
# SolarCapture C Bindings User Guide

SF-115721-CD , Issue 2

2016/05/17 14:55:14

Solarflare Communications Inc

## SolarCapture C Bindings User Guide



SF-115721-CD

Last Revised: 52016

Issue 2

# Contents

**1 Introduction** **1**

**2 Embedding SolarCapture** **3**

2.1 Sessions — struct sc_session . . . 3
2.2 Attributes — struct sc_attr . . . 3
2.3 Threads — struct sc_thread . . . 4
2.4 Virtual Interfaces — struct sc_vi . . . 4
2.5 Nodes — struct sc_node . . . 4
2.6 Mailboxes — struct sc_mailbox . . . 5
2.7 Built-In Nodes . . . 6

**3 Extending SolarCapture** **7**

3.1 Node factories — struct sc_node_factory . . . 7
3.2 Node types — struct sc_node_type . . . 8
3.3 Node libraries . . . 8
3.4 Insert a user-defined node between capture and sc_writer . . . 9

**4 Examples** **11**

4.1 c_api . . . 11
4.2 c_api_export . . . 11
4.3 extensions_api . . . 12
4.4 forwarding . . . 12
4.4.1 trivial_bridge.py . . . 12
4.4.2 bpf_firewall.py . . . 12
4.4.3 reflect.py . . . 12
4.5 py_api . . . 13
4.6 unmanaged . . . 13
4.6.1 unmanaged_mailbox.c . . . 13
4.6.2 unmanaged_thread.c . . . 13

  

**5 Nodes** **15**

5.1 sc_append_to_list Node Reference . . . 16
5.2 sc_arista_ts Node Reference . . . 17
5.3 sc_batch_limiter Node Reference . . . 20
5.4 sc_cpacket_encap Node Reference . . . 21
5.5 sc_cpacket_ts Node Reference . . . 21
5.6 sc_delay_line Node Reference . . . 22
5.7 sc_exit Node Reference . . . 23
5.8 sc_fd_reader Node Reference . . . 24
5.9 sc_fd_writer Node Reference . . . 25
5.10 sc_filter Node Reference . . . 26
5.11 sc_flow_balancer Node Reference . . . 27
5.12 sc_injector Node Reference . . . 28
5.13 sc_line_reader Node Reference . . . 28
5.14 sc_merge_sorter Node Reference . . . 30
5.15 sc_no_op Node Reference . . . 30
5.16 sc_pacer Node Reference . . . 30
5.17 sc_pass_n Node Reference . . . 31
5.18 sc_pcap_packer Node Reference . . . 31
5.19 sc_pool_forwarder Node Reference . . . 33
5.20 sc_ps_packer Node Reference . . . 34
5.21 sc_ps_unpacker Node Reference . . . 34
5.22 sc_range_filter Node Reference . . . 36
5.23 sc_rate_monitor Node Reference . . . 37
5.24 sc_reader Node Reference . . . 37
5.25 sc_repeater Node Reference . . . 38
5.26 sc_rr_gather Node Reference . . . 39
5.27 sc_rr_spreader Node Reference . . . 39
5.28 sc_rt_pacer Node Reference . . . 40

5.29 sc_shm_broadcast Node Reference . . . 41
5.30 sc_shm_export Node Reference . . . 43
5.31 sc_shm_import Node Reference . . . 44
5.32 sc_sim_work Node Reference . . . 45
5.33 sc_snap Node Reference . . . 46
5.34 sc_subnode_helper Node Reference . . . 46
5.35 sc_tap Node Reference . . . 47
5.36 sc_timestamp_filter Node Reference . . . 49
5.37 sc_token_bucket_shaper Node Reference . . . 50
5.38 sc_tracer Node Reference . . . 50
5.39 sc_ts_adjust Node Reference . . . 52
5.40 sc_tunnel Node Reference . . . 53
5.41 sc_tuntap Node Reference . . . 55
5.42 sc_vi_node Node Reference . . . 55
5.43 sc_vss Node Reference . . . 56
5.44 sc_writer Node Reference . . . 56

**6 Statistics** **59**

6.1 sc_arista_ts Statistics Reference . . . 59
6.2 sc_batch_limiter Statistics Reference . . . 61
6.3 sc_filter Statistics Reference . . . 61
6.4 sc_flow_balancer Statistics Reference . . . 61
6.5 sc_pcap_packer Statistics Reference . . . 62
6.6 sc_rate_monitor Statistics Reference . . . 62
6.7 sc_shm Statistics Reference . . . 62
6.8 sc_subnode_helper Statistics Reference . . . 63
6.9 sc_writer Statistics Reference . . . 63

**7 Data Structure Index** **65**

7.1 Data Structures . . . 65

  

**8 File Index** **67**

8.1 File List . . . 67

**9 Data Structure Documentation** **69**

9.1 sc_append_to_list Struct Reference . . . 69

9.1.1 Detailed Description . . . 69

9.1.2 Field Documentation . . . 69

9.1.2.1 append_to . . . 69

9.1.2.2 free_link . . . 69

9.1.2.3 links . . . 70

9.1.2.4 n_links . . . 70

9.2 sc_arg Struct Reference . . . 70

9.2.1 Detailed Description . . . 70

9.2.2 Field Documentation . . . 70

9.2.2.1 name . . . 70

9.2.2.2 type . . . 70

9.2.2.3 val . . . 71

9.3 sc_attr Struct Reference . . . 71

9.3.1 Detailed Description . . . 71

9.4 sc_callback Struct Reference . . . 71

9.4.1 Detailed Description . . . 72

9.4.2 Field Documentation . . . 72

9.4.2.1 cb_handler_fn . . . 72

9.4.2.2 cb_link . . . 72

9.4.2.3 cb_private . . . 72

9.5 sc_dlist Struct Reference . . . 72

9.5.1 Detailed Description . . . 73

9.5.2 Field Documentation . . . 73

9.5.2.1 next . . . 73

9.5.2.2 prev . . . 73

9.6 sc_hash_table Struct Reference . . . 73
9.6.1 Detailed Description . . . 73
9.7 sc_iovec_ptr Struct Reference . . . 73
9.7.1 Detailed Description . . . 74
9.7.2 Field Documentation . . . 74
9.7.2.1 io . . . 74
9.7.2.2 iov . . . 74
9.7.2.3 iovlen . . . 74
9.8 sc_node Struct Reference . . . 74
9.8.1 Detailed Description . . . 75
9.8.2 Field Documentation . . . 75
9.8.2.1 nd_name . . . 75
9.8.2.2 nd_private . . . 75
9.8.2.3 nd_type . . . 75
9.9 sc_node_factory Struct Reference . . . 75
9.9.1 Detailed Description . . . 75
9.9.2 Field Documentation . . . 76
9.9.2.1 nf_init_fn . . . 76
9.9.2.2 nf_name . . . 76
9.9.2.3 nf_node_api_ver . . . 76
9.9.2.4 nf_private . . . 76
9.9.2.5 nf_reserved . . . 76
9.9.2.6 nf_source_file . . . 76
9.10 sc_node_link Struct Reference . . . 76
9.10.1 Detailed Description . . . 77
9.10.2 Field Documentation . . . 77
9.10.2.1 name . . . 77
9.11 sc_node_type Struct Reference . . . 77
9.11.1 Detailed Description . . . 77

  

9.11.2 Field Documentation . . . 77
9.11.2.1 nt_add_link_fn . . . 77
9.11.2.2 nt_end_of_stream_fn . . . 78
9.11.2.3 nt_name . . . 78
9.11.2.4 nt_pkts_fn . . . 78
9.11.2.5 nt_prep_fn . . . 78
9.11.2.6 nt_private . . . 78
9.11.2.7 nt_select_subnode_fn . . . 78
9.12 sc_object Struct Reference . . . 78
9.12.1 Detailed Description . . . 78
9.13 sc_packed_packet Struct Reference . . . 79
9.13.1 Detailed Description . . . 79
9.13.2 Field Documentation . . . 79
9.13.2.1 ps_cap_len . . . 79
9.13.2.2 ps_flags . . . 79
9.13.2.3 ps_next_offset . . . 80
9.13.2.4 ps_orig_len . . . 80
9.13.2.5 ps_pkt_start_offset . . . 80
9.13.2.6 ps_ts_nsec . . . 80
9.13.2.7 ps_ts_sec . . . 80
9.14 sc_packet Struct Reference . . . 80
9.14.1 Detailed Description . . . 81
9.14.2 Field Documentation . . . 81
9.14.2.1 flags . . . 81
9.14.2.2 frags . . . 81
9.14.2.3 frags_n . . . 81
9.14.2.4 frags_tail . . . 81
9.14.2.5 frame_len . . . 81
9.14.2.6 iov . . . 81

9.14.2.7 iovlen . . . 81
9.14.2.8 metadata . . . 82
9.14.2.9 next . . . 82
9.14.2.10 reserved1 . . . 82
9.14.2.11 reserved2 . . . 82
9.14.2.12 ts_nsec . . . 82
9.14.2.13 ts_sec . . . 82
9.15 sc_packet_list Struct Reference . . . 82
9.15.1 Detailed Description . . . 83
9.15.2 Field Documentation . . . 83
9.15.2.1 head . . . 83
9.15.2.2 num_frags . . . 83
9.15.2.3 num_pkts . . . 83
9.15.2.4 tail . . . 83
9.16 sc_pkt_predicate Struct Reference . . . 83
9.16.1 Detailed Description . . . 83
9.16.2 Field Documentation . . . 84
9.16.2.1 pred_private . . . 84
9.16.2.2 pred_test_fn . . . 84
9.17 sc_session_error Struct Reference . . . 84
9.17.1 Detailed Description . . . 84
9.17.2 Field Documentation . . . 84
9.17.2.1 err_errno . . . 84
9.17.2.2 err_file . . . 84
9.17.2.3 err_func . . . 85
9.17.2.4 err_line . . . 85
9.17.2.5 err_msg . . . 85
9.18 sc_stream Struct Reference . . . 85
9.18.1 Detailed Description . . . 85

  

9.19 sc_subnode_helper Struct Reference . . . 86
9.19.1 Detailed Description . . . 86
9.19.2 Friends And Related Function Documentation . . . 86
9.19.2.1 sc_sh_handle_backlog_fn . . . 86
9.19.2.2 sc_sh_handle_end_of_stream_fn . . . 87
9.19.2.3 sc_subnode_helper_from_node . . . 87
9.19.2.4 sc_subnode_helper_request_callback . . . 87
9.19.3 Field Documentation . . . 88
9.19.3.1 sh_backlog . . . 88
9.19.3.2 sh_free_link . . . 88
9.19.3.3 sh_handle_backlog_fn . . . 88
9.19.3.4 sh_handle_end_of_stream_fn . . . 88
9.19.3.5 sh_links . . . 88
9.19.3.6 sh_n_links . . . 88
9.19.3.7 sh_node . . . 88
9.19.3.8 sh_poll_backlog_ns . . . 88
9.19.3.9 sh_pool . . . 88
9.19.3.10 sh_pool_threshold . . . 89
9.19.3.11 sh_private . . . 89
9.20 sc_vi Struct Reference . . . 89
9.20.1 Detailed Description . . . 89

**10 File Documentation** **91**

10.1 append_to_list.h File Reference . . . 91

10.1.1 Detailed Description . . . 91

10.2 args.h File Reference . . . 91

10.2.1 Detailed Description . . . 92

10.2.2 Enumeration Type Documentation . . . 92

10.2.2.1 sc_param_type . . . 92

10.2.3 Function Documentation . . . 92

10.2.3.1 SC_ARG_DBL . . . 92

10.2.3.2 SC_ARG_INT . . . 92

10.2.3.3 SC_ARG_OBJ . . . 93

10.2.3.4 SC_ARG_STR . . . 93

10.3 attr.h File Reference . . . 93

10.3.1 Detailed Description . . . 94

10.3.2 Function Documentation . . . 94

10.3.2.1 sc_attr_alloc . . . 94

10.3.2.2 sc_attr_doc . . . 94

10.3.2.3 sc_attr_dup . . . 95

10.3.2.4 sc_attr_free . . . 95

10.3.2.5 sc_attr_from_object . . . 95

10.3.2.6 sc_attr_reset . . . 95

10.3.2.7 sc_attr_set_from_fmt . . . 96

10.3.2.8 sc_attr_set_from_str . . . 96

10.3.2.9 sc_attr_set_int . . . 96

10.3.2.10 sc_attr_set_str . . . 97

10.3.2.11 sc_attr_to_object . . . 97

10.4 declare_types.h File Reference . . . 97

10.4.1 Detailed Description . . . 98

10.4.2 Macro Definition Documentation . . . 99

  

10.4.2.1 ST_CONSTANT . . . 99
10.4.2.2 ST_FIELD . . . 99
10.4.2.3 ST_FIELD_STR . . . 99
10.4.2.4 ST_STRUCT . . . 99
10.5 dlist.h File Reference . . . 100
10.5.1 Detailed Description . . . 101
10.5.2 Macro Definition Documentation . . . 101
10.5.2.1 SC_CONTAINER . . . 101
10.5.2.2 SC_DLIST_FOR_EACH_OBJ . . . 102
10.5.2.3 SC_DLIST_FOR_EACH_OBJ_SAFE . . . 103
10.5.3 Function Documentation . . . 103
10.5.3.1 sc_dlist_init . . . 103
10.5.3.2 sc_dlist_pop_head . . . 103
10.5.3.3 sc_dlist_pop_tail . . . 103
10.5.3.4 sc_dlist_push_head . . . 104
10.5.3.5 sc_dlist_push_tail . . . 104
10.5.3.6 sc_dlist_rehome . . . 104
10.5.3.7 sc_dlist_remove . . . 104
10.6 ethernet.h File Reference . . . 105
10.6.1 Detailed Description . . . 105
10.6.2 Macro Definition Documentation . . . 105
10.6.2.1 SC_8021Q_VID_MASK . . . 105
10.6.2.2 SC_ETHERTYPE_8021Q . . . 105
10.6.2.3 SC_ETHERTYPE_8021QinQ . . . 105
10.7 event.h File Reference . . . 105
10.7.1 Detailed Description . . . 106
10.7.2 Typedef Documentation . . . 106
10.7.2.1 sc_callback_handler_fn . . . 106
10.7.3 Function Documentation . . . 107

10.7.3.1 sc_callback_alloc . . . 107
10.7.3.2 sc_callback_alloc2 . . . 107
10.7.3.3 sc_callback_free . . . 107
10.7.3.4 sc_callback_is_active . . . 108
10.7.3.5 sc_callback_on_idle . . . 108
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 . . . 112
10.8.3.3 sc_node_init_fn . . . 113
10.8.3.4 sc_node_pkts_fn . . . 113
10.8.3.5 sc_node_prep_fn . . . 113
10.8.3.6 sc_node_select_subnode_fn . . . 114
10.8.4 Function Documentation . . . 114
10.8.4.1 sc_forward . . . 114
10.8.4.2 sc_forward2 . . . 115
10.8.4.3 sc_forward_list . . . 115
10.8.4.4 sc_forward_list2 . . . 115
10.8.4.5 sc_node_export_state . . . 115
10.8.4.6 sc_node_init_get_arg_dbl . . . 116
10.8.4.7 sc_node_init_get_arg_int . . . 116

10.8.4.8 sc_node_init_get_arg_int64 . . . 117
10.8.4.9 sc_node_init_get_arg_obj . . . 117
10.8.4.10 sc_node_init_get_arg_str . . . 118
10.8.4.11 sc_node_link_end_of_stream . . . 118
10.8.4.12 sc_node_link_end_of_stream2 . . . 118
10.8.4.13 sc_node_prep_check_links . . . 119
10.8.4.14 sc_node_prep_does_not_forward . . . 119
10.8.4.15 sc_node_prep_get_link . . . 119
10.8.4.16 sc_node_prep_get_link_or_free . . . 119
10.8.4.17 sc_node_prep_get_pool . . . 120
10.8.4.18 sc_node_prep_link_forwards_from_node . . . 120
10.8.4.19 sc_node_type_alloc . . . 121
10.9 ext_packet.h File Reference . . . 121
10.9.1 Detailed Description . . . 122
10.9.2 Macro Definition Documentation . . . 122
10.9.2.1 SC_MEMBER_OFFSET . . . 122
10.9.2.2 SC_MEMBER_SIZE . . . 123
10.9.3 Function Documentation . . . 123
10.9.3.1 sc_packet_bytes . . . 123
10.9.3.2 sc_packet_frags_tail . . . 123
10.9.3.3 sc_packet_prefetch_r . . . 123
10.9.3.4 sc_packet_prefetch_rw . . . 124
10.9.3.5 sc_packet_timespec . . . 124
10.10 ext_packet_list.h File Reference . . . 124
10.10.1 Detailed Description . . . 125
10.10.2 Function Documentation . . . 125
10.10.2.1 sc_packet_list_append . . . 125
10.10.2.2 sc_packet_list_append_list . . . 125
10.10.2.3 sc_packet_list_finalise . . . 125

10.10.2.4 sc_packet_list_init . . . 125
10.10.2.5 sc_packet_list_is_empty . . . 126
10.10.2.6 sc_packet_list_pop_head . . . 126
10.10.2.7 sc_packet_list_push_head . . . 126
10.10.2.8 sc_packet_list_tail . . . 126
10.11 hash_table.h File Reference . . . 127
10.11.1 Detailed Description . . . 127
10.11.2 Function Documentation . . . 127
10.11.2.1 sc_hash_table_alloc . . . 127
10.11.2.2 sc_hash_table_clear . . . 128
10.11.2.3 sc_hash_table_del . . . 128
10.11.2.4 sc_hash_table_del_val . . . 128
10.11.2.5 sc_hash_table_free . . . 128
10.11.2.6 sc_hash_table_get . . . 129
10.11.2.7 sc_hash_table_get_next_entry . . . 129
10.11.2.8 sc_hash_table_grow . . . 129
10.11.2.9 sc_hash_table_key_size . . . 130
10.11.2.10 sc_hash_table_num_entries . . . 130
10.11.2.11 sc_hash_table_val_size . . . 130
10.11.2.12 sc_hash_table_val_to_key . . . 130
10.12 iovec.h File Reference . . . 131
10.12.1 Detailed Description . . . 132
10.12.2 Function Documentation . . . 132
10.12.2.1 sc_iovec_copy_from_end . . . 132
10.12.2.2 sc_iovec_ptr_bytes . . . 133
10.12.2.3 sc_iovec_ptr_copy_out . . . 133
10.12.2.4 sc_iovec_ptr_find_chr . . . 133
10.12.2.5 sc_iovec_ptr_init . . . 133
10.12.2.6 sc_iovec_ptr_init_buf . . . 134

  

10.12.2.7 sc_iovec_ptr_init_packet . . . 134
10.12.2.8 sc_iovec_ptr_skip . . . 134
10.12.2.9 sc_iovec_trim_end . . . 134
10.13ip.h File Reference . . . 135
10.13.1 Detailed Description . . . 135
10.13.2 Macro Definition Documentation . . . 135
10.13.2.1 SC_IP4_FRAG_DONT . . . 135
10.13.2.2 SC_IP4_FRAG_MORE . . . 135
10.13.2.3 SC_IP4_OFFSET_MASK . . . 135
10.13.2.4 SC_TCP_ACK . . . 136
10.13.2.5 SC_TCP_FIN . . . 136
10.13.2.6 SC_TCP_PSH . . . 136
10.13.2.7 SC_TCP_RST . . . 136
10.13.2.8 SC_TCP_SYN . . . 136
10.13.2.9 SC_TCP_URG . . . 136
10.14mailbox.h File Reference . . . 136
10.14.1 Detailed Description . . . 137
10.14.2 Function Documentation . . . 137
10.14.2.1 sc_mailbox_alloc . . . 137
10.14.2.2 sc_mailbox_connect . . . 137
10.14.2.3 sc_mailbox_get_send_node . . . 137
10.14.2.4 sc_mailbox_poll . . . 138
10.14.2.5 sc_mailbox_send . . . 138
10.14.2.6 sc_mailbox_send_list . . . 138
10.14.2.7 sc_mailbox_set_recv . . . 138
10.15misc.h File Reference . . . 139
10.15.1 Detailed Description . . . 139
10.15.2 Function Documentation . . . 139
10.15.2.1 sc_join_mcast_group . . . 139

10.16 node.h File Reference . . . 140
10.16.1 Detailed Description . . . 140
10.16.2 Function Documentation . . . 140
10.16.2.1 sc_node_add_info_int . . . 140
10.16.2.2 sc_node_add_info_str . . . 141
10.16.2.3 sc_node_add_link . . . 141
10.16.2.4 sc_node_alloc . . . 141
10.16.2.5 sc_node_alloc_from_str . . . 142
10.16.2.6 sc_node_alloc_named . . . 142
10.16.2.7 sc_node_factory_lookup . . . 143
10.16.2.8 sc_node_from_object . . . 144
10.16.2.9 sc_node_get_thread . . . 144
10.16.2.10 sc_node_to_object . . . 145
10.17 object.h File Reference . . . 145
10.17.1 Detailed Description . . . 146
10.17.2 Enumeration Type Documentation . . . 146
10.17.2.1 sc_object_type . . . 146
10.17.3 Function Documentation . . . 146
10.17.3.1 sc_object_type . . . 146
10.17.3.2 sc_opaque_alloc . . . 146
10.17.3.3 sc_opaque_free . . . 146
10.17.3.4 sc_opaque_get_ptr . . . 147
10.17.3.5 sc_opaque_set_ptr . . . 147
10.18 packed_stream.h File Reference . . . 147
10.18.1 Detailed Description . . . 148
10.18.2 Macro Definition Documentation . . . 148
10.18.2.1 SC_PS_FLAG_BAD_FCS . . . 148
10.18.2.2 SC_PS_FLAG_BAD_L3_CSUM . . . 148
10.18.2.3 SC_PS_FLAG_BAD_L4_CSUM . . . 148

  

10.18.2.4 SC_PS_FLAG_CLOCK_IN_SYNC . . . 148
10.18.2.5 SC_PS_FLAG_CLOCK_SET . . . 149
10.18.3 Function Documentation . . . 149
10.18.3.1 sc_packed_packet_next . . . 149
10.18.3.2 sc_packed_packet_payload . . . 150
10.18.3.3 sc_packet_packed_end . . . 150
10.18.3.4 sc_packet_packed_first . . . 150
10.18.4 Variable Documentation . . . 151
10.18.4.1 ps_cap_len . . . 151
10.18.4.2 ps_flags . . . 151
10.18.4.3 ps_next_offset . . . 151
10.18.4.4 ps_orig_len . . . 151
10.18.4.5 ps_pkt_start_offset . . . 151
10.18.4.6 ps_ts_nsec . . . 151
10.18.4.7 ps_ts_sec . . . 151
10.19pkt_pool.h File Reference . . . 151
10.19.1 Detailed Description . . . 152
10.19.2 Function Documentation . . . 152
10.19.2.1 sc_packet_append_iovec_ptr . . . 152
10.19.2.2 sc_pool_duplicate_packed_packet . . . 153
10.19.2.3 sc_pool_duplicate_packet . . . 154
10.19.2.4 sc_pool_from_object . . . 154
10.19.2.5 sc_pool_get_buffer_size . . . 154
10.19.2.6 sc_pool_get_packets . . . 155
10.19.2.7 sc_pool_on_threshold . . . 155
10.19.2.8 sc_pool_return_packets . . . 155
10.19.2.9 sc_pool_set_refill_node . . . 156
10.19.2.10sc_pool_to_object . . . 156
10.19.2.11sc_pool_wraps_node . . . 156

10.20 predicate.h File Reference . . . 157
10.20.1 Detailed Description . . . 157
10.20.2 Function Documentation . . . 157
10.20.2.1 sc_pkt_predicate_alloc . . . 157
10.20.2.2 sc_pkt_predicate_from_object . . . 158
10.20.2.3 sc_pkt_predicate_to_object . . . 158
10.21 session.h File Reference . . . 158
10.21.1 Detailed Description . . . 159
10.21.2 Function Documentation . . . 159
10.21.2.1 sc_session_alloc . . . 159
10.21.2.2 sc_session_error_free . . . 159
10.21.2.3 sc_session_error_get . . . 160
10.21.2.4 sc_session_go . . . 160
10.21.2.5 sc_session_pause . . . 160
10.21.2.6 sc_session_prepare . . . 160
10.22 stream.h File Reference . . . 161
10.22.1 Detailed Description . . . 162
10.22.2 Function Documentation . . . 162
10.22.2.1 sc_stream_all . . . 162
10.22.2.2 sc_stream_alloc . . . 162
10.22.2.3 sc_stream_eth_dhost . . . 162
10.22.2.4 sc_stream_eth_shost . . . 163
10.22.2.5 sc_stream_eth_type . . . 163
10.22.2.6 sc_stream_eth_vlan_id . . . 163
10.22.2.7 sc_stream_free . . . 163
10.22.2.8 sc_stream_ip_dest_host . . . 164
10.22.2.9 sc_stream_ip_dest_hostport . . . 164
10.22.2.10 sc_stream_ip_dest_port . . . 164
10.22.2.11 sc_stream_ip_protocol . . . 165

  

10.22.2.12 sc_stream_ip_source_host . . . 165
10.22.2.13 sc_stream_ip_source_hostport . . . 165
10.22.2.14 sc_stream_ip_source_port . . . 165
10.22.2.15 sc_stream_mismatch . . . 166
10.22.2.16 sc_stream_reset . . . 166
10.22.2.17 sc_stream_set_str . . . 166
10.23 subnode_helper.h File Reference . . . 168
10.23.1 Detailed Description . . . 169
10.24 thread.h File Reference . . . 169
10.24.1 Detailed Description . . . 169
10.24.2 Function Documentation . . . 169
10.24.2.1 sc_thread_alloc . . . 169
10.24.2.2 sc_thread_calloc . . . 170
10.24.2.3 sc_thread_calloc_aligned . . . 170
10.24.2.4 sc_thread_get_time . . . 170
10.24.2.5 sc_thread_mfree . . . 171
10.24.2.6 sc_thread_poll . . . 171
10.24.2.7 sc_thread_poll_timers . . . 171
10.24.2.8 sc_thread_waitable_fd_get . . . 172
10.24.2.9 sc_thread_waitable_fd_prime . . . 172
10.25 time.h File Reference . . . 173
10.25.1 Detailed Description . . . 173
10.25.2 Function Documentation . . . 173
10.25.2.1 sc_ns_from_ms . . . 173
10.25.2.2 sc_ns_from_ts . . . 173
10.25.2.3 sc_ns_from_tv . . . 174
10.25.2.4 sc_ns_from_us . . . 174
10.25.2.5 sc_timer_expire_after_ns . . . 174
10.25.2.6 sc_timer_expire_at . . . 174

10.25.2.7 sc_timer_get_expiry_time . . . 175
10.25.2.8 sc_timer_push_back_ns . . . 175
10.26vi.h File Reference . . . 175
10.26.1 Detailed Description . . . 176
10.26.2 Function Documentation . . . 176
10.26.2.1 sc_vi_add_stream . . . 176
10.26.2.2 sc_vi_alloc . . . 176
10.26.2.3 sc_vi_alloc_from_group . . . 177
10.26.2.4 sc_vi_get_interface_name . . . 177
10.26.2.5 sc_vi_get_thread . . . 177
10.26.2.6 sc_vi_group_add_stream . . . 178
10.26.2.7 sc_vi_group_alloc . . . 178
10.26.2.8 sc_vi_group_get_session . . . 178
10.26.2.9 sc_vi_set_recv_node . . . 179

**Index** **181**

  

# 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.

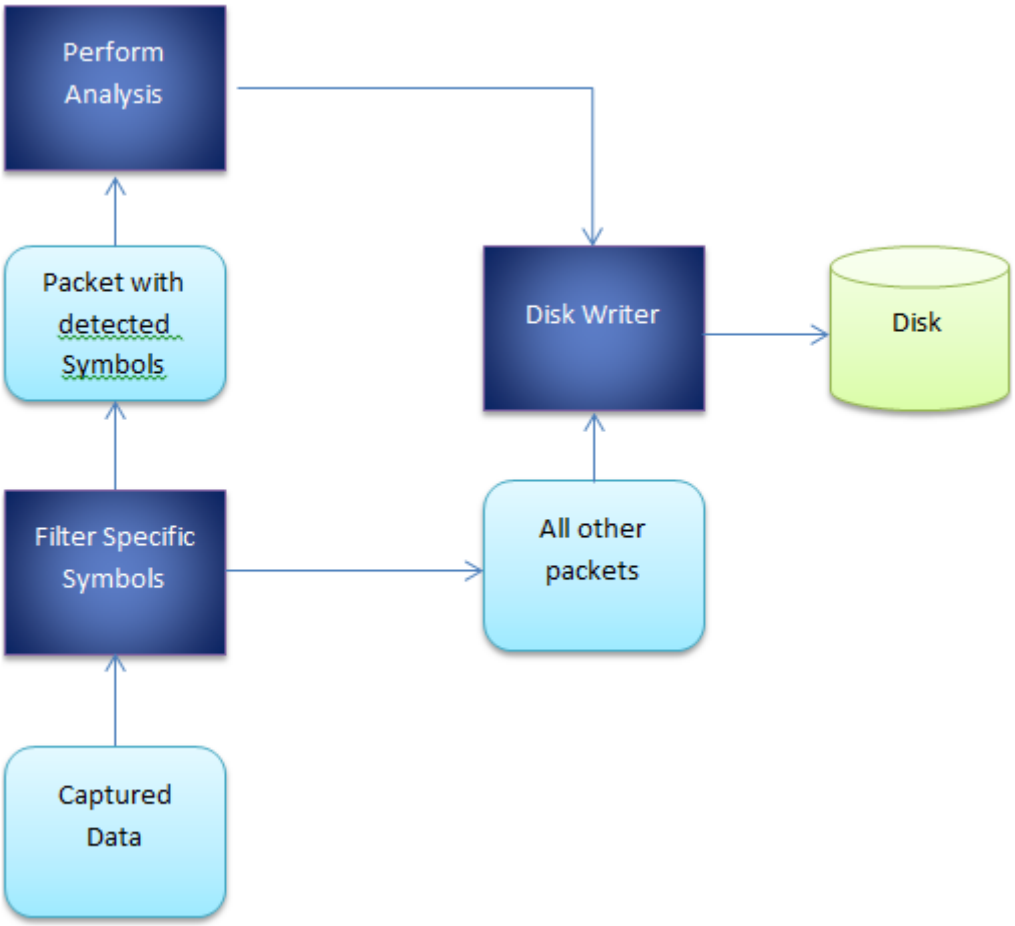


**Figure 2.1 Example Application Nodes**

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.

  

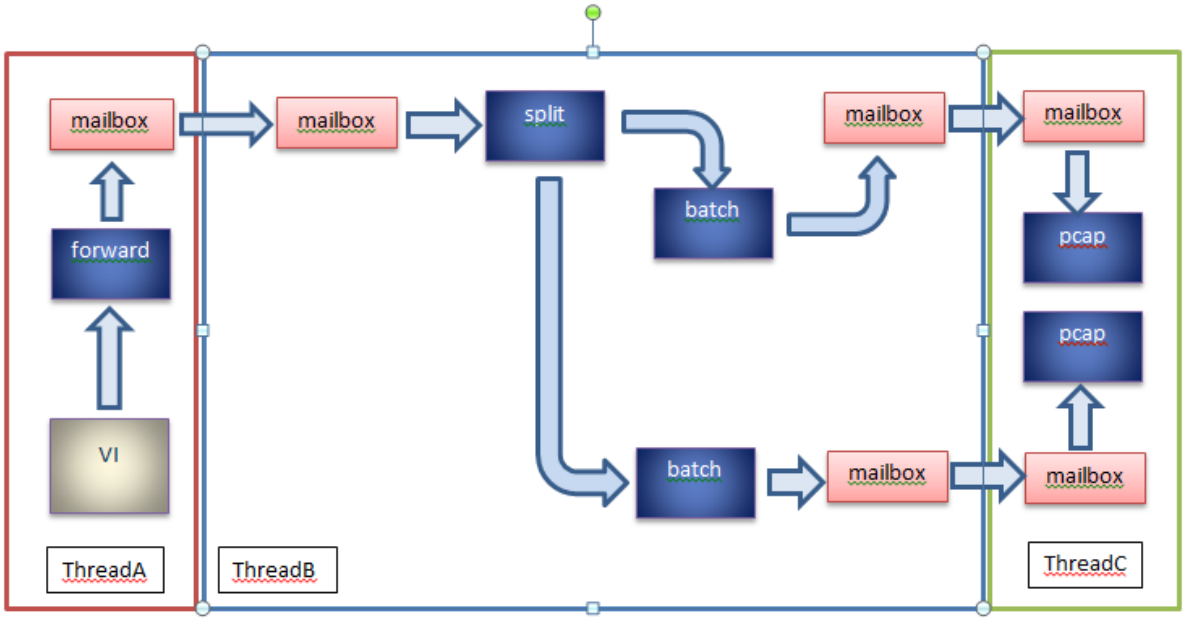


**Figure 2.2 Mailboxes**

## 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 let's 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 packet buffer timestamp with timestamp from Arista 7150 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 solar_capture_monitor. |
| 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. |
| 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 packet buffer timestamp with timestamp from Arista 7150 switch.

### Detailed Description

This node is used with Arista 7150 switches that are able to add hardware timestamps to packets. It replaces the SolarCapture timestamp with the timestamp generated by the switch.

#### 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. |

#### 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 |
| keyframes | no_sync | Used for keyframes |
| lldp | no_sync | 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 are exposed by the sc_arista_ts node.

| 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 skew zero packets received. |
| 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 | ev_count | Number of skews. |
| 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. |

## 5.3 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 |

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

If `num_lines` > 1:

- `usec/nsec 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 |
|---|---|---|---|---|
| num_lines | Yes | 1 | SC_PARAM_INT | Number of lines used in the hash. |
| usec | Yes | NULL | SC_PARAM_STR | Set this to a string of the form "<min_delay>[-<max_delay>]" to set the delay time of the node in microseconds. |
| nsec | Yes | NULL | SC_PARAM_STR | Set this to a string of the form "<min_delay>[-<max_delay>]" to set the delay time of the node in nanoseconds. |

Note: One and only one of usec and nsec 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 that reside within a single scope detect their exit condition, the process exits. 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 multiple sc_exit nodes are created when the process exits when all sc_exit nodes within a particular scope

Input packets are forwarded to the output unmodified.

#### Arguments

  

| Argument | Optional? | Default | Type | Description |
|---|---|---|---|---|
| end_of_stream | Yes | 1 | SC_PARAM_INT | Exit condition is met when end-of-stream signal is received. |
| scope | Yes | process | SC_PARAM_STR | See description. May be process, session or none. |
| predicate | Yes | | SC_PARAM_OBJ | Predicate is invoked on each input packet. Exit condition is met when it returns true. |

## 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 writev() 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 |
|---|---|---|---|---|
| copy_mode | No | | SC_PARAM_STR | Copy from input to output ('copy') or use zero-copy ('zc'). |
| mode | No | | SC_PARAM_STR | Balancing mode; 'estimate' or 'round-robin'. |
| flow_table_capacity | Yes | 1024 | SC_PARAM_STR | Initial capacity of the flow table. |
| flush_ns | Yes | 10000000 | SC_PARAM_STR | Flush timeout when copy_mode=copy. |

#### 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 |
|---|---|---|---|
| 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. |

## 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 |
|---|---|---|---|---|
| interface | No | | SC_PARAM_STR | The name of the network interface to use. |
| csum_ip | Yes | 0 | SC_PARAM_INT | Set to 1 to enable offload of the IPv4 header checksum. |
| csum_tcpudp | 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 |
|---|---|---|---|---|
| forward_truncated | 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. |
| lstrip | Yes | 1 | SC_PARAM_INT | Specifies whether whitespace should be stripped from the start of a line. |
| rstrip | Yes | 1 | SC_PARAM_INT | Specifies whether whitespace should be stripped from the end of a line. |
| strip_comments | 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 ('<br>') 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_CS-UM_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 expoential 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 sc_packet 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 |
|---|---|---|---|---|
| n_repeats | Yes | infinite | SC_PARAM_INT | 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). |

  

| | | | | |
|---|---|---|---|---|
| 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 |
|---|---|---|---|---|
| path | No | | SC_PARAM_STR | Location in the filesystem for the control socket. |
| reliable | Yes | 0 | SC_PARAM_INT | Set to 1 to request a reliable connection (which may cause head-of-line blocking). |
| active_connect | Yes | 1 | SC_PARAM_INT | 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 |
|---|---|---|---|
| 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. |

## 5.32 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 |
|---|---|---|---|---|
| with_pool | Yes | 0 | SC_PARAM_INT | Whether to allocate a pool. |
| with_free_link | Yes | 1 | SC_PARAM_INT | 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 |
|---|---|---|---|
| backlog_len | uint64_t | pkt_count | 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` argugment 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 thet 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 `max_bps`). |
| show_config | Yes | 0 | SC_PARAM_INT | When this is set to 1 the configuration is written to stderr 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 server_name, server_port, passive_open and connect_to are not used). |
| remote_args | Yes | | SC_PARAM_STR | Opaque message to send to remote side prior to starting sc_tunnel protocol. |
| passive_open | Yes | | SC_PARAM_INT | Should this node be opened in passive mode? Defaults to passive mode unless connect_to 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 ot 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 fsync() 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 | Arista timestamp statistics are exposed by the sc_arista_ts node. |
| 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

Arista timestamp statistics are exposed by the sc_arista_ts node.

| 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 skew zero packets received. |
| 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 | ev_count | Number of skews. |
| 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. |

## 6.2 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 . . . 69
sc_arg
Representation of an argument. Used by node init functions . . . 70
sc_attr
Attribute object . . . 71
sc_callback
A callback object . . . 71
sc_dlist
Doubly linked list pointers . . . 72
sc_hash_table
A hash table . . . 73
sc_iovec_ptr
An sc_iovec_ptr provides a convenient way to iterate over an iovec array without modifying it . . . 73
sc_node
Description of a node . . . 74
sc_node_factory
Struct to hold information about how to create an instance of this node . . . 75
sc_node_link
Description of a link the node has . . . 76
sc_node_type
Describes a type of node . . . 77
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 . . . 78
sc_packed_packet
A packed-stream packet . . . 79
sc_packet
Representation of a packet . . . 80
sc_packet_list
A list of packets or packet buffers . . . 82
sc_pkt_predicate
A packet predicate object . . . 83

  

sc_session_error
A SolarCapture session error object returned by sc_session_error_get . . . . . 84
sc_stream
A stream object . . . . . 85
sc_subnode_helper
sc_subnode_helper node private state . . . . . 86
sc_vi
A VI object . . . . . 89

  

# 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 . . . 91
args.h
Sc_arg: An argument to a node's initialisation function . . . 91
attr.h
Sc_attr: Control optional behaviours and tunables . . . 93
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 . . . 97
dlist.h
Sc_dlist: A doubly-linked list . . . 100
ethernet.h
Ethernet protocol definitions . . . 105
event.h
Sc_callback: Interface for event notification . . . 105
ext_node.h
Interface for writing custom nodes . . . 109
ext_packet.h
Sc_packet: The representation of a packet or other data . . . 121
ext_packet_list.h
Sc_packet_list: A list of packets . . . 124
hash_table.h
A hash table with open addressing and double hashing . . . 127
iovec.h
Sc_iovec_ptr: Supports iterating over a 'struct iovec' . . . 131
ip.h
IP protocol definitions . . . 135
mailbox.h
Sc_mailbox: A means to pass packets from one thread to another . . . 136
misc.h
Miscellaneous utility functions . . . 139
node.h
Sc_node: An object that processes packets . . . 140

  

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) . . . 145
packed_stream.h
Sc_packed_packet: The packed-stream encapsulation . . . 147
pkt_pool.h
Sc_pool: A pool of packet buffers . . . 151
predicate.h
Sc_pkt_predicate: Interface for testing properties of packets . . . 157
session.h
Sc_session: A set of threads and other objects . . . 158
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 . . . 161
subnode_helper.h
sc_subnode_helper node interface . . . 168
thread.h
Sc_thread: Representation of a thread in SolarCapture . . . 169
time.h
Functions for managing time . . . 173
vi.h
Sc_vi: Supports receiving packets from the network . . . 175

# 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

- struct sc_node_link * free_link
- 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 struct sc_packet_list* append_to

Application must point this at an initialised packet list.

#### 9.1.2.2 struct sc_node_link* free_link

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

#### 9.1.2.3 struct sc_node_link** links

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

#### 9.1.2.4 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**
  } val

### 9.2.1 Detailed Description

Representation of an argument. Used by node init functions.

### 9.2.2 Field Documentation

#### 9.2.2.1 const char* name

Parameter name

#### 9.2.2.2 enum sc_param_type type

Parameter type

#### 9.2.2.3 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 sc_callback_handler_fn* cb_handler_fn

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

#### 9.4.2.2 struct sc_dlist cb_link

Internal use only.

#### 9.4.2.3 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 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 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

- struct iovec * iov
- int iovlen
- 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 struct iovec io

Currently iterated iovec

#### 9.7.2.2 struct iovec* iov

Pointer to start of array

#### 9.7.2.3 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

- 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 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 void* nd_private

Set by node for local state

#### 9.8.2.3 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 sc_node_init_fn* nf_init_fn

Function called by SolarCapture core to initialise a new node.

#### 9.9.2.2 const char* nf_name

Name of this node factory.

#### 9.9.2.3 int nf_node_api_ver

Minimum version of SolarCapture that this node is compatible with.

#### 9.9.2.4 void* nf_private

Private state for the implementation.

#### 9.9.2.5 void* nf_reserved[8]

Reserved.

#### 9.9.2.6 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 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 sc_node_add_link_fn* nt_add_link_fn

(Optional) Add an outgoing link.

  

#### 9.11.2.2 sc_node_end_of_stream_fn* nt_end_of_stream_fn

(Optional) Handle end-of-stream signal.

#### 9.11.2.3 const char* nt_name

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

#### 9.11.2.4 sc_node_pkts_fn* nt_pkts_fn

(Optional) Handle incoming packets.

#### 9.11.2.5 sc_node_prep_fn* nt_prep_fn

(Optional) Prepare for packet processing.

#### 9.11.2.6 void* nt_private

Private state for the implementation.

#### 9.11.2.7 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 uint16_t ps_cap_len

Number of bytes of packet payload stored.

#### 9.13.2.2 uint8_t ps_flags

SC_PS_FLAG_* flags.

  

#### 9.13.2.3 uint16_t ps_next_offset

Offset of next packet from start of this struct.

#### 9.13.2.4 uint16_t ps_orig_len

Original length of the frame.

#### 9.13.2.5 uint8_t ps_pkt_start_offset

Offset of packet payload from start of this struct.

#### 9.13.2.6 uint32_t ps_ts_nsec

Timestamp (nanoseconds).

#### 9.13.2.7 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 uint16_t flags

flags defined below

#### 9.14.2.2 struct sc_packet* frags

list of chained fragments

#### 9.14.2.3 uint8_t frags_n

number of fragments in `frags` chain

#### 9.14.2.4 struct sc_packet** frags_tail

last fragment in chain

#### 9.14.2.5 uint16_t frame_len

original frame length in bytes

#### 9.14.2.6 struct iovec* iov

identifies packet data

#### 9.14.2.7 uint8_t iovlen

number of entries in `iov` array

  

#### 9.14.2.8 uintptr_t* metadata

packet metadata

#### 9.14.2.9 struct sc_packet* next

next packet in a packet list

#### 9.14.2.10 uint16_t reserved1

reserved

#### 9.14.2.11 uint32_t reserved2

reserved

#### 9.14.2.12 uint32_t ts_nsec

timestamp (nanoseconds)

#### 9.14.2.13 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 struct sc_packet* head

Head of list

#### 9.15.2.2 int num_frags

Number of pkt frags in the list

#### 9.15.2.3 int num_pkts

Number of pkts in the list

#### 9.15.2.4 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 void* pred_private

Field to hold state for the predicate function

#### 9.16.2.2 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 int err_errno

The errno for the error.

#### 9.17.2.2 char* err_file

The source file the error occurred in.

#### 9.17.2.3 char* err_func

The function the error occurred in.

#### 9.17.2.4 int err_line

The line number the error was issued from.

#### 9.17.2.5 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
- struct sc_node_link * sh_free_link
- 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 typedef void( sc_sh_handle_backlog_fn)(struct sc_subnode_helper *sh) [related]

Signature of sh_handle_backlog_fn.

**Parameters**

| | |
|---|---|
| *sh* | 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 typedef void( sc_sh_handle_end_of_stream_fn)(struct sc_subnode_helper *sh) [related]

Signature of sh_handle_end_of_stream_fn.

**Parameters**

| | |
|---|---|
| *sh* | 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 static struct sc_subnode_helper * sc_subnode_helper_from_node ( struct sc_node * *node* ) [related]

Get *sc_subnode_helper from sc_node*

**Parameters**

| | |
|---|---|
| *node* | Node of type sc_subnode_helper. |

**Returns**

The sc_subnode_helper from the node

#### 9.19.2.4 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**

| | |
|---|---|
| *sh* | An sc_subnode_helper instance |

### 9.19.3 Field Documentation

#### 9.19.3.1 struct sc_packet_list sh_backlog

Unprocessed incoming packets

#### 9.19.3.2 struct sc_node_link* sh_free_link

A node link for freeing packets (if requested)

#### 9.19.3.3 sc_sh_handle_backlog_fn* sh_handle_backlog_fn

Handler invoked to process the backlog

#### 9.19.3.4 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 struct sc_node_link** sh_links

Outgoing links

#### 9.19.3.6 int sh_n_links

Number of outgoing links

#### 9.19.3.7 struct sc_node* sh_node

The node

#### 9.19.3.8 uint64_t sh_poll_backlog_ns

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

#### 9.19.3.9 struct sc_pool* sh_pool

A packet pool (if requested)

#### 9.19.3.10 int sh_pool_threshold

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

#### 9.19.3.11 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 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*** sc_object pointer

***SC_PARAM_DBL*** native double type

### 10.2.3 Function Documentation

#### 10.2.3.1 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**

| | |
|---|---|
| *name* | Name of argument. |
| *val* | Value of argument. |

**Returns**

The constructed sc_arg struct

#### 10.2.3.2 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**

| | |
|---|---|
| *name* | Name of argument. |
| *val* | Value of argument. |

**Returns**

The constructed sc_arg struct

#### 10.2.3.3 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**

| | |
|---|---|
| *name* | Name of argument. |
| *val* | Value of argument. |

**Returns**

The constructed sc_arg struct

#### 10.2.3.4 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**

| | |
|---|---|
| *name* | Name of argument. |
| *val* | 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.*
- 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 int sc_attr_alloc ( struct sc_attr ** *attr_out* )

Allocate an attribute object.

**Parameters**

| | |
|---|---|
| *attr_out* | 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 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**

| | |
|---|---|
| *attr_name_opt* | The attribute name. |
| *docs_out* | On success, the resulting doc string output. |
| *docs_len_out* | On success, the length of the doc string output. |

**Returns**

0 on success, or a negative error code.

#### 10.3.2.3 int struct sc_attr* sc_attr_dup ( const struct sc_attr * *attr* )

Duplicate an attribute object.

**Parameters**

| | |
|---|---|
| *attr* | 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 void sc_attr_free ( struct sc_attr * *attr* )

Free an attribute object.

**Parameters**

| | |
|---|---|
| *attr* | The attribute object. |

#### 10.3.2.5 struct sc_attr* sc_attr_from_object ( struct sc_object * *obj* )

Convert an sc_object to an sc_attr.

**Parameters**

| | |
|---|---|
| *obj* | An sc_object 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 void sc_attr_reset ( struct sc_attr * *attr* )

Return attributes to their default values.

**Parameters**

| | |
|---|---|
| *attr* | The attribute object. |

#### 10.3.2.7 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**

| | |
|---|---|
| *attr* | The attribute object. |
| *name* | Name of the attribute. |
| *fmt* | 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 int sc_attr_set_from_str ( struct sc_attr * *attr*, const char * *name*, const char * *val* )

Set an attribute from a string value.

**Parameters**

| | |
|---|---|
| *attr* | The attribute object. |
| *name* | Name of the attribute. |
| *val* | 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 int sc_attr_set_int ( struct sc_attr * *attr*, const char * *name*, int64_t *val* )

Set an attribute to an integer value.

**Parameters**

| | |
|---|---|
| *attr* | The attribute object. |
| *name* | Name of the attribute. |
| *val* | 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 int sc_attr_set_str ( struct sc_attr * *attr*, const char * *name*, const char * *val* )

Set an attribute to a string value.

**Parameters**

| | |
|---|---|
| *attr* | The attribute object. |
| *name* | Name of the attribute. |
| *val* | 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 struct sc_object* sc_attr_to_object ( const struct sc_attr * *attr* )

Convert an sc_attr to an sc_object.

**Parameters**

| | |
|---|---|
| *attr* | An sc_attr 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

  *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 them 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 #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**

| | |
|---|---|
| *name* | The field name. |
| *val* | The constant. |

#### 10.4.2.2 #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**

| | |
|---|---|
| *type* | The basic data type. |
| *name* | The field name. |
| *kind* | A string to describe the kind of data. Examples used by SolarCapture nodes are pkt_count, ev_count, config, const, magnitude. |

#### 10.4.2.3 #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**

| | |
|---|---|
| *name* | The field name. |
| *len* | The length of the string. |
| *kind* | A string to describe the kind of data. Examples used by SolarCapture nodes are pkt_count, ev_count, config, const, magnitude. |

#### 10.4.2.4 #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**

| | |
|---|---|
| *name* | 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 #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**

| | |
|---|---|
| *c_type* | The container type. |
| *mbr_name* | The name of the member in c_type. |
| *p_mbr* | Pointer to the member. |

  

#### 10.5.2.2 #define SC_DLIST_FOR_EACH_OBJ( *list, iter, mbr* )

**Value:**

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

| | |
|---|---|
| *list* | A pointer to the head of the sc_dlist. |
| *iter* | A pointer of the same type as the container. |
| *mbr* | The name of the field in the container containing the sc_dlist struct. |

#### 10.5.2.3 #define SC_DLIST_FOR_EACH_OBJ_SAFE( *list, iter, next_entry, mbr* )

**Value:**

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

| | |
|---|---|
| *list* | A pointer to the head of the sc_dlist. |
| *iter* | A pointer of the same type as the container. |
| *next_entry* | A pointer of the same type as the container. |
| *mbr* | The name of the field in the container containing the sc_dlist struct. |

### 10.5.3 Function Documentation

#### 10.5.3.1 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**

| | |
|---|---|
| *list* | A pointer to the pre-allocated sc_dlist to be initialised. |

#### 10.5.3.2 static struct sc_dlist* sc_dlist_pop_head ( struct sc_dlist * *list* ) [static]

Pop off the head of a list.

**Parameters**

| | |
|---|---|
| *list* | The point to pop the head from. |

**Returns**

The item popped from `list`.

#### 10.5.3.3 static struct sc_dlist* sc_dlist_pop_tail ( struct sc_dlist * *list* ) [static]

Pop the tail of a list.

  

**Parameters**

| | |
|---|---|
| *list* | The point to pop the tail from. |

**Returns**

The item popped from `list`.

#### 10.5.3.4 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**

| | |
|---|---|
| *list* | The list to prepend to. |
| *l* | The item to prepend to `list`. |

#### 10.5.3.5 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**

| | |
|---|---|
| *list* | The list to append to. |
| *l* | The item to append to `list`. |

#### 10.5.3.6 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**

| | |
|---|---|
| *to_list* | The item to add to the list, replacing `from_list`. |
| *from_list* | The item to remove from the list. |

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

Remove an item from the list.

**Parameters**

| | |
|---|---|
| / | 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 #define SC_8021Q_VID_MASK 0xfff

Mask for VLAN identifier (VID)

#### 10.6.2.2 #define SC_ETHERTYPE_8021Q 0x8100

EtherType for IEEE 802.1Q

#### 10.6.2.3 #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 typedef void( sc_callback_handler_fn)(struct sc_callback *, void *event_info)

A callback handler function.

**Parameters**

| | |
|---|---|
| *callback* | The callback struct registered with this callback. |
| *event_info* | If callback was registered using sc_epoll_ctl this will contain the uint32_t epoll_events bitmask (see man 2 epoll_ctl) In all other cases this is not used. |

### 10.7.3 Function Documentation

#### 10.7.3.1 int sc_callback_alloc ( struct sc_callback ** *cb_out*, const struct sc_attr * *attr*, struct sc_thread * *thread* )

Allocate a callback object instance.

**Parameters**

| | |
|---|---|
| *cb_out* | The allocated callback object is returned here |
| *attr* | Attributes |
| *thread* | 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 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**

| | |
|---|---|
| *cb_out* | The allocated callback object is returned here |
| *attr* | Attributes |
| *thread* | The thread the callback will be used with |
| *description* | 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 void sc_callback_free ( struct sc_callback * *cb* )

Free a callback object instance.

  

**Parameters**

| | |
|---|---|
| *cb* | The callback object to free |

This function frees a callback object instance.

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

Returns true if a callback object is active.

**Parameters**

| | |
|---|---|
| *cb* | The callback object |

#### 10.7.3.5 void sc_callback_on_idle ( struct sc_callback * *cb* )

Request a callback when the thread is idle.

**Parameters**

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

Unregister a callback object from its event source.

**Parameters**

| | |
|---|---|
| *cb* | The callback object |

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

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

Set description of a callback.

**Parameters**

| | |
|---|---|
| *cb* | The callback object |
| *fmt* | 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 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**

| | |
|---|---|
| *thread* | The thread managing `fd` |
| *op* | EPOLL_CTL_ADD, EPOLL_CTL_MOD or EPOLL_CTL_DEL |
| *fd* | The file descriptor |
| *events* | Event flags (EPOLLIN, EPOLLOUT etc.) |
| *cb* | 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.*
- struct sc_node_link * sc_node_prep_get_link (struct sc_node *node, const char *link_name)

  *Find a named outgoing link.*
- 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 #define sc_node_fwd_error( *node, rc* ) __sc_node_fwd_error((node), __FILE__, __LINE__, __func__, (rc))

Forward error from a failed sc call.

**Parameters**

| | |
|---|---|
| *node* | The node that forwards the error |
| *rc* | The error code returned by the sc call |

Call this function to propagate an error generated by SolarCapture.

#### 10.8.2.2 #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**

  

| | |
|---|---|
| *node* | The node that originates the error |
| *errno_code* | 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 #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**

| | |
|---|---|
| *node* | The node that originates the error |
| *errno_code* | An error code from errno.h (or can be zero) |
| *fmt* | vprintf style format string |
| *args* | 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 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**

| | |
|---|---|
| *from_node* | The node being linked from |
| *link_name* | The name of the link |
| *to_node* | The node being linked to |
| *to_name_opt* | 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 typedef void( sc_node_end_of_stream_fn)(struct sc_node *node)

Signature of nt_end_of_stream_fn function.

**Parameters**

| | |
|---|---|
| *node* | 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 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**

| | |
|---|---|
| *node* | The node being initialised |
| *attr* | Attributes used to create the node |
| *factory* | 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 typedef void( sc_node_pkts_fn)(struct sc_node *node, struct sc_packet_list *packet_list)

Signature of nt_pkts_fn function.

**Parameters**

| | |
|---|---|
| *node* | The node receiving the packets |
| *packet_list* | 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 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**

| | |
|---|---|
| *node* | The node being prepared |
| *links* | Array of outgoing links the node has |
| *n_links* | 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 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**

| | |
|---|---|
| *node* | The node being linked to |
| *name_opt* | The name of the link (may be NULL) |
| *new_name_out* | 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 void sc_forward ( struct sc_node * *node*, const struct sc_node_link * *link*, struct sc_packet * *packet* )

Forward a single packet.

**Parameters**

| | |
|---|---|
| *node* | The node |
| *link* | The link to forward through |
| *packet* | The packet to forward |

See also sc_forward2.

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

Forward a single packet.

**Parameters**

| | |
|---|---|
| *link* | The link to forward through |
| *packet* | The packet to forward |

#### 10.8.4.3 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**

| | |
|---|---|
| *node* | The node |
| *link* | The link to forward through |
| *pl* | The list of packets to forward |

See also sc_forward_list2.

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

Forward a list of packets.

**Parameters**

| | |
|---|---|
| *link* | The link to forward through |
| *pl* | The list of packets to forward |

#### 10.8.4.5 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**

| | |
|---|---|
| *node* | The node exporting state |

  

| | |
|---|---|
| *type_name* | Name of the exported datastructure |
| *type_size* | Size in bytes of the exported datastructure |
| *pp_area* | 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 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**

| | |
|---|---|
| *v_out* | On success, the value is returned here |
| *node* | The node |
| *name* | The name of the argument |
| *v_default* | 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 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**

| | |
|---|---|
| *v_out* | On success, the value is returned here |
| *node* | The node |
| *name* | The name of the argument |
| *v_default* | 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 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**

| | |
|---|---|
| *v_out* | On success, the value is returned here |
| *node* | The node |
| *name* | The name of the argument |
| *v_default* | 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 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**

| | |
|---|---|
| *obj_out* | On success, the value is returned here |
| *node* | The node |

  

| | |
|---|---|
| *name* | The name of the argument |
| *obj_type* | 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 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**

| | |
|---|---|
| *v_out* | On success, the value is returned here |
| *node* | The node |
| *name* | The name of the argument |
| *v_default* | 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 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**

| | |
|---|---|
| *node* | The node |
| *link* | The link |

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

#### 10.8.4.12 void sc_node_link_end_of_stream2 ( const struct sc_node_link * *link* )

Indicate end-of-stream on a link.

  

**Parameters**

| | |
|---|---|
| *link* | The link |

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

#### 10.8.4.13 int sc_node_prep_check_links ( struct sc_node * *node* )

Check the node's links for any unused links.

**Parameters**

| | |
|---|---|
| *node* | 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 void sc_node_prep_does_not_forward ( struct sc_node * *node* )

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

**Parameters**

| | |
|---|---|
| *node* | 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 struct sc_node_link* sc_node_prep_get_link ( struct sc_node * *node,* const char * *link_name* )

Find a named outgoing link.

**Parameters**

| | |
|---|---|
| *node* | The node |
| *link_name* | 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 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**

| | |
|---|---|
| *node* | The node |
| *link_name* | 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 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**

| | |
|---|---|
| *pool_out* | On success, the pool is returned here |
| *attr* | Packet pool attributes (optional, may be NULL) |
| *node* | The node |
| *links* | The link(s) packets from the pool may be passed to (set to NULL for all) |
| *n_links* | 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 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**

| | |
|---|---|
| *node* | The node that `link` originates from |
| *link* | A link from `node` to another node |
| *from_node* | 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 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**

| | |
|---|---|
| *nt_out* | The allocated sc_node_type instance |
| *attr_opt* | Optional attributes (may be NULL) |
| *factory* | 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 #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**

| | |
|---|---|
| *c_type* | The struct type. |
| *mbr_name* | The field name to calculate the offset of. |

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

Calculate the size of a field within a struct.

**Parameters**

| | |
|---|---|
| *c_type* | The struct type. |
| *mbr_name* | The field to calculate the size of. |

### 10.9.3 Function Documentation

#### 10.9.3.1 static int sc_packet_bytes ( struct sc_packet ∗ *p* ) [inline], [static]

Return the size of the packet data in bytes.

**Parameters**

| | |
|---|---|
| *p* | A packet object. |

**Returns**

The size of the packet data in bytes.

#### 10.9.3.2 static struct sc_packet∗ sc_packet_frags_tail ( struct sc_packet ∗ *p* ) [static]

Return a packet's last fragment.

**Parameters**

| | |
|---|---|
| *p* | 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 static void sc_packet_prefetch_r ( struct sc_packet ∗ *p* ) [inline], [static]

Prefetch a packet for reading.

  

**Parameters**

| | |
|---|---|
| *p* | A packet object. |

#### 10.9.3.4 static void sc_packet_prefetch_rw ( struct sc_packet * *p* ) [inline], [static]

Prefetch a packet for reading and writing.

**Parameters**

| | |
|---|---|
| *p* | A packet object. |

#### 10.9.3.5 static struct timespec sc_packet_timespec ( const struct sc_packet * *p* ) [static]

Return the timestamp of the packet in timespec format.

**Parameters**

| | |
|---|---|
| *p* | 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 *l)
  *Initialise a list.*
- static int sc_packet_list_is_empty (const struct sc_packet_list *l)
  *Check if packet list is empty.*
- static void sc_packet_list_finalise (struct sc_packet_list *l)
  *Finalise a list.*
- static struct sc_packet * sc_packet_list_tail (struct sc_packet_list *l)
  *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 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**

| | |
|---|---|
| *l* | The packet list. |
| *p* | The packet. |

#### 10.10.2.2 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**

| | |
|---|---|
| *dest* | The list to be extended. |
| *src* | The list to be appended to `dest`. |

After this call `dest` is finalised if and only if `src` was finalised.

`src` must be non-empty.

#### 10.10.2.3 static void sc_packet_list_finalise ( struct sc_packet_list * *l* ) [inline], [static]

Finalise a list.

**Parameters**

| | |
|---|---|
| *l* | 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 static void sc_packet_list_init ( struct sc_packet_list * *l* ) [inline], [static]

Initialise a list.

**Parameters**

  

| | |
|---|---|
| *l* | The packet list. |

#### 10.10.2.5 static int sc_packet_list_is_empty ( const struct sc_packet_list * *l* ) [inline], [static]

Check if packet list is empty.

**Parameters**

| | |
|---|---|
| *l* | The packet list. |

**Returns**

True (1) if the packet list is empty, false (0) otherwise

#### 10.10.2.6 static struct sc_packet* sc_packet_list_pop_head ( struct sc_packet_list * *pl* ) [static]

Remove and return the head of the list.

**Parameters**

| | |
|---|---|
| *pl* | 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 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**

| | |
|---|---|
| *pl* | The packet list. |
| *p* | The packet. |

#### 10.10.2.8 static struct sc_packet* sc_packet_list_tail ( struct sc_packet_list * *l* ) [static]

Return the tail of current tail of the list.

**Parameters**

| | |
|---|---|
| *l* | 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 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**

| | |
|---|---|
| *table_out* | The allocated sc_hash_table is returned here |
| *key_size* | The size in bytes of keys |
| *val_size* | The size in bytes of values |
| *capacity* | 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 void sc_hash_table_clear ( struct sc_hash_table * *table* )

Clear all entries from an sc_hash_table.

**Parameters**

| | |
|---|---|
| *table* | A hash table |

#### 10.11.2.3 int sc_hash_table_del ( struct sc_hash_table * *table*, const void * *key* )

Remove an entry from an sc_hash_table by key.

**Parameters**

| | |
|---|---|
| *table* | A hash table |
| *key* | The key to remove |

**Returns**

- 0 on success
- -ENOENT if `key` was not found

#### 10.11.2.4 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**

| | |
|---|---|
| *table* | A hash table |
| *val* | 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 void sc_hash_table_free ( struct sc_hash_table * *table* )

Free an sc_hash_table.

**Parameters**

| | |
|---|---|
| *table* | A hash table |

#### 10.11.2.6 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**

| | |
|---|---|
| *table* | A hash table |
| *key* | The key to look for in the table |
| *insert_if_not_found* | If true then an entry is inserted if not found |
| *val_out* | 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 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**

| | |
|---|---|
| *table* | A hash table |
| *key_out* | Pointer to the next key returned here |
| *val_out* | Pointer to the next value returned here |
| *iterator* | 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 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**

| | |
|---|---|
| *table* | A hash table |
| *max_size* | 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 unsigned sc_hash_table_key_size ( struct sc_hash_table * *table* )

Get the size in bytes of a hash table's keys.

**Parameters**

| | |
|---|---|
| *table* | A hash table |

**Returns**

The size in bytes of the hash table's keys.

#### 10.11.2.10 unsigned sc_hash_table_num_entries ( struct sc_hash_table * *table* )

Get the number of entries in an sc_hash_table.

**Parameters**

| | |
|---|---|
| *table* | A hash table |

**Returns**

The number of entries in the hash table.

#### 10.11.2.11 unsigned sc_hash_table_val_size ( struct sc_hash_table * *table* )

Get the size in bytes of each value buffer.

**Parameters**

| | |
|---|---|
| *table* | A hash table |

**Returns**

The size in bytes of each value buffer.

#### 10.11.2.12 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**

| | |
|---|---|
| *table* | A hash table |
| *val* | 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 *iov, 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 (void *dest_buf, const struct iovec *iov, int iovlen, int bytes)

  *Copy data out of the end of a sc_iovec_ptr.*
- static void sc_iovec_trim_end (struct iovec *iov, uint8_t *iovlen, 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 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**

| | |
|---|---|
| *dest_buf* | Buffer to copy to. |
| *iov* | A pointer to an array of iovec objects. |
| *iovlen* | The number of iovec objects in `iov`. This must be > 0. |
| *bytes* | Number of bytes to copy. |

Note: The caller must ensure that at least `bytes` of data are available in `iov`

#### 10.12.2.2 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**

| | |
|---|---|
| *iovp* | The sc_iovec_ptr |

**Returns**

The number of bytes represented by the sc_iovec_ptr

#### 10.12.2.3 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**

| | |
|---|---|
| *dest* | Buffer to copy to |
| *iovp* | An sc_iovec_ptr |
| *max_bytes* | Max number of bytes to copy (length of `dest`) |

**Returns**

The number of bytes copied.

#### 10.12.2.4 int sc_iovec_ptr_find_chr ( const struct sc_iovec_ptr * *iovp*, int *c* )

Find offset of character in iovec.

**Parameters**

| | |
|---|---|
| *iovp* | An sc_iovec_ptr |
| *c* | Character to find |

**Returns**

The offset of first occurrence of character `c` in the memory reference by `iovp`, or -1 if not found.

#### 10.12.2.5 static void sc_iovec_ptr_init ( struct sc_iovec_ptr * *iovp*, const struct iovec * *iov*, int *iovlen* ) [inline], [static]

Initialise a struct sc_iovec_ptr.

  

**Parameters**

| | |
|---|---|
| *iovp* | The sc_iovec_ptr to initialise |
| *iov* | Pointer to array of 'struct iovec's |
| *iovlen* | Length of the `iov` array |

#### 10.12.2.6 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**

| | |
|---|---|
| *iovp* | The sc_iovec_ptr to initialise |
| *buf* | Pointer to the start of the buffer |
| *len* | Length of the buffer |

#### 10.12.2.7 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**

| | |
|---|---|
| *iovp* | The sc_iovec_ptr to initialise |
| *packet* | The packet |

#### 10.12.2.8 int sc_iovec_ptr_skip ( struct sc_iovec_ptr * *iovp,* int *bytes_to_skip* )

Skip forward over an iovec.

**Parameters**

| | |
|---|---|
| *iovp* | An sc_iovec_ptr |
| *bytes_to_skip* | 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.

#### 10.12.2.9 static void sc_iovec_trim_end ( struct iovec * *iov,* uint8_t * *iovlen,* int *bytes* ) [inline], [static]

Remove data from the end of an iovec.

**Parameters**

| | |
|---|---|
| *iov* | A pointer to an array of iovec objects. |
| *iovlen* | The number of iovec objects in `iov`. |
| *bytes* | 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 #define SC_IP4_FRAG_DONT 0x4000

Mask for Don't Fragment flag in IP header

#### 10.13.2.2 #define SC_IP4_FRAG_MORE 0x2000

Mask for More Fragments flag in IP header

#### 10.13.2.3 #define SC_IP4_OFFSET_MASK 0x1fff

Mask for Fragment Offset field in IP header

  

#### 10.13.2.4 #define SC_TCP_ACK 0x10

Mask for ACK flag in TCP header

#### 10.13.2.5 #define SC_TCP_FIN 0x01

Mask for FIN flag in TCP header

#### 10.13.2.6 #define SC_TCP_PSH 0x08

Mask for PSH flag in TCP header

#### 10.13.2.7 #define SC_TCP_RST 0x04

Mask for RST flag in TCP header

#### 10.13.2.8 #define SC_TCP_SYN 0x02

Mask for SYN flag in TCP header

#### 10.13.2.9 #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 int sc_mailbox_alloc ( struct sc_mailbox ** *mb_out,* const struct sc_attr * *attr,* struct sc_thread * *thread* )

Allocate a mailbox.

**Parameters**

| | |
|---|---|
| *mb_out* | The allocated mailbox is returned here. |
| *attr* | Attributes (see sc_attr). |
| *thread* | 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 int sc_mailbox_connect ( struct sc_mailbox * *mb1,* struct sc_mailbox * *mb2* )

Connect a pair of mailboxes.

**Parameters**

| | |
|---|---|
| *mb1* | The first mailbox. |
| *mb2* | 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 struct sc_node* sc_mailbox_get_send_node ( struct sc_mailbox * *mailbox* )

Return a mailbox's "send node".

  

**Parameters**

| | |
|---|---|
| *mailbox* | 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 int sc_mailbox_poll ( struct sc_mailbox * *mailbox,* struct sc_packet_list * *list* )

Poll a mailbox.

**Parameters**

| | |
|---|---|
| *mailbox* | The mailbox to poll. |
| *list* | 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 void sc_mailbox_send ( struct sc_mailbox * *mailbox,* struct sc_packet * *packet* )

Send a packet through a mailbox to another thread.

**Parameters**

| | |
|---|---|
| *mailbox* | The mailbox. |
| *packet* | 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 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**

| | |
|---|---|
| *mailbox* | The mailbox. |
| *list* | 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 int sc_mailbox_set_recv ( struct sc_mailbox * *mailbox,* struct sc_node * *node,* const char * *name_opt* )

Connect a mailbox to a node.

**Parameters**

| | |
|---|---|
| *mailbox* | The mailbox. |
| *node* | The node. |
| *name_opt* | 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 int sc_join_mcast_group ( struct sc_session ∗ *scs,* const char ∗ *interface,* const char ∗ *group* )

Join a multicast group.

**Parameters**

| | |
|---|---|
| *scs* | A session |
| *interface* | The network interface to join on |
| *group* | 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 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**

| | |
|---|---|
| *node* | The node exporting state. |
| *field_name* | Name of field. |
| *field_val* | 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 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**

| | |
|---|---|
| *node* | The node exporting state. |
| *field_name* | Name of field. |
| *field_val* | 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 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**

| | |
|---|---|
| *from_node* | Node to connect from. |
| *link_name* | Name of the `from_node's` egress link. |
| *to_node* | Node to connect to. |
| *to_name_opt* | 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 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**

| | |
|---|---|
| *node_out* | The allocated node is returned here |
| *attr* | Attributes |
| *thread* | The thread the node will be in |
| *factory* | A node factory |
| *args* | An array of arguments for node initialisation |
| *n_args* | 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 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**

| | |
|---|---|
| *node_out* | The allocated node is returned here. |
| *attr* | Attributes. |
| *thread* | The thread the node will be in. |
| *node_spec* | 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 [":" ARGS] ARGS := NAME=VAL [";" ARGS]

Example: "sc_vi_node:interface=eth4;streams=all"

#### 10.16.2.6 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**

| | |
|---|---|
| *node_out* | The allocated node is returned here. |
| *attr* | Attributes. |
| *thread* | The thread the node will be in. |
| *factory_name* | Name of the node factory. |
| *lib_name* | Name of the node library (may be NULL). |
| *args* | An array of arguments for node initialisation. |
| *n_args* | 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 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**

| | |
|---|---|
| *factory_out* | The node factory found. |
| *session* | The SolarCapture session. |
| *factory_name* | Name of the node factory. |
| *lib_name* | 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 struct sc_node* sc_node_from_object ( struct sc_object * *obj* )

Convert an sc_object to an sc_node.

**Parameters**

| | |
|---|---|
| *obj* | An sc_object 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 struct sc_thread* sc_node_get_thread ( const struct sc_node * *node* )

Return the thread associated with a node.

**Parameters**

| | |
|---|---|
| *node* | The node. |

**Returns**

The thread associated with the node.

#### 10.16.2.10 struct sc_object* sc_node_to_object ( struct sc_node * *node* )

Convert an sc_node to an sc_object.

**Parameters**

| | |
|---|---|
| *node* | An sc_node 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 enum sc_object_type

The type of data the sc_object contains.

**Enumerator**

***SC_OBJ_OPAQUE*** An opaque pointer

***SC_OBJ_PKT_PREDICATE*** A packet predicate (see sc_pkt_predicate)

***SC_OBJ_C_ATTR*** Const attributes (see sc_attr)

***SC_OBJ_NODE*** A node (see sc_node)

***SC_OBJ_POOL*** A packet pool

### 10.17.3 Function Documentation

#### 10.17.3.1 enum sc_object_type sc_object_type ( struct sc_object * *obj* )

Return the type of data contained within the sc_object.

**Parameters**

| | |
|---|---|
| *obj* | The object to check the data type of. |

**Returns**

The type of data contained within the sc_object.

#### 10.17.3.2 int sc_opaque_alloc ( struct sc_object ** *obj_out*, void * *opaque* )

Allocate memory for an opaque sc_object.

**Parameters**

| | |
|---|---|
| *obj_out* | On success the allocated object. |
| *opaque* | A pointer to the data to be wrapped by the object. |

**Returns**

0 on success.

#### 10.17.3.3 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**

| | |
|---|---|
| *obj* | The object to free |

#### 10.17.3.4 void* sc_opaque_get_ptr ( const struct sc_object * *obj* )

Get the opaque pointer stored in an sc_object.

**Parameters**

| | |
|---|---|
| *obj* | The object to fetch the opaque pointer from. |

**Returns**

The opaque pointer.

#### 10.17.3.5 void sc_opaque_set_ptr ( struct sc_object * *obj*, void * *opaque* )

Set the opaque pointer in an sc_object.

**Parameters**

| | |
|---|---|
| *obj* | The object to set the pointer on |
| *opaque* | 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.*

### 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 #define SC_PS_FLAG_BAD_FCS 0x4

Mask for sc_packed_packet flags, bad FCS

#### 10.18.2.2 #define SC_PS_FLAG_BAD_L3_CSUM 0x10

Mask for sc_packed_packet flags, bad layer 3 checksum

#### 10.18.2.3 #define SC_PS_FLAG_BAD_L4_CSUM 0x8

Mask for sc_packed_packet flags, bad layer 4 checksum

#### 10.18.2.4 #define SC_PS_FLAG_CLOCK_IN_SYNC 0x2

Mask for sc_packed_packet flags, clock in sync

#### 10.18.2.5 #define SC_PS_FLAG_CLOCK_SET 0x1

Mask for sc_packed_packet flags, clock set

### 10.18.3 Function Documentation

#### 10.18.3.1 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**

| | |
|---|---|
| *ps_pkt* | A packed-stream packet header |

**Returns**

The next packed-stream packet in the buffer.

#### 10.18.3.2 static void* sc_packed_packet_payload ( const struct sc_packed_packet * *ps_pkt* ) [inline], [static]

Return a pointer to the packet payload.

**Parameters**

| | |
|---|---|
| *ps_pkt* | A packed-stream packet header |

**Returns**

The start of the packet payload.

#### 10.18.3.3 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**

| | |
|---|---|
| *pkt* | 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 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**

| | |
|---|---|
| *pkt* | 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 uint16_t ps_cap_len

Number of bytes of packet payload stored.

#### 10.18.4.2 uint8_t ps_flags

SC_PS_FLAG_∗ flags.

#### 10.18.4.3 uint16_t ps_next_offset

Offset of next packet from start of this struct.

#### 10.18.4.4 uint16_t ps_orig_len

Original length of the frame.

#### 10.18.4.5 uint8_t ps_pkt_start_offset

Offset of packet payload from start of this struct.

#### 10.18.4.6 uint32_t ps_ts_nsec

Timestamp (nanoseconds).

#### 10.18.4.7 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 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**

| | |
|---|---|
| *packet* | The packet to append data to |
| *pool* | Packet pool to allocate frag buffers from (optional) |
| *iovp* | Identifies the data to copy in |
| *snap* | 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 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**

| | |
|---|---|
| *pool* | The pool to allocate buffers from |
| *psp* | The packed-stream packet to duplicate |
| *snap* | The maximum number of bytes to copy |

**Returns**

The duplicated packet or NULL if insufficient buffers.

#### 10.19.2.3 struct sc_packet* sc_pool_duplicate_packet ( struct sc_pool * *pool*, struct sc_packet * *packet*, int *snap* )

Duplicate a packet.

**Parameters**

| | |
|---|---|
| *pool* | The pool to allocate buffers from |
| *packet* | The packet to duplicate |
| *snap* | The maximum number of bytes to copy |

**Returns**

The duplicated packet or NULL if insufficient buffers.

#### 10.19.2.4 struct sc_pool* sc_pool_from_object ( struct sc_object * *obj* )

Convert an sc_object to an sc_pool.

**Parameters**

| | |
|---|---|
| *obj* | 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 uint64_t sc_pool_get_buffer_size ( struct sc_pool * *pool* )

Get the minimum buffer size provided by this pool.

**Parameters**

| | |
|---|---|
| *pool* | 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 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**

| | |
|---|---|
| *list* | List where retrieved packets are placed |
| *pool* | The packet pool |
| *min_packets* | Minimum number of buffers to be returned |
| *max_packets* | 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 void sc_pool_on_threshold ( struct sc_pool * *pool*, struct sc_callback * *event*, int *threshold* )

Request a callback when the pool is refilled.

**Parameters**

| | |
|---|---|
| *pool* | The packet pool |
| *event* | The event object |
| *threshold* | Event fires when pool has >= 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 void sc_pool_return_packets ( struct sc_pool * *pool*, struct sc_packet_list * *list* )

Return packets to a pool.

  

**Parameters**

| | |
|---|---|
| *pool* | The packet pool |
| *list* | 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 struct sc_node* sc_pool_set_refill_node ( struct sc_pool * *pool*, struct sc_node * *node* )

Set the refill node for a pool.

**Parameters**

| | |
|---|---|
| *pool* | A packet pool |
| *node* | 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 struct sc_object* sc_pool_to_object ( struct sc_pool * *pool* )

Convert an sc_pool to an sc_object.

**Parameters**

| | |
|---|---|
| *pool* | An sc_pool instance or NULL |

**Returns**

NULL if `pool` is NULL otherwise the sc_object.

#### 10.19.2.11 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**

| | |
|---|---|
| *pool* | A packet pool |

| | |
|---|---|
| *node* | 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 int sc_pkt_predicate_alloc ( struct sc_pkt_predicate ** *pred_out*, int *private_bytes* )

Allocate a packet predicate object.

  

**Parameters**

| | |
|---|---|
| *pred_out* | On success the allocated sc_pkt_predicate object. |
| *private_bytes* | 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 struct sc_pkt_predicate* sc_pkt_predicate_from_object ( struct sc_object * *obj* )

Convert a sc_object into a sc_pkt_predicate.

**Parameters**

| | |
|---|---|
| *obj* | An sc_object instance or NULL |

**Returns**

NULL if `obj` is NULL otherwise the converted sc_pkt_predicate.

#### 10.20.2.3 struct sc_object* sc_pkt_predicate_to_object ( struct sc_pkt_predicate * *pred* )

Convert a sc_pkt_predicate into a sc_object.

**Parameters**

| | |
|---|---|
| *pred* | An sc_pkt_predicate 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_prepare (struct sc_session *scs)
  *Prepare a SolarCapture session.*
- int sc_session_go (struct sc_session *scs)
  *Start a SolarCapture session.*
- int sc_session_pause (struct sc_session *scs)
  *Pause a SolarCapture session.*
- 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 int sc_session_alloc ( struct sc_session ** *scs_out,* const struct sc_attr * *attr* )

Allocate a SolarCapture session.

**Parameters**

| | |
|---|---|
| *scs_out* | The allocated session object is returned here |
| *attr* | 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 void sc_session_error_free ( struct sc_session * *scs,* struct sc_session_error * *err* )

Frees an error object.

**Parameters**

| | |
|---|---|
| *scs* | The session |
| *err* | The error |

Frees a sc_session_error pointer returned by sc_session_error_get.

#### 10.21.2.3 struct sc_session_error* sc_session_error_get ( struct sc_session * *scs* )

Returns an error from a SolarCapture session.

**Parameters**

| | |
|---|---|
| *scs* | 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 once done with it.
If no error has occurred, this function returns NULL.

#### 10.21.2.4 int sc_session_go ( struct sc_session * *scs* )

Start a SolarCapture session.

**Parameters**

| | |
|---|---|
| *scs* | 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.

#### 10.21.2.5 int sc_session_pause ( struct sc_session * *scs* )

Pause a SolarCapture session.

**Parameters**

| | |
|---|---|
| *scs* | The session |

**Returns**

0 on success, or a negative error code.

Pause the threads managed by session `scs`.

#### 10.21.2.6 int sc_session_prepare ( struct sc_session * *scs* )

Prepare a SolarCapture session.

**Parameters**

| | |
|---|---|
| *scs* | 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.

## 10.22 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 int sc_stream_all ( struct sc_stream * *stream* )

Configure stream to match packets not explicitly steered elsewhere.

**Parameters**

| | |
|---|---|
| *stream* | 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 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**

| | |
|---|---|
| *stream_out* | On success, the created stream. |
| *attr* | Attributes to pass in. |
| *scs* | The session this stream is for. |

**Returns**

0 on success, or a negative error code.

#### 10.22.2.3 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**

| | |
|---|---|
| *stream* | A stream object. |
| *mac_addr* | The destination MAC address to match. |

**Returns**

0 on success, or a negative error code.

#### 10.22.2.4 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**

| | |
|---|---|
| *stream* | A stream object. |
| *mac_addr* | The source MAC address to match. |

**Returns**

0 on success, or a negative error code.

#### 10.22.2.5 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**

| | |
|---|---|
| *stream* | A stream object. |
| *eth_type* | The ether_type to match (host endian). |

**Returns**

0 on success, or a negative error code.

#### 10.22.2.6 int sc_stream_eth_vlan_id ( struct sc_stream * *stream*, int *vlan_id* )

Add the VLAN ID to the set of fields matched.

**Parameters**

| | |
|---|---|
| *stream* | A stream object. |
| *vlan_id* | The VLAN ID to match against. |

**Returns**

0 on success, or a negative error code.

#### 10.22.2.7 int sc_stream_free ( struct sc_stream * *stream* )

Free a previously created stream.

  

**Parameters**

| | |
|---|---|
| *stream* | The stream to free. |

**Returns**

0, always.

#### 10.22.2.8 int sc_stream_ip_dest_host ( struct sc_stream * *stream*, const char * *dhost* )

Add the IPv4 destination to the set of fields matched.

**Parameters**

| | |
|---|---|
| *stream* | A stream object. |
| *dhost* | The destination host name or IP to match. |

**Returns**

0 on success, or a negative error code.

#### 10.22.2.9 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**

| | |
|---|---|
| *stream* | A stream object. |
| *protocol* | The transport layer protocol to match against. |
| *dhost* | The destination hostname to match against. |
| *dport* | The destination port to match against. |

**Returns**

0 on success, or a negative error code.

#### 10.22.2.10 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**

| | |
|---|---|
| *stream* | A stream object. |

| | |
|---|---|
| *dport* | The destination port to match. |

**Returns**

0 on success, or a negative error code.

#### 10.22.2.11 int sc_stream_ip_protocol ( struct sc_stream * *stream*, int *protocol* )

Add the IP protocol to the set of fields matched.

**Parameters**

| | |
|---|---|
| *stream* | A stream object. |
| *protocol* | The IP protocol to match. |

**Returns**

0 on success, or a negative error code.

#### 10.22.2.12 int sc_stream_ip_source_host ( struct sc_stream * *stream*, const char * *shost* )

Add the IPv4 source to the set of fields matched.

**Parameters**

| | |
|---|---|
| *stream* | A stream object. |
| *shost* | The source host name or IP to match. |

**Returns**

0 on success, or a negative error code.

#### 10.22.2.13 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**

| | |
|---|---|
| *stream* | A stream object. |
| *shost* | The source hostname to match against. |
| *sport* | The source port to match against. |

**Returns**

0 on success, or a negative error code.

#### 10.22.2.14 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**

| | |
|---|---|
| *stream* | A stream object. |
| *sport* | The source port to match. |

**Returns**

0 on success, or a negative error code.

#### 10.22.2.15 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**

| | |
|---|---|
| *stream* | 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 int sc_stream_reset ( struct sc_stream * *stream* )

Reinitialise a stream.

**Parameters**

| | |
|---|---|
| *stream* | The stream to reinitialise. |

**Returns**

0 on success, or a negative error code.

#### 10.22.2.17 int sc_stream_set_str ( struct sc_stream * *stream,* const char * *str* )

Set the stream to match packets identified by a string.

**Parameters**

| | |
|---|---|
| *stream* | A stream object. |
| *str* | 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 |
|---|---|
| eth:<dest-mac> | FF ULL CPS |
| eth:vid=<vlan>,<dest-mac> | FF ULL |
| {udp\|tcp}:<dest-host>:<dest-port> | FF ULL CPS |
| {udp\|tcp}:<dest-host>:<dest-port>,<source-host>-:<source-port> | FF ULL |
| {udp\|tcp}:vid=<vlan>,<dest-host>:<dest-port> | FF |
| {udp\|tcp}:vid=<vlan>,<dest-host>:<dest-port>,<source-host>:<source-port> | 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 |
|---|---|
| dmac=xx:xx:xx:xx:xx:xx | Match Ethernet destination MAC address. |
| smac=xx:xx:xx:xx:xx:xx | Match Ethernet source MAC addres. |
| vid=INT | Match Ethernet outer VLAN ID. |
| eth_type=ip\|arp\|INT | Match Ethernet ether_type. |
| shost=hostname | Match IPv4 source host. |
| dhost=hostname | Match IPv4 destination host. |
| ip_protocol=udp\|tcp\|INT | Match IPv4 protocol (implies eth_type=ip). |
| sport=INT | Match TCP or UDP source port. |
| dport=INT | Match TCP or UDP destination port. |
| all | All packets not steered elsewhere. |
| mismatch | All packets not steered elsewhere and not requested by the kernel network stack. |
| ip | Shorthand for eth_type=ip. |
| tcp | Shorthand for ip_protocol=tcp. |
| udp | Shorthand for ip_protocol=udp. |

IPv4 addresses may be given as a dotted quad or a host name that can resolved with getaddrinfo().

Supported combinations of keys are shown in the table below, together with the firmware variants required:

| Key combination | SFN7xxx |
|---|---|
| all | FF ULL CPS |
| mismatch | FF ULL CPS |
| vid | FF CPS |
| dmac | FF ULL CPS |
| dmac, vid | FF ULL |
| ip_protocol, dhost, dport | FF ULL CPS |
| ip_protocol, dhost, dport, shost, sport | FF ULL |
| [vid,] [dmac,] ip_protocol, dhost, dport | FF |
| [vid,] [dmac,] ip_protocol, dhost, dport, shost, sport | FF |
| eth_type | FF ULL |
| eth_type, vid | FF ULL CPS |
| eth_type, dmac | FF ULL |
| ip_protocol | FF ULL |
| ip_protocol, vid | FF ULL CPS |
| [vid,] ip_protocol, dmac | 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 int sc_thread_alloc ( struct sc_thread ** *thread_out,* const struct sc_attr * *attr,* struct sc_session * *scs* )

Allocate a SolarCapture thread.

  

**Parameters**

| | |
|---|---|
| *thread_out* | The allocated thread object is returned here |
| *attr* | Attributes |
| *scs* | 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 void* sc_thread_calloc ( struct sc_thread * *thread,* size_t *bytes* )

Allocate memory to be used by a thread.

**Parameters**

| | |
|---|---|
| *thread* | The thread |
| *bytes* | 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 void* sc_thread_calloc_aligned ( struct sc_thread * *thread,* size_t *bytes,* int *align* )

Allocate memory to be used by a thread.

**Parameters**

| | |
|---|---|
| *thread* | The thread |
| *bytes* | Size of memory area wanted |
| *align* | 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 void sc_thread_get_time ( const struct sc_thread * *thread,* struct timespec * *time_out* )

Return a thread's "current time".

**Parameters**

| | |
|---|---|
| *thread* | The thread |
| *time_out* | 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 void sc_thread_mfree ( struct sc_thread * *thread,* void * *mem* )

Free memory.

**Parameters**

| | |
|---|---|
| *thread* | The thread |
| *mem* | The memory area |

Use this function to free memory allocated with sc_thread_calloc or sc_thread_calloc_aligned.

#### 10.24.2.6 int sc_thread_poll ( struct sc_thread * *thread* )

Poll a thread.

**Parameters**

| | |
|---|---|
| *thread* | 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 int sc_thread_poll_timers ( struct sc_thread * *thread* )

Poll a thread's timers.

  

**Parameters**

| | |
|---|---|
| *thread* | 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 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 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 static uint64_t sc_ns_from_ms ( uint64_t *ms* ) [inline], [static]

Convert milliseconds to nanoseconds.

**Parameters**

| | |
|---|---|
| *ms* | The time in milliseconds to convert |

**Returns**

Time in nanoseconds

#### 10.25.2.2 static uint64_t sc_ns_from_ts ( const struct timespec * *ts* ) [inline], [static]

Convert a timespec struct to nanoseconds.

  

**Parameters**

| | |
|---|---|
| *ts* | The timespec struct to convert |

**Returns**

Time in nanoseconds

#### 10.25.2.3 static uint64_t sc_ns_from_tv ( const struct timeval * *tv* ) [inline], [static]

Convert a timeval struct to nanoseconds.

**Parameters**

| | |
|---|---|
| *tv* | The timeval struct to convert |

**Returns**

Time in nanoseconds

#### 10.25.2.4 static uint64_t sc_ns_from_us ( uint64_t *us* ) [inline], [static]

Convert microseconds to nanoseconds.

**Parameters**

| | |
|---|---|
| *us* | The time in microseconds to convert |

**Returns**

Time in nanoseconds

#### 10.25.2.5 void sc_timer_expire_after_ns ( struct sc_callback * *cb,* int64_t *delta_ns* )

Request a callback in the future.

**Parameters**

| | |
|---|---|
| *cb* | A callback object |
| *delta_ns* | 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 void sc_timer_expire_at ( struct sc_callback * *cb,* const struct timespec * *time* )

Request a callback at a given time.

**Parameters**

| | |
|---|---|
| *cb* | A callback object |
| *time* | 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 int sc_timer_get_expiry_time ( const struct sc_callback * *cb*, struct timespec * *ts_out* )

Return the expiry time of a timer callback.

**Parameters**

| | |
|---|---|
| *cb* | A callback object |
| *ts_out* | The expiry time is returned here |

**Returns**

Zero if `cb` is a timer else -1

#### 10.25.2.8 void sc_timer_push_back_ns ( struct sc_callback * *cb*, int64_t *delta_ns* )

Push the expiry time further into the future.

**Parameters**

| | |
|---|---|
| *cb* | A callback object |
| *delta_ns* | 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 int sc_vi_add_stream ( struct sc_vi * *vi,* struct sc_stream * *stream* )

Direct a packet stream to a VI.

**Parameters**

| | |
|---|---|
| *vi* | The VI receiving packets |
| *stream* | 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 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**

| | |
|---|---|
| *vi_out* | The allocated VI is returned here |
| *attr* | Attributes |
| *thread* | The thread the VI will be in |
| *interface* | 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 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**

| | |
|---|---|
| *vi_out* | The allocated VI is returned here |
| *attr* | Attributes |
| *thread* | The thread the VI will be in |
| *vi_group* | 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 const char* sc_vi_get_interface_name ( const struct sc_vi * *vi* )

Return the name of the network interface associated with a VI.

**Parameters**

| | |
|---|---|
| *vi* | 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 struct sc_thread* sc_vi_get_thread ( const struct sc_vi * *vi* )

Return the thread associated with a VI.

  

**Parameters**

| | |
|---|---|
| *vi* | The VI |

**Returns**

The thread associated with the VI.

#### 10.26.2.6 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**

| | |
|---|---|
| *vi_group* | The VI group receiving packets |
| *stream* | 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 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**

| | |
|---|---|
| *vi_out* | The allocated VI is returned here |
| *attr* | Attributes |
| *session* | The SolarCapture session |
| *interface* | The network interface to receive packets from |
| *num_vis* | 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 struct sc_session* sc_vi_group_get_session ( const struct sc_vi_group * *vi_group* )

Return the session associated with a VI group.

**Parameters**

| | |
|---|---|
| *vi_group* | The VI group |

**Returns**

The session associated with the VI group.

#### 10.26.2.9 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.

**Parameters**

| | |
|---|---|
| *vi* | The VI receiving packets |
| *node* | The node to deliver packets to |
| *name_opt* | 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, 69
append_to_list.h, 91
args.h
    SC_PARAM_DBL, 92
    SC_PARAM_INT, 92
    SC_PARAM_OBJ, 92
    SC_PARAM_STR, 92
args.h, 91
    SC_ARG_DBL, 92
    SC_ARG_INT, 92
    SC_ARG_OBJ, 93
    SC_ARG_STR, 93
    sc_param_type, 92
attr.h, 93
    sc_attr_alloc, 94
    sc_attr_doc, 94
    sc_attr_dup, 95
    sc_attr_free, 95
    sc_attr_from_object, 95
    sc_attr_reset, 95
    sc_attr_set_from_fmt, 96
    sc_attr_set_from_str, 96
    sc_attr_set_int, 96
    sc_attr_set_str, 97
    sc_attr_to_object, 97

cb_handler_fn
    sc_callback, 72
cb_link
    sc_callback, 72
cb_private
    sc_callback, 72

declare_types.h, 97
    ST_CONSTANT, 99
    ST_FIELD, 99
    ST_FIELD_STR, 99
    ST_STRUCT, 99
dlist.h, 100
    SC_CONTAINER, 101
    sc_dlist_init, 103
    sc_dlist_pop_head, 103
    sc_dlist_pop_tail, 103
    sc_dlist_push_head, 104
    sc_dlist_push_tail, 104
    sc_dlist_rehome, 104
    sc_dlist_remove, 104

err_errno
    sc_session_error, 84
err_file
    sc_session_error, 84
err_func
    sc_session_error, 84
err_line
    sc_session_error, 85
err_msg
    sc_session_error, 85
ethernet.h, 105
    SC_8021Q_VID_MASK, 105
    SC_ETHERTYPE_8021Q, 105
    SC_ETHERTYPE_8021QinQ, 105
event.h, 105
    sc_callback_alloc, 107
    sc_callback_alloc2, 107
    sc_callback_free, 107
    sc_callback_handler_fn, 106
    sc_callback_is_active, 108
    sc_callback_on_idle, 108
    sc_callback_remove, 108
    sc_callback_set_description, 108
    sc_epoll_ctl, 108
ext_node.h, 109
    sc_forward, 114
    sc_forward2, 115
    sc_forward_list, 115
    sc_forward_list2, 115
    sc_node_add_link_fn, 112
    sc_node_end_of_stream_fn, 112
    sc_node_export_state, 115
    sc_node_fwd_error, 111
    sc_node_init_fn, 113
    sc_node_init_get_arg_dbl, 116
    sc_node_init_get_arg_int, 116
    sc_node_init_get_arg_int64, 117
    sc_node_init_get_arg_obj, 117
    sc_node_init_get_arg_str, 118
    sc_node_link_end_of_stream, 118
    sc_node_link_end_of_stream2, 118
    sc_node_pkts_fn, 113
    sc_node_prep_check_links, 119
    sc_node_prep_does_not_forward, 119
    sc_node_prep_fn, 113
    sc_node_prep_get_link, 119
    sc_node_prep_get_link_or_free, 119

sc_node_prep_get_pool, 120
sc_node_prep_link_forwards_from_node, 120
sc_node_select_subnode_fn, 114
sc_node_set_error, 111
sc_node_set_errorv, 112
sc_node_type_alloc, 121
ext_packet.h, 121
SC_MEMBER_OFFSET, 122
SC_MEMBER_SIZE, 123
sc_packet_bytes, 123
sc_packet_frags_tail, 123
sc_packet_prefetch_r, 123
sc_packet_prefetch_rw, 124
sc_packet_timespec, 124
ext_packet_list.h, 124
sc_packet_list_append, 125
sc_packet_list_append_list, 125
sc_packet_list_finalise, 125
sc_packet_list_init, 125
sc_packet_list_is_empty, 126
sc_packet_list_pop_head, 126
sc_packet_list_push_head, 126
sc_packet_list_tail, 126

flags
sc_packet, 81
frags
sc_packet, 81
frags_n
sc_packet, 81
frags_tail
sc_packet, 81
frame_len
sc_packet, 81
free_link
sc_append_to_list, 69

hash_table.h, 127
sc_hash_table_alloc, 127
sc_hash_table_clear, 128
sc_hash_table_del, 128
sc_hash_table_del_val, 128
sc_hash_table_free, 128
sc_hash_table_get, 129
sc_hash_table_get_next_entry, 129
sc_hash_table_grow, 129
sc_hash_table_key_size, 130
sc_hash_table_num_entries, 130
sc_hash_table_val_size, 130
sc_hash_table_val_to_key, 130
head
sc_packet_list, 83

io
sc_iovec_ptr, 74
iov
sc_iovec_ptr, 74
sc_packet, 81
iovec.h, 131
sc_iovec_copy_from_end, 132
sc_iovec_ptr_bytes, 133
sc_iovec_ptr_copy_out, 133
sc_iovec_ptr_find_chr, 133
sc_iovec_ptr_init, 133
sc_iovec_ptr_init_buf, 134
sc_iovec_ptr_init_packet, 134
sc_iovec_ptr_skip, 134
sc_iovec_trim_end, 134
iovlen
sc_iovec_ptr, 74
sc_packet, 81
ip.h, 135
SC_IP4_FRAG_DONT, 135
SC_IP4_FRAG_MORE, 135
SC_IP4_OFFSET_MASK, 135
SC_TCP_ACK, 135
SC_TCP_FIN, 136
SC_TCP_PSH, 136
SC_TCP_RST, 136
SC_TCP_SYN, 136
SC_TCP_URG, 136

links
sc_append_to_list, 69

mailbox.h, 136
sc_mailbox_alloc, 137
sc_mailbox_connect, 137
sc_mailbox_get_send_node, 137
sc_mailbox_poll, 138
sc_mailbox_send, 138
sc_mailbox_send_list, 138
sc_mailbox_set_recv, 138
metadata
sc_packet, 81
misc.h, 139
sc_join_mcast_group, 139

n_links
sc_append_to_list, 70
name
sc_arg, 70
sc_node_link, 77
nd_name
sc_node, 75
nd_private
sc_node, 75
nd_type
sc_node, 75
next
sc_dlist, 73
sc_packet, 82
nf_init_fn
sc_node_factory, 76

nf_name
sc_node_factory, 76
nf_node_api_ver
sc_node_factory, 76
nf_private
sc_node_factory, 76
nf_reserved
sc_node_factory, 76
nf_source_file
sc_node_factory, 76
node.h, 140
sc_node_add_info_int, 140
sc_node_add_info_str, 141
sc_node_add_link, 141
sc_node_alloc, 141
sc_node_alloc_from_str, 142
sc_node_alloc_named, 142
sc_node_factory_lookup, 143
sc_node_from_object, 144
sc_node_get_thread, 144
sc_node_to_object, 145
nt_add_link_fn
sc_node_type, 77
nt_end_of_stream_fn
sc_node_type, 77
nt_name
sc_node_type, 78
nt_pkts_fn
sc_node_type, 78
nt_prep_fn
sc_node_type, 78
nt_private
sc_node_type, 78
nt_select_subnode_fn
sc_node_type, 78
num_frags
sc_packet_list, 83
num_pkts
sc_packet_list, 83

object.h
SC_OBJ_C_ATTR, 146
SC_OBJ_NODE, 146
SC_OBJ_OPAQUE, 146
SC_OBJ_PKT_PREDICATE, 146
SC_OBJ_POOL, 146
object.h, 145
sc_object_type, 146
sc_opaque_alloc, 146
sc_opaque_free, 146
sc_opaque_get_ptr, 147
sc_opaque_set_ptr, 147

packed_stream.h, 147
ps_cap_len, 151
ps_flags, 151
ps_next_offset, 151
ps_orig_len, 151
ps_pkt_start_offset, 151
ps_ts_nsec, 151
ps_ts_sec, 151
sc_packed_packet_next, 149
sc_packed_packet_payload, 150
sc_packet_packed_end, 150
sc_packet_packed_first, 150
pkt_pool.h, 151
sc_packet_append_iovec_ptr, 152
sc_pool_duplicate_packed_packet, 152
sc_pool_duplicate_packet, 154
sc_pool_from_object, 154
sc_pool_get_buffer_size, 154
sc_pool_get_packets, 155
sc_pool_on_threshold, 155
sc_pool_return_packets, 155
sc_pool_set_refill_node, 156
sc_pool_to_object, 156
sc_pool_wraps_node, 156
pred_private
sc_pkt_predicate, 84
pred_test_fn
sc_pkt_predicate, 84
predicate.h, 157
sc_pkt_predicate_alloc, 157
sc_pkt_predicate_from_object, 158
sc_pkt_predicate_to_object, 158
prev
sc_dlist, 73
ps_cap_len
packed_stream.h, 151
sc_packed_packet, 79
ps_flags
packed_stream.h, 151
sc_packed_packet, 79
ps_next_offset
packed_stream.h, 151
sc_packed_packet, 79
ps_orig_len
packed_stream.h, 151
sc_packed_packet, 80
ps_pkt_start_offset
packed_stream.h, 151
sc_packed_packet, 80
ps_ts_nsec
packed_stream.h, 151
sc_packed_packet, 80
ps_ts_sec
packed_stream.h, 151
sc_packed_packet, 80

reserved1
sc_packet, 82
reserved2
sc_packet, 82

  

SC_OBJ_C_ATTR
object.h, 146
SC_OBJ_NODE
object.h, 146
SC_OBJ_OPAQUE
object.h, 146
SC_OBJ_PKT_PREDICATE
object.h, 146
SC_OBJ_POOL
object.h, 146
SC_PARAM_DBL
args.h, 92
SC_PARAM_INT
args.h, 92
SC_PARAM_OBJ
args.h, 92
SC_PARAM_STR
args.h, 92
SC_8021Q_VID_MASK
ethernet.h, 105
SC_ARG_DBL
args.h, 92
SC_ARG_INT
args.h, 92
SC_ARG_OBJ
args.h, 93
SC_ARG_STR
args.h, 93
SC_CONTAINER
dlist.h, 101
SC_ETHERTYPE_8021Q
ethernet.h, 105
SC_ETHERTYPE_8021QinQ
ethernet.h, 105
SC_IP4_FRAG_DONT
ip.h, 135
SC_IP4_FRAG_MORE
ip.h, 135
SC_IP4_OFFSET_MASK
ip.h, 135
SC_MEMBER_OFFSET
ext_packet.h, 122
SC_MEMBER_SIZE
ext_packet.h, 123
SC_TCP_ACK
ip.h, 135
SC_TCP_FIN
ip.h, 136
SC_TCP_PSH
ip.h, 136
SC_TCP_RST
ip.h, 136
SC_TCP_SYN
ip.h, 136
SC_TCP_URG
ip.h, 136
ST_CONSTANT
declare_types.h, 99
ST_FIELD
declare_types.h, 99
ST_FIELD_STR
declare_types.h, 99
ST_STRUCT
declare_types.h, 99
sc_append_to_list, 69
append_to, 69
free_link, 69
links, 69
n_links, 70
sc_arg, 70
name, 70
type, 70
val, 70
sc_attr, 71
sc_attr_alloc
attr.h, 94
sc_attr_doc
attr.h, 94
sc_attr_dup
attr.h, 95
sc_attr_free
attr.h, 95
sc_attr_from_object
attr.h, 95
sc_attr_reset
attr.h, 95
sc_attr_set_from_fmt
attr.h, 96
sc_attr_set_from_str
attr.h, 96
sc_attr_set_int
attr.h, 96
sc_attr_set_str
attr.h, 97
sc_attr_to_object
attr.h, 97
sc_callback, 71
cb_handler_fn, 72
cb_link, 72
cb_private, 72
sc_callback_alloc
event.h, 107
sc_callback_alloc2
event.h, 107
sc_callback_free
event.h, 107
sc_callback_handler_fn
event.h, 106
sc_callback_is_active
event.h, 108
sc_callback_on_idle
event.h, 108

sc_callback_remove
event.h, 108
sc_callback_set_description
event.h, 108
sc_dlist, 72
next, 73
prev, 73
sc_dlist_init
dlist.h, 103
sc_dlist_pop_head
dlist.h, 103
sc_dlist_pop_tail
dlist.h, 103
sc_dlist_push_head
dlist.h, 104
sc_dlist_push_tail
dlist.h, 104
sc_dlist_rehome
dlist.h, 104
sc_dlist_remove
dlist.h, 104
sc_epoll_ctl
event.h, 108
sc_forward
ext_node.h, 114
sc_forward2
ext_node.h, 115
sc_forward_list
ext_node.h, 115
sc_forward_list2
ext_node.h, 115
sc_hash_table, 73
sc_hash_table_alloc
hash_table.h, 127
sc_hash_table_clear
hash_table.h, 128
sc_hash_table_del
hash_table.h, 128
sc_hash_table_del_val
hash_table.h, 128
sc_hash_table_free
hash_table.h, 128
sc_hash_table_get
hash_table.h, 129
sc_hash_table_get_next_entry
hash_table.h, 129
sc_hash_table_grow
hash_table.h, 129
sc_hash_table_key_size
hash_table.h, 130
sc_hash_table_num_entries
hash_table.h, 130
sc_hash_table_val_size
hash_table.h, 130
sc_hash_table_val_to_key
hash_table.h, 130
sc_iovec_copy_from_end
iovec.h, 132
sc_iovec_ptr, 73
io, 74
iov, 74
iovlen, 74
sc_iovec_ptr_bytes
iovec.h, 133
sc_iovec_ptr_copy_out
iovec.h, 133
sc_iovec_ptr_find_chr
iovec.h, 133
sc_iovec_ptr_init
iovec.h, 133
sc_iovec_ptr_init_buf
iovec.h, 134
sc_iovec_ptr_init_packet
iovec.h, 134
sc_iovec_ptr_skip
iovec.h, 134
sc_iovec_trim_end
iovec.h, 134
sc_join_mcast_group
misc.h, 139
sc_mailbox_alloc
mailbox.h, 137
sc_mailbox_connect
mailbox.h, 137
sc_mailbox_get_send_node
mailbox.h, 137
sc_mailbox_poll
mailbox.h, 138
sc_mailbox_send
mailbox.h, 138
sc_mailbox_send_list
mailbox.h, 138
sc_mailbox_set_recv
mailbox.h, 138
sc_node, 74
nd_name, 75
nd_private, 75
nd_type, 75
sc_node_add_info_int
node.h, 140
sc_node_add_info_str
node.h, 141
sc_node_add_link
node.h, 141
sc_node_add_link_fn
ext_node.h, 112
sc_node_alloc
node.h, 141
sc_node_alloc_from_str
node.h, 142
sc_node_alloc_named
node.h, 142

  

sc_node_end_of_stream_fn
ext_node.h, 112
sc_node_export_state
ext_node.h, 115
sc_node_factory, 75
nf_init_fn, 76
nf_name, 76
nf_node_api_ver, 76
nf_private, 76
nf_reserved, 76
nf_source_file, 76
sc_node_factory_lookup
node.h, 143
sc_node_from_object
node.h, 144
sc_node_fwd_error
ext_node.h, 111
sc_node_get_thread
node.h, 144
sc_node_init_fn
ext_node.h, 113
sc_node_init_get_arg_dbl
ext_node.h, 116
sc_node_init_get_arg_int
ext_node.h, 116
sc_node_init_get_arg_int64
ext_node.h, 117
sc_node_init_get_arg_obj
ext_node.h, 117
sc_node_init_get_arg_str
ext_node.h, 118
sc_node_link, 76
name, 77
sc_node_link_end_of_stream
ext_node.h, 118
sc_node_link_end_of_stream2
ext_node.h, 118
sc_node_pkts_fn
ext_node.h, 113
sc_node_prep_check_links
ext_node.h, 119
sc_node_prep_does_not_forward
ext_node.h, 119
sc_node_prep_fn
ext_node.h, 113
sc_node_prep_get_link
ext_node.h, 119
sc_node_prep_get_link_or_free
ext_node.h, 119
sc_node_prep_get_pool
ext_node.h, 120
sc_node_prep_link_forwards_from_node
ext_node.h, 120
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, 145
sc_node_type, 77
nt_add_link_fn, 77
nt_end_of_stream_fn, 77
nt_name, 78
nt_pkts_fn, 78
nt_prep_fn, 78
nt_private, 78
nt_select_subnode_fn, 78
sc_node_type_alloc
ext_node.h, 121
sc_ns_from_ms
time.h, 173
sc_ns_from_ts
time.h, 173
sc_ns_from_tv
time.h, 174
sc_ns_from_us
time.h, 174
sc_object, 78
sc_object_type
object.h, 146
sc_opaque_alloc
object.h, 146
sc_opaque_free
object.h, 146
sc_opaque_get_ptr
object.h, 147
sc_opaque_set_ptr
object.h, 147
sc_packed_packet, 79
ps_cap_len, 79
ps_flags, 79
ps_next_offset, 79
ps_orig_len, 80
ps_pkt_start_offset, 80
ps_ts_nsec, 80
ps_ts_sec, 80
sc_packed_packet_next
packed_stream.h, 149
sc_packed_packet_payload
packed_stream.h, 150
sc_packet, 80
flags, 81
frags, 81
frags_n, 81
frags_tail, 81
frame_len, 81
iov, 81
iovlen, 81
metadata, 81
next, 82

reserved1, 82
reserved2, 82
ts_nsec, 82
ts_sec, 82
sc_packet_append_iovec_ptr
pkt_pool.h, 152
sc_packet_bytes
ext_packet.h, 123
sc_packet_frags_tail
ext_packet.h, 123
sc_packet_list, 82
head, 83
num_frags, 83
num_pkts, 83
tail, 83
sc_packet_list_append
ext_packet_list.h, 125
sc_packet_list_append_list
ext_packet_list.h, 125
sc_packet_list_finalise
ext_packet_list.h, 125
sc_packet_list_init
ext_packet_list.h, 125
sc_packet_list_is_empty
ext_packet_list.h, 126
sc_packet_list_pop_head
ext_packet_list.h, 126
sc_packet_list_push_head
ext_packet_list.h, 126
sc_packet_list_tail
ext_packet_list.h, 126
sc_packet_packed_end
packed_stream.h, 150
sc_packet_packed_first
packed_stream.h, 150
sc_packet_prefetch_r
ext_packet.h, 123
sc_packet_prefetch_rw
ext_packet.h, 124
sc_packet_timespec
ext_packet.h, 124
sc_param_type
args.h, 92
sc_pkt_predicate, 83
pred_private, 84
pred_test_fn, 84
sc_pkt_predicate_alloc
predicate.h, 157
sc_pkt_predicate_from_object
predicate.h, 158
sc_pkt_predicate_to_object
predicate.h, 158
sc_pool_duplicate_packed_packet
pkt_pool.h, 152
sc_pool_duplicate_packet
pkt_pool.h, 154
sc_pool_from_object
pkt_pool.h, 154
sc_pool_get_buffer_size
pkt_pool.h, 154
sc_pool_get_packets
pkt_pool.h, 155
sc_pool_on_threshold
pkt_pool.h, 155
sc_pool_return_packets
pkt_pool.h, 155
sc_pool_set_refill_node
pkt_pool.h, 156
sc_pool_to_object
pkt_pool.h, 156
sc_pool_wraps_node
pkt_pool.h, 156
sc_session_alloc
session.h, 159
sc_session_error, 84
err_errno, 84
err_file, 84
err_func, 84
err_line, 85
err_msg, 85
sc_session_error_free
session.h, 159
sc_session_error_get
session.h, 160
sc_session_go
session.h, 160
sc_session_pause
session.h, 160
sc_session_prepare
session.h, 160
sc_sh_handle_backlog_fn
sc_subnode_helper, 86
sc_sh_handle_end_of_stream_fn
sc_subnode_helper, 87
sc_stream, 85
sc_stream_all
stream.h, 162
sc_stream_alloc
stream.h, 162
sc_stream_eth_dhost
stream.h, 162
sc_stream_eth_shost
stream.h, 163
sc_stream_eth_type
stream.h, 163
sc_stream_eth_vlan_id
stream.h, 163
sc_stream_free
stream.h, 163
sc_stream_ip_dest_host
stream.h, 164
sc_stream_ip_dest_hostport

  

stream.h, 164
sc_stream_ip_dest_port
stream.h, 164
sc_stream_ip_protocol
stream.h, 165
sc_stream_ip_source_host
stream.h, 165
sc_stream_ip_source_hostport
stream.h, 165
sc_stream_ip_source_port
stream.h, 165
sc_stream_mismatch
stream.h, 166
sc_stream_reset
stream.h, 166
sc_stream_set_str
stream.h, 166
sc_subnode_helper, 86
sc_sh_handle_backlog_fn, 86
sc_sh_handle_end_of_stream_fn, 87
sc_subnode_helper_from_node, 87
sc_subnode_helper_request_callback, 87
sh_backlog, 88
sh_free_link, 88
sh_handle_backlog_fn, 88
sh_handle_end_of_stream_fn, 88
sh_links, 88
sh_n_links, 88
sh_node, 88
sh_poll_backlog_ns, 88
sh_pool, 88
sh_pool_threshold, 88
sh_private, 89
sc_subnode_helper_from_node
sc_subnode_helper, 87
sc_subnode_helper_request_callback
sc_subnode_helper, 87
sc_thread_alloc
thread.h, 169
sc_thread_calloc
thread.h, 170
sc_thread_calloc_aligned
thread.h, 170
sc_thread_get_time
thread.h, 170
sc_thread_mfree
thread.h, 171
sc_thread_poll
thread.h, 171
sc_thread_poll_timers
thread.h, 171
sc_thread_waitable_fd_get
thread.h, 172
sc_thread_waitable_fd_prime
thread.h, 172
sc_timer_expire_after_ns
time.h, 174
sc_timer_expire_at
time.h, 174
sc_timer_get_expiry_time
time.h, 175
sc_timer_push_back_ns
time.h, 175
sc_vi, 89
sc_vi_add_stream
vi.h, 176
sc_vi_alloc
vi.h, 176
sc_vi_alloc_from_group
vi.h, 177
sc_vi_get_interface_name
vi.h, 177
sc_vi_get_thread
vi.h, 177
sc_vi_group_add_stream
vi.h, 178
sc_vi_group_alloc
vi.h, 178
sc_vi_group_get_session
vi.h, 178
sc_vi_set_recv_node
vi.h, 179
session.h, 158
sc_session_alloc, 159
sc_session_error_free, 159
sc_session_error_get, 160
sc_session_go, 160
sc_session_pause, 160
sc_session_prepare, 160
sh_backlog
sc_subnode_helper, 88
sh_free_link
sc_subnode_helper, 88
sh_handle_backlog_fn
sc_subnode_helper, 88
sh_handle_end_of_stream_fn
sc_subnode_helper, 88
sh_links
sc_subnode_helper, 88
sh_n_links
sc_subnode_helper, 88
sh_node
sc_subnode_helper, 88
sh_poll_backlog_ns
sc_subnode_helper, 88
sh_pool
sc_subnode_helper, 88
sh_pool_threshold
sc_subnode_helper, 88
sh_private
sc_subnode_helper, 89
stream.h, 161

sc_stream_all, 162
sc_stream_alloc, 162
sc_stream_eth_dhost, 162
sc_stream_eth_shost, 163
sc_stream_eth_type, 163
sc_stream_eth_vlan_id, 163
sc_stream_free, 163
sc_stream_ip_dest_host, 164
sc_stream_ip_dest_hostport, 164
sc_stream_ip_dest_port, 164
sc_stream_ip_protocol, 165
sc_stream_ip_source_host, 165
sc_stream_ip_source_hostport, 165
sc_stream_ip_source_port, 165
sc_stream_mismatch, 166
sc_stream_reset, 166
sc_stream_set_str, 166
subnode_helper.h, 168

tail
sc_packet_list, 83
thread.h, 169
sc_thread_alloc, 169
sc_thread_calloc, 170
sc_thread_calloc_aligned, 170
sc_thread_get_time, 170
sc_thread_mfree, 171
sc_thread_poll, 171
sc_thread_poll_timers, 171
sc_thread_waitable_fd_get, 172
sc_thread_waitable_fd_prime, 172
time.h, 173
sc_ns_from_ms, 173
sc_ns_from_ts, 173
sc_ns_from_tv, 174
sc_ns_from_us, 174
sc_timer_expire_after_ns, 174
sc_timer_expire_at, 174
sc_timer_get_expiry_time, 175
sc_timer_push_back_ns, 175
ts_nsec
sc_packet, 82
ts_sec
sc_packet, 82
type
sc_arg, 70

val
sc_arg, 70
vi.h, 175
sc_vi_add_stream, 176
sc_vi_alloc, 176
sc_vi_alloc_from_group, 177
sc_vi_get_interface_name, 177
sc_vi_get_thread, 177
sc_vi_group_add_stream, 178
sc_vi_group_alloc, 178
sc_vi_group_get_session, 178
sc_vi_set_recv_node, 179

  