
#define sc_node_set_error(  
    node,  
    errno_code,  
    ... )
```

Value:

```
__sc_node_set_error((node), __FILE__, __LINE__, __func__, \  
    (errno_code), __VA_ARGS__)
```

Set error from within the implementation of a node.

Parameters

<i>node</i>	The node that originates the error
<i>errno_code</i>	An error code from errno.h (or can be zero)

Call this function when returning an error to SolarCapture from a node. The value returned by this function should be passed on to the caller of the function reporting the error.

10.8.2.3 sc_node_set_errorv

```
#define sc_node_set_errorv(
    node,
    errno_code,
    fmt,
    args )
```

Value:

```
__sc_node_set_errorv((node), __FILE__, __LINE__, __func__, \
    (errno_code), (fmt), (args))
```

Set error from within the implementation of a node.

Parameters

<i>node</i>	The node that originates the error
<i>errno_code</i>	An error code from errno.h (or can be zero)
<i>fmt</i>	vprintf style format string
<i>args</i>	vprintf arguments matching format string

Call this function when returning an error to SolarCapture from a node. The value returned by this function should be passed on to the caller of the function reporting the error.

See also [sc_node_set_error](#).

10.8.3 Typedef Documentation
10.8.3.1 sc_node_add_link_fn

```
typedef int() sc_node_add_link_fn(struct sc_node *from_node, const char *link_name, struct
sc_node *to_node, const char *to_name_opt)
```

Signature of nt_add_link_fn function.

Parameters

<i>from_node</i>	The node being linked from
<i>link_name</i>	The name of the link
<i>to_node</i>	The node being linked to
<i>to_name_opt</i>	Optional name of ingress link

This method is optional and supports compound nodes. It is invoked on *from_node* when [sc_node_add_link](#)(*from_node*, *link_name*, *to_node*, *to_name_opt*) is called, and gives the implementation an opportunity to select the subnode(s) to be linked from or issue an error if an attempt is made to create an unwanted link.

The implementation of this function should invoke [sc_node_add_link\(\)](#) on *from_node* or on a subnode, or should return an error. If it returns an error it should do so by calling [sc_node_set_error](#) with a suitable error message, and return the value returned by [sc_node_set_error](#).

The *to_node* and *to_name_opt* arguments should be passed unmodified. The implementation may invoke [sc_node_add_link\(\)](#) multiple times to create links from multiple subnodes.

10.8.3.2 sc_node_end_of_stream_fn

```
typedef void() sc_node_end_of_stream_fn(struct sc_node *node)
```

Signature of [nt_end_of_stream_fn](#) function.

Parameters

<i>node</i>	The node.
-------------	-----------

This method is invoked when all incoming upstream nodes have indicated end-of-stream. After this method has been called [sc_node_pkts_fn](#) will not be called again. The implementation of this function may propagate end-of-stream through its outgoing links by calling [sc_node_link_end_of_stream\(\)](#). If this function is not provided end-of-stream will not propagate further through the node graph.

After the node has propagated end-of-stream to a node through its outgoing link it should not pass any more packets to this node.

This method is optional.

10.8.3.3 sc_node_init_fn

```
typedef int() sc_node_init_fn(struct sc_node *node, const struct sc_attr *attr, const struct sc_node_factory *factory)
```

Signature of function to initialise a node.

Parameters

<i>node</i>	The node being initialised
<i>attr</i>	Attributes used to create the node
<i>factory</i>	The node factory

This callback is used to initialise the private state of a node instance. It is called in response to [sc_node_alloc\(\)](#) (or similar).

This function must set [sc_node::nd_type](#) before invoking any other function call on the node.

The lifetime of the *attr* argument is limited to this call only. Use [sc_attr_dup\(\)](#) if a copy is needed after this call returns.

10.8.3.4 `sc_node_pkts_fn`

```
typedef void() sc_node_pkts_fn(struct sc_node *node, struct sc_packet_list *packet_list)
```

Signature of `nt_pkts_fn` function.

Parameters

<i>node</i>	The node receiving the packets
<i>packet_list</i>	List of packets

This function will be called when packets are received on **any** incoming link to the node. It is not possible to distinguish which incoming link the packets arrived from directly. If the node needs to distinguish between incoming streams then either upstream nodes must append metadata to the packets or the node must be constructed from subnodes with each subnode connected to a subset of incoming links.

Once this function is invoked the node gets ownership of the packets. Ownership is relinquished by invoking [sc_forward_list](#), [sc_forward_list2](#), [sc_forward](#) or [sc_forward2](#) to forward or free the packets.

10.8.3.5 `sc_node_prep_fn`

```
typedef int() sc_node_prep_fn(struct sc_node *node, const struct sc_node_link *const *links,
int n_links)
```

Signature of `nt_prep_fn` function.

Parameters

<i>node</i>	The node being prepared
<i>links</i>	Array of outgoing links the node has
<i>n_links</i>	Number of outgoing links in the array

This callback is invoked to prepare `node` for live packet processing. The implementation typically checks the egress links and saves them to private storage.

Any initialisation that could not be done in `sc_node_init` should be done here.

If the node needs to create subnodes and establish links it should be done **before** this stage in one of [sc_node_init_fn](#), [sc_node_select_subnode_fn](#) or [sc_node_add_link_fn](#).

Note that the array `links` is only valid for the duration of this function call, but the [sc_node_link](#) objects are valid for at least the lifetime of the node.

10.8.3.6 `sc_node_select_subnode_fn`

```
typedef struct sc_node*() sc_node_select_subnode_fn(struct sc_node *node, const char *name_opt,
char **new_name_out)
```

Signature of `nt_select_subnode_fn` function.

Parameters

<i>node</i>	The node being linked to
<i>name_opt</i>	The name of the link (may be NULL)
<i>new_name_out</i>	Use to set a different name for sub-node

This method is optional and supports compound nodes. It is invoked on *to_node* when [sc_node_add_link](#)(*from_node*, *link_name*, *to_node*, *to_name_opt*) is called, and gives the implementation an opportunity to select an alternative subnode that should be linked to or issue an error if an attempt is made to create an unwanted link.

The implementation should return *node* or a subnode, or NULL to indicate that *name_opt2* is not valid for this node. If returning NULL the implementation should first call [sc_node_set_error](#)() to give the reason for the error.

name_opt2 comes from the *to_name_opt* argument passed to [sc_node_add_link](#)(), and may be NULL. If multiple links are added with the same non-NULL *name_opt2* then they should be treated as being logically the same link.

If a new name is specified via *new_name_out* then ownership is passed to the caller, and it will be freed with [free](#)() .

10.8.4 Function Documentation

10.8.4.1 sc_forward()

```
void sc_forward (
    struct sc_node * node,
    const struct sc_node_link * link,
    struct sc_packet * packet )
```

Forward a single packet.

Parameters

<i>node</i>	The node
<i>link</i>	The link to forward through
<i>packet</i>	The packet to forward

See also [sc_forward2](#).

10.8.4.2 sc_forward2()

```
void sc_forward2 (
    const struct sc_node_link * link,
    struct sc_packet * packet )
```

Forward a single packet.

Parameters

<i>link</i>	The link to forward through
<i>packet</i>	The packet to forward

10.8.4.3 sc_forward_list()

```
void sc_forward_list (
    struct sc_node * node,
    const struct sc_node_link * link,
    struct sc_packet_list * pl )
```

Forward a list of packets.

Parameters

<i>node</i>	The node
<i>link</i>	The link to forward through
<i>pl</i>	The list of packets to forward

See also [sc_forward_list2](#).

10.8.4.4 sc_forward_list2()

```
void sc_forward_list2 (
    const struct sc_node_link * link,
    struct sc_packet_list * pl )
```

Forward a list of packets.

Parameters

<i>link</i>	The link to forward through
<i>pl</i>	The list of packets to forward

10.8.4.5 sc_node_export_state()

```
int sc_node_export_state (
    struct sc_node * node,
    const char * type_name,
    int type_size,
    void * pp_area )
```

Export dynamic state to solar_capture_monitor.

Parameters

<i>node</i>	The node exporting state
<i>type_name</i>	Name of the exported datastructure
<i>type_size</i>	Size in bytes of the exported datastructure
<i>pp_area</i>	Pointer to memory is returned here

Returns

0 on success, or a negative error code.

Use this function to export dynamic runtime information about a node to `solar_capture_monitor`. The information can include configuration information, statistics and/or other runtime state.

`pp_area` gives the address of a pointer that is overwritten with a pointer to the memory area large enough for an instance of `type_name`. So `pp_area` should be of type `'struct type_name**'`.

The type `type_name` must already have been declared by creating the `type_name_declare()` function using `declare_types.h::SC_DECLARE_TYPES` and calling it.

See also [sc_node_add_info_str\(\)](#) and [sc_node_add_info_int\(\)](#), which are useful for exporting static data.

10.8.4.6 sc_node_init_get_arg_dbl()

```
int sc_node_init_get_arg_dbl (
    double * v_out,
    struct sc_node * node,
    const char * name,
    double v_default )
```

Get a double argument.

Parameters

<i>v_out</i>	On success, the value is returned here
<i>node</i>	The node
<i>name</i>	The name of the argument
<i>v_default</i>	Default returned if arg not found

Returns

- 0 on success
- 1 if the argument is not found (in which case `v_default` is copied to `v_out`)
- 1 if the argument was found but is of the wrong type.

This may only be called from [sc_node_init_fn](#).

10.8.4.7 sc_node_init_get_arg_int()

```
int sc_node_init_get_arg_int (
    int * v_out,
    struct sc_node * node,
    const char * name,
    int v_default )
```

Get an integer argument.

Parameters

<i>v_out</i>	On success, the value is returned here
<i>node</i>	The node
<i>name</i>	The name of the argument
<i>v_default</i>	Default returned if arg not found

Returns

- 0 on success
- 1 if the argument is not found (in which case *v_default* is copied to *v_out*)
- 1 if the argument was found but is of the wrong type.

This may only be called from [sc_node_init_fn](#).

10.8.4.8 sc_node_init_get_arg_int64()

```
int sc_node_init_get_arg_int64 (
    int64_t * v_out,
    struct sc_node * node,
    const char * name,
    int64_t v_default )
```

Get a 64 bit integer argument.

Parameters

<i>v_out</i>	On success, the value is returned here
<i>node</i>	The node
<i>name</i>	The name of the argument
<i>v_default</i>	Default returned if arg not found

Returns

- 0 on success
- 1 if the argument is not found (in which case *v_default* is copied to *v_out*)
- 1 if the argument was found but is of the wrong type.

This may only be called from [sc_node_init_fn](#).

10.8.4.9 sc_node_init_get_arg_obj()

```
int sc_node_init_get_arg_obj (
    struct sc_object ** obj_out,
    struct sc_node * node,
    const char * name,
    enum sc_object_type obj_type )
```

Get an [sc_object](#) argument.

Parameters

<i>obj_out</i>	On success, the value is returned here
<i>node</i>	The node
<i>name</i>	The name of the argument
<i>obj_type</i>	The type of object wanted, or SC_OBJ_ANY

Returns

- 0 on success
- 1 if the argument is not found (in which case *v_default* is copied to *v_out*)
- 1 if the argument was found but is of the wrong type.

This may only be called from [sc_node_init_fn](#).

10.8.4.10 `sc_node_init_get_arg_str()`

```
int sc_node_init_get_arg_str (
    const char ** v_out,
    struct sc_node * node,
    const char * name,
    const char * v_default )
```

Get an string argument.

Parameters

<i>v_out</i>	On success, the value is returned here
<i>node</i>	The node
<i>name</i>	The name of the argument
<i>v_default</i>	Default returned if arg not found

Returns

- 0 on success
- 1 if the argument is not found (in which case *v_default* is copied to *v_out*)
- 1 if the argument was found but is of the wrong type.

This may only be called from [sc_node_init_fn](#).

The string returned is valid only until the [sc_node_init_fn](#) call returns.

10.8.4.11 `sc_node_link_end_of_stream()`

```
void sc_node_link_end_of_stream (
    struct sc_node * node,
    const struct sc_node_link * link )
```

Indicate end-of-stream on a link.

Parameters

<i>node</i>	The node
<i>link</i>	The link

It is a fatal error to forward any further packets through the link after calling this function.

10.8.4.12 `sc_node_link_end_of_stream2()`

```
void sc_node_link_end_of_stream2 (
    const struct sc\_node\_link * link )
```

Indicate end-of-stream on a link.

Parameters

<i>link</i>	The link
-------------	----------

It is a fatal error to forward any further packets through the link after calling this function.

10.8.4.13 `sc_node_prep_check_links()`

```
int sc_node_prep_check_links (
    struct sc\_node * node )
```

Check the node's links for any unused links.

Parameters

<i>node</i>	The node
-------------	----------

Returns

- 0 if all is fine (or only warnings are needed)
- 1 on error, which should be propagated out of [sc_node_prep_fn\(\)](#).

This may only be called from [sc_node_prep_fn\(\)](#), and should only be used by nodes that find their links by calling [sc_node_prep_get_link\(\)](#).

This function will complain about any links added to the node that have not been queried by [sc_node_prep_get_link\(\)](#). It may emit a warning, or generate an error.

10.8.4.14 `sc_node_prep_does_not_forward()`

```
void sc_node_prep_does_not_forward (
    struct sc\_node * node )
```

Indicate that this node does not forward to all of its links.

Parameters

<i>node</i>	A node
-------------	--------

By default it is assumed that packets arriving at a node may be forwarded through any of the node's outgoing links. The effect of this call is to break that assumption. SolarCapture will assume that packets arriving at `node` are not forwarded via any of the outgoing links, unless overridden by [sc_node_prep_link_forwards_from_node](#).

10.8.4.15 `sc_node_prep_get_link()`

```
const struct sc\_node\_link* sc_node_prep_get_link (
    struct sc\_node * node,
    const char * link_name )
```

Find a named outgoing link.

Parameters

<i>node</i>	The node
<i>link_name</i>	Name of the link

Returns

The named link, or NULL if the named link doesn't exist.

A node's [sc_node_prep_fn](#) can either use this mechanism to query its links, or it can simply iterate over the links passed as arguments to [sc_node_prep_fn](#).

This function may only be called from [sc_node_prep_fn](#).

See also [sc_node_prep_check_links\(\)](#).

10.8.4.16 `sc_node_prep_get_link_or_free()`

```
const struct sc\_node\_link* sc_node_prep_get_link_or_free (
    struct sc\_node * node,
    const char * link_name )
```

Find a named outgoing link or return a link for freeing.

Parameters

<i>node</i>	The node
<i>link_name</i>	Name of the link

Returns

The named link, or a special link that frees packets if the named link doesn't exist.

This function behaves just like [sc_node_prep_get_link\(\)](#), except that if no link of that name has been added to the node, a special link is returned that frees packets.

`link_name` may be NULL, in which case a link for freeing packets is returned.

10.8.4.17 `sc_node_prep_get_pool()`

```
int sc_node_prep_get_pool (
    struct sc_pool ** pool_out,
    const struct sc_attr * attr,
    struct sc_node * node,
    const struct sc_node_link *const * links,
    int n_links )
```

Get a packet pool that can be used to obtain empty packet buffers that can be passed to any of the given set of links.

Parameters

<i>pool_out</i>	On success, the pool is returned here
<i>attr</i>	Packet pool attributes (optional, may be NULL)
<i>node</i>	The node
<i>links</i>	The link(s) packets from the pool may be passed to (set to NULL for all)
<i>n_links</i>	Number of links in 'links' (set to 0 for all)

Returns

0 on success, or a negative error code.

The node must only forward packets from the returned pool over the links identified by `links` and `n_links`. If `n_links` is 0 then it is assumed that packets from the pool may be forwarded over any of the node's links.

Restricting the links packets can be sent along allows SolarCapture to optimise the releasing of packets back to the pool when the node graph is finished with them.

This may only be called from `sc_node_prep_fn`.

10.8.4.18 `sc_node_prep_link_forwards_from_node()`

```
void sc_node_prep_link_forwards_from_node (
    struct sc_node * node,
    const struct sc_node_link * link,
    struct sc_node * from_node )
```

Indicate that packets arriving at a node pass through a link.

Parameters

<i>node</i>	The node that <code>link</code> originates from
<i>link</i>	A link from <code>node</code> to another node
<i>from_node</i>	Node at which packets arrive

This call tells SolarCapture that packets arriving at `from_node` are forwarded via `link`.

You will also need to call `sc_node_prep_does_not_forward` to cancel the default assumption that all links are used for forwarding.

Note that most nodes do not need to use this function, because SolarCapture assumes by default that packets arriving at a node may be forwarded through any of the node's outgoing links. This call is useful when either (a) only a subset of links are used for forwarding or (b) a node forwards packets that arrived at a different node.

10.8.4.19 `sc_node_type_alloc()`

```
int sc_node_type_alloc (
    struct sc_node_type ** nt_out,
    const struct sc_attr * attr_opt,
    const struct sc_node_factory * factory )
```

Allocate an `sc_node_type` instance.

Parameters

<code>nt_out</code>	The allocated <code>sc_node_type</code> instance
<code>attr_opt</code>	Optional attributes (may be NULL)
<code>factory</code>	The factory that created the node

Returns

0 on success, or a negative error code.

At the time of writing `attr_opt` is not used and this call always succeeds. In future it may fail if the attributes are invalid in some way.

10.9 `ext_packet.h` File Reference

`sc_packet`: The representation of a packet or other data.

```
#include <stdlib.h>
#include <stdint.h>
#include <sys/uio.h>
```

Data Structures

- struct `sc_packet`
Representation of a packet.

Macros

- #define `SC_MEMBER_OFFSET`(`c_type`, `mbr_name`) ((uint32_t) (uintptr_t) (&((c_type*)0)->`mbr_name`))
Calculate memory offset of a field within a struct.
- #define `SC_MEMBER_SIZE`(`c_type`, `mbr_name`) (sizeof(((c_type*)0)->`mbr_name`))
Calculate the size of a field within a struct.
- #define `SC_FRAME_LEN_LARGE` `UINT16_MAX`
struct `sc_packet.frame_len` holds this special value to indicate that the frame is "large". (Meaning it would overflow `sc_packet.frame_len`).
- #define `SC_CSUM_ERROR` (1 << 0)
struct `sc_packet.flags` will have this set if the packet has a checksum error
- #define `SC_CRC_ERROR` (1 << 1)
struct `sc_packet.flags` will have this set if the packet has a crc error
- #define `SC_TRUNCATED` (1 << 2)
struct `sc_packet.flags` will have this set if the packet has been truncated
- #define `SC_MCAST_MISMATCH` (1 << 3)
struct `sc_packet.flags` will have this set if the packet is for a multicast group the host hasn't joined
- #define `SC_UCAST_MISMATCH` (1 << 4)
struct `sc_packet.flags` will have this set if the packet is for a unicast address not matching the host's

Functions

- static int `sc_packet_bytes` (struct `sc_packet` *p)
Return the size of the packet data in bytes.
- static struct `sc_packet` * `sc_packet_frags_tail` (struct `sc_packet` *p)
Return a packet's last fragment.
- static void `sc_packet_prefetch_r` (struct `sc_packet` *p)
Prefetch a packet for reading.
- static void `sc_packet_prefetch_rw` (struct `sc_packet` *p)
Prefetch a packet for reading and writing.
- static struct timespec `sc_packet_timespec` (const struct `sc_packet` *p)
Return the timestamp of the packet in timespec format.

10.9.1 Detailed Description

`sc_packet`: The representation of a packet or other data.

10.9.2 Macro Definition Documentation

10.9.2.1 SC_MEMBER_OFFSET

```
#define SC_MEMBER_OFFSET(  
    c_type,  
    mbr_name ) ((uint32_t) (uintptr_t) (&((c_type*)0)->mbr_name))
```

Calculate memory offset of a field within a struct.

Parameters

<code>c_type</code>	The struct type.
<code>mbr_name</code>	The field name to calculate the offset of.

10.9.2.2 SC_MEMBER_SIZE

```
#define SC_MEMBER_SIZE(  
    c_type,  
    mbr_name ) (sizeof(((c_type*)0)->mbr_name))
```

Calculate the size of a field within a struct.

Parameters

<i>c_type</i>	The struct type.
<i>mbr_name</i>	The field to calculate the size of.

10.9.3 Function Documentation

10.9.3.1 `sc_packet_bytes()`

```
static int sc_packet_bytes (  
    struct sc_packet * p ) [inline], [static]
```

Return the size of the packet data in bytes.

Parameters

<i>p</i>	A packet object.
----------	------------------

Returns

The size of the packet data in bytes.

10.9.3.2 `sc_packet_fragments_tail()`

```
static struct sc_packet* sc_packet_fragments_tail (  
    struct sc_packet * p ) [static]
```

Return a packet's last fragment.

Parameters

<i>p</i>	A packet object.
----------	------------------

Returns

The packet's last fragment.

The result is only valid if the packet has at least one fragment.

10.9.3.3 `sc_packet_prefetch_r()`

```
static void sc_packet_prefetch_r (  
    struct sc_packet * p ) [inline], [static]
```

Prefetch a packet for reading.

Parameters

<i>p</i>	A packet object.
----------	------------------

10.9.3.4 sc_packet_prefetch_rw()

```
static void sc_packet_prefetch_rw (  
    struct sc\_packet * p ) [inline], [static]
```

Prefetch a packet for reading and writing.

Parameters

<i>p</i>	A packet object.
----------	------------------

10.9.3.5 sc_packet_timespec()

```
static struct timespec sc_packet_timespec (  
    const struct sc\_packet * p ) [static]
```

Return the timestamp of the packet in timespec format.

Parameters

<i>p</i>	A packet object.
----------	------------------

Returns

The timestamp of the packet in timespec format.

10.10 ext_packet_list.h File Reference

[sc_packet_list](#): A list of packets.

Data Structures

- struct [sc_packet_list](#)
A list of packets or packet buffers.

Functions

- static void `sc_packet_list_init` (struct `sc_packet_list` *)
Initialise a list.
- static int `sc_packet_list_is_empty` (const struct `sc_packet_list` *)
Check if packet list is empty.
- static void `sc_packet_list_finalise` (struct `sc_packet_list` *)
Finalise a list.
- static struct `sc_packet` * `sc_packet_list_tail` (struct `sc_packet_list` *)
Return the tail of current tail of the list.
- static void `sc_packet_list_push_head` (struct `sc_packet_list` *pl, struct `sc_packet` *p)
Push a packet to the head of a list.
- static void `sc_packet_list_append` (struct `sc_packet_list` *l, struct `sc_packet` *p)
Append a packet to a list and finalise.
- static void `sc_packet_list_append_list` (struct `sc_packet_list` *dest, struct `sc_packet_list` *src)
Append a list to a list.
- static struct `sc_packet` * `sc_packet_list_pop_head` (struct `sc_packet_list` *pl)
Remove and return the head of the list.

10.10.1 Detailed Description

`sc_packet_list`: A list of packets.

10.10.2 Function Documentation

10.10.2.1 `sc_packet_list_append()`

```
static void sc_packet_list_append (  
    struct sc_packet_list * l,  
    struct sc_packet * p ) [inline], [static]
```

Append a packet to a list and finalise.

Parameters

<i>l</i>	The packet list.
<i>p</i>	The packet.

10.10.2.2 `sc_packet_list_append_list()`

```
static void sc_packet_list_append_list (  
    struct sc_packet_list * dest,  
    struct sc_packet_list * src ) [inline], [static]
```

Append a list to a list.

Parameters

<i>dest</i>	The list to be extended.
<i>src</i>	The list to be appended to <i>dest</i> .

After this call *dest* is finalised if and only if *src* was finalised.

src must be non-empty.

10.10.2.3 sc_packet_list_finalise()

```
static void sc_packet_list_finalise (
    struct sc_packet_list * l ) [inline], [static]
```

Finalise a list.

Parameters

/	The packet list.
---	------------------

If a list is not finalised, it is possible that the next pointer of tail is not NULL.

10.10.2.4 sc_packet_list_init()

```
static void sc_packet_list_init (
    struct sc_packet_list * l ) [inline], [static]
```

Initialise a list.

Parameters

/	The packet list.
---	------------------

10.10.2.5 sc_packet_list_is_empty()

```
static int sc_packet_list_is_empty (
    const struct sc_packet_list * l ) [inline], [static]
```

Check if packet list is empty.

Parameters

/	The packet list.
---	------------------

Returns

True (1) if the packet list is empty, false (0) otherwise

10.10.2.6 sc_packet_list_pop_head()

```
static struct sc_packet* sc_packet_list_pop_head (
    struct sc_packet_list * pl ) [static]
```

Remove and return the head of the list.

Parameters

<i>pl</i>	The packet list.
-----------	------------------

Returns

The removed head of the packet list.

This must only be invoked on a non-empty list.

10.10.2.7 `sc_packet_list_push_head()`

```
static void sc_packet_list_push_head (  
    struct sc_packet_list * pl,  
    struct sc_packet * p ) [inline], [static]
```

Push a packet to the head of a list.

Parameters

<i>pl</i>	The packet list.
<i>p</i>	The packet.

10.10.2.8 `sc_packet_list_tail()`

```
static struct sc_packet* sc_packet_list_tail (  
    struct sc_packet_list * l ) [static]
```

Return the tail of current tail of the list.

Parameters

<i>l</i>	The packet list.
----------	------------------

Returns

The tail of current tail of the list.

10.11 `hash_table.h` File Reference

A hash table with open addressing and double hashing.

Functions

- int `sc_hash_table_alloc` (struct `sc_hash_table` **table_out, unsigned key_size, unsigned val_size, unsigned capacity)
Allocate an `sc_hash_table`.
- void `sc_hash_table_free` (struct `sc_hash_table` *table)
Free an `sc_hash_table`.
- int `sc_hash_table_grow` (struct `sc_hash_table` *table, size_t max_size)
Increase the capacity of a hash table.
- int `sc_hash_table_get` (struct `sc_hash_table` *table, const void *key, bool insert_if_not_found, void **val_out)
Lookup or insert an entry in a hash table.
- int `sc_hash_table_del` (struct `sc_hash_table` *table, const void *key)
Remove an entry from an `sc_hash_table` by key.
- int `sc_hash_table_del_val` (struct `sc_hash_table` *table, const void *val)
Remove an entry from an `sc_hash_table` by value.
- void `sc_hash_table_clear` (struct `sc_hash_table` *table)
Clear all entries from an `sc_hash_table`.
- const void * `sc_hash_table_val_to_key` (struct `sc_hash_table` *table, const void *val)
Return the key associated with a given value.
- unsigned `sc_hash_table_key_size` (struct `sc_hash_table` *table)
Get the size in bytes of a hash table's keys.
- unsigned `sc_hash_table_val_size` (struct `sc_hash_table` *table)
Get the size in bytes of each value buffer.
- unsigned `sc_hash_table_num_entries` (struct `sc_hash_table` *table)
Get the number of entries in an `sc_hash_table`.
- int `sc_hash_table_get_next_entry` (struct `sc_hash_table` *table, void **key_out, void **val_out, unsigned *iterator)
Iterate over key-value pairs in an `sc_hash_table`.

10.11.1 Detailed Description

A hash table with open addressing and double hashing.

10.11.2 Function Documentation

10.11.2.1 `sc_hash_table_alloc()`

```
int sc_hash_table_alloc (
    struct sc_hash_table ** table_out,
    unsigned key_size,
    unsigned val_size,
    unsigned capacity )
```

Allocate an `sc_hash_table`.

Parameters

<i>table_out</i>	The allocated sc_hash_table is returned here
<i>key_size</i>	The size in bytes of keys
<i>val_size</i>	The size in bytes of values
<i>capacity</i>	The desired number of entries in the hash table.

Returns

0 on success, or a negative error code.

Note the underlying table is sized so that there is a high probability you be able to insert `capacity` entries, but this cannot be guaranteed.

10.11.2.2 `sc_hash_table_clear()`

```
void sc_hash_table_clear (
    struct sc_hash_table * table )
```

Clear all entries from an [sc_hash_table](#).

Parameters

<i>table</i>	A hash table
--------------	--------------

10.11.2.3 `sc_hash_table_del()`

```
int sc_hash_table_del (
    struct sc_hash_table * table,
    const void * key )
```

Remove an entry from an [sc_hash_table](#) by key.

Parameters

<i>table</i>	A hash table
<i>key</i>	The key to remove

Returns

- 0 on success
- -ENOENT if `key` was not found

10.11.2.4 `sc_hash_table_del_val()`

```
int sc_hash_table_del_val (
    struct sc_hash_table * table,
    const void * val )
```

Remove an entry from an [sc_hash_table](#) by value.

Parameters

<i>table</i>	A hash table
<i>val</i>	Pointer to the value of an entry in the hash table

val must be a valid pointer to an existing value stored in the hash table. ie. It must have been returned by [sc_hash_table_get\(\)](#) or [sc_hash_table_get_next_entry\(\)](#).

Returns

- 0 if the key was successfully removed
- -ENOENT if the value was not in the table

10.11.2.5 sc_hash_table_free()

```
void sc_hash_table_free (
    struct sc_hash_table * table )
```

Free an [sc_hash_table](#).

Parameters

<i>table</i>	A hash table
--------------	--------------

10.11.2.6 sc_hash_table_get()

```
int sc_hash_table_get (
    struct sc_hash_table * table,
    const void * key,
    bool insert_if_not_found,
    void ** val_out )
```

Lookup or insert an entry in a hash table.

If the entry matching *key* is found then a pointer to the corresponding value is returned in *val_out*. Otherwise if *insert_if_not_found* is true, then a new entry is inserted.

Parameters

<i>table</i>	A hash table
<i>key</i>	The key to look for in the table
<i>insert_if_not_found</i>	If true then an entry is inserted if not found
<i>val_out</i>	Pointer to value buffer returned here

Returns

- 0 if the matching entry was found
- 1 if *insert_if_not_found* was true and a new entry was added
- -ENOENT if an entry was not found and *insert_if_not_found* was false
- -ENOSPC if an entry was not found and it was not possible to insert a new entry

10.11.2.7 `sc_hash_table_get_next_entry()`

```
int sc_hash_table_get_next_entry (
    struct sc_hash_table * table,
    void ** key_out,
    void ** val_out,
    unsigned * iterator )
```

Iterate over key-value pairs in an `sc_hash_table`.

Parameters

<code>table</code>	A hash table
<code>key_out</code>	Pointer to the next key returned here
<code>val_out</code>	Pointer to the next value returned here
<code>iterator</code>	State used by the implementation to iterate over entries

`iterator` must point to storage allocated by the caller and initialised to zero before the first call.

NOTE: This function is relatively inefficient for hash tables with a low fill level because it scans entries linearly.

Returns

- 0 on successfully finding the entry
- -ENOENT when no further entries remain

10.11.2.8 `sc_hash_table_grow()`

```
int sc_hash_table_grow (
    struct sc_hash_table * table,
    size_t max_size )
```

Increase the capacity of a hash table.

After this call all key and value pointers will be stale.

Parameters

<code>table</code>	A hash table
<code>max_size</code>	Maximum size of storage in bytes, or 0 for unlimited

Returns

- 0 On success
- -ENOSPC if it is not possible to grow the table further

10.11.2.9 `sc_hash_table_key_size()`

```
unsigned sc_hash_table_key_size (
    struct sc_hash_table * table )
```

Get the size in bytes of a hash table's keys.

Parameters

<i>table</i>	A hash table
--------------	--------------

Returns

The size in bytes of the hash table's keys.

10.11.2.10 `sc_hash_table_num_entries()`

```
unsigned sc_hash_table_num_entries (  
    struct sc_hash_table * table )
```

Get the number of entries in an [sc_hash_table](#).

Parameters

<i>table</i>	A hash table
--------------	--------------

Returns

The number of entries in the hash table.

10.11.2.11 `sc_hash_table_val_size()`

```
unsigned sc_hash_table_val_size (  
    struct sc_hash_table * table )
```

Get the size in bytes of each value buffer.

Parameters

<i>table</i>	A hash table
--------------	--------------

Returns

The size in bytes of each value buffer.

10.11.2.12 `sc_hash_table_val_to_key()`

```
const void* sc_hash_table_val_to_key (  
    struct sc_hash_table * table,  
    const void * val )
```

Return the key associated with a given value.

Parameters

<i>table</i>	A hash table
<i>val</i>	Pointer to the value of an entries in the hash table

val must be a valid pointer to an existing value stored in the hash table. ie. It must have been returned by [sc_hash_table_get\(\)](#) or [sc_hash_table_get_next_entry\(\)](#).

Returns

A pointer the key in the hash table

NOTE: This function cannot check the value pointer was valid.

10.12 iovec.h File Reference

[sc_iovec_ptr](#): Supports iterating over a 'struct iovec'.

```
#include <string.h>
```

Data Structures

- struct [sc_iovec_ptr](#)
An [sc_iovec_ptr](#) provides a convenient way to iterate over an iovec array without modifying it.

Functions

- static void [sc_iovec_ptr_init](#) (struct [sc_iovec_ptr](#) *iovp, const struct iovec *iovec, int iovlen)
Initialise a struct [sc_iovec_ptr](#).
- static void [sc_iovec_ptr_init_buf](#) (struct [sc_iovec_ptr](#) *iovp, void *buf, int len)
Initialise a struct [sc_iovec_ptr](#) with a contiguous buffer.
- static void [sc_iovec_ptr_init_packet](#) (struct [sc_iovec_ptr](#) *iovp, const struct [sc_packet](#) *packet)
Initialise a struct [sc_iovec_ptr](#) to point at packet data.
- static int [sc_iovec_ptr_bytes](#) (const struct [sc_iovec_ptr](#) *iovp)
Returns the number of bytes represented by an [sc_iovec_ptr](#).
- int [sc_iovec_ptr_skip](#) (struct [sc_iovec_ptr](#) *iovp, int bytes_to_skip)
Skip forward over an iovec.
- int [sc_iovec_ptr_find_chr](#) (const struct [sc_iovec_ptr](#) *iovp, int c)
Find offset of character in iovec.
- int [sc_iovec_ptr_copy_out](#) (void *dest, struct [sc_iovec_ptr](#) *iovp, int max_bytes)
Copy data out of an [sc_iovec_ptr](#).
- static void [sc_iovec_copy_from_end_offset](#) (void *dest_buf, const struct iovec *iovec, int iovlen, int bytes, int offset)
Copy data out of the end with the offset of a [sc_iovec_ptr](#).
- static void [sc_iovec_copy_to_end_offset](#) (struct iovec *iovec, const void *src_buf, int iovlen, int bytes, int offset)
Copy data to the end with the offset of a [sc_iovec_ptr](#), overwriting any existing data.
- static void [sc_iovec_copy_from_end](#) (void *dest_buf, const struct iovec *iovec, int iovlen, int bytes)
Copy data out of the end of a [sc_iovec_ptr](#).
- static void [sc_iovec_trim_end](#) (struct iovec *iovec, uint8_t *iovec, int bytes)
Remove data from the end of an iovec.

10.12.1 Detailed Description

[sc_iovec_ptr](#): Supports iterating over a 'struct iovec'.

10.12.2 Function Documentation

10.12.2.1 sc_iovec_copy_from_end()

```
static void sc_iovec_copy_from_end (
    void * dest_buf,
    const struct iovec * iov,
    int iovlen,
    int bytes ) [inline], [static]
```

Copy data out of the end of a [sc_iovec_ptr](#).

Parameters

<i>dest_buf</i>	Buffer to copy to.
<i>iov</i>	A pointer to an array of iovec objects.
<i>iovlen</i>	The number of iovec objects in <i>iov</i> . This must be > 0.
<i>bytes</i>	Number of bytes to copy.

Note: The caller must ensure that at least *bytes* of data are available in *iov*

10.12.2.2 sc_iovec_copy_from_end_offset()

```
static void sc_iovec_copy_from_end_offset (
    void * dest_buf,
    const struct iovec * iov,
    int iovlen,
    int bytes,
    int offset ) [inline], [static]
```

Copy data out of the end with the offset of a [sc_iovec_ptr](#).

Parameters

<i>dest_buf</i>	Buffer to copy to.
<i>iov</i>	A pointer to an array of iovec objects.
<i>iovlen</i>	The number of iovec objects in <i>iov</i> . This must be > 0.
<i>bytes</i>	Number of bytes to copy.
<i>offset</i>	Number of bytes to offset.

Note: The caller must ensure that at least *bytes* + *offset* of data are available in *iov*

10.12.2.3 `sc_iovec_copy_to_end_offset()`

```
static void sc_iovec_copy_to_end_offset (
    struct iovec * iovec,
    const void * src_buf,
    int iovec_len,
    int bytes,
    int offset ) [inline], [static]
```

Copy data to the end with the offset of a `sc_iovec_ptr`, overwriting any existing data.

Parameters

<i>iovec</i>	A pointer to an array of iovec objects.
<i>src_buf</i>	Buffer to copy from.
<i>iovec_len</i>	The number of iovec objects in <i>iovec</i> . This must be > 0.
<i>bytes</i>	Number of bytes to copy.
<i>offset</i>	Number of bytes to offset.

Note: The caller must ensure that at least `bytes + offset` of data are available in *iovec*

10.12.2.4 `sc_iovec_ptr_bytes()`

```
static int sc_iovec_ptr_bytes (
    const struct sc_iovec_ptr * iovp ) [inline], [static]
```

Returns the number of bytes represented by an `sc_iovec_ptr`.

Parameters

<i>iovp</i>	The <code>sc_iovec_ptr</code>
-------------	-------------------------------

Returns

The number of bytes represented by the `sc_iovec_ptr`

10.12.2.5 `sc_iovec_ptr_copy_out()`

```
int sc_iovec_ptr_copy_out (
    void * dest,
    struct sc_iovec_ptr * iovp,
    int max_bytes )
```

Copy data out of an `sc_iovec_ptr`.

Parameters

<i>dest</i>	Buffer to copy to
<i>iovp</i>	An sc_iovec_ptr
<i>max_bytes</i>	Max number of bytes to copy (length of <i>dest</i>)

Returns

The number of bytes copied.

10.12.2.6 [sc_iovec_ptr_find_chr\(\)](#)

```
int sc_iovec_ptr_find_chr (
    const struct sc\_iovec\_ptr * iovp,
    int c )
```

Find offset of character in iovec.

Parameters

<i>iovp</i>	An sc_iovec_ptr
<i>c</i>	Character to find

Returns

The offset of first occurrence of character *c* in the memory reference by *iovp*, or -1 if not found.

10.12.2.7 [sc_iovec_ptr_init\(\)](#)

```
static void sc_iovec_ptr_init (
    struct sc\_iovec\_ptr * iovp,
    const struct iovec * iov,
    int ioflen ) [inline], [static]
```

Initialise a struct [sc_iovec_ptr](#).

Parameters

<i>iovp</i>	The sc_iovec_ptr to initialise
<i>iov</i>	Pointer to array of 'struct iovec's
<i>ioflen</i>	Length of the <i>iov</i> array

10.12.2.8 [sc_iovec_ptr_init_buf\(\)](#)

```
static void sc_iovec_ptr_init_buf (
    struct sc\_iovec\_ptr * iovp,
    void * buf,
    int len ) [inline], [static]
```

Initialise a struct [sc_iovec_ptr](#) with a contiguous buffer.

Parameters

<i>iovp</i>	The sc_iovec_ptr to initialise
<i>buf</i>	Pointer to the start of the buffer
<i>len</i>	Length of the buffer

10.12.2.9 [sc_iovec_ptr_init_packet\(\)](#)

```
static void sc_iovec_ptr_init_packet (
    struct sc\_iovec\_ptr * iovp,
    const struct sc\_packet * packet ) [inline], [static]
```

Initialise a struct [sc_iovec_ptr](#) to point at packet data.

Parameters

<i>iovp</i>	The sc_iovec_ptr to initialise
<i>packet</i>	The packet

10.12.2.10 [sc_iovec_ptr_skip\(\)](#)

```
int sc_iovec_ptr_skip (
    struct sc\_iovec\_ptr * iovp,
    int bytes_to_skip )
```

Skip forward over an iovec.

Parameters

<i>iovp</i>	An sc_iovec_ptr
<i>bytes_to_skip</i>	Number of bytes to skip over

Returns

The number of bytes skipped, which may be fewer than *bytes_to_skip* if the total amount of memory referenced by *iovp* is less.

10.12.2.11 [sc_iovec_trim_end\(\)](#)

```
static void sc_iovec_trim_end (
    struct iovec * iov,
    uint8_t * iovl,
    int bytes ) [inline], [static]
```

Remove data from the end of an iovec.

Parameters

<i>iov</i>	A pointer to an array of iovec objects.
<i>iovlen</i>	The number of iovec objects in <i>iov</i> .
<i>bytes</i>	The number of bytes to trim.

Note: Caller must ensure that at least `bytes` of data are available in `iov`.

10.13 ip.h File Reference

IP protocol definitions.

Macros

- `#define SC_IP4_OFFSET_MASK 0x1fff`
- `#define SC_IP4_FRAG_MORE 0x2000`
- `#define SC_IP4_FRAG_DONT 0x4000`
- `#define SC_TCP_FIN 0x01`
- `#define SC_TCP_SYN 0x02`
- `#define SC_TCP_RST 0x04`
- `#define SC_TCP_PSH 0x08`
- `#define SC_TCP_ACK 0x10`
- `#define SC_TCP_URG 0x20`

10.13.1 Detailed Description

IP protocol definitions.

10.13.2 Macro Definition Documentation

10.13.2.1 SC_IP4_FRAG_DONT

```
#define SC_IP4_FRAG_DONT 0x4000
```

Mask for Don't Fragment flag in IP header

10.13.2.2 SC_IP4_FRAG_MORE

```
#define SC_IP4_FRAG_MORE 0x2000
```

Mask for More Fragments flag in IP header

10.13.2.3 SC_IP4_OFFSET_MASK

```
#define SC_IP4_OFFSET_MASK 0x1fff
```

Mask for Fragment Offset field in IP header

10.13.2.4 SC_TCP_ACK

```
#define SC_TCP_ACK 0x10
```

Mask for ACK flag in TCP header

10.13.2.5 SC_TCP_FIN

```
#define SC_TCP_FIN 0x01
```

Mask for FIN flag in TCP header

10.13.2.6 SC_TCP_PSH

```
#define SC_TCP_PSH 0x08
```

Mask for PSH flag in TCP header

10.13.2.7 SC_TCP_RST

```
#define SC_TCP_RST 0x04
```

Mask for RST flag in TCP header

10.13.2.8 SC_TCP_SYN

```
#define SC_TCP_SYN 0x02
```

Mask for SYN flag in TCP header

10.13.2.9 SC_TCP_URG

```
#define SC_TCP_URG 0x20
```

Mask for URG flag in TCP header

10.14 mailbox.h File Reference

sc_mailbox: A means to pass packets from one thread to another.

Functions

- int [sc_mailbox_alloc](#) (struct sc_mailbox **mb_out, const struct [sc_attr](#) *attr, struct sc_thread *thread)
Allocate a mailbox.
- int [sc_mailbox_connect](#) (struct sc_mailbox *mb1, struct sc_mailbox *mb2)
Connect a pair of mailboxes.
- int [sc_mailbox_set_recv](#) (struct sc_mailbox *mailbox, struct [sc_node](#) *node, const char *name_opt)
Connect a mailbox to a node.
- struct [sc_node](#) * [sc_mailbox_get_send_node](#) (struct sc_mailbox *mailbox)
Return a mailbox's "send node".
- int [sc_mailbox_poll](#) (struct sc_mailbox *mailbox, struct [sc_packet_list](#) *list)
Poll a mailbox.
- void [sc_mailbox_send](#) (struct sc_mailbox *mailbox, struct [sc_packet](#) *packet)
Send a packet through a mailbox to another thread.
- void [sc_mailbox_send_list](#) (struct sc_mailbox *mailbox, struct [sc_packet_list](#) *list)
Send a list of packets through a mailbox to another thread.

10.14.1 Detailed Description

sc_mailbox: A means to pass packets from one thread to another.

10.14.2 Function Documentation

10.14.2.1 sc_mailbox_alloc()

```
int sc_mailbox_alloc (
    struct sc_mailbox ** mb_out,
    const struct sc\_attr * attr,
    struct sc_thread * thread )
```

Allocate a mailbox.

Parameters

<i>mb_out</i>	The allocated mailbox is returned here.
<i>attr</i>	Attributes (see sc_attr).
<i>thread</i>	The thread the mailbox will be in.

Returns

0 on success, or a negative error code.

Mailboxes are used to pass packets between threads. To communicate you need a mailbox in each thread, and together they form a bi-directional link.

From SolarCapture 1.1 onwards it is not usually necessary to create mailboxes explicitly: They are created automatically when objects in different threads are connected together.

10.14.2.2 `sc_mailbox_connect()`

```
int sc_mailbox_connect (
    struct sc_mailbox * mb1,
    struct sc_mailbox * mb2 )
```

Connect a pair of mailboxes.

Parameters

<i>mb1</i>	The first mailbox.
<i>mb2</i>	The second mailbox.

Returns

0 on success, or a negative error code.

Link a pair of mailboxes so that they can communicate. A mailbox can only be connected once.

10.14.2.3 `sc_mailbox_get_send_node()`

```
struct sc_node* sc_mailbox_get_send_node (
    struct sc_mailbox * mailbox )
```

Return a mailbox's "send node".

Parameters

<i>mailbox</i>	The mailbox.
----------------	--------------

Returns

The mailbox's send-node. Packets passed to this send-node are forwarded to the paired mailboxes recv-node.

10.14.2.4 `sc_mailbox_poll()`

```
int sc_mailbox_poll (
    struct sc_mailbox * mailbox,
    struct sc_packet_list * list )
```

Poll a mailbox.

Parameters

<i>mailbox</i>	The mailbox to poll.
<i>list</i>	Received packets are appended to this list.

Returns

0 on success, or a negative error code.

This function should only be invoked on an unmanaged mailbox. It is necessary to poll a mailbox in order to receive packets from other threads, and to ensure that sent packets are delivered.

10.14.2.5 `sc_mailbox_send()`

```
void sc_mailbox_send (
    struct sc_mailbox * mailbox,
    struct sc_packet * packet )
```

Send a packet through a mailbox to another thread.

Parameters

<i>mailbox</i>	The mailbox.
<i>packet</i>	The packet to send.

This function should only be invoked on an unmanaged mailbox.

Invoke this function to place a packet on a mailbox's send queue. NB. The packet may not actually be delivered to the remote thread until a later call to [sc_mailbox_poll\(\)](#).

10.14.2.6 `sc_mailbox_send_list()`

```
void sc_mailbox_send_list (
    struct sc_mailbox * mailbox,
    struct sc_packet_list * list )
```

Send a list of packets through a mailbox to another thread.

Parameters

<i>mailbox</i>	The mailbox.
<i>list</i>	The packets to send.

This function should only be invoked on an unmanaged mailbox.

Invoke this function to place packets on a mailbox's send queue. NB. The packets may not actually be delivered to the remote thread until a later call to [sc_mailbox_poll\(\)](#).

10.14.2.7 `sc_mailbox_set_recv()`

```
int sc_mailbox_set_recv (
    struct sc_mailbox * mailbox,
    struct sc_node * node,
    const char * name_opt )
```

Connect a mailbox to a node.

Parameters

<i>mailbox</i>	The mailbox.
<i>node</i>	The node.
<i>name_opt</i>	Optional ingress port name (may be NULL).

Returns

0 on success, or a negative error code.

Connect the output of a mailbox to a node. Packets passed to the send-node of the paired mailbox are passed to node.

10.15 misc.h File Reference

Miscellaneous utility functions.

Functions

- `int sc_join_mcast_group (struct sc_session *scs, const char *interface, const char *group)`
Join a multicast group.

10.15.1 Detailed Description

Miscellaneous utility functions.

10.15.2 Function Documentation

10.15.2.1 sc_join_mcast_group()

```
int sc_join_mcast_group (  
    struct sc_session * scs,  
    const char * interface,  
    const char * group )
```

Join a multicast group.

Parameters

<i>scs</i>	A session
<i>interface</i>	The network interface to join on
<i>group</i>	The multicast group to join

Returns

0 on success, or a negative error code.

This function joins multicast group `group` on interface `interface`. This is needed when you need to use the IGMP protocol to arrange that multicast packets are delivered to the adapter.

10.16 node.h File Reference

`Sc_node`: An object that processes packets.

Functions

- `int sc_node_alloc` (struct `sc_node` **node_out, const struct `sc_attr` *attr, struct `sc_thread` *thread, const struct `sc_node_factory` *factory, const struct `sc_arg` *args, int n_args)
Allocate a packet processing node.
- `int sc_node_alloc_named` (struct `sc_node` **node_out, const struct `sc_attr` *attr, struct `sc_thread` *thread, const char *factory_name, const char *lib_name, const struct `sc_arg` *args, int n_args)
Allocate a packet processing node by name.
- `int sc_node_alloc_from_str` (struct `sc_node` **node_out, const struct `sc_attr` *attr, struct `sc_thread` *thread, const char *node_spec)
Allocate a packet processing node using a string specification.
- `int sc_node_add_link` (struct `sc_node` *from_node, const char *link_name, struct `sc_node` *to_node, const char *to_name_opt)
Add a link from one node to another.
- `struct sc_thread * sc_node_get_thread` (const struct `sc_node` *node)
Return the thread associated with a node.
- `int sc_node_factory_lookup` (const struct `sc_node_factory` **factory_out, struct `sc_session` *session, const char *factory_name, const char *lib_name)
Find a node factory.
- `void sc_node_add_info_str` (struct `sc_node` *node, const char *field_name, const char *field_val)
Export information to solar_capture_monitor.
- `void sc_node_add_info_int` (struct `sc_node` *node, const char *field_name, int64_t field_val)
Export information to solar_capture_monitor.
- `struct sc_object * sc_node_to_object` (struct `sc_node` *node)
Convert an `sc_node` to an `sc_object`.
- `struct sc_node * sc_node_from_object` (struct `sc_object` *obj)
Convert an `sc_object` to an `sc_node`.

10.16.1 Detailed Description

`Sc_node`: An object that processes packets.

10.16.2 Function Documentation

10.16.2.1 `sc_node_add_info_int()`

```
void sc_node_add_info_int (
    struct sc_node * node,
    const char * field_name,
    int64_t field_val )
```

Export information to `solar_capture_monitor`.

Parameters

<code>node</code>	The node exporting state.
<code>field_name</code>	Name of field.
<code>field_val</code>	State to export.

Use this function to export static runtime information about a node to `solar_capture_monitor`. This function can be used in the implementation of a new node type, or in an application using a node.

See also [sc_node_add_info_str\(\)](#) and [sc_node_export_state\(\)](#).

10.16.2.2 `sc_node_add_info_str()`

```
void sc_node_add_info_str (
    struct sc_node * node,
    const char * field_name,
    const char * field_val )
```

Export information to `solar_capture_monitor`.

Parameters

<i>node</i>	The node exporting state.
<i>field_name</i>	Name of field.
<i>field_val</i>	State to export.

Use this function to export static runtime information about a node to `solar_capture_monitor`. This function can be used in the implementation of a new node type, or in an application using a node.

See also [sc_node_add_info_int\(\)](#) and [sc_node_export_state\(\)](#).

10.16.2.3 `sc_node_add_link()`

```
int sc_node_add_link (
    struct sc_node * from_node,
    const char * link_name,
    struct sc_node * to_node,
    const char * to_name_opt )
```

Add a link from one node to another.

Parameters

<i>from_node</i>	Node to connect from.
<i>link_name</i>	Name of the <code>from_node</code> 's egress link.
<i>to_node</i>	Node to connect to.
<i>to_name_opt</i>	Optional ingress port name (may be NULL).

Returns

0 on success, or a negative error code.

Packets flow from node to node along links. This function adds a link from `from_node` to `to_node`.

`link_name` identifies `from_node`'s egress link. By convention the default egress link is named "".

Some node types support multiple ingress ports so that the node can receive and separate multiple incoming packet streams. The name of the ingress port is given by `to_name_opt`.

Since SolarCapture 1.1, if the nodes are in different threads then this function automatically creates a link between the threads using mailboxes.

10.16.2.4 `sc_node_alloc()`

```
int sc_node_alloc (
    struct sc_node ** node_out,
    const struct sc_attr * attr,
    struct sc_thread * thread,
    const struct sc_node_factory * factory,
    const struct sc_arg * args,
    int n_args )
```

Allocate a packet processing node.

Parameters

<i>node_out</i>	The allocated node is returned here
<i>attr</i>	Attributes
<i>thread</i>	The thread the node will be in
<i>factory</i>	A node factory
<i>args</i>	An array of arguments for node initialisation
<i>n_args</i>	The number of arguments

Returns

0 on success, or a negative error code.

Nodes perform packet processing services such as filtering, packet modification, import/export and packet injection.

A node factory allocates nodes of a particular type, and the argument list provides configuration for the node instance.

Use this function when you have a pointer to the node factory. For built-in nodes or nodes in a separate library it is simpler to use [sc_node_alloc_named\(\)](#).

10.16.2.5 `sc_node_alloc_from_str()`

```
int sc_node_alloc_from_str (
    struct sc_node ** node_out,
    const struct sc_attr * attr,
    struct sc_thread * thread,
    const char * node_spec )
```

Allocate a packet processing node using a string specification.

Parameters

<i>node_out</i>	The allocated node is returned here.
<i>attr</i>	Attributes.
<i>thread</i>	The thread the node will be in.
<i>node_spec</i>	String giving the node type and arguments.

Returns

0 on success, or a negative error code.

This function allocates a node as specified in `node_spec`, which is formatted as follows:

`NODE_SPEC := NODE_TYPE [":" ARGV] ARGV := NAME=VAL [":" ARGV]`

Example: "sc_vi_node:interface=eth4;streams=all"

10.16.2.6 `sc_node_alloc_named()`

```
int sc_node_alloc_named (
    struct sc_node ** node_out,
    const struct sc_attr * attr,
    struct sc_thread * thread,
    const char * factory_name,
    const char * lib_name,
    const struct sc_arg * args,
    int n_args )
```

Allocate a packet processing node by name.

Parameters

<i>node_out</i>	The allocated node is returned here.
<i>attr</i>	Attributes.
<i>thread</i>	The thread the node will be in.
<i>factory_name</i>	Name of the node factory.
<i>lib_name</i>	Name of the node library (may be NULL).
<i>args</i>	An array of arguments for node initialisation.
<i>n_args</i>	The number of arguments.

Returns

0 on success, or a negative error code.

Nodes perform packet processing services such as filtering, packet modification, import/export and packet injection.

This function allocates a node of type `factory_name`, and the argument list provides configuration for the node instance.

This function is a short-cut for `sc_node_factory_lookup()` followed by `sc_node_alloc()`.

10.16.2.7 `sc_node_factory_lookup()`

```
int sc_node_factory_lookup (
    const struct sc_node_factory ** factory_out,
    struct sc_session * session,
    const char * factory_name,
    const char * lib_name )
```

Find a node factory.

Parameters

<i>factory_out</i>	The node factory found.
<i>session</i>	The SolarCapture session.
<i>factory_name</i>	Name of the node factory.
<i>lib_name</i>	Name of the node library (may be NULL).

Returns

0 on success, or a negative error code.

Finds the factory of name `factory_name`. It may be a built-in factory (in which case `lib_name` should be NULL) or a factory in an external library.

A factory library is a shared object file that contains one or more node factory instances.

`lib_name` may be NULL, in which case it defaults to being the same as the `factory_name`.

If `lib_name` contains a '/' character it is treated as the full path to the library object file.

Otherwise `lib_name` is the name of the library object file (either with or without a `.so` suffix). This function will search for the library object file in the following directories (in order):

- the current working directory
- directories specified by the `SC_NODE_PATH` environment variable
- `/usr/lib64/solar_capture/site-nodes`
- `/usr/lib/x86_64-linux-gnu/solar_capture/site-nodes`
- `/usr/lib64/solar_capture/nodes`
- `/usr/lib/x86_64-linux-gnu/solar_capture/nodes`

Depending on the target system, not all of the above directories may exist. In particular, the subdirectories of `/usr/lib/x86_64-linux-gnu/` will only exist on Debian-derived systems using the multiarch structure for library folders. This is not expected to cause a problem at runtime.

If we decide to support 32-bit builds again, these directories will be searched instead (in order):

- the current working directory
- directories specified by the `SC_NODE_PATH` environment variable
- `/usr/lib/solar_capture/site-nodes`
- `/usr/lib/i386-linux-gnu/solar_capture/site-nodes`
- `/usr/lib/solar_capture/nodes`
- `/usr/lib/i386-linux-gnu/solar_capture/nodes`

If a library containing the named factory is not found by this search, the built-in nodes are searched last.

10.16.2.8 `sc_node_from_object()`

```
struct sc_node* sc_node_from_object (
    struct sc_object * obj )
```

Convert an `sc_object` to an `sc_node`.

Parameters

<code>obj</code>	An <code>sc_object</code> instance or NULL
------------------	--

Returns

NULL if `obj` is NULL otherwise the `sc_node`.

Also returns NULL if `obj` is not of type `SC_OBJ_NODE`.

10.16.2.9 `sc_node_get_thread()`

```
struct sc_thread* sc_node_get_thread (
    const struct sc_node * node )
```

Return the thread associated with a node.

Parameters

<code>node</code>	The node.
-------------------	-----------

Returns

The thread associated with the node.

10.16.2.10 `sc_node_to_object()`

```
struct sc_object* sc_node_to_object (  
    struct sc_node * node )
```

Convert an `sc_node` to an `sc_object`.

Parameters

<code>node</code>	An <code>sc_node</code> instance or NULL
-------------------	--

Returns

NULL if `node` is NULL otherwise the `sc_object`.

10.17 `object.h` File Reference

`sc_object`: Opaque object interface. Use this to pass all types of data that are not ints, doubles or char arrays (see `SC_PARAM_INT`, `SC_PARAM_DBL` and `SC_PARAM_STR` respectively for these).

Enumerations

- enum `sc_object_type` {
 SC_OBJ_ANY, **SC_OBJ_OPAQUE**, **SC_OBJ_PKT_PREDICATE**, **SC_OBJ_C_ATTR**,
 SC_OBJ_NODE, **SC_OBJ_POOL** }

The type of data the `sc_object` contains.

Functions

- enum `sc_object_type` `sc_object_type` (struct `sc_object` *obj)
Return the type of data contained within the `sc_object`.
- int `sc_opaque_alloc` (struct `sc_object` **obj_out, void *opaque)
Allocate memory for an opaque `sc_object`.
- void `sc_opaque_free` (struct `sc_object` *obj)
Free an `sc_object` previously allocated using `sc_opaque_alloc`. Only use this to free an opaque `sc_object`. The underlying data wrapped by this object will not be freed.
- void `sc_opaque_set_ptr` (struct `sc_object` *obj, void *opaque)
Set the opaque pointer in an `sc_object`.
- void * `sc_opaque_get_ptr` (const struct `sc_object` *obj)
Get the opaque pointer stored in an `sc_object`.

10.17.1 Detailed Description

`sc_object`: Opaque object interface. Use this to pass all types of data that are not ints, doubles or char arrays (see `SC_PARAM_INT`, `SC_PARAM_DBL` and `SC_PARAM_STR` respectively for these).

10.17.2 Enumeration Type Documentation

10.17.2.1 `sc_object_type`

```
enum sc_object_type
```

The type of data the `sc_object` contains.

Enumerator

<code>SC_OBJ_OPAQUE</code>	An opaque pointer
<code>SC_OBJ_PKT_PREDICATE</code>	A packet predicate (see <code>sc_pkt_predicate</code>)
<code>SC_OBJ_C_ATTR</code>	Const attributes (see <code>sc_attr</code>)
<code>SC_OBJ_NODE</code>	A node (see <code>sc_node</code>)
<code>SC_OBJ_POOL</code>	A packet pool

10.17.3 Function Documentation

10.17.3.1 `sc_object_type()`

```
enum sc_object_type sc_object_type (
    struct sc_object * obj )
```

Return the type of data contained within the `sc_object`.

Parameters

<code>obj</code>	The object to check the data type of.
------------------	---------------------------------------

Returns

The type of data contained within the `sc_object`.

10.17.3.2 `sc_opaque_alloc()`

```
int sc_opaque_alloc (
    struct sc_object ** obj_out,
    void * opaque )
```

Allocate memory for an opaque `sc_object`.

Parameters

<i>obj_out</i>	On success the allocated object.
<i>opaque</i>	A pointer to the data to be wrapped by the object.

Returns

0 on success.

10.17.3.3 `sc_opaque_free()`

```
void sc_opaque_free (
    struct sc_object * obj )
```

Free an `sc_object` previously allocated using `sc_opaque_alloc`. Only use this to free an opaque `sc_object`. The underlying data wrapped by this object will not be freed.

Parameters

<i>obj</i>	The object to free
------------	--------------------

10.17.3.4 `sc_opaque_get_ptr()`

```
void* sc_opaque_get_ptr (
    const struct sc_object * obj )
```

Get the opaque pointer stored in an `sc_object`.

Parameters

<i>obj</i>	The object to fetch the opaque pointer from.
------------	--

Returns

The opaque pointer.

10.17.3.5 `sc_opaque_set_ptr()`

```
void sc_opaque_set_ptr (
    struct sc_object * obj,
    void * opaque )
```

Set the opaque pointer in an `sc_object`.

Parameters

<i>obj</i>	The object to set the pointer on
<i>opaque</i>	The new opaque pointer to use

10.18 packed_stream.h File Reference

[sc_packed_packet](#): The packed-stream encapsulation.

Data Structures

- struct [sc_packed_packet](#)
A packed-stream packet.

Macros

- #define [SC_PS_FLAG_CLOCK_SET](#) 0x1
- #define [SC_PS_FLAG_CLOCK_IN_SYNC](#) 0x2
- #define [SC_PS_FLAG_BAD_FCS](#) 0x4
- #define [SC_PS_FLAG_BAD_L4_CSUM](#) 0x8
- #define [SC_PS_FLAG_BAD_L3_CSUM](#) 0x10

Functions

- struct [sc_packed_packet](#) **__attribute__** ((packed))
- static struct [sc_packed_packet](#) * [sc_packed_packet_next](#) (const struct [sc_packed_packet](#) *ps_pkt)
Iterate from one packed-stream header to the next.
- static void * [sc_packed_packet_payload](#) (const struct [sc_packed_packet](#) *ps_pkt)
Return a pointer to the packet payload.
- static struct [sc_packed_packet](#) * [sc_packet_packed_first](#) (struct [sc_packet](#) *pkt)
Return the first packet header in a packed-stream buffer.
- static struct [sc_packed_packet](#) * [sc_packet_packed_end](#) (struct [sc_packet](#) *pkt)
Return a pointer to the end of a packed-stream buffer.
- static struct [sc_packed_packet](#) * [sc_packet_iov_packed_first](#) (struct [sc_packet](#) *pkt, unsigned iov_i)
- static struct [sc_packed_packet](#) * [sc_packet_iov_packed_end](#) (struct [sc_packet](#) *pkt, unsigned iov_i)

Variables

- uint16_t [ps_next_offset](#)
- uint8_t [ps_pkt_start_offset](#)
- uint8_t [ps_flags](#)
- uint16_t [ps_cap_len](#)
- uint16_t [ps_orig_len](#)
- uint32_t [ps_ts_sec](#)
- uint32_t [ps_ts_nsec](#)

10.18.1 Detailed Description

[sc_packed_packet](#): The packed-stream encapsulation.

10.18.2 Macro Definition Documentation

10.18.2.1 SC_PS_FLAG_BAD_FCS

```
#define SC_PS_FLAG_BAD_FCS 0x4
```

Mask for [sc_packed_packet](#) flags, bad FCS

10.18.2.2 SC_PS_FLAG_BAD_L3_CSUM

```
#define SC_PS_FLAG_BAD_L3_CSUM 0x10
```

Mask for [sc_packed_packet](#) flags, bad layer 3 checksum

10.18.2.3 SC_PS_FLAG_BAD_L4_CSUM

```
#define SC_PS_FLAG_BAD_L4_CSUM 0x8
```

Mask for [sc_packed_packet](#) flags, bad layer 4 checksum

10.18.2.4 SC_PS_FLAG_CLOCK_IN_SYNC

```
#define SC_PS_FLAG_CLOCK_IN_SYNC 0x2
```

Mask for [sc_packed_packet](#) flags, clock in sync

10.18.2.5 SC_PS_FLAG_CLOCK_SET

```
#define SC_PS_FLAG_CLOCK_SET 0x1
```

Mask for [sc_packed_packet](#) flags, clock set

10.18.3 Function Documentation

10.18.3.1 sc_packed_packet_next()

```
static struct sc\_packed\_packet* sc_packed_packet_next (  
    const struct sc\_packed\_packet * ps_pkt ) [static]
```

Iterate from one packed-stream header to the next.

Parameters

<i>ps_pkt</i>	A packed-stream packet header
---------------	-------------------------------

Returns

The next packed-stream packet in the buffer.

10.18.3.2 `sc_packed_packet_payload()`

```
static void* sc_packed_packet_payload (
    const struct sc\_packed\_packet * ps_pkt ) [inline], [static]
```

Return a pointer to the packet payload.

Parameters

<i>ps_pkt</i>	A packed-stream packet header
---------------	-------------------------------

Returns

The start of the packet payload.

10.18.3.3 `sc_packet_packed_end()`

```
static struct sc\_packed\_packet* sc_packet_packed_end (
    struct sc\_packet * pkt ) [static]
```

Return a pointer to the end of a packed-stream buffer.

Parameters

<i>pkt</i>	An sc_packet containing packed-stream encoded packets
------------	---

Returns

A pointer to the end of the buffer. This can be compared with the pointer returned by [sc_packed_packet_next\(\)](#) to determine whether the last packet has been consumed.

10.18.3.4 `sc_packet_packed_first()`

```
static struct sc\_packed\_packet* sc_packet_packed_first (
    struct sc\_packet * pkt ) [static]
```

Return the first packet header in a packed-stream buffer.

Parameters

<i>pkt</i>	An sc_packet containing packed-stream encoded packets
------------	---

Returns

The [sc_packed_packet](#) header for the first packet.

10.18.4 Variable Documentation

10.18.4.1 ps_cap_len

`uint16_t ps_cap_len`

Number of bytes of packet payload stored.

10.18.4.2 ps_flags

`uint8_t ps_flags`

SC_PS_FLAG_* flags.

10.18.4.3 ps_next_offset

`uint16_t ps_next_offset`

Offset of next packet from start of this struct.

10.18.4.4 ps_orig_len

`uint16_t ps_orig_len`

Original length of the frame.

10.18.4.5 ps_pkt_start_offset

`uint8_t ps_pkt_start_offset`

Offset of packet payload from start of this struct.

10.18.4.6 ps_ts_nsec

uint32_t ps_ts_nsec

Timestamp (nanoseconds).

10.18.4.7 ps_ts_sec

uint32_t ps_ts_sec

Timestamp (seconds).

10.19 pkt_pool.h File Reference

sc_pool: A pool of packet buffers.

Functions

- int [sc_pool_get_packets](#) (struct [sc_packet_list](#) *list, struct [sc_pool](#) *pool, int min_packets, int max_packets)
Get packet buffers from a pool.
- void [sc_pool_return_packets](#) (struct [sc_pool](#) *pool, struct [sc_packet_list](#) *list)
Return packets to a pool.
- void [sc_pool_on_threshold](#) (struct [sc_pool](#) *pool, struct [sc_callback](#) *event, int threshold)
Request a callback when the pool is refilled.
- struct [sc_packet](#) * [sc_pool_duplicate_packet](#) (struct [sc_pool](#) *pool, struct [sc_packet](#) *packet, int snap)
Duplicate a packet.
- struct [sc_packet](#) * [sc_pool_duplicate_packed_packet](#) (struct [sc_pool](#) *pool, const struct [sc_packed_packet](#) *psp, int snap)
Duplicate a packed-stream packet.
- int [sc_packet_append_iovec_ptr](#) (struct [sc_packet](#) *packet, struct [sc_pool](#) *pool, struct [sc_iovec_ptr](#) *iovp, int snap)
Append data to a packet.
- struct [sc_node](#) * [sc_pool_set_refill_node](#) (struct [sc_pool](#) *pool, struct [sc_node](#) *node)
Set the refill node for a pool.
- int [sc_pool_wraps_node](#) (struct [sc_pool](#) *pool, struct [sc_node](#) *node)
Indicate that a pool is used to wrap packets from a node.
- struct [sc_object](#) * [sc_pool_to_object](#) (struct [sc_pool](#) *pool)
Convert an [sc_pool](#) to an [sc_object](#).
- struct [sc_pool](#) * [sc_pool_from_object](#) (struct [sc_object](#) *obj)
Convert an [sc_object](#) to an [sc_pool](#).
- uint64_t [sc_pool_get_buffer_size](#) (struct [sc_pool](#) *pool)
Get the minimum buffer size provided by this pool.

10.19.1 Detailed Description

sc_pool: A pool of packet buffers.

10.19.2 Function Documentation

10.19.2.1 `sc_packet_append_iovec_ptr()`

```
int sc_packet_append_iovec_ptr (
    struct sc_packet * packet,
    struct sc_pool * pool,
    struct sc_iovec_ptr * iovp,
    int snap )
```

Append data to a packet.

Parameters

<i>packet</i>	The packet to append data to
<i>pool</i>	Packet pool to allocate frag buffers from (optional)
<i>iovp</i>	Identifies the data to copy in
<i>snap</i>	The maximum number of bytes to copy in

Returns

- 0 if all the requested data could be appended.
- 1 if more space was needed and it was not possible to allocate fragment buffers
- 2 if the packet runs out of space (ie. the fragments chain would exceed the maximum chain length).

If you need to know the number of bytes appended, compare the packet `frame_len` before and after the call.

10.19.2.2 `sc_pool_duplicate_packed_packet()`

```
struct sc_packet* sc_pool_duplicate_packed_packet (
    struct sc_pool * pool,
    const struct sc_packed_packet * psp,
    int snap )
```

Duplicate a packed-stream packet.

Parameters

<i>pool</i>	The pool to allocate buffers from
<i>psp</i>	The packed-stream packet to duplicate
<i>snap</i>	The maximum number of bytes to copy

Returns

The duplicated packet or NULL if insufficient buffers.

10.19.2.3 `sc_pool_duplicate_packet()`

```
struct sc_packet* sc_pool_duplicate_packet (
    struct sc_pool * pool,
    struct sc_packet * packet,
    int snap )
```

Duplicate a packet.

Parameters

<i>pool</i>	The pool to allocate buffers from
<i>packet</i>	The packet to duplicate
<i>snap</i>	The maximum number of bytes to copy

Returns

The duplicated packet or NULL if insufficient buffers.

10.19.2.4 sc_pool_from_object()

```
struct sc_pool* sc_pool_from_object (
    struct sc_object * obj )
```

Convert an [sc_object](#) to an [sc_pool](#).

Parameters

<i>obj</i>	An sc_object instance or NULL
------------	---

Returns

NULL if *obj* is NULL otherwise the [sc_pool](#).

Also returns NULL if *obj* is not of type [SC_OBJ_POOL](#).

10.19.2.5 sc_pool_get_buffer_size()

```
uint64_t sc_pool_get_buffer_size (
    struct sc_pool * pool )
```

Get the minimum buffer size provided by this pool.

Parameters

<i>pool</i>	An sc_pool instance
-------------	-------------------------------------

Returns

The minimum buffer size provided by this pool.

If called at prep time, the size returned returned may be less than the size of buffers provided by this pool.

10.19.2.6 sc_pool_get_packets()

```
int sc_pool_get_packets (
    struct sc_packet_list * list,
    struct sc_pool * pool,
    int min_packets,
    int max_packets )
```

Get packet buffers from a pool.

Parameters

<i>list</i>	List where retrieved packets are placed
<i>pool</i>	The packet pool
<i>min_packets</i>	Minimum number of buffers to be returned
<i>max_packets</i>	Maximum number of buffers to be returned

Returns

The number of buffers added to `list`, or -1 if the minimum could not be satisfied.

`list` must be initialised on entry (and may already contain some packets), but need not be finalised. The list is finalised on return unless an error is returned (in which case the list is not modified).

Each packet returned is initialised as follows: `pkt->flags = 0`; `pkt->frame_len = 0`; `pkt->iovlen = 1`; `pkt->iov[0]` gives the base and extent of the DMA area The fragment list is empty

The following packet fields have undefined values: `ts_sec`, `ts_nsec`.

10.19.2.7 `sc_pool_on_threshold()`

```
void sc_pool_on_threshold (
    struct sc_pool * pool,
    struct sc_callback * event,
    int threshold )
```

Request a callback when the pool is refilled.

Parameters

<i>pool</i>	The packet pool
<i>event</i>	The event object
<i>threshold</i>	Event fires when pool has \geq threshold buffers

Registers an event handler that is invoked when the pool fill level reaches the specified threshold. If the pool fill level is already at or above the threshold, the handler will be invoked as soon as possible.

10.19.2.8 `sc_pool_return_packets()`

```
void sc_pool_return_packets (
    struct sc_pool * pool,
    struct sc_packet_list * list )
```

Return packets to a pool.

Parameters

<i>pool</i>	The packet pool
<i>list</i>	List of packets to return

`list` must be initialised on entry, but can be empty. The packets on the list can have frags.

10.19.2.9 `sc_pool_set_refill_node()`

```
struct sc_node* sc_pool_set_refill_node (
    struct sc_pool * pool,
    struct sc_node * node )
```

Set the refill node for a pool.

Parameters

<code>pool</code>	A packet pool
<code>node</code>	A refill node

Returns

The refill node This function sets `node` to be the refill node for `pool`. SolarCapture sets up the necessary links so that when packet buffers from `pool` are freed, they will be forwarded to `node`.

It is expected that `node` will normally return packets to the pool by calling `sc_pool_return_packets`.

This call is only needed if some action needs to be taken before returning freed buffers to the pool. The builtin nodes `sc_wrap_undo` and `sc_ref_count_undo` can be used as pool refill nodes.

10.19.2.10 `sc_pool_to_object()`

```
struct sc_object* sc_pool_to_object (
    struct sc_pool * pool )
```

Convert an `sc_pool` to an `sc_object`.

Parameters

<code>pool</code>	An <code>sc_pool</code> instance or NULL
-------------------	--

Returns

NULL if `pool` is NULL otherwise the `sc_object`.

10.19.2.11 `sc_pool_wraps_node()`

```
int sc_pool_wraps_node (
    struct sc_pool * pool,
    struct sc_node * node )
```

Indicate that a pool is used to wrap packets from a node.

Parameters

<i>pool</i>	A packet pool
<i>node</i>	A node

Returns

0 on success, or a negative error code.

This function is used to indicate that packets from `pool` are used to wrap packets that are delivered to `node`.

This allows SolarCapture to ensure that the packet pools sending packets to `node` can be configured appropriately. For example, if these wrapped packets reach an `sc_injector`, it may be necessary to DMA map the underlying packet buffers.

10.20 predicate.h File Reference

[sc_pkt_predicate](#): Interface for testing properties of packets.

Data Structures

- struct [sc_pkt_predicate](#)
A packet predicate object.

Typedefs

- typedef int() [sc_pkt_predicate_test_fn](#)(struct [sc_pkt_predicate](#) *, struct [sc_packet](#) *)
A packet predicate test function. It should return 1 (true), or 0 (false).

Functions

- int [sc_pkt_predicate_alloc](#) (struct [sc_pkt_predicate](#) **pred_out, int private_bytes)
Allocate a packet predicate object.
- struct [sc_object](#) * [sc_pkt_predicate_to_object](#) (struct [sc_pkt_predicate](#) *pred)
Convert a [sc_pkt_predicate](#) into a [sc_object](#).
- struct [sc_pkt_predicate](#) * [sc_pkt_predicate_from_object](#) (struct [sc_object](#) *obj)
Convert a [sc_object](#) into a [sc_pkt_predicate](#).

Variables

- struct [sc_pkt_predicate](#) [__attribute__](#)

10.20.1 Detailed Description

`sc_pkt_predicate`: Interface for testing properties of packets.

10.20.2 Function Documentation

10.20.2.1 `sc_pkt_predicate_alloc()`

```
int sc_pkt_predicate_alloc (
    struct sc_pkt_predicate ** pred_out,
    int private_bytes )
```

Allocate a packet predicate object.

Parameters

<code>pred_out</code>	On success the allocated <code>sc_pkt_predicate</code> object.
<code>private_bytes</code>	Size of private memory area wanted.

Returns

0 on success.

Packet predicates are used to test packets against some criteria. The test function should return true (1) or false (0).

If `private_bytes` is non-zero then `pred_private` is initialised with a pointer to a region of memory of size `private_bytes`. The `pred_private` field may be used by the implementation to hold state.

10.20.2.2 `sc_pkt_predicate_from_object()`

```
struct sc_pkt_predicate* sc_pkt_predicate_from_object (
    struct sc_object * obj )
```

Convert a `sc_object` into a `sc_pkt_predicate`.

Parameters

<code>obj</code>	An <code>sc_object</code> instance or NULL
------------------	--

Returns

NULL if `obj` is NULL otherwise the converted `sc_pkt_predicate`.

10.20.2.3 `sc_pkt_predicate_to_object()`

```
struct sc_object* sc_pkt_predicate_to_object (
    struct sc_pkt_predicate * pred )
```

Convert a `sc_pkt_predicate` into a `sc_object`.

Parameters

<code>pred</code>	An <code>sc_pkt_predicate</code> instance or NULL
-------------------	---

Returns

NULL if `pred` is NULL otherwise the converted `sc_object`.

10.21 session.h File Reference

`sc_session`: A set of threads and other objects.

Data Structures

- struct `sc_session_error`
A SolarCapture session error object returned by `sc_session_error_get`.

Functions

- int `sc_session_alloc` (struct `sc_session **scs_out`, const struct `sc_attr *attr`)
Allocate a SolarCapture session.
- int `sc_session_destroy` (struct `sc_session *scs`)
Destroy a SolarCapture session.
- int `sc_session_prepare` (struct `sc_session *scs`)
Prepare a SolarCapture session.
- int `sc_session_go` (struct `sc_session *scs`)
Start a SolarCapture session.
- int `sc_session_run` (struct `sc_session *scs`, int `*exit_code_out`)
Start a SolarCapture session and wait until it stops.
- int `sc_session_pause` (struct `sc_session *scs`)
Pause a SolarCapture session.
- int `sc_session_stop` (struct `sc_session *scs`, int `exit_code`)
Stop a SolarCapture session, causing `sc_session_run()` to return.
- struct `sc_session_error * sc_session_error_get` (struct `sc_session *scs`)
Returns an error from a SolarCapture session.
- void `sc_session_error_free` (struct `sc_session *scs`, struct `sc_session_error *err`)
Frees an error object.

10.21.1 Detailed Description

`sc_session`: A set of threads and other objects.

10.21.2 Function Documentation

10.21.2.1 `sc_session_alloc()`

```
int sc_session_alloc (
    struct sc_session ** scs_out,
    const struct sc_attr * attr )
```

Allocate a SolarCapture session.

Parameters

<code>scs_out</code>	The allocated session object is returned here
<code>attr</code>	Attributes for the new session

Returns

0 on success, or a negative error code.

This function allocates a SolarCapture session.

A session comprises a set of threads, VIs, nodes and/or other SolarCapture objects.

10.21.2.2 `sc_session_destroy()`

```
int sc_session_destroy (
    struct sc_session * scs )
```

Destroy a SolarCapture session.

Parameters

<code>scs</code>	The session
------------------	-------------

This call stops the session and frees all of the associated resources, including threads, nodes etc.

10.21.2.3 `sc_session_error_free()`

```
void sc_session_error_free (
    struct sc_session * scs,
    struct sc_session_error * err )
```

Frees an error object.

Parameters

<i>scs</i>	The session
<i>err</i>	The error

Frees a [sc_session_error](#) pointer returned by [sc_session_error_get](#).

10.21.2.4 [sc_session_error_get\(\)](#)

```
struct sc\_session\_error* sc\_session\_error\_get (  
    struct sc\_session * scs )
```

Returns an error from a SolarCapture session.

Parameters

<i>scs</i>	The session
------------	-------------

Returns

A pointer to a [sc_session_error](#) struct representing the error encountered by session *scs*. The caller should pass the pointer to [sc_session_error_free](#) once done with it. If no error has occurred, this function returns NULL.

10.21.2.5 [sc_session_go\(\)](#)

```
int sc\_session\_go (  
    struct sc\_session * scs )
```

Start a SolarCapture session.

Parameters

<i>scs</i>	The session
------------	-------------

Returns

0 on success, or a negative error code.

Prepare the session *scs* (if necessary) and start the managed threads. This is usually called just once, after allocating resources. It can also be called after [sc_session_pause\(\)](#) to restart a paused session.

10.21.2.6 [sc_session_pause\(\)](#)

```
int sc\_session\_pause (  
    struct sc\_session * scs )
```

Pause a SolarCapture session.

Parameters

<code>scs</code>	The session
------------------	-------------

Returns

0 on success, or a negative error code.

Pause the threads managed by session `scs`.

This function must not be invoked by a SolarCapture managed thread.

10.21.2.7 `sc_session_prepare()`

```
int sc_session_prepare (
    struct sc_session * scs )
```

Prepare a SolarCapture session.

Parameters

<code>scs</code>	The session
------------------	-------------

Returns

0 on success, or a negative error code.

Prepare the session `scs`. This step includes finalising resource allocations, preparing nodes, and starting packet capture. Managed threads are started in the "paused" state.

Note that although packet capture is started, you may get packet loss if the threads managing '`sc_vi`'s are not started soon afterwards.

Call [sc_session_go\(\)](#) to start the managed threads and begin packet processing.

10.21.2.8 `sc_session_run()`

```
int sc_session_run (
    struct sc_session * scs,
    int * exit_code_out )
```

Start a SolarCapture session and wait until it stops.

Parameters

<code>scs</code>	The session
<code>exit_code_out</code>	Exit code from sc_session_stop() returned here

Returns

0 on success, or a negative error code.

This function calls [sc_session_go\(\)](#), and then waits until [sc_session_stop\(\)](#) is called. The exit code passed to [sc_session_stop\(\)](#) is returned via `p_exit_code` (which can be NULL if the exit code is not wanted).

Calling [sc_session_run\(\)](#) changes the default action of the `sc_exit` node so that it calls [sc_session_stop\(\)](#) when the exit condition is met.

10.21.2.9 sc_session_stop()

```
int sc_session_stop (
    struct sc_session * scs,
    int exit_code )
```

Stop a SolarCapture session, causing [sc_session_run\(\)](#) to return.

Parameters

<code>scs</code>	The session
<code>exit_code</code>	Exit code passed to sc_session_run()

Returns

>= 0 on success, or a negative error code.

This function calls [sc_session_pause\(\)](#), and also causes [sc_session_run\(\)](#) to stop waiting and return `exit_code`.

This function can be invoked in an application thread or a SolarCapture managed thread. In the latter case it will return immediately and the work will be deferred to a background thread.

10.22 stream.h File Reference

This header file defines [sc_stream](#) objects for directing packets to a [sc_vi](#) instance. A packet must match all the stream criteria for it to be directed by the stream to an [sc_vi](#) instance.

Functions

- int [sc_stream_alloc](#) (struct [sc_stream](#) **stream_out, const struct [sc_attr](#) *attr, struct [sc_session](#) *scs)
Create a new stream object for this session.
- int [sc_stream_free](#) (struct [sc_stream](#) *stream)
Free a previously created stream.
- int [sc_stream_reset](#) (struct [sc_stream](#) *stream)
Reinitialise a stream.
- int [sc_stream_set_str](#) (struct [sc_stream](#) *stream, const char *str)
Set the stream to match packets identified by a string.
- int [sc_stream_all](#) (struct [sc_stream](#) *stream)
Configure stream to match packets not explicitly steered elsewhere.
- int [sc_stream_mismatch](#) (struct [sc_stream](#) *stream)
Configure stream to match packets not steered elsewhere and not requested by the kernel network stack.
- int [sc_stream_ip_dest_hostport](#) (struct [sc_stream](#) *stream, int protocol, const char *dhost, const char *dport)
Configure this stream to capture all packets with the matching protocol, destination hostname and destination port.
- int [sc_stream_ip_source_hostport](#) (struct [sc_stream](#) *stream, const char *shost, const char *sport)
Configure this stream to capture all packets with the matching protocol, source hostname and source port.
- int [sc_stream_eth_dhost](#) (struct [sc_stream](#) *stream, const uint8_t *mac_addr)

- Add the destination MAC address to the set of fields matched.*

 - int `sc_stream_eth_vlan_id` (struct `sc_stream` *stream, int vlan_id)

Add the VLAN ID to the set of fields matched.
- int `sc_stream_eth_shost` (struct `sc_stream` *stream, const uint8_t *mac_addr)

Add the source MAC address to the set of fields matched.
- int `sc_stream_eth_type` (struct `sc_stream` *stream, uint16_t eth_type)

Add the Ethernet ether_type field to the set of fields matched.
- int `sc_stream_ip_dest_host` (struct `sc_stream` *stream, const char *dhost)

Add the IPv4 destination to the set of fields matched.
- int `sc_stream_ip_dest_port` (struct `sc_stream` *stream, const char *dport)

Add the TCP or UDP destination port to the set of fields matched.
- int `sc_stream_ip_source_host` (struct `sc_stream` *stream, const char *shost)

Add the IPv4 source to the set of fields matched.
- int `sc_stream_ip_source_port` (struct `sc_stream` *stream, const char *sport)

Add the TCP or UDP source port to the set of fields matched.
- int `sc_stream_ip_protocol` (struct `sc_stream` *stream, int protocol)

Add the IP protocol to the set of fields matched.

10.22.1 Detailed Description

This header file defines `sc_stream` objects for directing packets to a `sc_vi` instance. A packet must match all the stream criteria for it to be directed by the stream to an `sc_vi` instance.

10.22.2 Function Documentation

10.22.2.1 `sc_stream_all()`

```
int sc_stream_all (
    struct sc_stream * stream )
```

Configure stream to match packets not explicitly steered elsewhere.

Parameters

<code>stream</code>	A stream object.
---------------------	------------------

This stream captures packets that would otherwise be delivered to the OS kernel network stack, and also packets that would normally be discarded by the adapter when not in promiscuous mode.

Returns

0 on success, or a negative error code.

10.22.2.2 `sc_stream_alloc()`

```
int sc_stream_alloc (
    struct sc_stream ** stream_out,
    const struct sc_attr * attr,
    struct sc_session * scs )
```

Create a new stream object for this session.

Parameters

<i>stream_out</i>	On success, the created stream.
<i>attr</i>	Attributes to pass in.
<i>scs</i>	The session this stream is for.

Returns

0 on success, or a negative error code.

10.22.2.3 `sc_stream_eth_dhost()`

```
int sc_stream_eth_dhost (
    struct sc_stream * stream,
    const uint8_t * mac_addr )
```

Add the destination MAC address to the set of fields matched.

Parameters

<i>stream</i>	A stream object.
<i>mac_addr</i>	The destination MAC address to match.

Returns

0 on success, or a negative error code.

10.22.2.4 `sc_stream_eth_shost()`

```
int sc_stream_eth_shost (
    struct sc_stream * stream,
    const uint8_t * mac_addr )
```

Add the source MAC address to the set of fields matched.

Parameters

<i>stream</i>	A stream object.
<i>mac_addr</i>	The source MAC address to match.

Returns

0 on success, or a negative error code.

10.22.2.5 `sc_stream_eth_type()`

```
int sc_stream_eth_type (
    struct sc_stream * stream,
    uint16_t eth_type )
```

Add the Ethernet `ether_type` field to the set of fields matched.

Parameters

<i>stream</i>	A stream object.
<i>eth_type</i>	The ether_type to match (host endian).

Returns

0 on success, or a negative error code.

10.22.2.6 sc_stream_eth_vlan_id()

```
int sc_stream_eth_vlan_id (
    struct sc_stream * stream,
    int vlan_id )
```

Add the VLAN ID to the set of fields matched.

Parameters

<i>stream</i>	A stream object.
<i>vlan_id</i>	The VLAN ID to match against.

Returns

0 on success, or a negative error code.

10.22.2.7 sc_stream_free()

```
int sc_stream_free (
    struct sc_stream * stream )
```

Free a previously created stream.

Parameters

<i>stream</i>	The stream to free.
---------------	---------------------

Returns

0, always.

10.22.2.8 sc_stream_ip_dest_host()

```
int sc_stream_ip_dest_host (
    struct sc_stream * stream,
    const char * dhost )
```

Add the IPv4 destination to the set of fields matched.

Parameters

<i>stream</i>	A stream object.
<i>dhost</i>	The destination host name or IP to match.

Returns

0 on success, or a negative error code.

10.22.2.9 `sc_stream_ip_dest_hostport()`

```
int sc_stream_ip_dest_hostport (
    struct sc_stream * stream,
    int protocol,
    const char * dhost,
    const char * dport )
```

Configure this stream to capture all packets with the matching protocol, destination hostname and destination port.

Parameters

<i>stream</i>	A stream object.
<i>protocol</i>	The transport layer protocol to match against.
<i>dhost</i>	The destination hostname to match against.
<i>dport</i>	The destination port to match against.

Returns

0 on success, or a negative error code.

10.22.2.10 `sc_stream_ip_dest_port()`

```
int sc_stream_ip_dest_port (
    struct sc_stream * stream,
    const char * dport )
```

Add the TCP or UDP destination port to the set of fields matched.

Parameters

<i>stream</i>	A stream object.
<i>dport</i>	The destination port to match.

Returns

0 on success, or a negative error code.

10.22.2.11 `sc_stream_ip_protocol()`

```
int sc_stream_ip_protocol (
    struct sc_stream * stream,
    int protocol )
```

Add the IP protocol to the set of fields matched.

Parameters

<i>stream</i>	A stream object.
<i>protocol</i>	The IP protocol to match.

Returns

0 on success, or a negative error code.

10.22.2.12 `sc_stream_ip_source_host()`

```
int sc_stream_ip_source_host (
    struct sc_stream * stream,
    const char * shost )
```

Add the IPv4 source to the set of fields matched.

Parameters

<i>stream</i>	A stream object.
<i>shost</i>	The source host name or IP to match.

Returns

0 on success, or a negative error code.

10.22.2.13 `sc_stream_ip_source_hostport()`

```
int sc_stream_ip_source_hostport (
    struct sc_stream * stream,
    const char * shost,
    const char * sport )
```

Configure this stream to capture all packets with the matching protocol, source hostname and source port.

Parameters

<i>stream</i>	A stream object.
<i>shost</i>	The source hostname to match against.
<i>sport</i>	The source port to match against.

Returns

0 on success, or a negative error code.

10.22.2.14 `sc_stream_ip_source_port()`

```
int sc_stream_ip_source_port (
    struct sc_stream * stream,
    const char * sport )
```

Add the TCP or UDP source port to the set of fields matched.

Parameters

<i>stream</i>	A stream object.
<i>sport</i>	The source port to match.

Returns

0 on success, or a negative error code.

10.22.2.15 `sc_stream_mismatch()`

```
int sc_stream_mismatch (
    struct sc_stream * stream )
```

Configure stream to match packets not steered elsewhere and not requested by the kernel network stack.

Parameters

<i>stream</i>	A stream object.
---------------	------------------

This stream matches packets that would normally be discarded by the network adapter when it is not in promiscuous mode.

Returns

0 on success, or a negative error code.

10.22.2.16 `sc_stream_reset()`

```
int sc_stream_reset (
    struct sc_stream * stream )
```

Reinitialise a stream.

Parameters

<i>stream</i>	The stream to reinitialise.
---------------	-----------------------------

Returns

0 on success, or a negative error code.

10.22.2.17 `sc_stream_set_str()`

```
int sc_stream_set_str (
    struct sc_stream * stream,
    const char * str )
```

Set the stream to match packets identified by a string.

Parameters

<i>stream</i>	A stream object.
<i>str</i>	Match criteria.

Returns

0 on success, or a negative error code.

This call is the preferred way of configuring a stream, since it offers the most flexibility.

Different adapter types support matching on different combinations of header fields. The combinations supported also depend on firmware version and firmware variant. (The firmware variant is selected using the `sfboot` utility). In the tables below the firmware variants are identified as follows:

- FF: Full-featured firmware variant
- ULL: Ultra-low latency firmware variant
- CPS: Capture-packed-stream firmware variant

The abbreviated syntax uses one of the formats shown in the table below. The table also shows the adapter firmware variants that support each format.

Abbreviated syntax	SFN7xxx
<code>eth:<dest-mac></code>	FF ULL CPS
<code>eth:vid=<vlan>,<dest-mac></code>	FF ULL
<code>{udp tcp}:<dest-host>:<dest-port></code>	FF ULL CPS
<code>{udp tcp}:<dest-host>:<dest-port>,<source-host>:<source-port></code>	FF ULL
<code>{udp tcp}:vid=<vlan>,<dest-host>:<dest-port></code>	FF
<code>{udp tcp}:vid=<vlan>,<dest-host>:<dest-port>,<source-host>:<source-port></code>	FF

The full syntax allows more flexibility. A stream is constructed as a comma separated list of key-value pairs, except for the special cases "all", "mismatch", "ip", "tcp", and "udp". Available keys are shown in the table below:

Key or key-value pairs	Description
<code>dmac=xx:xx:xx:xx:xx:xx</code>	Match Ethernet destination MAC address.
<code>smac=xx:xx:xx:xx:xx:xx</code>	Match Ethernet source MAC address.
<code>vid=INT</code>	Match Ethernet outer VLAN ID.
<code>eth_type=ip arp INT</code>	Match Ethernet ether_type.
<code>shost=hostname</code>	Match IPv4 source host.
<code>dhost=hostname</code>	Match IPv4 destination host.
<code>ip_protocol=udp tcp INT</code>	Match IPv4 protocol (implies <code>eth_type=ip</code>).
<code>sport=INT</code>	Match TCP or UDP source port.
<code>dport=INT</code>	Match TCP or UDP destination port.
<code>all</code>	All packets not steered elsewhere.
<code>mismatch</code>	All packets not steered elsewhere and not requested by the kernel network stack.
<code>ip</code>	Shorthand for <code>eth_type=ip</code> .
<code>tcp</code>	Shorthand for <code>ip_protocol=tcp</code> .
<code>udp</code>	Shorthand for <code>ip_protocol=udp</code> .

IPv4 addresses may be given as a dotted quad or a host name that can be resolved with `getaddrinfo()`.

Supported combinations of keys are shown in the table below, together with the firmware variants required:

Key combination	SFN7xxx
<code>all</code>	FF ULL CPS
<code>mismatch</code>	FF ULL CPS
<code>vid</code>	FF CPS
<code>dmac</code>	FF ULL CPS
<code>dmac, vid</code>	FF ULL
<code>ip_protocol, dhost, dport</code>	FF ULL CPS
<code>ip_protocol, dhost, dport, shost, sport</code>	FF ULL
<code>[vid,] [dmac,] ip_protocol, dhost, dport</code>	FF
<code>[vid,] [dmac,] ip_protocol, dhost, dport, shost, sport</code>	FF
<code>eth_type</code>	FF ULL
<code>eth_type, vid</code>	FF ULL CPS
<code>eth_type, dmac</code>	FF ULL
<code>ip_protocol</code>	FF ULL
<code>ip_protocol, vid</code>	FF ULL CPS
<code>[vid,] ip_protocol, dmac</code>	FF

10.23 subnode_helper.h File Reference

[sc_subnode_helper](#) node interface.

Data Structures

- struct [sc_subnode_helper](#)
sc_subnode_helper node private state.

10.23.1 Detailed Description

`sc_subnode_helper` node interface.

10.24 thread.h File Reference

`sc_thread`: Representation of a thread in SolarCapture.

```
#include <stdbool.h>
```

Functions

- int `sc_thread_alloc` (struct `sc_thread **thread_out`, const struct `sc_attr *attr`, struct `sc_session *scs`)
Allocate a SolarCapture thread.
- struct `sc_session *` `sc_thread_get_session` (const struct `sc_thread *thread`)
Return the session associated with a thread.
- void `sc_thread_get_time` (const struct `sc_thread *thread`, struct `timespec *time_out`)
Return a thread's "current time".
- void * `sc_thread_calloc` (struct `sc_thread *thread`, `size_t` bytes)
Allocate memory to be used by a thread.
- void * `sc_thread_calloc_aligned` (struct `sc_thread *thread`, `size_t` bytes, int align)
Allocate memory to be used by a thread.
- void `sc_thread_mfree` (struct `sc_thread *thread`, void *mem)
Free memory.
- int `sc_thread_poll` (struct `sc_thread *thread`)
Poll a thread.
- int `sc_thread_poll_timers` (struct `sc_thread *thread`)
Poll a thread's timers.
- int `sc_thread_waitable_fd_get` (struct `sc_thread *thread`, bool edge_triggered)
Return a file descriptor which an application can wait on until the SolarCapture thread is ready to be polled.
- void `sc_thread_waitable_fd_prime` (struct `sc_thread *thread`)
Primes the thread's waitable FD.

10.24.1 Detailed Description

`sc_thread`: Representation of a thread in SolarCapture.

10.24.2 Function Documentation

10.24.2.1 `sc_thread_alloc()`

```
int sc_thread_alloc (  
    struct sc_thread ** thread_out,  
    const struct sc_attr * attr,  
    struct sc_session * scs )
```

Allocate a SolarCapture thread.

Parameters

<i>thread_out</i>	The allocated thread object is returned here
<i>attr</i>	Attributes
<i>scs</i>	The session

Returns

0 on success, or a negative error code.

This function allocates a SolarCapture thread.

Normally SolarCapture creates an OS thread for the `sc_thread` object, and starts the thread when `sc_session_go()` is called. If the 'managed' attribute is set to false, then it is up to the application to create an underlying thread.

10.24.2.2 `sc_thread_calloc()`

```
void* sc_thread_calloc (
    struct sc_thread * thread,
    size_t bytes )
```

Allocate memory to be used by a thread.

Parameters

<i>thread</i>	The thread
<i>bytes</i>	Size of memory area to allocate

This function is intended to be used for allocating small amounts of memory that are used on performance critical paths, such as the private state used by the implementation of a node.

The memory region may overlap cache lines used by other allocations from this API for the same `thread`.

10.24.2.3 `sc_thread_calloc_aligned()`

```
void* sc_thread_calloc_aligned (
    struct sc_thread * thread,
    size_t bytes,
    int align )
```

Allocate memory to be used by a thread.

Parameters

<i>thread</i>	The thread
<i>bytes</i>	Size of memory area wanted
<i>align</i>	Alignment of memory area wanted

This function is intended to be used for allocating small amounts of memory that are used on performance critical paths, such as the private state used by the implementation of a node.

The memory region may overlap cache lines used by other allocations from this API for the same `thread`.

10.24.2.4 `sc_thread_get_time()`

```
void sc_thread_get_time (
    const struct sc_thread * thread,
    struct timespec * time_out )
```

Return a thread's "current time".

Parameters

<i>thread</i>	The thread
<i>time_out</i>	The current time is returned here

Each thread's current time is updated by the polling loop, so may or may not be up-to-date when you call this function. The clock used as the time base is `CLOCK_REALTIME`.

10.24.2.5 `sc_thread_mfree()`

```
void sc_thread_mfree (
    struct sc_thread * thread,
    void * mem )
```

Free memory.

Parameters

<i>thread</i>	The thread
<i>mem</i>	The memory area

Use this function to free memory allocated with `sc_thread_calloc` or `sc_thread_calloc_aligned`.

10.24.2.6 `sc_thread_poll()`

```
int sc_thread_poll (
    struct sc_thread * thread )
```

Poll a thread.

Parameters

<i>thread</i>	The thread.
---------------	-------------

Returns

0 if no work was available to do, or non-zero if work was done (see description).

Use this function to poll an unmanaged SolarCapture thread, causing it to do I/O, push packets through the node graph and perform other work.

The return value indicates whether any work was done. If the return is non-zero, then one of the following has happened: Packets have been received by an `sc_vi`; Packets have been received by an `sc_mailbox`; Packets or messages have been forwarded between nodes; A timer has expired; Other work has been done (such as handling I/O on a managed file descriptor).

This call returns after doing a batch of work. The application should invoke `sc_thread_poll()` repeatedly until it returns 0 to do all work presently available.

Note: For managed threads this functionality is provided internally by `solar_capture`. It is illegal to invoke `sc_thread_poll()` on a managed thread.

10.24.2.7 `sc_thread_poll_timers()`

```
int sc_thread_poll_timers (
    struct sc_thread * thread )
```

Poll a thread's timers.

Parameters

<code>thread</code>	The thread.
---------------------	-------------

Returns

Returns non-zero if any timers expired.

This function polls an unmanaged thread's timers. It should always be invoked once after `sc_session_prepare()` or `sc_session_go()`, and before calling `sc_thread_poll()` for the first time.

It is also good practice to call this function if the thread has not been polled for a long period of time.

10.24.2.8 `sc_thread_waitable_fd_get()`

```
int sc_thread_waitable_fd_get (
    struct sc_thread * thread,
    bool edge_triggered )
```

Return a file descriptor which an application can wait on until the SolarCapture thread is ready to be polled.

The FD returned by this call is typically used with I/O multiplexors such as `select()`, `poll()` and `epoll_wait()`. See also `sc_thread_waitable_fd_prime()`.

In level triggered mode: The FD returned by this call is not yet "primed" and is in the readable state.

In edge triggered mode: The FD returned is not yet "primed" and may or may not be readable. The caller should invoke `sc_thread_poll()` until it returns 0 and call `sc_thread_waitable_fd_prime()` before waiting on the FD.

This call is only supported on unmanaged threads.

Returns an FD on success or -1 on error.

10.24.2.9 `sc_thread_waitable_fd_prime()`

```
void sc_thread_waitable_fd_prime (
    struct sc_thread * thread )
```

Primes the thread's waitable FD.

The application should invoke `sc_thread_waitable_fd_prime()` before waiting on the waitable FD. If there is no outstanding work to do in the associated thread, then this call makes the waitable FD become unready, and it will become ready again once there is work to do.

This call should only be invoked after `sc_thread_poll()` has returned false, indicating that there is no further work for the thread to do. If this rule is not observed then it is possible for there to be further work for the thread to do even while the waitable FD is not ready.

In level triggered mode, once the waitable FD becomes ready it remains ready until `sc_thread_waitable_fd_prime()` is invoked.

In edge triggered mode, the waitable FD may become unready as a side effect of `sc_thread_poll()`.

Once the thread's FD becomes readable, it will remain readable until this function is called. After this call returns, it will be readable only if the thread still has work to do.

NOTE: To be sure the thread has no more work to do, call `sc_thread_poll` in a loop until it returns 0.

Before calling this function the application must have called `sc_thread_waitable_fd_get()`.

10.25 time.h File Reference

Functions for managing time.

Functions

- void `sc_timer_expire_at` (struct `sc_callback` *cb, const struct timespec *time)
Request a callback at a given time.
- void `sc_timer_expire_after_ns` (struct `sc_callback` *cb, int64_t delta_ns)
Request a callback in the future.
- void `sc_timer_push_back_ns` (struct `sc_callback` *cb, int64_t delta_ns)
Push the expiry time further into the future.
- int `sc_timer_get_expiry_time` (const struct `sc_callback` *cb, struct timespec *ts_out)
Return the expiry time of a timer callback.
- static uint64_t `sc_ns_from_ts` (const struct timespec *ts)
Convert a timespec struct to nanoseconds.
- static uint64_t `sc_ns_from_tv` (const struct timeval *tv)
Convert a timeval struct to nanoseconds.
- static uint64_t `sc_ns_from_ms` (uint64_t ms)
Convert milliseconds to nanoseconds.
- static uint64_t `sc_ns_from_us` (uint64_t us)
Convert microseconds to nanoseconds.

10.25.1 Detailed Description

Functions for managing time.

10.25.2 Function Documentation

10.25.2.1 sc_ns_from_ms()

```
static uint64_t sc_ns_from_ms (
    uint64_t ms ) [inline], [static]
```

Convert milliseconds to nanoseconds.

Parameters

<i>ms</i>	The time in milliseconds to convert
-----------	-------------------------------------

Returns

Time in nanoseconds

10.25.2.2 sc_ns_from_ts()

```
static uint64_t sc_ns_from_ts (
    const struct timespec * ts ) [inline], [static]
```

Convert a timespec struct to nanoseconds.

Parameters

<i>ts</i>	The timespec struct to convert
-----------	--------------------------------

Returns

Time in nanoseconds

10.25.2.3 `sc_ns_from_tv()`

```
static uint64_t sc_ns_from_tv (  
    const struct timeval * tv ) [inline], [static]
```

Convert a timeval struct to nanoseconds.

Parameters

<i>tv</i>	The timeval struct to convert
-----------	-------------------------------

Returns

Time in nanoseconds

10.25.2.4 `sc_ns_from_us()`

```
static uint64_t sc_ns_from_us (  
    uint64_t us ) [inline], [static]
```

Convert microseconds to nanoseconds.

Parameters

<i>us</i>	The time in microseconds to convert
-----------	-------------------------------------

Returns

Time in nanoseconds

10.25.2.5 `sc_timer_expire_after_ns()`

```
void sc_timer_expire_after_ns (  
    struct sc_callback * cb,  
    int64_t delta_ns )
```

Request a callback in the future.

Parameters

<i>cb</i>	A callback object
<i>delta_ns</i>	How far in the future in nanoseconds.

The callback will be invoked at or after the specified time delta in nanoseconds. If *delta_ns* is zero or negative then the handler function will be invoked as soon as possible.

10.25.2.6 sc_timer_expire_at()

```
void sc_timer_expire_at (
    struct sc_callback * cb,
    const struct timespec * time )
```

Request a callback at a given time.

Parameters

<i>cb</i>	A callback object
<i>time</i>	Time at which callback is requested

The callback *cb* will be invoked at or after the specified time. If the time is in the past, then the handler function will be invoked as soon as possible.

The time is relative to the system realtime clock (CLOCK_REALTIME), which is the same clock returned by [sc_thread_get_time\(\)](#).

10.25.2.7 sc_timer_get_expiry_time()

```
int sc_timer_get_expiry_time (
    const struct sc_callback * cb,
    struct timespec * ts_out )
```

Return the expiry time of a timer callback.

Parameters

<i>cb</i>	A callback object
<i>ts_out</i>	The expiry time is returned here

Returns

Zero if *cb* is a timer else -1

10.25.2.8 sc_timer_push_back_ns()

```
void sc_timer_push_back_ns (
    struct sc_callback * cb,
    int64_t delta_ns )
```

Push the expiry time further into the future.

Parameters

<i>cb</i>	A callback object
<i>delta_ns</i>	How far in the future in nanoseconds.

This function pushes the expiry time of a timer callback further into the future.

The callback *cb* must either be a currently active timer registered with [sc_timer_expire_at\(\)](#) or [sc_timer_expire_after_ns\(\)](#), or it must be an inactive timer. ie. The most recent use of *cb* must have been as a timer callback.

If *cb* is active, then it is rescheduled at its current expiry time plus *delta_ns*. If it is not active then it is scheduled at its previous expiry time plus *delta_ns*.

10.26 vi.h File Reference

[sc_vi](#): Supports receiving packets from the network.

Functions

- int [sc_vi_alloc](#) (struct [sc_vi](#) **vi_out, const struct [sc_attr](#) *attr, struct [sc_thread](#) *thread, const char *interface)
Allocate a VI instance.
- int [sc_vi_set_recv_node](#) (struct [sc_vi](#) *vi, struct [sc_node](#) *node, const char *name_opt)
Set the node a VI should deliver its received packets to.
- int [sc_vi_add_stream](#) (struct [sc_vi](#) *vi, struct [sc_stream](#) *stream)
Direct a packet stream to a VI.
- struct [sc_thread](#) * [sc_vi_get_thread](#) (const struct [sc_vi](#) *vi)
Return the thread associated with a VI.
- const char * [sc_vi_get_interface_name](#) (const struct [sc_vi](#) *vi)
Return the name of the network interface associated with a VI.
- int [sc_vi_group_alloc](#) (struct [sc_vi_group](#) **vi_out, const struct [sc_attr](#) *attr, struct [sc_session](#) *session, const char *interface, int num_vis)
Allocate a VI group.
- struct [sc_session](#) * [sc_vi_group_get_session](#) (const struct [sc_vi_group](#) *vi_group)
Return the session associated with a VI group.
- int [sc_vi_alloc_from_group](#) (struct [sc_vi](#) **vi_out, const struct [sc_attr](#) *attr, struct [sc_thread](#) *thread, struct [sc_vi_group](#) *vi_group)
Allocate a VI instance from a VI group.
- int [sc_vi_group_add_stream](#) (struct [sc_vi_group](#) *vi_group, struct [sc_stream](#) *stream)
Direct a packet stream to a group of VIs.

10.26.1 Detailed Description

[sc_vi](#): Supports receiving packets from the network.

10.26.2 Function Documentation

10.26.2.1 `sc_vi_add_stream()`

```
int sc_vi_add_stream (
    struct sc_vi * vi,
    struct sc_stream * stream )
```

Direct a packet stream to a VI.

Parameters

<i>vi</i>	The VI receiving packets
<i>stream</i>	The packet stream

Returns

0 on success, or a negative error code.

Arrange for the packet stream identified by *stream* to be copied or steered to *vi*.

10.26.2.2 `sc_vi_alloc()`

```
int sc_vi_alloc (
    struct sc_vi ** vi_out,
    const struct sc_attr * attr,
    struct sc_thread * thread,
    const char * interface )
```

Allocate a VI instance.

Parameters

<i>vi_out</i>	The allocated VI is returned here
<i>attr</i>	Attributes
<i>thread</i>	The thread the VI will be in
<i>interface</i>	The network interface to receive packets from

Returns

0 on success, or a negative error code.

A VI is a "virtual network interface" and supports receiving packets from the network. Packets received by a VI are passed to nodes (`sc_node`) for processing.

10.26.2.3 `sc_vi_alloc_from_group()`

```
int sc_vi_alloc_from_group (
    struct sc_vi ** vi_out,
    const struct sc_attr * attr,
    struct sc_thread * thread,
    struct sc_vi_group * vi_group )
```

Allocate a VI instance from a VI group.

Parameters

<i>vi_out</i>	The allocated VI is returned here
<i>attr</i>	Attributes
<i>thread</i>	The thread the VI will be in
<i>vi_group</i>	The VI group

Returns

0 on success, or a negative error code.

See also [sc_vi_group_alloc\(\)](#) and [sc_vi_alloc\(\)](#).

10.26.2.4 `sc_vi_get_interface_name()`

```
const char* sc_vi_get_interface_name (  
    const struct sc\_vi * vi )
```

Return the name of the network interface associated with a VI.

Parameters

<i>vi</i>	The VI
-----------	--------

Returns

The name of the network interface associated with the [sc_vi](#) object

This call returns the name of the network interface associated with the [sc_vi](#) object. This can be different from the interface name used to create the [sc_vi](#) when application clustering is used.

The network interface name is most often needed so that the application can create an injector on the same interface as a VI.

10.26.2.5 `sc_vi_get_thread()`

```
struct sc\_thread* sc_vi_get_thread (  
    const struct sc\_vi * vi )
```

Return the thread associated with a VI.

Parameters

<i>vi</i>	The VI
-----------	--------

Returns

The thread associated with the VI.

10.26.2.6 `sc_vi_group_add_stream()`

```
int sc_vi_group_add_stream (  
    struct sc\_vi\_group * vi_group,  
    struct sc\_stream * stream )
```

Direct a packet stream to a group of VIs.

Parameters

<i>vi_group</i>	The VI group receiving packets
<i>stream</i>	The packet stream

Returns

The session associated with the VI group.

Arrange for the packet stream identified by *stream* to be copied or steered to the VIs that comprise *vi_group*.

Note that packets are spread over the VIs in a group by computing a hash on the addresses in the packet headers. Normally the hash is computed over the IP addresses, and for TCP packets also the port numbers. The hash selects a VI within the group, so that packets with the same addresses are consistently delivered to the same VI.

If *stream* identifies a set of packets that all have the same source and destination IP addresses (and ports in the case of TCP) then they will all be received by a single VI.

10.26.2.7 `sc_vi_group_alloc()`

```
int sc_vi_group_alloc (
    struct sc_vi_group ** vi_out,
    const struct sc_attr * attr,
    struct sc_session * session,
    const char * interface,
    int num_vis )
```

Allocate a VI group.

Parameters

<i>vi_out</i>	The allocated VI is returned here
<i>attr</i>	Attributes
<i>session</i>	The SolarCapture session
<i>interface</i>	The network interface to receive packets from
<i>num_vis</i>	The number of VIs in the group

Returns

0 on success, or a negative error code.

A VI group provides a way to distribute packet capture over multiple threads. A VI group consists of a set of VIs, each of which receives a distinct subset of the streams directed at the group.

Streams are directed to a group by calling `sc_vi_group_add_stream()`.

While a VI allocated from a group receives packets from streams directed to the group (`sc_vi_group_add_stream()`), it is also possible to use `sc_vi_add_stream()` to direct a specific stream to a specific member of the group.

10.26.2.8 `sc_vi_group_get_session()`

```
struct sc_session* sc_vi_group_get_session (
    const struct sc_vi_group * vi_group )
```

Return the session associated with a VI group.

Parameters

<i>vi_group</i>	The VI group
-----------------	--------------

Returns

The session associated with the VI group.

10.26.2.9 `sc_vi_set_rcv_node()`

```
int sc_vi_set_rcv_node (
    struct sc_vi * vi,
    struct sc_node * node,
    const char * name_opt )
```

Set the node a VI should deliver its received packets to.

Parameters

<i>vi</i>	The VI receiving packets
<i>node</i>	The node to deliver packets to
<i>name_opt</i>	Optional ingress port name (may be NULL)

Returns

0 on success, or a negative error code.

Since SolarCapture 1.1, if `node` is in a different thread from `vi`, then this function automatically creates a link between the threads using mailboxes.

Index

- append_to
 - sc_append_to_list, [61](#)
- append_to_list.h, [87](#)
- args.h, [87](#)
 - SC_ARG_DBL, [88](#)
 - SC_ARG_INT, [88](#)
 - SC_ARG_OBJ, [89](#)
 - SC_ARG_STR, [89](#)
 - sc_param_type, [88](#)
- args.h
 - SC_PARAM_DBL, [88](#)
 - SC_PARAM_INT, [88](#)
 - SC_PARAM_OBJ, [88](#)
 - SC_PARAM_STR, [88](#)
- attr.h, [90](#)
 - sc_attr_alloc, [91](#)
 - sc_attr_doc, [91](#)
 - sc_attr_dup, [91](#)
 - sc_attr_free, [92](#)
 - sc_attr_from_object, [92](#)
 - sc_attr_reset, [92](#)
 - sc_attr_set_from_fmt, [93](#)
 - sc_attr_set_from_str, [93](#)
 - sc_attr_set_int, [93](#)
 - sc_attr_set_str, [94](#)
 - sc_attr_to_object, [94](#)
- cb_handler_fn
 - sc_callback, [64](#)
- cb_link
 - sc_callback, [64](#)
- cb_private
 - sc_callback, [64](#)
- declare_types.h, [95](#)
 - ST_CONSTANT, [96](#)
 - ST_FIELD_STR, [97](#)
 - ST_FIELD, [96](#)
 - ST_STRUCT, [97](#)
- dlist.h, [97](#)
 - SC_CONTAINER, [99](#)
 - SC_DLIST_FOR_EACH_OBJ_SAFE, [100](#)
 - SC_DLIST_FOR_EACH_OBJ, [99](#)
 - sc_dlist_init, [101](#)
 - sc_dlist_pop_head, [102](#)
 - sc_dlist_pop_tail, [102](#)
 - sc_dlist_push_head, [102](#)
 - sc_dlist_push_tail, [103](#)
 - sc_dlist_rehome, [103](#)
 - sc_dlist_remove, [103](#)
- err_errno
 - sc_session_error, [79](#)
- err_file
 - sc_session_error, [79](#)
- err_func
 - sc_session_error, [79](#)
- err_line
 - sc_session_error, [79](#)
- err_msg
 - sc_session_error, [79](#)
- ethernet.h, [104](#)
 - SC_8021Q_VID_MASK, [104](#)
 - SC_ETHERTYPE_8021QinQ, [104](#)
 - SC_ETHERTYPE_8021Q, [104](#)
- event.h, [104](#)
 - sc_callback_alloc, [106](#)
 - sc_callback_alloc2, [106](#)
 - sc_callback_free, [107](#)
 - sc_callback_handler_fn, [105](#)
 - sc_callback_is_active, [107](#)
 - sc_callback_on_idle, [107](#)
 - sc_callback_remove, [108](#)
 - sc_callback_set_description, [108](#)
 - sc_epoll_ctl, [108](#)
- ext_node.h, [109](#)
 - sc_forward, [115](#)
 - sc_forward2, [115](#)
 - sc_forward_list, [116](#)
 - sc_forward_list2, [116](#)
 - sc_node_add_link_fn, [112](#)
 - sc_node_end_of_stream_fn, [113](#)
 - sc_node_export_state, [116](#)
 - sc_node_fwd_error, [111](#)
 - sc_node_init_fn, [113](#)
 - sc_node_init_get_arg_dbl, [117](#)
 - sc_node_init_get_arg_int, [117](#)
 - sc_node_init_get_arg_int64, [118](#)
 - sc_node_init_get_arg_obj, [118](#)
 - sc_node_init_get_arg_str, [119](#)
 - sc_node_link_end_of_stream, [119](#)
 - sc_node_link_end_of_stream2, [120](#)
 - sc_node_pkts_fn, [113](#)
 - sc_node_prep_check_links, [120](#)
 - sc_node_prep_does_not_forward, [120](#)

- sc_node_prep_fn, 114
- sc_node_prep_get_link, 121
- sc_node_prep_get_link_or_free, 121
- sc_node_prep_get_pool, 121
- sc_node_prep_link_forwards_from_node, 122
- sc_node_select_subnode_fn, 114
- sc_node_set_error, 111
- sc_node_set_errorv, 112
- sc_node_type_alloc, 122
- ext_packet.h, 123
 - SC_MEMBER_OFFSET, 124
 - SC_MEMBER_SIZE, 124
 - sc_packet_bytes, 125
 - sc_packet_fragments_tail, 125
 - sc_packet_prefetch_r, 125
 - sc_packet_prefetch_rw, 126
 - sc_packet_timespec, 126
- ext_packet_list.h, 126
 - sc_packet_list_append, 127
 - sc_packet_list_append_list, 127
 - sc_packet_list_finalise, 128
 - sc_packet_list_init, 128
 - sc_packet_list_is_empty, 128
 - sc_packet_list_pop_head, 128
 - sc_packet_list_push_head, 129
 - sc_packet_list_tail, 129
- flags
 - sc_packet, 75
- frags
 - sc_packet, 75
- frags_n
 - sc_packet, 75
- frags_tail
 - sc_packet, 75
- frame_len
 - sc_packet, 75
- free_link
 - sc_append_to_list, 61
- hash_table.h, 129
 - sc_hash_table_alloc, 130
 - sc_hash_table_clear, 131
 - sc_hash_table_del, 131
 - sc_hash_table_del_val, 131
 - sc_hash_table_free, 132
 - sc_hash_table_get, 132
 - sc_hash_table_get_next_entry, 132
 - sc_hash_table_grow, 133
 - sc_hash_table_key_size, 133
 - sc_hash_table_num_entries, 134
 - sc_hash_table_val_size, 134
 - sc_hash_table_val_to_key, 134
- head
 - sc_packet_list, 77
- io
 - sc_iovec_ptr, 66
- iov
 - sc_iovec_ptr, 66
 - sc_packet, 75
- iovec.h, 135
 - sc_iovec_copy_from_end, 136
 - sc_iovec_copy_from_end_offset, 136
 - sc_iovec_copy_to_end_offset, 136
 - sc_iovec_ptr_bytes, 137
 - sc_iovec_ptr_copy_out, 137
 - sc_iovec_ptr_find_chr, 138
 - sc_iovec_ptr_init, 138
 - sc_iovec_ptr_init_buf, 138
 - sc_iovec_ptr_init_packet, 139
 - sc_iovec_ptr_skip, 139
 - sc_iovec_trim_end, 139
- iovlen
 - sc_iovec_ptr, 66
 - sc_packet, 75
- ip.h, 140
 - SC_IP4_FRAG_DONT, 140
 - SC_IP4_FRAG_MORE, 140
 - SC_IP4_OFFSET_MASK, 140
 - SC_TCP_ACK, 141
 - SC_TCP_FIN, 141
 - SC_TCP_PSH, 141
 - SC_TCP_RST, 141
 - SC_TCP_SYN, 141
 - SC_TCP_URG, 141
- links
 - sc_append_to_list, 62
- mailbox.h, 142
 - sc_mailbox_alloc, 142
 - sc_mailbox_connect, 142
 - sc_mailbox_get_send_node, 143
 - sc_mailbox_poll, 143
 - sc_mailbox_send, 143
 - sc_mailbox_send_list, 144
 - sc_mailbox_set_recv, 144
- metadata
 - sc_packet, 75
- misc.h, 145
 - sc_join_mcast_group, 145
- n_links
 - sc_append_to_list, 62
- name
 - sc_arg, 63
 - sc_node_link, 70
- nd_name
 - sc_node, 67
- nd_private
 - sc_node, 67
- nd_type
 - sc_node, 67

- next
 - sc_dlist, [65](#)
 - sc_packet, [76](#)
- nf_init_fn
 - sc_node_factory, [68](#)
- nf_name
 - sc_node_factory, [68](#)
- nf_node_api_ver
 - sc_node_factory, [68](#)
- nf_private
 - sc_node_factory, [69](#)
- nf_reserved
 - sc_node_factory, [69](#)
- nf_source_file
 - sc_node_factory, [69](#)
- node.h, [145](#)
 - sc_node_add_info_int, [146](#)
 - sc_node_add_info_str, [146](#)
 - sc_node_add_link, [147](#)
 - sc_node_alloc, [147](#)
 - sc_node_alloc_from_str, [149](#)
 - sc_node_alloc_named, [149](#)
 - sc_node_factory_lookup, [151](#)
 - sc_node_from_object, [152](#)
 - sc_node_get_thread, [152](#)
 - sc_node_to_object, [153](#)
- nt_add_link_fn
 - sc_node_type, [70](#)
- nt_end_of_stream_fn
 - sc_node_type, [70](#)
- nt_name
 - sc_node_type, [71](#)
- nt_pkts_fn
 - sc_node_type, [71](#)
- nt_prep_fn
 - sc_node_type, [71](#)
- nt_private
 - sc_node_type, [71](#)
- nt_select_subnode_fn
 - sc_node_type, [71](#)
- num_frags
 - sc_packet_list, [77](#)
- num_pkts
 - sc_packet_list, [77](#)
- object.h, [153](#)
 - sc_object_type, [154](#)
 - sc_opaque_alloc, [154](#)
 - sc_opaque_free, [155](#)
 - sc_opaque_get_ptr, [155](#)
 - sc_opaque_set_ptr, [155](#)
- object.h
 - SC_OBJ_C_ATTR, [154](#)
 - SC_OBJ_NODE, [154](#)
 - SC_OBJ_OPAQUE, [154](#)
 - SC_OBJ_PKT_PREDICATE, [154](#)
 - SC_OBJ_POOL, [154](#)
- packed_stream.h, [156](#)
 - ps_cap_len, [159](#)
 - ps_flags, [159](#)
 - ps_next_offset, [159](#)
 - ps_orig_len, [159](#)
 - ps_pkt_start_offset, [159](#)
 - ps_ts_nsec, [159](#)
 - ps_ts_sec, [160](#)
 - SC_PS_FLAG_BAD_FCS, [157](#)
 - SC_PS_FLAG_BAD_L3_CSUM, [157](#)
 - SC_PS_FLAG_BAD_L4_CSUM, [157](#)
 - SC_PS_FLAG_CLOCK_IN_SYNC, [157](#)
 - SC_PS_FLAG_CLOCK_SET, [157](#)
 - sc_packed_packet_next, [157](#)
 - sc_packed_packet_payload, [158](#)
 - sc_packet_packed_end, [158](#)
 - sc_packet_packed_first, [158](#)
- pkt_pool.h, [160](#)
 - sc_packet_append_iovec_ptr, [161](#)
 - sc_pool_duplicate_packed_packet, [161](#)
 - sc_pool_duplicate_packet, [161](#)
 - sc_pool_from_object, [162](#)
 - sc_pool_get_buffer_size, [162](#)
 - sc_pool_get_packets, [162](#)
 - sc_pool_on_threshold, [163](#)
 - sc_pool_return_packets, [163](#)
 - sc_pool_set_refill_node, [163](#)
 - sc_pool_to_object, [164](#)
 - sc_pool_wraps_node, [164](#)
- pred_private
 - sc_pkt_predicate, [78](#)
- pred_test_fn
 - sc_pkt_predicate, [78](#)
- predicate.h, [165](#)
 - sc_pkt_predicate_alloc, [166](#)
 - sc_pkt_predicate_from_object, [166](#)
 - sc_pkt_predicate_to_object, [166](#)
- prev
 - sc_dlist, [65](#)
- ps_cap_len
 - packed_stream.h, [159](#)
 - sc_packed_packet, [73](#)
- ps_flags
 - packed_stream.h, [159](#)
 - sc_packed_packet, [73](#)
- ps_next_offset
 - packed_stream.h, [159](#)
 - sc_packed_packet, [73](#)
- ps_orig_len
 - packed_stream.h, [159](#)
 - sc_packed_packet, [73](#)
- ps_pkt_start_offset
 - packed_stream.h, [159](#)
 - sc_packed_packet, [73](#)

ps_ts_nsec
 packed_stream.h, 159
 sc_packed_packet, 73
 ps_ts_sec
 packed_stream.h, 160
 sc_packed_packet, 73

 reserved1
 sc_packet, 76
 reserved2
 sc_packet, 76

 SC_8021Q_VID_MASK
 ethernet.h, 104
 SC_ARG_DBL
 args.h, 88
 SC_ARG_INT
 args.h, 88
 SC_ARG_OBJ
 args.h, 89
 SC_ARG_STR
 args.h, 89
 SC_CONTAINER
 dlist.h, 99
 SC_DLIST_FOR_EACH_OBJ_SAFE
 dlist.h, 100
 SC_DLIST_FOR_EACH_OBJ
 dlist.h, 99
 SC_ETHERTYPE_8021QinQ
 ethernet.h, 104
 SC_ETHERTYPE_8021Q
 ethernet.h, 104
 SC_IP4_FRAG_DONT
 ip.h, 140
 SC_IP4_FRAG_MORE
 ip.h, 140
 SC_IP4_OFFSET_MASK
 ip.h, 140
 SC_MEMBER_OFFSET
 ext_packet.h, 124
 SC_MEMBER_SIZE
 ext_packet.h, 124
 SC_PS_FLAG_BAD_FCS
 packed_stream.h, 157
 SC_PS_FLAG_BAD_L3_CSUM
 packed_stream.h, 157
 SC_PS_FLAG_BAD_L4_CSUM
 packed_stream.h, 157
 SC_PS_FLAG_CLOCK_IN_SYNC
 packed_stream.h, 157
 SC_PS_FLAG_CLOCK_SET
 packed_stream.h, 157
 SC_TCP_ACK
 ip.h, 141
 SC_TCP_FIN
 ip.h, 141
 SC_TCP_PSH
 ip.h, 141
 SC_TCP_RST
 ip.h, 141
 SC_TCP_SYN
 ip.h, 141
 SC_TCP_URG
 ip.h, 141
 ST_CONSTANT
 declare_types.h, 96
 ST_FIELD_STR
 declare_types.h, 97
 ST_FIELD
 declare_types.h, 96
 ST_STRUCT
 declare_types.h, 97
 sc_append_to_list, 61
 append_to, 61
 free_link, 61
 links, 62
 n_links, 62
 sc_arg, 62
 name, 63
 type, 63
 val, 63
 sc_attr, 63
 sc_attr_alloc
 attr.h, 91
 sc_attr_doc
 attr.h, 91
 sc_attr_dup
 attr.h, 91
 sc_attr_free
 attr.h, 92
 sc_attr_from_object
 attr.h, 92
 sc_attr_reset
 attr.h, 92
 sc_attr_set_from_fmt
 attr.h, 93
 sc_attr_set_from_str
 attr.h, 93
 sc_attr_set_int
 attr.h, 93
 sc_attr_set_str
 attr.h, 94
 sc_attr_to_object
 attr.h, 94
 sc_callback, 64
 cb_handler_fn, 64
 cb_link, 64
 cb_private, 64
 sc_callback_alloc
 event.h, 106
 sc_callback_alloc2
 event.h, 106
 sc_callback_free

- event.h, [107](#)
- sc_callback_handler_fn
 - event.h, [105](#)
- sc_callback_is_active
 - event.h, [107](#)
- sc_callback_on_idle
 - event.h, [107](#)
- sc_callback_remove
 - event.h, [108](#)
- sc_callback_set_description
 - event.h, [108](#)
- sc_dlist, [65](#)
 - next, [65](#)
 - prev, [65](#)
- sc_dlist_init
 - dlist.h, [101](#)
- sc_dlist_pop_head
 - dlist.h, [102](#)
- sc_dlist_pop_tail
 - dlist.h, [102](#)
- sc_dlist_push_head
 - dlist.h, [102](#)
- sc_dlist_push_tail
 - dlist.h, [103](#)
- sc_dlist_rehome
 - dlist.h, [103](#)
- sc_dlist_remove
 - dlist.h, [103](#)
- sc_epoll_ctl
 - event.h, [108](#)
- sc_forward
 - ext_node.h, [115](#)
- sc_forward2
 - ext_node.h, [115](#)
- sc_forward_list
 - ext_node.h, [116](#)
- sc_forward_list2
 - ext_node.h, [116](#)
- sc_hash_table, [65](#)
- sc_hash_table_alloc
 - hash_table.h, [130](#)
- sc_hash_table_clear
 - hash_table.h, [131](#)
- sc_hash_table_del
 - hash_table.h, [131](#)
- sc_hash_table_del_val
 - hash_table.h, [131](#)
- sc_hash_table_free
 - hash_table.h, [132](#)
- sc_hash_table_get
 - hash_table.h, [132](#)
- sc_hash_table_get_next_entry
 - hash_table.h, [132](#)
- sc_hash_table_grow
 - hash_table.h, [133](#)
- sc_hash_table_key_size
 - hash_table.h, [133](#)
- sc_hash_table_num_entries
 - hash_table.h, [134](#)
- sc_hash_table_val_size
 - hash_table.h, [134](#)
- sc_hash_table_val_to_key
 - hash_table.h, [134](#)
- sc_iovec_copy_from_end
 - iovec.h, [136](#)
- sc_iovec_copy_from_end_offset
 - iovec.h, [136](#)
- sc_iovec_copy_to_end_offset
 - iovec.h, [136](#)
- sc_iovec_ptr, [66](#)
 - io, [66](#)
 - iov, [66](#)
 - iovlen, [66](#)
- sc_iovec_ptr_bytes
 - iovec.h, [137](#)
- sc_iovec_ptr_copy_out
 - iovec.h, [137](#)
- sc_iovec_ptr_find_chr
 - iovec.h, [138](#)
- sc_iovec_ptr_init
 - iovec.h, [138](#)
- sc_iovec_ptr_init_buf
 - iovec.h, [138](#)
- sc_iovec_ptr_init_packet
 - iovec.h, [139](#)
- sc_iovec_ptr_skip
 - iovec.h, [139](#)
- sc_iovec_trim_end
 - iovec.h, [139](#)
- sc_join_mcast_group
 - misc.h, [145](#)
- sc_mailbox_alloc
 - mailbox.h, [142](#)
- sc_mailbox_connect
 - mailbox.h, [142](#)
- sc_mailbox_get_send_node
 - mailbox.h, [143](#)
- sc_mailbox_poll
 - mailbox.h, [143](#)
- sc_mailbox_send
 - mailbox.h, [143](#)
- sc_mailbox_send_list
 - mailbox.h, [144](#)
- sc_mailbox_set_rcv
 - mailbox.h, [144](#)
- sc_node, [67](#)
 - nd_name, [67](#)
 - nd_private, [67](#)
 - nd_type, [67](#)
- sc_node_add_info_int
 - node.h, [146](#)
- sc_node_add_info_str

[node.h](#), [146](#)
[sc_node_add_link](#)
 [node.h](#), [147](#)
[sc_node_add_link_fn](#)
 [ext_node.h](#), [112](#)
[sc_node_alloc](#)
 [node.h](#), [147](#)
[sc_node_alloc_from_str](#)
 [node.h](#), [149](#)
[sc_node_alloc_named](#)
 [node.h](#), [149](#)
[sc_node_end_of_stream_fn](#)
 [ext_node.h](#), [113](#)
[sc_node_export_state](#)
 [ext_node.h](#), [116](#)
[sc_node_factory](#), [68](#)
 [nf_init_fn](#), [68](#)
 [nf_name](#), [68](#)
 [nf_node_api_ver](#), [68](#)
 [nf_private](#), [69](#)
 [nf_reserved](#), [69](#)
 [nf_source_file](#), [69](#)
[sc_node_factory_lookup](#)
 [node.h](#), [151](#)
[sc_node_from_object](#)
 [node.h](#), [152](#)
[sc_node_fwd_error](#)
 [ext_node.h](#), [111](#)
[sc_node_get_thread](#)
 [node.h](#), [152](#)
[sc_node_init_fn](#)
 [ext_node.h](#), [113](#)
[sc_node_init_get_arg_dbl](#)
 [ext_node.h](#), [117](#)
[sc_node_init_get_arg_int](#)
 [ext_node.h](#), [117](#)
[sc_node_init_get_arg_int64](#)
 [ext_node.h](#), [118](#)
[sc_node_init_get_arg_obj](#)
 [ext_node.h](#), [118](#)
[sc_node_init_get_arg_str](#)
 [ext_node.h](#), [119](#)
[sc_node_link](#), [69](#)
 [name](#), [70](#)
[sc_node_link_end_of_stream](#)
 [ext_node.h](#), [119](#)
[sc_node_link_end_of_stream2](#)
 [ext_node.h](#), [120](#)
[sc_node_pkts_fn](#)
 [ext_node.h](#), [113](#)
[sc_node_prep_check_links](#)
 [ext_node.h](#), [120](#)
[sc_node_prep_does_not_forward](#)
 [ext_node.h](#), [120](#)
[sc_node_prep_fn](#)
 [ext_node.h](#), [114](#)
[sc_node_prep_get_link](#)
 [ext_node.h](#), [121](#)
[sc_node_prep_get_link_or_free](#)
 [ext_node.h](#), [121](#)
[sc_node_prep_get_pool](#)
 [ext_node.h](#), [121](#)
[sc_node_prep_link_forwards_from_node](#)
 [ext_node.h](#), [122](#)
[sc_node_select_subnode_fn](#)
 [ext_node.h](#), [114](#)
[sc_node_set_error](#)
 [ext_node.h](#), [111](#)
[sc_node_set_errorv](#)
 [ext_node.h](#), [112](#)
[sc_node_to_object](#)
 [node.h](#), [153](#)
[sc_node_type](#), [70](#)
 [nt_add_link_fn](#), [70](#)
 [nt_end_of_stream_fn](#), [70](#)
 [nt_name](#), [71](#)
 [nt_pkts_fn](#), [71](#)
 [nt_prep_fn](#), [71](#)
 [nt_private](#), [71](#)
 [nt_select_subnode_fn](#), [71](#)
[sc_node_type_alloc](#)
 [ext_node.h](#), [122](#)
[sc_ns_from_ms](#)
 [time.h](#), [184](#)
[sc_ns_from_ts](#)
 [time.h](#), [184](#)
[sc_ns_from_tv](#)
 [time.h](#), [185](#)
[sc_ns_from_us](#)
 [time.h](#), [185](#)
[sc_object](#), [71](#)
[sc_object_type](#)
 [object.h](#), [154](#)
[sc_opaque_alloc](#)
 [object.h](#), [154](#)
[sc_opaque_free](#)
 [object.h](#), [155](#)
[sc_opaque_get_ptr](#)
 [object.h](#), [155](#)
[sc_opaque_set_ptr](#)
 [object.h](#), [155](#)
[sc_packed_packet](#), [72](#)
 [ps_cap_len](#), [73](#)
 [ps_flags](#), [73](#)
 [ps_next_offset](#), [73](#)
 [ps_orig_len](#), [73](#)
 [ps_pkt_start_offset](#), [73](#)
 [ps_ts_nsec](#), [73](#)
 [ps_ts_sec](#), [73](#)
[sc_packed_packet_next](#)
 [packed_stream.h](#), [157](#)
[sc_packed_packet_payload](#)

- packed_stream.h, 158
- sc_packet, 74
 - flags, 75
 - frags, 75
 - frags_n, 75
 - frags_tail, 75
 - frame_len, 75
 - iov, 75
 - iovlen, 75
 - metadata, 75
 - next, 76
 - reserved1, 76
 - reserved2, 76
 - ts_nsec, 76
 - ts_sec, 76
- sc_packet_append_iovec_ptr
 - pkt_pool.h, 161
- sc_packet_bytes
 - ext_packet.h, 125
- sc_packet_fragments_tail
 - ext_packet.h, 125
- sc_packet_list, 76
 - head, 77
 - num_fragments, 77
 - num_pkts, 77
 - tail, 77
- sc_packet_list_append
 - ext_packet_list.h, 127
- sc_packet_list_append_list
 - ext_packet_list.h, 127
- sc_packet_list_finalise
 - ext_packet_list.h, 128
- sc_packet_list_init
 - ext_packet_list.h, 128
- sc_packet_list_is_empty
 - ext_packet_list.h, 128
- sc_packet_list_pop_head
 - ext_packet_list.h, 128
- sc_packet_list_push_head
 - ext_packet_list.h, 129
- sc_packet_list_tail
 - ext_packet_list.h, 129
- sc_packet_packed_end
 - packed_stream.h, 158
- sc_packet_packed_first
 - packed_stream.h, 158
- sc_packet_prefetch_r
 - ext_packet.h, 125
- sc_packet_prefetch_rw
 - ext_packet.h, 126
- sc_packet_timespec
 - ext_packet.h, 126
- sc_param_type
 - args.h, 88
- sc_pkt_predicate, 78
 - pred_private, 78
 - pred_test_fn, 78
- sc_pkt_predicate_alloc
 - predicate.h, 166
- sc_pkt_predicate_from_object
 - predicate.h, 166
- sc_pkt_predicate_to_object
 - predicate.h, 166
- sc_pool_duplicate_packed_packet
 - pkt_pool.h, 161
- sc_pool_duplicate_packet
 - pkt_pool.h, 161
- sc_pool_from_object
 - pkt_pool.h, 162
- sc_pool_get_buffer_size
 - pkt_pool.h, 162
- sc_pool_get_packets
 - pkt_pool.h, 162
- sc_pool_on_threshold
 - pkt_pool.h, 163
- sc_pool_return_packets
 - pkt_pool.h, 163
- sc_pool_set_refill_node
 - pkt_pool.h, 163
- sc_pool_to_object
 - pkt_pool.h, 164
- sc_pool_wraps_node
 - pkt_pool.h, 164
- sc_session_alloc
 - session.h, 168
- sc_session_destroy
 - session.h, 168
- sc_session_error, 78
 - err_errno, 79
 - err_file, 79
 - err_func, 79
 - err_line, 79
 - err_msg, 79
- sc_session_error_free
 - session.h, 168
- sc_session_error_get
 - session.h, 169
- sc_session_go
 - session.h, 169
- sc_session_pause
 - session.h, 169
- sc_session_prepare
 - session.h, 170
- sc_session_run
 - session.h, 170
- sc_session_stop
 - session.h, 170
- sc_sh_handle_backlog_fn
 - sc_subnode_helper, 81
- sc_sh_handle_end_of_stream_fn
 - sc_subnode_helper, 82
- sc_stream, 80

- sc_stream_all
 - stream.h, [172](#)
- sc_stream_alloc
 - stream.h, [172](#)
- sc_stream_eth_dhost
 - stream.h, [173](#)
- sc_stream_eth_shost
 - stream.h, [173](#)
- sc_stream_eth_type
 - stream.h, [173](#)
- sc_stream_eth_vlan_id
 - stream.h, [174](#)
- sc_stream_free
 - stream.h, [174](#)
- sc_stream_ip_dest_host
 - stream.h, [174](#)
- sc_stream_ip_dest_hostport
 - stream.h, [175](#)
- sc_stream_ip_dest_port
 - stream.h, [175](#)
- sc_stream_ip_protocol
 - stream.h, [175](#)
- sc_stream_ip_source_host
 - stream.h, [176](#)
- sc_stream_ip_source_hostport
 - stream.h, [176](#)
- sc_stream_ip_source_port
 - stream.h, [177](#)
- sc_stream_mismatch
 - stream.h, [177](#)
- sc_stream_reset
 - stream.h, [177](#)
- sc_stream_set_str
 - stream.h, [178](#)
- sc_subnode_helper, [80](#)
 - sc_sh_handle_backlog_fn, [81](#)
 - sc_sh_handle_end_of_stream_fn, [82](#)
 - sc_subnode_helper_from_node, [82](#)
 - sc_subnode_helper_request_callback, [83](#)
 - sh_backlog, [83](#)
 - sh_free_link, [83](#)
 - sh_handle_backlog_fn, [83](#)
 - sh_handle_end_of_stream_fn, [83](#)
 - sh_links, [84](#)
 - sh_n_links, [84](#)
 - sh_node, [84](#)
 - sh_poll_backlog_ns, [84](#)
 - sh_pool, [84](#)
 - sh_pool_threshold, [84](#)
 - sh_private, [84](#)
- sc_subnode_helper_from_node
 - sc_subnode_helper, [82](#)
- sc_subnode_helper_request_callback
 - sc_subnode_helper, [83](#)
- sc_thread_alloc
 - thread.h, [180](#)
- sc_thread_calloc
 - thread.h, [181](#)
- sc_thread_calloc_aligned
 - thread.h, [181](#)
- sc_thread_get_time
 - thread.h, [181](#)
- sc_thread_mfree
 - thread.h, [182](#)
- sc_thread_poll
 - thread.h, [182](#)
- sc_thread_poll_timers
 - thread.h, [182](#)
- sc_thread_waitable_fd_get
 - thread.h, [183](#)
- sc_thread_waitable_fd_prime
 - thread.h, [183](#)
- sc_timer_expire_after_ns
 - time.h, [185](#)
- sc_timer_expire_at
 - time.h, [186](#)
- sc_timer_get_expiry_time
 - time.h, [186](#)
- sc_timer_push_back_ns
 - time.h, [186](#)
- sc_vi, [85](#)
- sc_vi_add_stream
 - vi.h, [188](#)
- sc_vi_alloc
 - vi.h, [188](#)
- sc_vi_alloc_from_group
 - vi.h, [188](#)
- sc_vi_get_interface_name
 - vi.h, [189](#)
- sc_vi_get_thread
 - vi.h, [189](#)
- sc_vi_group_add_stream
 - vi.h, [189](#)
- sc_vi_group_alloc
 - vi.h, [190](#)
- sc_vi_group_get_session
 - vi.h, [190](#)
- sc_vi_set_rcv_node
 - vi.h, [191](#)
- SC_OBJ_C_ATTR
 - object.h, [154](#)
- SC_OBJ_NODE
 - object.h, [154](#)
- SC_OBJ_OPAQUE
 - object.h, [154](#)
- SC_OBJ_PKT_PREDICATE
 - object.h, [154](#)
- SC_OBJ_POOL
 - object.h, [154](#)
- SC_PARAM_DBL
 - args.h, [88](#)
- SC_PARAM_INT

- args.h, 88
- SC_PARAM_OBJ
 - args.h, 88
- SC_PARAM_STR
 - args.h, 88
- session.h, 167
 - sc_session_alloc, 168
 - sc_session_destroy, 168
 - sc_session_error_free, 168
 - sc_session_error_get, 169
 - sc_session_go, 169
 - sc_session_pause, 169
 - sc_session_prepare, 170
 - sc_session_run, 170
 - sc_session_stop, 170
- sh_backlog
 - sc_subnode_helper, 83
- sh_free_link
 - sc_subnode_helper, 83
- sh_handle_backlog_fn
 - sc_subnode_helper, 83
- sh_handle_end_of_stream_fn
 - sc_subnode_helper, 83
- sh_links
 - sc_subnode_helper, 84
- sh_n_links
 - sc_subnode_helper, 84
- sh_node
 - sc_subnode_helper, 84
- sh_poll_backlog_ns
 - sc_subnode_helper, 84
- sh_pool
 - sc_subnode_helper, 84
- sh_pool_threshold
 - sc_subnode_helper, 84
- sh_private
 - sc_subnode_helper, 84
- stream.h, 171
 - sc_stream_all, 172
 - sc_stream_alloc, 172
 - sc_stream_eth_dhost, 173
 - sc_stream_eth_shost, 173
 - sc_stream_eth_type, 173
 - sc_stream_eth_vlan_id, 174
 - sc_stream_free, 174
 - sc_stream_ip_dest_host, 174
 - sc_stream_ip_dest_hostport, 175
 - sc_stream_ip_dest_port, 175
 - sc_stream_ip_protocol, 175
 - sc_stream_ip_source_host, 176
 - sc_stream_ip_source_hostport, 176
 - sc_stream_ip_source_port, 177
 - sc_stream_mismatch, 177
 - sc_stream_reset, 177
 - sc_stream_set_str, 178
- subnode_helper.h, 179
- tail
 - sc_packet_list, 77
- thread.h, 180
 - sc_thread_alloc, 180
 - sc_thread_calloc, 181
 - sc_thread_calloc_aligned, 181
 - sc_thread_get_time, 181
 - sc_thread_mfree, 182
 - sc_thread_poll, 182
 - sc_thread_poll_timers, 182
 - sc_thread_waitable_fd_get, 183
 - sc_thread_waitable_fd_prime, 183
- time.h, 184
 - sc_ns_from_ms, 184
 - sc_ns_from_ts, 184
 - sc_ns_from_tv, 185
 - sc_ns_from_us, 185
 - sc_timer_expire_after_ns, 185
 - sc_timer_expire_at, 186
 - sc_timer_get_expiry_time, 186
 - sc_timer_push_back_ns, 186
- ts_nsec
 - sc_packet, 76
- ts_sec
 - sc_packet, 76
- type
 - sc_arg, 63
- val
 - sc_arg, 63
- vi.h, 187
 - sc_vi_add_stream, 188
 - sc_vi_alloc, 188
 - sc_vi_alloc_from_group, 188
 - sc_vi_get_interface_name, 189
 - sc_vi_get_thread, 189
 - sc_vi_group_add_stream, 189
 - sc_vi_group_alloc, 190
 - sc_vi_group_get_session, 190
 - sc_vi_set_rcv_node, 191