# PULSAR

2.0.0

Generated by Doxygen 1.8.4

Thu Nov 20 2014 18:08:33

# Contents


**1 Module Index** **1**

1.1 Modules . . . 1

**2 Data Structure Index** **3**

2.1 Data Structures . . . 3

**3 File Index** **5**

3.1 File List . . . 5

**4 Module Documentation** **7**

4.1 PRT API - core interface . . . 7

4.1.1 Detailed Description . . . 8

4.1.2 Function Documentation . . . 8

4.1.2.1 prt_channel_new . . . 8

4.1.2.2 prt_tuple_new . . . 8

4.1.2.3 prt_vdp_channel_insert . . . 9

4.1.2.4 prt_vdp_channel_pop . . . 9

4.1.2.5 prt_vdp_channel_push . . . 10

4.1.2.6 prt_vdp_new . . . 10

4.1.2.7 prt_vdp_packet_new . . . 11

4.1.2.8 prt_vdp_packet_release . . . 11

4.1.2.9 prt_vsa_delete . . . 12

4.1.2.10 prt_vsa_new . . . 13

4.1.2.11 prt_vsa_run . . . 14

4.1.2.12 prt_vsa_vdp_insert . . . 15

4.2 PRT API - auxiliary interface . . . 17

4.2.1 Detailed Description . . . 17

4.2.2 Function Documentation . . . 17

4.2.2.1 prt_vdp_channel_off . . . 17

4.2.2.2 prt_vdp_channel_on . . . 17
4.2.2.3 prt_vsa_config_set . . . 18
4.2.2.4 prt_vsa_thread_warmup_func_set . . . 18
4.3 PRT API - accelerator interface . . . 19
4.3.1 Detailed Description . . . 19
4.3.2 Function Documentation . . . 19
4.3.2.1 prt_vdp_packet_new_host_to_device . . . 19
4.3.2.2 prt_vsa_device_warmup_func_set . . . 20

**5 Data Structure Documentation** **21**

5.1 gpu_malloc_s Struct Reference . . . 21
5.1.1 Detailed Description . . . 21
5.2 icl_deque_s Struct Reference . . . 22
5.2.1 Detailed Description . . . 22
5.3 icl_entry_s Struct Reference . . . 22
5.3.1 Detailed Description . . . 23
5.4 icl_hash_s Struct Reference . . . 23
5.4.1 Detailed Description . . . 23
5.5 icl_list_s Struct Reference . . . 24
5.5.1 Detailed Description . . . 24
5.6 MPI_Request Struct Reference . . . 24
5.6.1 Detailed Description . . . 24
5.7 MPI_Status Struct Reference . . . 24
5.7.1 Detailed Description . . . 25
5.8 prt_callback_finish_s Struct Reference . . . 25
5.8.1 Detailed Description . . . 25
5.9 prt_callback_queue_s Struct Reference . . . 26
5.9.1 Detailed Description . . . 26
5.10 prt_callback_release_s Struct Reference . . . 26
5.10.1 Detailed Description . . . 27
5.11 prt_channel_s Struct Reference . . . 27
5.11.1 Detailed Description . . . 28
5.12 prt_config_s Struct Reference . . . 29
5.12.1 Detailed Description . . . 29
5.13 prt_device_s Struct Reference . . . 29
5.13.1 Detailed Description . . . 30
5.14 prt_mapping_s Struct Reference . . . 30

5.14.1 Detailed Description . . . 30
5.15 prt_packet_s Struct Reference . . . 30
5.15.1 Detailed Description . . . 31
5.16 prt_proxy_s Struct Reference . . . 31
5.16.1 Detailed Description . . . 32
5.17 prt_request_s Struct Reference . . . 32
5.17.1 Detailed Description . . . 33
5.18 prt_thread_s Struct Reference . . . 34
5.18.1 Detailed Description . . . 34
5.19 prt_transfer_s Struct Reference . . . 34
5.19.1 Detailed Description . . . 35
5.20 prt_vdp_s Struct Reference . . . 35
5.20.1 Detailed Description . . . 36
5.21 prt_vsa_s Struct Reference . . . 37
5.21.1 Detailed Description . . . 38
5.22 segment Struct Reference . . . 38
5.22.1 Detailed Description . . . 38

**6 File Documentation** **39**

6.1 cuda_stubs.c File Reference . . . 39
6.1.1 Detailed Description . . . 40
6.2 cuda_stubs.h File Reference . . . 40
6.2.1 Detailed Description . . . 42
6.3 gpu_malloc.c File Reference . . . 42
6.3.1 Detailed Description . . . 43
6.3.2 Function Documentation . . . 43
6.3.2.1 gpu_free . . . 43
6.3.2.2 gpu_malloc . . . 43
6.3.2.3 gpu_malloc_fini . . . 44
6.3.2.4 gpu_malloc_init . . . 44
6.4 gpu_malloc.h File Reference . . . 45
6.4.1 Detailed Description . . . 46
6.4.2 Function Documentation . . . 46
6.4.2.1 gpu_free . . . 46
6.4.2.2 gpu_malloc . . . 47
6.4.2.3 gpu_malloc_fini . . . 47
6.4.2.4 gpu_malloc_init . . . 48

6.5 icl_deque.c File Reference . . . 48
6.5.1 Detailed Description . . . 49
6.5.2 Function Documentation . . . 50
6.5.2.1 icl_deque_append . . . 50
6.5.2.2 icl_deque_delete . . . 51
6.5.2.3 icl_deque_destroy . . . 51
6.5.2.4 icl_deque_first . . . 52
6.5.2.5 icl_deque_new . . . 53
6.5.2.6 icl_deque_next . . . 54
6.5.2.7 icl_deque_prepend . . . 54
6.5.2.8 icl_deque_size . . . 55
6.6 icl_deque.h File Reference . . . 55
6.6.1 Detailed Description . . . 57
6.6.2 Function Documentation . . . 57
6.6.2.1 icl_deque_append . . . 57
6.6.2.2 icl_deque_delete . . . 58
6.6.2.3 icl_deque_destroy . . . 59
6.6.2.4 icl_deque_first . . . 60
6.6.2.5 icl_deque_new . . . 60
6.6.2.6 icl_deque_next . . . 61
6.6.2.7 icl_deque_prepend . . . 62
6.6.2.8 icl_deque_size . . . 62
6.7 icl_hash.c File Reference . . . 63
6.7.1 Detailed Description . . . 64
6.7.2 Function Documentation . . . 64
6.7.2.1 icl_hash_create . . . 64
6.7.2.2 icl_hash_delete . . . 65
6.7.2.3 icl_hash_destroy . . . 65
6.7.2.4 icl_hash_dump . . . 66
6.7.2.5 icl_hash_find . . . 66
6.7.2.6 icl_hash_insert . . . 67
6.7.2.7 icl_hash_update_insert . . . 68
6.8 icl_hash.h File Reference . . . 68
6.8.1 Detailed Description . . . 69
6.8.2 Macro Definition Documentation . . . 70
6.8.2.1 icl_hash_foreach . . . 70
6.8.3 Function Documentation . . . 70

6.8.3.1 icl_hash_create . . . 70
6.8.3.2 icl_hash_delete . . . 71
6.8.3.3 icl_hash_destroy . . . 72
6.8.3.4 icl_hash_dump . . . 72
6.8.3.5 icl_hash_find . . . 73
6.8.3.6 icl_hash_insert . . . 73
6.8.3.7 icl_hash_update_insert . . . 74
6.9 icl_list.c File Reference . . . 74
6.9.1 Detailed Description . . . 76
6.9.2 Function Documentation . . . 76
6.9.2.1 icl_list_append . . . 76
6.9.2.2 icl_list_concat . . . 77
6.9.2.3 icl_list_delete . . . 77
6.9.2.4 icl_list_destroy . . . 78
6.9.2.5 icl_list_first . . . 78
6.9.2.6 icl_list_insert . . . 79
6.9.2.7 icl_list_isort . . . 80
6.9.2.8 icl_list_last . . . 80
6.9.2.9 icl_list_new . . . 81
6.9.2.10 icl_list_next . . . 81
6.9.2.11 icl_list_prepend . . . 82
6.9.2.12 icl_list_prev . . . 83
6.9.2.13 icl_list_search . . . 83
6.9.2.14 icl_list_size . . . 84
6.10 icl_list.h File Reference . . . 84
6.10.1 Detailed Description . . . 86
6.10.2 Function Documentation . . . 86
6.10.2.1 icl_list_append . . . 86
6.10.2.2 icl_list_concat . . . 87
6.10.2.3 icl_list_delete . . . 87
6.10.2.4 icl_list_destroy . . . 88
6.10.2.5 icl_list_first . . . 88
6.10.2.6 icl_list_insert . . . 89
6.10.2.7 icl_list_isort . . . 90
6.10.2.8 icl_list_last . . . 90
6.10.2.9 icl_list_new . . . 91
6.10.2.10 icl_list_next . . . 91

6.10.2.11 icl_list_prepend . . . 92
6.10.2.12 icl_list_prev . . . 93
6.10.2.13 icl_list_search . . . 93
6.10.2.14 icl_list_size . . . 94
6.11 mpi_stubs.c File Reference . . . 94
6.11.1 Detailed Description . . . 95
6.12 mpi_stubs.h File Reference . . . 96
6.12.1 Detailed Description . . . 97
6.13 prt.h File Reference . . . 97
6.13.1 Detailed Description . . . 99
6.14 prt_assert.c File Reference . . . 99
6.14.1 Detailed Description . . . 100
6.14.2 Function Documentation . . . 100
6.14.2.1 prt_assert_line_file . . . 100
6.14.2.2 prt_error_line_file . . . 100
6.14.2.3 prt_warning_line_file . . . 101
6.15 prt_assert.h File Reference . . . 101
6.15.1 Detailed Description . . . 102
6.15.2 Function Documentation . . . 102
6.15.2.1 prt_assert_line_file . . . 102
6.15.2.2 prt_error_line_file . . . 103
6.15.2.3 prt_warning_line_file . . . 103
6.16 prt_callback.c File Reference . . . 104
6.16.1 Detailed Description . . . 104
6.16.2 Function Documentation . . . 105
6.16.2.1 prt_callback_finish_delete . . . 105
6.16.2.2 prt_callback_finish_handler . . . 105
6.16.2.3 prt_callback_finish_new . . . 106
6.16.2.4 prt_callback_queue_delete . . . 107
6.16.2.5 prt_callback_queue_handler . . . 107
6.16.2.6 prt_callback_queue_new . . . 108
6.16.2.7 prt_callback_release_delete . . . 109
6.16.2.8 prt_callback_release_handler . . . 109
6.16.2.9 prt_callback_release_new . . . 110
6.17 prt_callback.h File Reference . . . 111
6.17.1 Detailed Description . . . 112
6.17.2 Function Documentation . . . 112

6.17.2.1 prt_callback_finish_delete . . . 112
6.17.2.2 prt_callback_finish_handler . . . 113
6.17.2.3 prt_callback_finish_new . . . 113
6.17.2.4 prt_callback_queue_delete . . . 114
6.17.2.5 prt_callback_queue_handler . . . 114
6.17.2.6 prt_callback_queue_new . . . 115
6.17.2.7 prt_callback_release_delete . . . 116
6.17.2.8 prt_callback_release_handler . . . 116
6.17.2.9 prt_callback_release_new . . . 117
6.18 prt_channel.c File Reference . . . 118
6.18.1 Detailed Description . . . 118
6.18.2 Function Documentation . . . 119
6.18.2.1 prt_channel_compare . . . 119
6.18.2.2 prt_channel_delete . . . 120
6.18.2.3 prt_channel_empty . . . 121
6.18.2.4 prt_channel_off . . . 122
6.18.2.5 prt_channel_on . . . 122
6.18.2.6 prt_channel_pop . . . 123
6.18.2.7 prt_channel_push_device . . . 123
6.18.2.8 prt_channel_push_host . . . 124
6.19 prt_channel.h File Reference . . . 125
6.19.1 Detailed Description . . . 127
6.19.2 Typedef Documentation . . . 127
6.19.2.1 prt_channel_t . . . 127
6.19.3 Function Documentation . . . 127
6.19.3.1 prt_channel_compare . . . 127
6.19.3.2 prt_channel_delete . . . 128
6.19.3.3 prt_channel_empty . . . 128
6.19.3.4 prt_channel_off . . . 129
6.19.3.5 prt_channel_on . . . 130
6.19.3.6 prt_channel_pop . . . 131
6.19.3.7 prt_channel_push_device . . . 132
6.19.3.8 prt_channel_push_host . . . 133
6.20 prt_config.c File Reference . . . 133
6.20.1 Detailed Description . . . 134
6.20.2 Function Documentation . . . 134
6.20.2.1 prt_config_delete . . . 134

6.20.2.2 prt_config_new . . . 134
6.21 prt_config.h File Reference . . . 135
6.21.1 Detailed Description . . . 136
6.21.2 Function Documentation . . . 136
6.21.2.1 prt_config_delete . . . 136
6.21.2.2 prt_config_new . . . 137
6.22 prt_device.c File Reference . . . 137
6.22.1 Detailed Description . . . 138
6.22.2 Function Documentation . . . 138
6.22.2.1 prt_device_cycle . . . 138
6.22.2.2 prt_device_delete . . . 139
6.22.2.3 prt_device_new . . . 139
6.23 prt_device.h File Reference . . . 140
6.23.1 Detailed Description . . . 141
6.23.2 Typedef Documentation . . . 141
6.23.2.1 prt_device_t . . . 141
6.23.3 Function Documentation . . . 142
6.23.3.1 prt_device_cycle . . . 142
6.23.3.2 prt_device_delete . . . 142
6.23.3.3 prt_device_new . . . 143
6.24 prt_packet.c File Reference . . . 144
6.24.1 Detailed Description . . . 145
6.24.2 Function Documentation . . . 145
6.24.2.1 prt_packet_device_mpi_to_host . . . 145
6.24.2.2 prt_packet_device_to_device . . . 146
6.24.2.3 prt_packet_device_to_device_direct . . . 147
6.24.2.4 prt_packet_device_to_host . . . 147
6.24.2.5 prt_packet_host_to_device . . . 148
6.24.2.6 prt_packet_new_device . . . 149
6.24.2.7 prt_packet_new_host . . . 150
6.24.2.8 prt_packet_release_device . . . 151
6.24.2.9 prt_packet_release_host . . . 152
6.24.2.10 prt_packet_resize_host . . . 153
6.25 prt_packet.h File Reference . . . 154
6.25.1 Detailed Description . . . 155
6.25.2 Typedef Documentation . . . 155
6.25.2.1 prt_packet_t . . . 155

6.25.3 Function Documentation . . . 156
6.25.3.1 prt_packet_device_mpi_to_host . . . 156
6.25.3.2 prt_packet_device_to_device . . . 156
6.25.3.3 prt_packet_device_to_device_direct . . . 157
6.25.3.4 prt_packet_device_to_host . . . 158
6.25.3.5 prt_packet_host_to_device . . . 158
6.25.3.6 prt_packet_new_device . . . 159
6.25.3.7 prt_packet_new_host . . . 160
6.25.3.8 prt_packet_release_device . . . 161
6.25.3.9 prt_packet_release_host . . . 162
6.25.3.10 prt_packet_resize_host . . . 163
6.26 prt_proxy.c File Reference . . . 164
6.26.1 Detailed Description . . . 165
6.26.2 Function Documentation . . . 165
6.26.2.1 prt_proxy_cuda . . . 165
6.26.2.2 prt_proxy_delete . . . 166
6.26.2.3 prt_proxy_max_channel_size . . . 167
6.26.2.4 prt_proxy_mpi . . . 168
6.26.2.5 prt_proxy_new . . . 169
6.26.2.6 prt_proxy_recv . . . 170
6.26.2.7 prt_proxy_run . . . 171
6.27 prt_proxy.h File Reference . . . 172
6.27.1 Detailed Description . . . 173
6.27.2 Typedef Documentation . . . 173
6.27.2.1 prt_proxy_t . . . 173
6.27.3 Function Documentation . . . 173
6.27.3.1 prt_proxy_cuda . . . 173
6.27.3.2 prt_proxy_delete . . . 174
6.27.3.3 prt_proxy_max_channel_size . . . 175
6.27.3.4 prt_proxy_mpi . . . 176
6.27.3.5 prt_proxy_new . . . 177
6.27.3.6 prt_proxy_recv . . . 178
6.27.3.7 prt_proxy_run . . . 179
6.28 prt_request.c File Reference . . . 180
6.28.1 Detailed Description . . . 180
6.28.2 Function Documentation . . . 181
6.28.2.1 prt_request_cancel . . . 181

6.28.2.2 prt_request_delete . . . 181
6.28.2.3 prt_request_new . . . 182
6.28.2.4 prt_request_recv . . . 182
6.28.2.5 prt_request_send . . . 183
6.28.2.6 prt_request_test . . . 184
6.29 prt_request.h File Reference . . . 184
6.29.1 Detailed Description . . . 186
6.29.2 Function Documentation . . . 186
6.29.2.1 prt_request_cancel . . . 186
6.29.2.2 prt_request_delete . . . 186
6.29.2.3 prt_request_new . . . 187
6.29.2.4 prt_request_recv . . . 187
6.29.2.5 prt_request_send . . . 188
6.29.2.6 prt_request_test . . . 189
6.30 prt_thread.c File Reference . . . 189
6.30.1 Detailed Description . . . 190
6.30.2 Function Documentation . . . 190
6.30.2.1 prt_thread_delete . . . 190
6.30.2.2 prt_thread_new . . . 191
6.30.2.3 prt_thread_run . . . 192
6.31 prt_thread.h File Reference . . . 193
6.31.1 Detailed Description . . . 194
6.31.2 Typedef Documentation . . . 194
6.31.2.1 prt_thread_t . . . 194
6.31.3 Function Documentation . . . 195
6.31.3.1 prt_thread_delete . . . 195
6.31.3.2 prt_thread_new . . . 195
6.31.3.3 prt_thread_run . . . 196
6.32 prt_transfer.c File Reference . . . 197
6.32.1 Detailed Description . . . 198
6.32.2 Function Documentation . . . 198
6.32.2.1 prt_transfer_delete . . . 198
6.32.2.2 prt_transfer_new . . . 198
6.33 prt_transfer.h File Reference . . . 199
6.33.1 Detailed Description . . . 200
6.33.2 Function Documentation . . . 200
6.33.2.1 prt_transfer_delete . . . 200

6.34 prt_tuple.c File Reference . . . 200
6.34.1 Detailed Description . . . 201
6.34.2 Function Documentation . . . 202
6.34.2.1 prt_tuple_cat . . . 202
6.34.2.2 prt_tuple_compare . . . 203
6.34.2.3 prt_tuple_copy . . . 204
6.34.2.4 prt_tuple_delete . . . 204
6.34.2.5 prt_tuple_equal . . . 204
6.34.2.6 prt_tuple_hash . . . 205
6.34.2.7 prt_tuple_len . . . 206
6.34.2.8 prt_tuple_print . . . 206
6.35 prt_tuple.h File Reference . . . 207
6.35.1 Detailed Description . . . 208
6.35.2 Function Documentation . . . 208
6.35.2.1 prt_tuple_cat . . . 208
6.35.2.2 prt_tuple_compare . . . 209
6.35.2.3 prt_tuple_copy . . . 209
6.35.2.4 prt_tuple_delete . . . 210
6.35.2.5 prt_tuple_equal . . . 210
6.35.2.6 prt_tuple_hash . . . 210
6.35.2.7 prt_tuple_len . . . 211
6.35.2.8 prt_tuple_print . . . 211
6.36 prt_vdp.c File Reference . . . 211
6.36.1 Detailed Description . . . 213
6.36.2 Function Documentation . . . 213
6.36.2.1 prt_vdp_annihilate . . . 213
6.36.2.2 prt_vdp_delete . . . 213
6.36.2.3 prt_vdp_ready . . . 214
6.37 prt_vdp.h File Reference . . . 215
6.37.1 Detailed Description . . . 216
6.37.2 Function Documentation . . . 217
6.37.2.1 prt_vdp_annihilate . . . 217
6.37.2.2 prt_vdp_delete . . . 218
6.37.2.3 prt_vdp_ready . . . 219
6.38 prt_vsa.c File Reference . . . 219
6.38.1 Detailed Description . . . 221
6.38.2 Function Documentation . . . 221

6.38.2.1 prt_tuple_equal . . . 221
6.38.2.2 prt_tuple_hash . . . 222
6.38.2.3 prt_vsa_channel_streams . . . 222
6.38.2.4 prt_vsa_channel_tags . . . 223
6.38.2.5 prt_vsa_devices_warmup . . . 224
6.38.2.6 prt_vsa_vdp_merge_channels . . . 224
6.38.2.7 prt_vsa_vdp_track_tags . . . 225
6.39 prt_vsa.h File Reference . . . 226
6.39.1 Detailed Description . . . 227
6.39.2 Function Documentation . . . 228
6.39.2.1 prt_vsa_channel_streams . . . 228
6.39.2.2 prt_vsa_channel_tags . . . 229
6.39.2.3 prt_vsa_devices_warmup . . . 230
6.39.2.4 prt_vsa_vdp_merge_channels . . . 230
6.39.2.5 prt_vsa_vdp_track_tags . . . 231
6.40 svg_trace.c File Reference . . . 232
6.40.1 Detailed Description . . . 233
6.40.2 Function Documentation . . . 233
6.40.2.1 get_time_of_day . . . 233
6.40.2.2 svg_trace_init . . . 233
6.40.2.3 svg_trace_memory_device . . . 234
6.40.2.4 svg_trace_memory_host . . . 234
6.40.2.5 svg_trace_start_cpu . . . 235
6.40.2.6 svg_trace_start_dma . . . 236
6.40.2.7 svg_trace_start_gpu . . . 237
6.40.2.8 svg_trace_stop_cpu . . . 237
6.40.2.9 svg_trace_stop_dma . . . 238
6.40.2.10 svg_trace_stop_gpu . . . 239
6.41 svg_trace.h File Reference . . . 239
6.41.1 Detailed Description . . . 242
6.41.2 Function Documentation . . . 242
6.41.2.1 get_time_of_day . . . 242
6.41.2.2 svg_trace_init . . . 242
6.41.2.3 svg_trace_memory_device . . . 243
6.41.2.4 svg_trace_memory_host . . . 243
6.41.2.5 svg_trace_start_cpu . . . 244
6.41.2.6 svg_trace_start_dma . . . 245

6.41.2.7 svg_trace_start_gpu . . . . . 245
6.41.2.8 svg_trace_stop_cpu . . . . . 245
6.41.2.9 svg_trace_stop_dma . . . . . 246
6.41.2.10 svg_trace_stop_gpu . . . . . 247

**Index** **248**

# Chapter 1

# Module Index

## 1.1 Modules

Here is a list of all modules:

PRT API - core interface . . . . . 7
PRT API - auxiliary interface . . . . . 17
PRT API - accelerator interface . . . . . 19

# Chapter 2

# Data Structure Index

## 2.1 Data Structures

Here are the data structures with brief descriptions:


gpu_malloc_s . . . 21
icl_deque_s . . . 22
icl_entry_s . . . 22
icl_hash_s . . . 23
icl_list_s . . . 24
MPI_Request . . . 24
MPI_Status . . . 24
prt_callback_finish_s
Callback data for finishing a local communication . . . 25
prt_callback_queue_s
Callback data for queueing a local communication . . . 26
prt_callback_release_s
Callback data for releasing a device packet . . . 26
prt_channel_s
VDP's data channel. Implements a data link between a pair of VDPs. Identifies the source and destination VDPs by tuples. Contains a thread-safe list of data packets . . . 27
prt_config_s
PRT configuration . . . 29
prt_device_s
VSA's accelerator device. Represents a hardware accelerator. Currently synonymous with an Nvidia GPU . . . 29
prt_mapping_s
Mapping of VDPs to hardware . . . 30
prt_packet_s
VDP's data packet A packet of data transferred through VDP's channels . . . 30
prt_proxy_s
VSA's proxy . . . 31
prt_request_s
MPI communication request for a packet. Contains a packet, some info, MPI request and MPI status 32
prt_thread_s
VSA's worker thread. Represents a single CPU core or a collection of cores . . . 34
prt_transfer_s
Local transfer object . . . 34

prt_vdp_s
Virtual Data Processor (VDP). Is uniquely identified by a tuple. Fires for a predefined number of cycles. Has a fixed number of input and output channels. Has a persistent local store. Has access to read-only global store . . . . . 35
prt_vsa_s
Virtual Systolic Array (VSA) VSA contains global informationa about the system, a local communication proxy, an array of local worker threads, and an array of local accelerator devices . . . . . 37
segment . . . . . 38

# Chapter 3

# File Index

## 3.1 File List

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


cuda_stubs.c
Stubs for a no-CUDA build . . . 39
cuda_stubs.h
Stubs for a no-CUDA build . . . 40
gpu_malloc.c
Simple device memory allocator . . . 42
gpu_malloc.h
Simple device memory allocator . . . 45
icl_deque.c
Thread-safe double-ended queue . . . 48
icl_deque.h
Thread-safe double-ended queue . . . 55
icl_hash.c
Dependency-free hash table . . . 63
icl_hash.h
Dependency-free hash table . . . 68
icl_list.c
Dependency-free linked list . . . 74
icl_list.h
Dependency-free linked list . . . 84
**make.inc** . . . **??**
mpi_stubs.c
Stubs for a no-MPI build . . . 94
mpi_stubs.h
Stubs for a no-MPI build . . . 96
prt.h
PULSAR Runtime (PRT) . . . 97
prt_assert.c
PRT exception handling . . . 99
prt_assert.h
PRT exception handling . . . 101
prt_callback.c
PRT callback . . . 104

prt_callback.h
PRT callback . . . 111
prt_channel.c
PRT data channel . . . 118
prt_channel.h
PRT data channel . . . 125
prt_config.c
PRT configuration . . . 133
prt_config.h
PRT configuration . . . 135
prt_device.c
PRT device . . . 137
prt_device.h
PRT device . . . 140
prt_packet.c
PRT data packet . . . 144
prt_packet.h
PRT data packet . . . 154
prt_proxy.c
PRT communication proxy . . . 164
prt_proxy.h
PRT communication proxy . . . 172
prt_request.c
PRT communication request . . . 180
prt_request.h
PRT communication request . . . 184
prt_thread.c
PRT thread . . . 189
prt_thread.h
PRT thread . . . 193
prt_transfer.c
PRT local transfer . . . 197
prt_transfer.h
PRT local transfer . . . 199
prt_tuple.c
PRT tuple . . . 200
prt_tuple.h
PRT tuple . . . 207
prt_vdp.c
Virtual Data Processor . . . 211
prt_vdp.h
Virtual Data Processor . . . 215
prt_vsa.c
Virtual Systolic Array . . . 219
prt_vsa.h
Virtual Systolic Array . . . 226
svg_trace.c
SVG tracing . . . 232
svg_trace.h
SVG tracing . . . 239

# Chapter 4

# Module Documentation

## 4.1 PRT API - core interface

### Functions

- prt_channel_t * prt_channel_new (size_t size, int *src_tuple, int src_slot, int *dst_tuple, int dst_slot)

  *Creates a new channel. Channel size cannot be larger than INT_MAX, because all data typea are packed inside messages of type MPI_BYTE.*

- int * prt_tuple_new (int len,...)

  *Creates a new tuple. Allocates memory for the tuple plus the termination symbol (INT_MAX). Fills out the tuple with the integers on the list. There is also a set of macros, prt_tuple_new1/2/3/4/5/6, where the length of the tuple is indicated by the number in the name. Because this is such a tiny function, and is mostly intended to be accessed through macros, skipping error checks for input parameters.*

- prt_vdp_t * prt_vdp_new (int *tuple, int counter, prt_vdp_function_t function, size_t local_store_size, int num_inputs, int num_outputs, int color)

  *Creates a new VDP.*

- void prt_vdp_channel_insert (prt_vdp_t *vdp, prt_channel_t *channel, prt_channel_direction_t direction, int slot)

  *Inserts a new channel into a VDP.*

- prt_packet_t * prt_vdp_packet_new (prt_vdp_t *vdp, size_t size, void *data)

  *Creates a new packet. Allocates the size amount of data if a NULL pointer is passed. The size cannot be larger than INT_MAX, because all data typea are packed inside messages of type MPI_BYTE. Calls host constructor or device constructor depending on the VDP's location.*

- void prt_vdp_packet_release (prt_vdp_t *vdp, prt_packet_t *packet)

  *Releases a packet. Decrements the number of active references. Destroys the packet when the number of references goes down to zero. For device packets, puts a callback in the VDP's stream.*

- void prt_vdp_channel_push (prt_vdp_t *vdp, int channel_num, prt_packet_t *packet)

  *Pushes a packet in a channel.*

- prt_packet_t * prt_vdp_channel_pop (prt_vdp_t *vdp, int channel_num)

  *Fetches a packet from a channel.*

- prt_vsa_t * prt_vsa_new (int num_threads, int num_devices, void *global_store, struct prt_mapping_s(*vdp_mapping)(int *, void *, int, int))

  *Creates a new VSA.*

- void prt_vsa_delete (prt_vsa_t *vsa)

  *Destroys a VSA.*

- void prt_vsa_vdp_insert (prt_vsa_t *vsa, prt_vdp_t *vdp)

*Inserts a VDP in a VSA. Destroys VDPs that do not belong to this node. Puts the VDP in the list of VDPs of the owner thread or device. Connects corresponding input and output channels of intra-node VDPs. Builds the list of channel connections to other nodes. For a device VDP, creates a CUDA stream with the cudaStreamNonBlocking flag. This indicates no synchronization with the default stream (stream 0). Stream 0 is not used anywere in PRT.*

- double prt_vsa_run (prt_vsa_t *vsa)

  *Implements the VSA's production cycle. Launches worker threads. Sends the master thread in the proxy production cycle. Joins the worker threads.*

### 4.1.1 Detailed Description

### 4.1.2 Function Documentation

#### 4.1.2.1 prt_channel_t* prt_channel_new ( size_t *size*, int * *src_tuple*, int *src_slot*, int * *dst_tuple*, int *dst_slot* )

Creates a new channel. Channel size cannot be larger than INT_MAX, because all data typea are packed inside messages of type MPI_BYTE.

**Parameters**

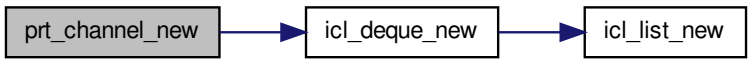
| | |
|---|---|
| *size* | – The size of packets in bytes. |
| *src_tuple* | – The tuple of the source VDP. |
| *src_slot* | – The slot number in the source VDP. |
| *dst_tuple* | – The tuple of the destination VDP. |
| *dst_slot* | – The slot number in the destination VDP. |

**Returns**

A new channel.

Definition at line 28 of file prt_channel.c.

Here is the call graph for this function:




#### 4.1.2.2 int* prt_tuple_new ( int *len*, ... )

Creates a new tuple. Allocates memory for the tuple plus the termination symbol (INT_MAX). Fills out the tuple with the integers on the list. There is also a set of macros, prt_tuple_new1/2/3/4/5/6, where the length of the tuple is indicated by the number in the name. Because this is such a tiny function, and is mostly intended to be accessed through macros, skipping error checks for input parameters.

**Parameters**

| | |
|---|---|
| *len* | – The length of the tuple. |
| *...* | – A list of elements of type int. |

**Returns**

A pointer to an array of integers terminated by INT_MAX.

Definition at line 31 of file prt_tuple.c.

#### 4.1.2.3 void prt_vdp_channel_insert ( prt_vdp_t * *vdp*, prt_channel_t * *channel*, prt_channel_direction_t *direction*, int *slot* )

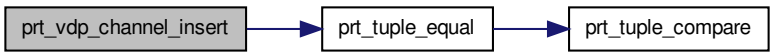
Inserts a new channel into a VDP.

**Parameters**

| | |
|---|---|
| *vdp* | – The VDP to insert the channel into. |
| *channel* | – The channel to insert. |
| *direction* | – The direction of the channel. |
| *slot* | – The slot number. |

Definition at line 200 of file prt_vdp.c.

Here is the call graph for this function:




#### 4.1.2.4 prt_packet_t* prt_vdp_channel_pop ( prt_vdp_t * *vdp*, int *channel_num* )

Fetches a packet from a channel.

**Parameters**

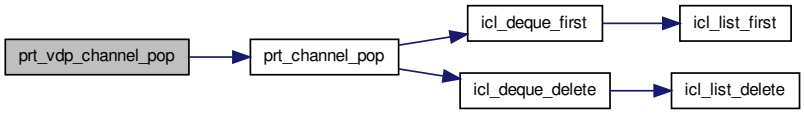
| | |
|---|---|
| *vdp* | – The VDP fetching the packet. |
| *channel_num* | – The number of the channel to fetch from. |

**Returns**

A packet.

Definition at line 393 of file prt_vdp.c.

Here is the call graph for this function:




#### 4.1.2.5 void prt_vdp_channel_push ( prt_vdp_t * *vdp*, int *channel_num*, prt_packet_t * *packet* )

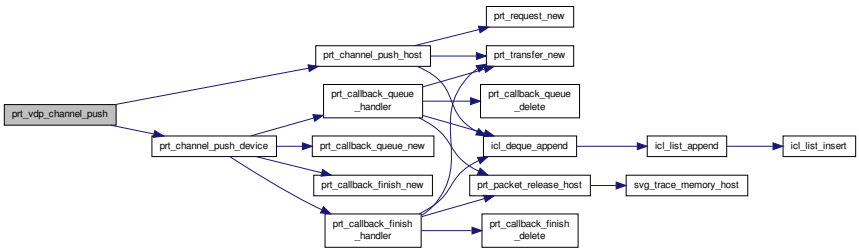
Pushes a packet in a channel.

**Parameters**

| | |
|---|---|
| *vdp* | – The VDP pushing to the channel. |
| *channel_num* | – The number of the channel to push to. |
| *packet* | – The packet to push. |

Definition at line 360 of file prt_vdp.c.

Here is the call graph for this function:




#### 4.1.2.6 prt_vdp_t* prt_vdp_new ( int * *tuple*, int *counter*, prt_vdp_function_t *function*, size_t *local_store_size*, int *num_inputs*, int *num_outputs*, int *color* )

Creates a new VDP.

**Parameters**

| | |
|---|---|
| *tuple* | – A unique identifier of the VDP. |
| *counter* | – The number of times to fire the VDP. |
| *function* | – The function implementing the VDP's actions. |
| *local_store_size* | – The size of VDP's persistent local store in bytes. |
| *num_inputs* | – The number of input channels. |
| *num_outputs* | – The number of output channels. |
| *color* | – The VDP's color in the SVG traces. |

**Returns**

A new VDP.

Definition at line 28 of file prt_vdp.c.

#### 4.1.2.7 prt_packet_t* prt_vdp_packet_new ( prt_vdp_t * *vdp*, size_t *size*, void * *data* )

Creates a new packet. Allocates the size amount of data if a NULL pointer is passed. The size cannot be larger than INT_MAX, because all data typea are packed inside messages of type MPI_BYTE. Calls host constructor or device constructor depending on the VDP's location.

**Parameters**

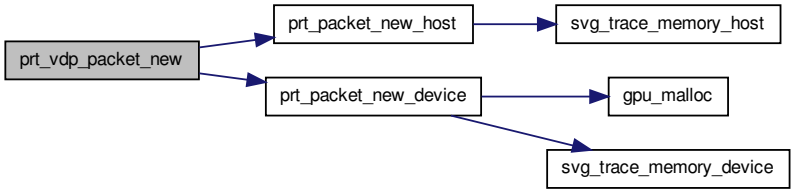
| | |
|---|---|
| *vdp* | – The VDP creating the packet. |
| *size* | – The size of the packet in bytes. |
| *data* | – The data payload of the packet. |

**Returns**

A new packet.

Definition at line 258 of file prt_vdp.c.

Here is the call graph for this function:




#### 4.1.2.8 void prt_vdp_packet_release ( prt_vdp_t * *vdp*, prt_packet_t * *packet* )

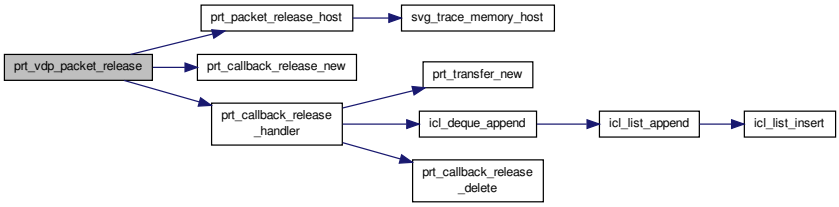
Releases a packet. Decrements the number of active references. Destroys the packet when the number of references goes down to zero. For device packets, puts a callback in the VDP's stream.

**Parameters**

| | |
|---|---|
| *vdp* | – The VDP releasing the packet. |
| *packet* | – The packet to release. |

Definition at line 330 of file prt_vdp.c.

Here is the call graph for this function:




#### 4.1.2.9 void prt_vsa_delete ( prt_vsa_t * *vsa* )

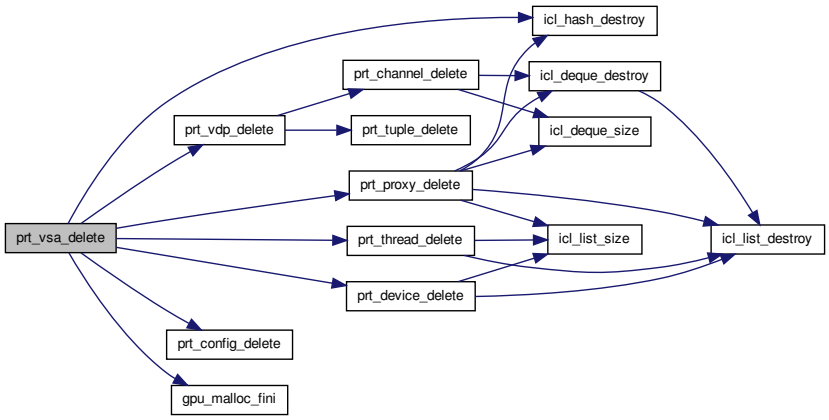
Destroys a VSA.

**Parameters**

| | |
|---|---|
| *VSA* | – The VSA to destroy. |

Definition at line 140 of file prt_vsa.c.

Here is the call graph for this function:



4.1.2.10 **prt_vsa_t∗ prt_vsa_new ( int *num_threads,* int *num_devices,* void ∗ *global_store,* struct prt_mapping_s(∗)(int ∗, void ∗, int, int) *vdp_mapping* )**

Creates a new VSA.

**Parameters**

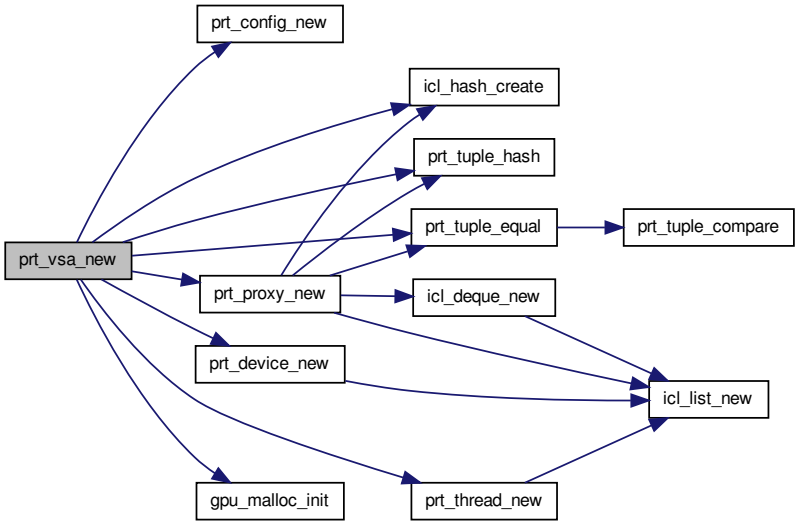
| | |
|---|---|
| *num_threads* | – The number of local CPU threads. |
| *num_devices* | – The number of local GPU devices. |
| *global_store* | – VSA's global store, accessible to all VDPs. |
| *vdp_mapping* | – The function for mapping VDPs to cores and accelerators. |

**Returns**

A new VSA.

Definition at line 28 of file prt_vsa.c.

Here is the call graph for this function:




#### 4.1.2.11 double prt_vsa_run ( prt_vsa_t * *vsa* )

Implements the VSA's production cycle. Launches worker threads. Sends the master thread in the proxy production cycle. Joins the worker threads.

**Parameters**

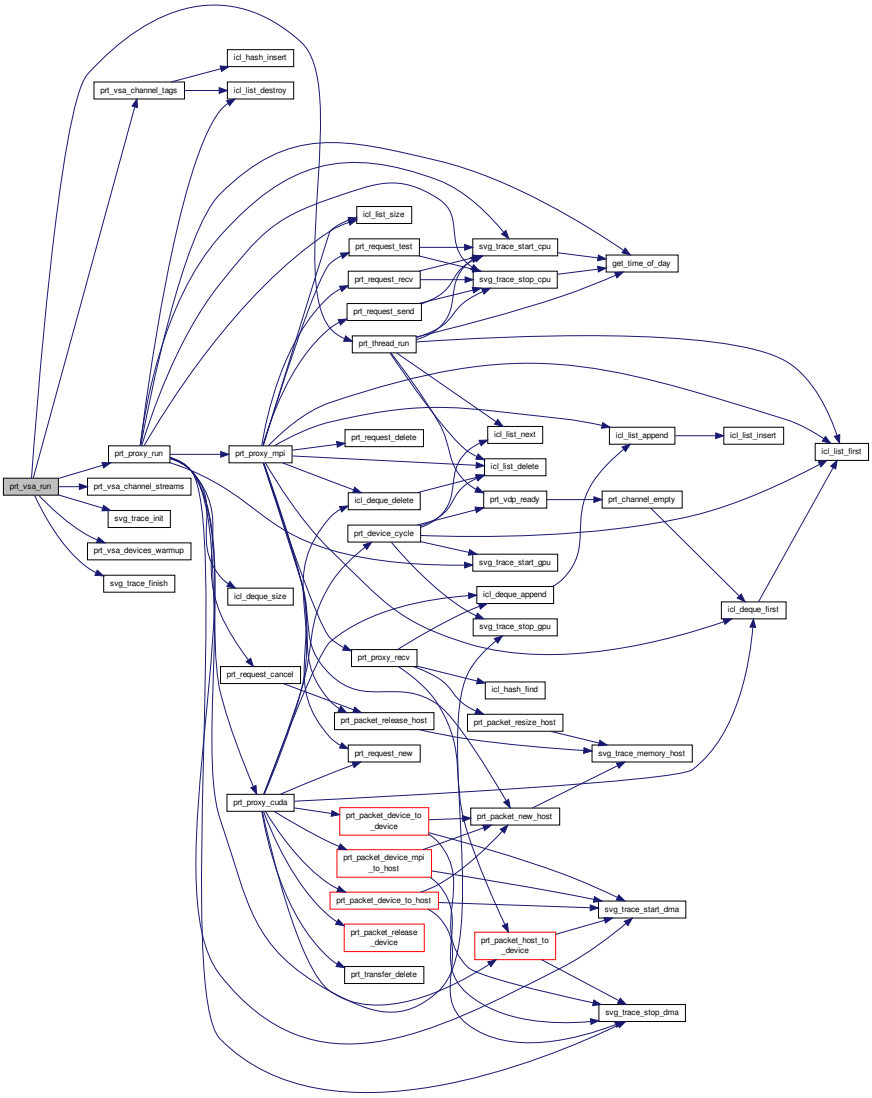
| | |
|---|---|
| *vsa* | – The VSA to run. |

**Returns**

The VSA's execution time in seconds.

Definition at line 546 of file prt_vsa.c.

Here is the call graph for this function:




#### 4.1.2.12 void prt_vsa_vdp_insert ( prt_vsa_t * *vsa*, prt_vdp_t * *vdp* )

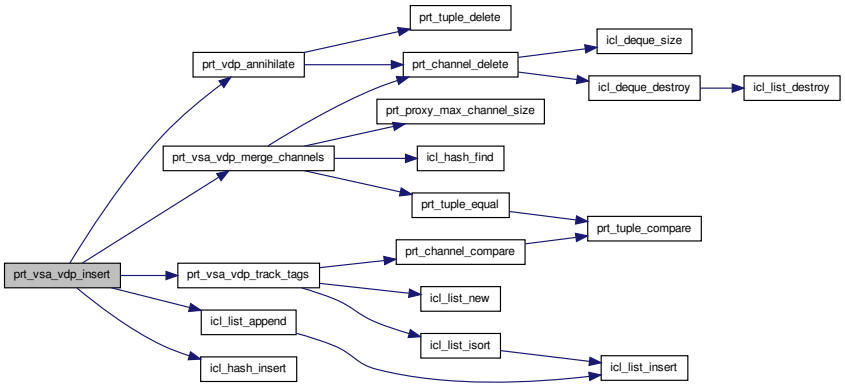
Inserts a VDP in a VSA. Destroys VDPs that do not belong to this node. Puts the VDP in the list of VDPs of the owner thread or device. Connects corresponding input and output channels of intra-node VDPs. Builds the list of channel connections to other nodes. For a device VDP, creates a CUDA stream with the cudaStreamNonBlocking flag. This indicates no synchronization with the default stream (stream 0). Stream 0 is not used anywere in PRT.

**Parameters**

| | |
|---|---|
| *vsa* | – The VSA to insert into. |
| *vdp* | – The VDP to insert. |

Definition at line 200 of file prt_vsa.c.

Here is the call graph for this function:



## 4.2 PRT API - auxiliary interface

### Functions

- void prt_vdp_channel_off (prt_vdp_t *vdp, int channel_num)

  *Deactivates a channel.*
- void prt_vdp_channel_on (prt_vdp_t *vdp, int channel_num)

  *Activates a channel.*
- void prt_vsa_config_set (prt_vsa_t *vsa, prt_config_param_t param, prt_config_value_t value)

  *Sets a VSA configuration parameter.*
- void prt_vsa_thread_warmup_func_set (prt_vsa_t *vsa, void(*func)())

  *Sets a thread warmup function. If set, the thread warmup function is called by each thread right after launching and before threads are barriered and the timer is started. Allows for excluding the time for initialization procedures of libraries, such as loading of dynamic libraries, internal memory allocations, possibly time consuming pinned memory allocations, etc. A NULL function pointer can be passed to remove the thread warmup function.*

### 4.2.1 Detailed Description

### 4.2.2 Function Documentation

#### 4.2.2.1 void prt_vdp_channel_off ( prt_vdp_t * *vdp*, int *channel_num* )

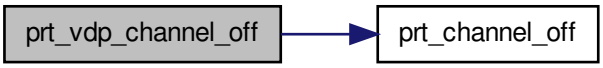
Deactivates a channel.

**Parameters**

| | |
|---|---|
| *vdp* | – The VDP deactivating the channel. |
| *channel_num* | – The number of the channel to be deactivated. |

Definition at line 414 of file prt_vdp.c.

Here is the call graph for this function:




#### 4.2.2.2 void prt_vdp_channel_on ( prt_vdp_t * *vdp*, int *channel_num* )

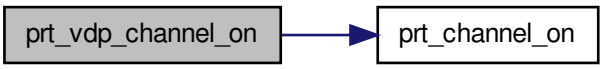
Activates a channel.

**Parameters**

| | |
|---|---|
| *vdp* | – The VDP activating the channel. |
| *channel_num* | – The channel to be activated. |

Definition at line 433 of file prt_vdp.c.

Here is the call graph for this function:




#### 4.2.2.3 void prt_vsa_config_set ( prt_vsa_t * *vsa*, prt_config_param_t *param*, prt_config_value_t *value* )

Sets a VSA configuration parameter.

**Parameters**

| | |
|---|---|
| *vsa* | – The VSA to configure. |
| *param* | – The parameter to set. |
| *value* | – The new value for the parameter. |

Definition at line 606 of file prt_vsa.c.

#### 4.2.2.4 void prt_vsa_thread_warmup_func_set ( prt_vsa_t * *vsa*, void(*)() *func* )

Sets a thread warmup function. If set, the thread warmup function is called by each thread right after launching and before threads are barriered and the timer is started. Allows for excluding the time for initialization procedures of libraries, such as loading of dynamic libraries, internal memory allocations, possibly time consuming pinned memory allocations, etc. A NULL function pointer can be passed to remove the thread warmup function.

**Parameters**

| | |
|---|---|
| *vsa* | – The VSA to set the function for. |
| *func* | – The thread (CPU) warmup function. |

Definition at line 656 of file prt_vsa.c.

## 4.3 PRT API - accelerator interface

### Functions

- prt_packet_t ∗ prt_vdp_packet_new_host_to_device (prt_vdp_t ∗vdp, size_t size, void ∗data)

  *Creates a new packet and queues a host-to-device transfer. The size cannot be larger than INT_MAX, because all data typea are packed inside messages of type MPI_BYTE. Expects a non-NULL pointer to the data in host memory. Right now, device memory is allocated immediately. Potentially, it could also be done in the VDP's stream.*

- void prt_vsa_device_warmup_func_set (prt_vsa_t ∗vsa, void(∗func)())

  *Sets a device warmup function. If set, the device warmup function is called by each device right after launching and before devices are barriered and the timer is started. Allows for excluding the time for initialization procedures of libraries, such as loading of dynamic libraries, internal memory allocations, possibly time consuming pinned memory allocations, etc. A NULL function pointer can be passed to remove the device warmup function.*

### 4.3.1 Detailed Description

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

### 4.3.2 Function Documentation

#### 4.3.2.1 prt_packet_t∗ prt_vdp_packet_new_host_to_device ( prt_vdp_t ∗ *vdp*, size_t *size*, void ∗ *data* )

Creates a new packet and queues a host-to-device transfer. The size cannot be larger than INT_MAX, because all data typea are packed inside messages of type MPI_BYTE. Expects a non-NULL pointer to the data in host memory. Right now, device memory is allocated immediately. Potentially, it could also be done in the VDP's stream.

**Parameters**

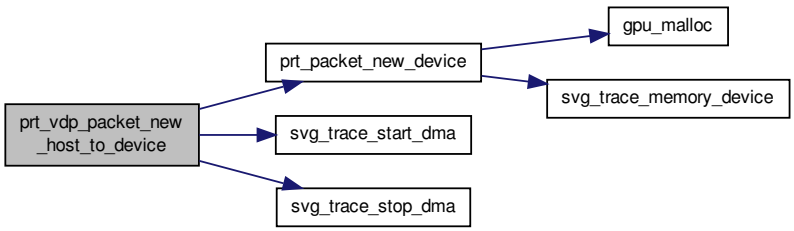
| | |
|---|---|
| *vdp* | – The VDP creating the packet. |
| *size* | – Teh size of the packet in bytes. |
| *data* | – The data payload of the packet. |

**Returns**

A new packet.

Definition at line 297 of file prt_vdp.c.

Here is the call graph for this function:



#### 4.3.2.2 void prt_vsa_device_warmup_func_set ( prt_vsa_t * *vsa*, void(*)() *func* )

Sets a device warmup function. If set, the device warmup function is called by each device right after launching and before devices are barriered and the timer is started. Allows for excluding the time for initialization procedures of libraries, such as loading of dynamic libraries, internal memory allocations, possibly time consuming pinned memory allocations, etc. A NULL function pointer can be passed to remove the device warmup function.

**Parameters**

| | |
|---|---|
| *vsa* | – The VSA to set the function for. |
| *func* | – The device (GPU) warmup function. |

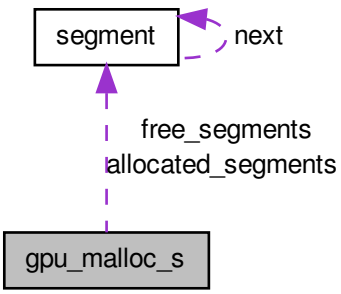
Definition at line 679 of file prt_vsa.c.

# Chapter 5

# Data Structure Documentation

## 5.1 gpu_malloc_s Struct Reference

Collaboration diagram for gpu_malloc_s:




### Data Fields

- char * **base**
- segment_t * **allocated_segments**
- segment_t * **free_segments**
- size_t **unit_size**
- int **max_segment**

### 5.1.1 Detailed Description

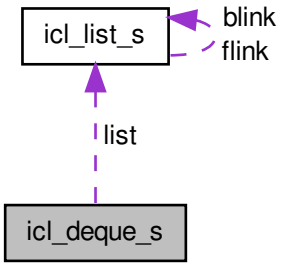
Definition at line 36 of file gpu_malloc.h.

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

- gpu_malloc.h

## 5.2 icl_deque_s Struct Reference

Collaboration diagram for icl_deque_s:




### Data Fields

- pthread_spinlock_t **spinlock**
- icl_list_t * **list**
- int **size**

### 5.2.1 Detailed Description

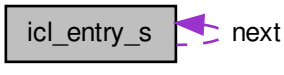
Definition at line 24 of file icl_deque.h.

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

- icl_deque.h

## 5.3 icl_entry_s Struct Reference

Collaboration diagram for icl_entry_s:



### Data Fields

- void * **key**
- void * **data**
- struct icl_entry_s * **next**

#### 5.3.1 Detailed Description

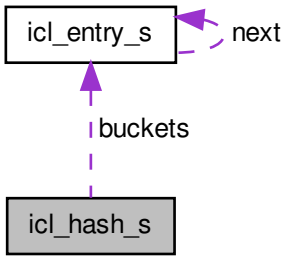
Definition at line 18 of file icl_hash.h.

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

- icl_hash.h

## 5.4 icl_hash_s Struct Reference

Collaboration diagram for icl_hash_s:




### Data Fields

- int **nbuckets**
- int **nentries**
- icl_entry_t ** **buckets**
- unsigned int(* **hash_function** )(void *)
- int(* **hash_key_compare** )(void *, void *)

#### 5.4.1 Detailed Description

Definition at line 24 of file icl_hash.h.

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

- icl_hash.h

## 5.5 icl_list_s Struct Reference

Collaboration diagram for icl_list_s:


### Data Fields

- void * **data**
- struct icl_list_s * **flink**
- struct icl_list_s * **blink**

### 5.5.1 Detailed Description

Definition at line 18 of file icl_list.h.

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

- icl_list.h

## 5.6 MPI_Request Struct Reference

### 5.6.1 Detailed Description

Definition at line 30 of file mpi_stubs.h.

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

- mpi_stubs.h

## 5.7 MPI_Status Struct Reference

### Data Fields

- int **MPI_TAG**
- int **MPI_SOURCE**

### 5.7.1 Detailed Description

Definition at line 25 of file mpi_stubs.h.

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

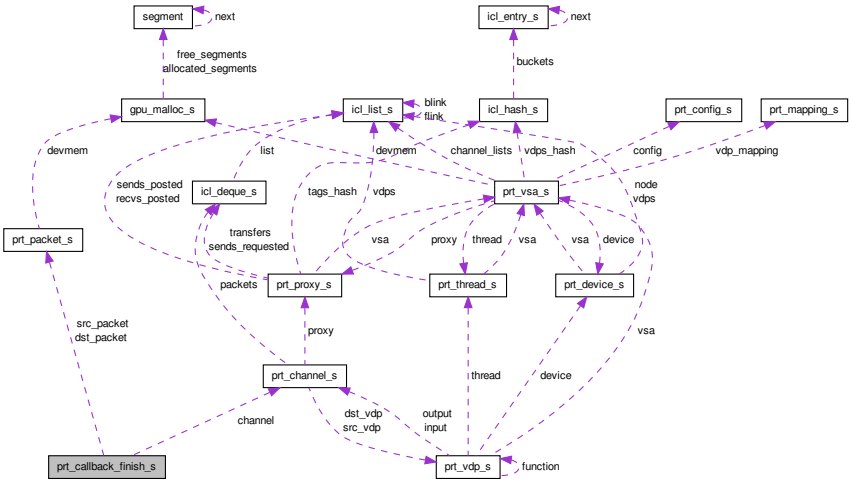
- mpi_stubs.h

## 5.8 prt_callback_finish_s Struct Reference

Callback data for finishing a local communication.

`#include <prt_callback.h>`

Collaboration diagram for prt_callback_finish_s:




### Data Fields

- struct prt_packet_s * **src_packet**
- struct prt_packet_s * **dst_packet**
- struct prt_channel_s * **channel**

### 5.8.1 Detailed Description

Callback data for finishing a local communication.

Definition at line 23 of file prt_callback.h.

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

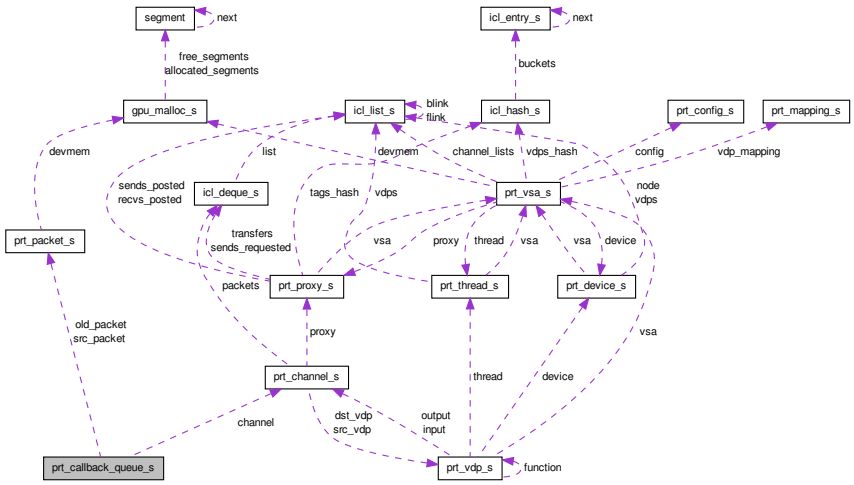
- prt_callback.h

## 5.9 prt_callback_queue_s Struct Reference

Callback data for queueing a local communication.

`#include <prt_callback.h>`

Collaboration diagram for prt_callback_queue_s:




### Data Fields

- struct prt_packet_s * **old_packet**
- struct prt_packet_s * **src_packet**
- struct prt_channel_s * **channel**
- prt_direction_t **direction**
- int **agent**

### 5.9.1 Detailed Description

Callback data for queueing a local communication.

Definition at line 32 of file prt_callback.h.

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

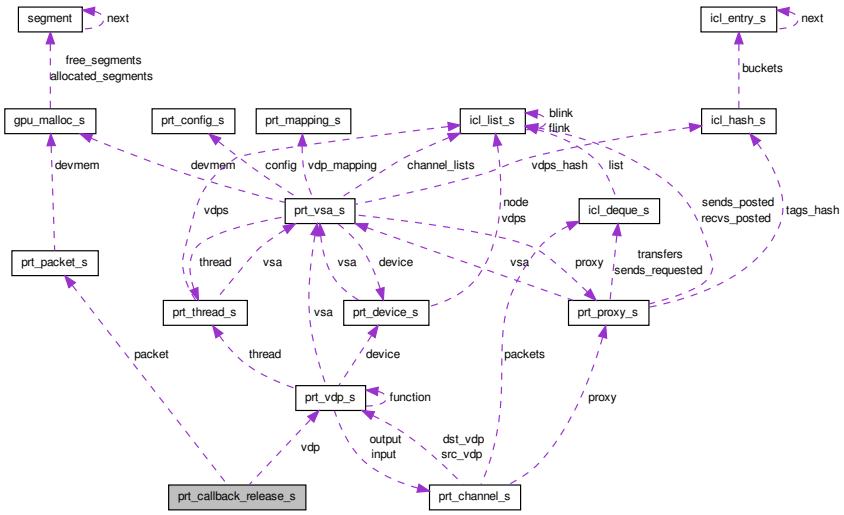
- prt_callback.h

## 5.10 prt_callback_release_s Struct Reference

Callback data for releasing a device packet.

`#include <prt_callback.h>`

Collaboration diagram for prt_callback_release_s:




### Data Fields

- struct prt_vdp_s * **vdp**
- struct prt_packet_s * **packet**

#### 5.10.1 Detailed Description

Callback data for releasing a device packet.

Definition at line 43 of file prt_callback.h.

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

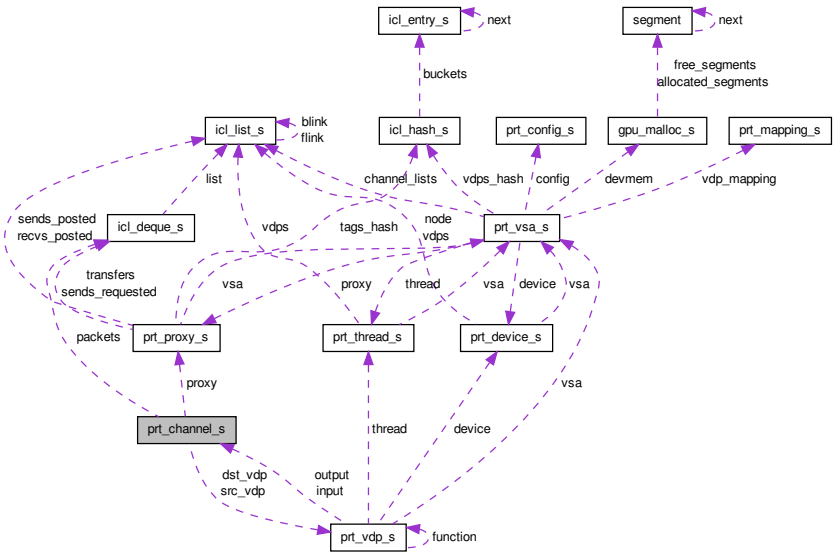
- prt_callback.h

## 5.11 prt_channel_s Struct Reference

VDP's data channel. Implements a data link between a pair of VDPs. Identifies the source and destination VDPs by tuples. Contains a thread-safe list of data packets.

`#include <prt_channel.h>`

Collaboration diagram for prt_channel_s:




## Data Fields

- struct prt_vdp_s * **dst_vdp**
- struct prt_vdp_s * **src_vdp**
- struct prt_proxy_s * **proxy**
- size_t **size**
- int * **src_tuple**
- int **src_slot**
- int * **dst_tuple**
- int **dst_slot**
- int **src_node**
- int **dst_node**
- int **tag**
- icl_deque_t * **packets**
- int **active**
- cudaStream_t **in_stream**
- cudaStream_t **out_stream**

### 5.11.1 Detailed Description

VDP's data channel. Implements a data link between a pair of VDPs. Identifies the source and destination VDPs by tuples. Contains a thread-safe list of data packets.

The in_stream is used when the recipient device pulls: host->device, device->device (second stage). The out_stream is used when the sender device pushes: device->host, device->device (first stage).

Definition at line 34 of file prt_channel.h.

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

- prt_channel.h

## 5.12 prt_config_s Struct Reference

PRT configuration.

`#include <prt_config.h>`

### Data Fields

- int **vdp_scheduling**
- int **svg_tracing**

### 5.12.1 Detailed Description

PRT configuration.

Definition at line 41 of file prt_config.h.

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

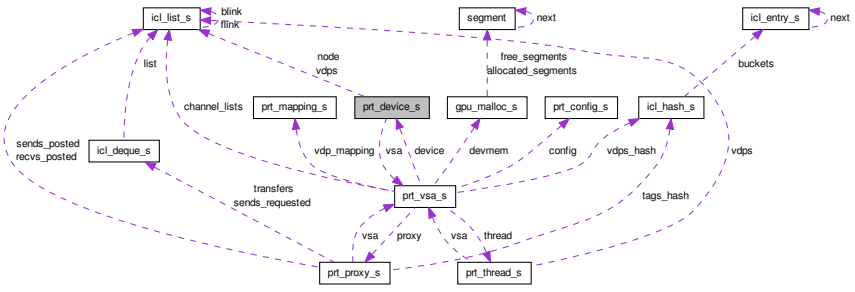
- prt_config.h

## 5.13 prt_device_s Struct Reference

VSA's accelerator device. Represents a hardware accelerator. Currently synonymous with an Nvidia GPU.

`#include <prt_device.h>`

Collaboration diagram for prt_device_s:




### Data Fields

- struct prt_vsa_s * **vsa**
- int **rank**
- int **accelerator**
- icl_list_t * **vdps**

- icl_list_t * **node**
- volatile int **finished**
- int **agent_rank**

### 5.13.1 Detailed Description

VSA's accelerator device. Represents a hardware accelerator. Currently synonymous with an Nvidia GPU.

"finished" is a one-directional synchronization variable. Therefore declered volatile, but no need for atomic access.

Definition at line 30 of file prt_device.h.

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

- prt_device.h

## 5.14 prt_mapping_s Struct Reference

Mapping of VDPs to hardware.

`#include <prt.h>`

### Data Fields

- prt_location_t **location**
- int **rank**

### 5.14.1 Detailed Description

Mapping of VDPs to hardware.

Definition at line 43 of file prt.h.

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

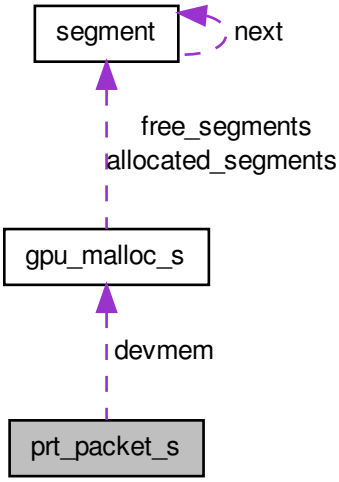
- prt.h

## 5.15 prt_packet_s Struct Reference

VDP's data packet A packet of data transferred through VDP's channels.

`#include <prt_packet.h>`

Collaboration diagram for prt_packet_s:




### Data Fields

- void * **data**
- size_t **size**
- volatile int **num_refs**
- prt_location_t **location**
- int **device_rank**
- struct gpu_malloc_s * **devmem**

#### 5.15.1 Detailed Description

VDP's data packet A packet of data transferred through VDP's channels.

"num_refs" is a multi-access synchronization variable. Therefore, declared as volatile and accessed with atomics.

Definition at line 31 of file prt_packet.h.

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

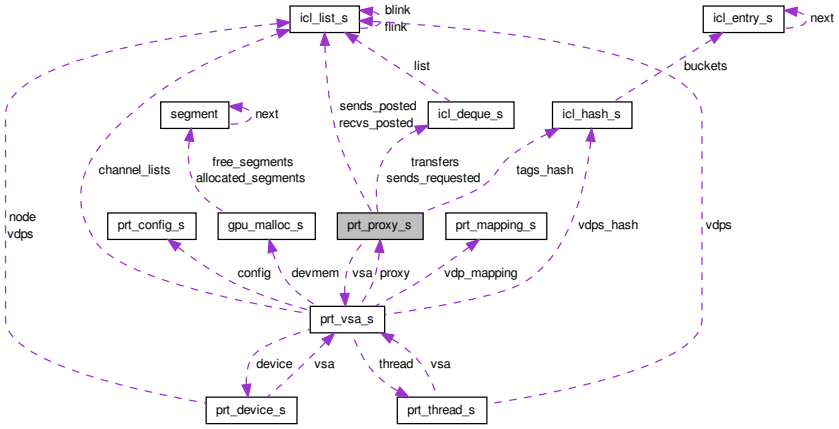
- prt_packet.h

## 5.16 prt_proxy_s Struct Reference

VSA's proxy.

`#include <prt_proxy.h>`

Collaboration diagram for prt_proxy_s:




### Data Fields

- struct prt_vsa_s * **vsa**
- int **num_agents**
- icl_hash_t * **tags_hash**
- icl_deque_t ** **sends_requested**
- icl_list_t ** **sends_posted**
- icl_list_t * **recvs_posted**
- icl_deque_t * **transfers**
- size_t **max_channel_size**
- volatile int **num_callbacks**

#### 5.16.1 Detailed Description

VSA's proxy.

The reason for the num_callbacks counter is the following: Empty transfers queue does not mean there is nothing pending. Communication requests may be sitting in a stream waiting to be queued.

Definition at line 49 of file prt_proxy.h.

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

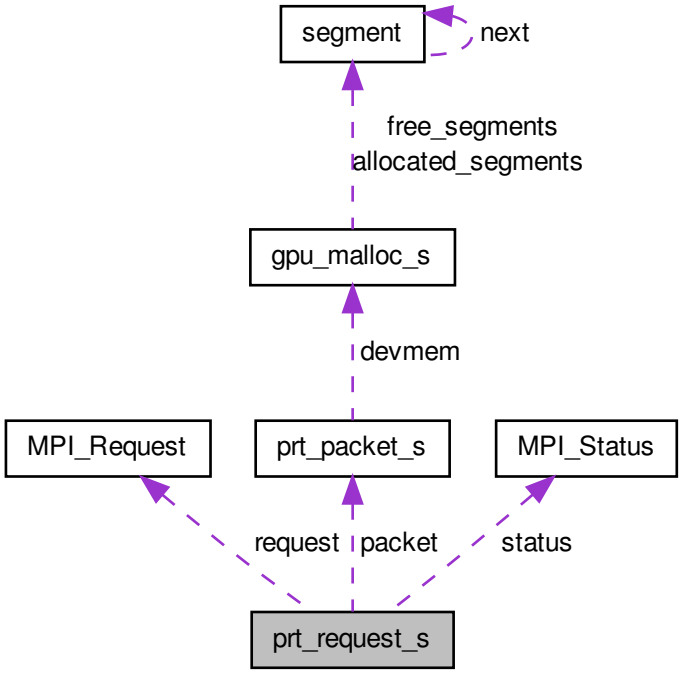
- prt_proxy.h

## 5.17 prt_request_s Struct Reference

MPI communication request for a packet. Contains a packet, some info, MPI request and MPI status.

`#include <prt_request.h>`

Collaboration diagram for prt_request_s:




## Data Fields

- struct prt_packet_s * **packet**
- size_t **size**
- int **peer**
- int **tag**
- MPI_Request **request**
- MPI_Status **status**

### 5.17.1 Detailed Description

MPI communication request for a packet. Contains a packet, some info, MPI request and MPI status.

Definition at line 26 of file prt_request.h.

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

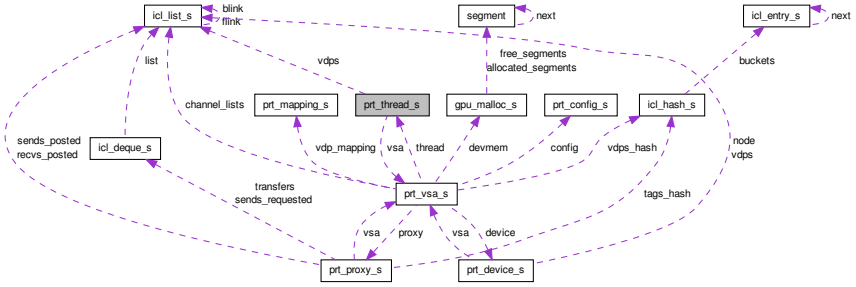
- prt_request.h

## 5.18 prt_thread_s Struct Reference

VSA's worker thread. Represents a single CPU core or a collection of cores.

`#include <prt_thread.h>`

Collaboration diagram for prt_thread_s:




### Data Fields

- struct prt_vsa_s * **vsa**
- int **rank**
- int **core**
- pthread_t **id**
- icl_list_t * **vdps**
- volatile int **finished**
- int **agent_rank**
- double **time**

### 5.18.1 Detailed Description

VSA's worker thread. Represents a single CPU core or a collection of cores.

"finished" is a one-directional synchronization variable. Therefore declered volatile, but no need for atomic access.

Definition at line 29 of file prt_thread.h.

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

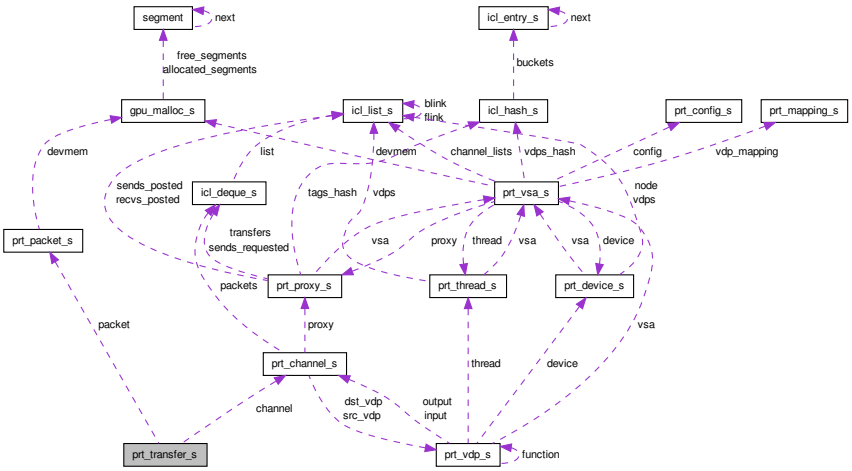
- prt_thread.h

## 5.19 prt_transfer_s Struct Reference

Local transfer object.

`#include <prt_transfer.h>`

Collaboration diagram for prt_transfer_s:




### Data Fields

- struct prt_packet_s * **packet**
- struct prt_channel_s * **channel**
- prt_direction_t **direction**
- int **agent**

### 5.19.1 Detailed Description

Local transfer object.

Definition at line 23 of file prt_transfer.h.

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

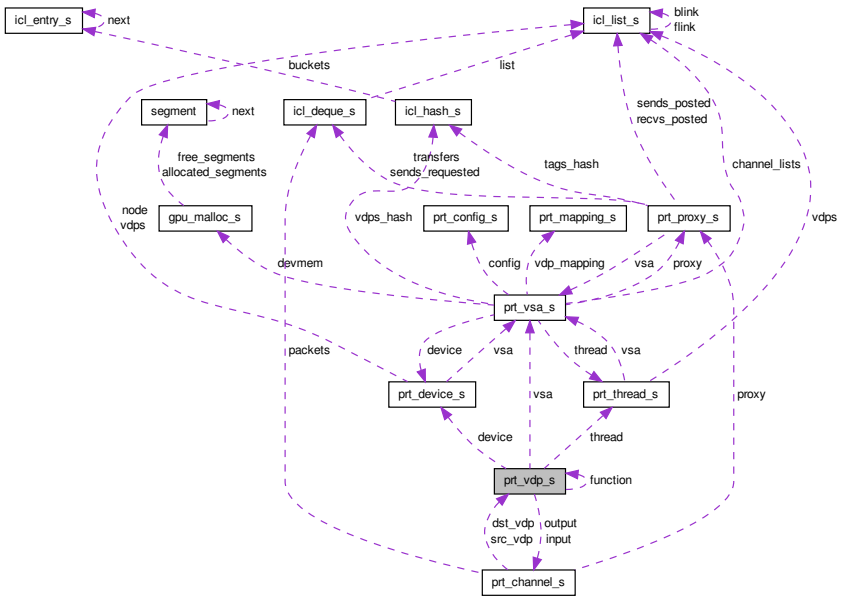
- prt_transfer.h

## 5.20 prt_vdp_s Struct Reference

Virtual Data Processor (VDP). Is uniquely identified by a tuple. Fires for a predefined number of cycles. Has a fixed number of input and output channels. Has a persistent local store. Has access to read-only global store.

`#include <prt_vdp.h>`

Collaboration diagram for prt_vdp_s:




## Data Fields

- prt_location_t **location**
- struct prt_thread_s * **thread**
- struct prt_device_s * **device**
- struct prt_vsa_s * **vsa**
- int * **tuple**
- int **counter**
- int **num_inputs**
- struct prt_channel_s ** **input**
- int **num_outputs**
- struct prt_channel_s ** **output**
- prt_vdp_function_t **function**
- void * **local_store**
- void * **global_store**
- int **color**
- cudaStream_t **stream**

### 5.20.1 Detailed Description

Virtual Data Processor (VDP). Is uniquely identified by a tuple. Fires for a predefined number of cycles. Has a fixed number of input and output channels. Has a persistent local store. Has access to read-only global store.

Definition at line 39 of file prt_vdp.h.

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

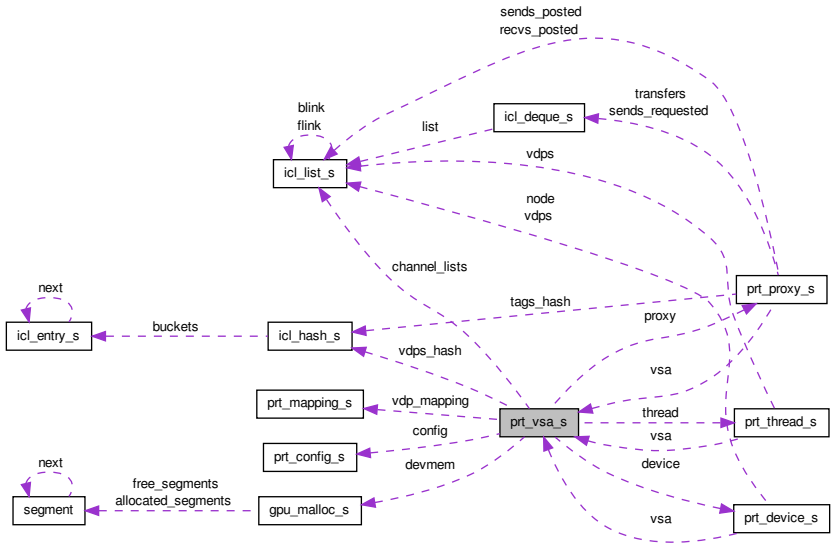
- prt_vdp.h

## 5.21 prt_vsa_s Struct Reference

Virtual Systolic Array (VSA) VSA contains global informationa about the system, a local communication proxy, an array of local worker threads, and an array of local accelerator devices.

`#include <prt_vsa.h>`

Collaboration diagram for prt_vsa_s:




### Data Fields

- int **node_rank**
- int **num_nodes**
- int **num_threads**
- int **num_cores**
- int **concurrency**
- pthread_attr_t **thread_attr**
- struct prt_thread_s ** **thread**
- pthread_barrier_t **barrier**
- void * **global_store**
- prt_vdp_mapping_t **vdp_mapping**
- icl_hash_t * **vdps_hash**
- struct prt_config_s * **config**
- struct prt_proxy_s * **proxy**
- icl_list_t ** **channel_lists**
- void(* **thread_warmup_func** )()
- int **num_devices**
- int **num_accelerators**
- struct prt_device_s ** **device**
- void(* **device_warmup_func** )()
- struct gpu_malloc_s ** **devmem**

### 5.21.1 Detailed Description

Virtual Systolic Array (VSA) VSA contains global informationa about the system, a local communication proxy, an array of local worker threads, and an array of local accelerator devices.

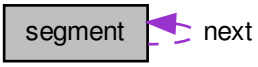
Definition at line 55 of file prt_vsa.h.

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

- prt_vsa.h

## 5.22 segment Struct Reference

Collaboration diagram for segment:




### Data Fields

- int **start_index**
- int **nb_units**
- int **nb_free**
- struct segment * **next**

### 5.22.1 Detailed Description

Definition at line 29 of file gpu_malloc.h.

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

- gpu_malloc.h

# Chapter 6

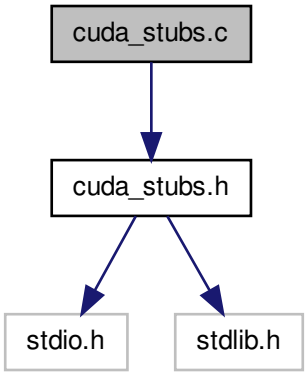
# File Documentation

## 6.1 cuda_stubs.c File Reference

Stubs for a no-CUDA build.

```
#include "cuda_stubs.h"
```

Include dependency graph for cuda_stubs.c:




### Functions

- cudaError_t **cudaSetDevice** (int device)
- cudaError_t **cudaGetDevice** (int ∗device)
- cudaError_t **cudaFree** (void ∗devPtr)
- cudaError_t **cudaMalloc** (void ∗∗devPtr, size_t size)
- cudaError_t **cudaMemGetInfo** (size_t ∗free, size_t ∗total)
- cudaError_t **cudaStreamDestroy** (cudaStream_t stream)
- cudaError_t **cudaStreamCreateWithFlags** (cudaStream_t ∗pStream, unsigned int flags)

- cudaError_t **cudaStreamAddCallback** (cudaStream_t stream, cudaStreamCallback_t callback, void ∗userData, unsigned int flags)
- cudaError_t **cudaEventCreate** (cudaEvent_t ∗event)
- cudaError_t **cudaEventRecord** (cudaEvent_t event, cudaStream_t stream)
- cudaError_t **cudaEventElapsedTime** (float ∗ms, cudaEvent_t start, cudaEvent_t end)
- cudaError_t **cudaMemcpyAsync** (void ∗dst, const void ∗src, size_t count, cudaMemcpyKind kind, cudaStream_t stream)
- cudaError_t **cudaMemcpyPeerAsync** (void ∗dst, int dstDevice, const void ∗src, int srcDevice, size_t count, cudaStream_t stream)
- cudaError_t **cudaDeviceSynchronize** (void)
- const __cudart_builtin__ char ∗ **cudaGetErrorString** (cudaError_t error)

### 6.1.1 Detailed Description

Stubs for a no-CUDA build.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

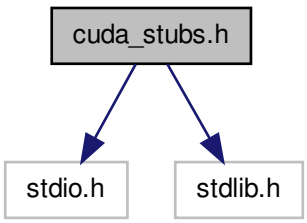
Definition in file cuda_stubs.c.

## 6.2 cuda_stubs.h File Reference

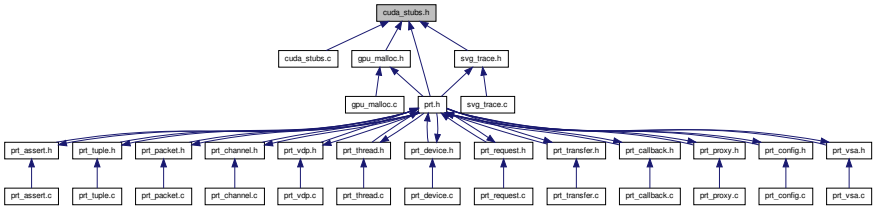
Stubs for a no-CUDA build.

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

Include dependency graph for cuda_stubs.h:



This graph shows which files directly or indirectly include this file:




## Typedefs

- typedef int **cudaError_t**
- typedef int **cudaEvent_t**
- typedef int **cudaStream_t**
- typedef int **cudaMemcpyKind**
- typedef void CUDART_CB(∗ **cudaStreamCallback_t** )(cudaStream_t, cudaError_t, void ∗)

## Enumerations

- enum { **cudaSuccess**, **cudaMemcpyHostToDevice**, **cudaMemcpyDeviceToHost**, **cudaStreamNonBlocking** }

## Functions

- cudaError_t **cudaSetDevice** (int device)
- cudaError_t **cudaGetDevice** (int ∗device)
- cudaError_t **cudaFree** (void ∗devPtr)
- cudaError_t **cudaMalloc** (void ∗∗devPtr, size_t size)
- cudaError_t **cudaMemGetInfo** (size_t ∗free, size_t ∗total)
- cudaError_t **cudaStreamDestroy** (cudaStream_t stream)
- cudaError_t **cudaStreamCreateWithFlags** (cudaStream_t ∗pStream, unsigned int flags)
- cudaError_t **cudaStreamAddCallback** (cudaStream_t stream, cudaStreamCallback_t callback, void ∗userData, unsigned int flags)
- cudaError_t **cudaEventCreate** (cudaEvent_t ∗event)
- cudaError_t **cudaEventRecord** (cudaEvent_t event, cudaStream_t stream)
- cudaError_t **cudaEventElapsedTime** (float ∗ms, cudaEvent_t start, cudaEvent_t end)
- cudaError_t **cudaMemcpyAsync** (void ∗dst, const void ∗src, size_t count, cudaMemcpyKind kind, cudaStream_t stream)
- cudaError_t **cudaMemcpyPeerAsync** (void ∗dst, int dstDevice, const void ∗src, int srcDevice, size_t count, cudaStream_t stream)
- cudaError_t **cudaDeviceSynchronize** (void)
- const __cudart_builtin__ char ∗ **cudaGetErrorString** (cudaError_t error)

### 6.2.1 Detailed Description

Stubs for a no-CUDA build.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

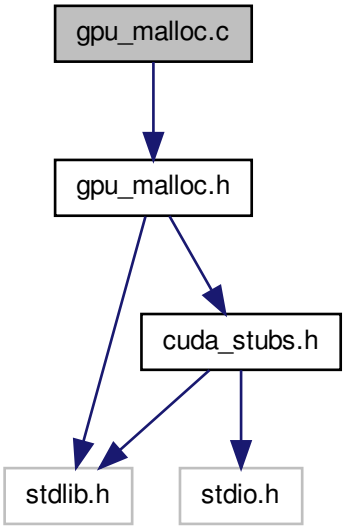
Definition in file cuda_stubs.h.

## 6.3 gpu_malloc.c File Reference

Simple device memory allocator.

```
#include "gpu_malloc.h"
```

Include dependency graph for gpu_malloc.c:




### Functions

- gpu_malloc_t * gpu_malloc_init (int _max_segment, size_t _unit_size)

  *Creates a new device allocator.*
- int gpu_malloc_fini (gpu_malloc_t *gdata)

  *Destroys a device allocator.*
- void * gpu_malloc (gpu_malloc_t *gdata, size_t size)

  *Allocates device memory.*

- int gpu_free (gpu_malloc_t *gdata, void *add)
  *Frees device memory.*

### 6.3.1 Detailed Description

Simple device memory allocator.

**Author**

Aurelien Bouteiller
Thomas Herault
George Bosilca

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file gpu_malloc.c.

### 6.3.2 Function Documentation

#### 6.3.2.1 int gpu_free ( gpu_malloc_t * *gdata*, void * *add* )

Frees device memory.

**Parameters**

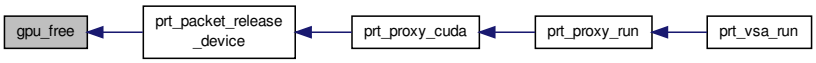
| | |
|---|---|
| *gdata* | – The allocator to use. |
| *add* | – The pointer to the memory to free. |

**Return values**

| | |
|---|---|
| *0* | on success. |
| *-1* | on error. |

Definition at line 143 of file gpu_malloc.c.

Here is the caller graph for this function:




#### 6.3.2.2 void* gpu_malloc ( gpu_malloc_t * *gdata*, size_t *size* )

Allocates device memory.

**Parameters**

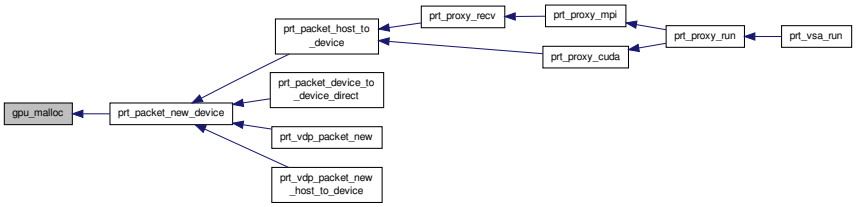
| | |
|---|---|
| *gdata* | – The allocator to use. |
| *size* | – The size in bytes to allocate. |

**Returns**

A pointer to the allocated memory. NULL on error.

Definition at line 106 of file gpu_malloc.c.

Here is the caller graph for this function:




#### 6.3.2.3 int gpu_malloc_fini ( gpu_malloc_t * *gdata* )

Destroys a device allocator.

**Parameters**

| | |
|---|---|
| *gdata* | – The allocator to destroy. |

**Return values**

| | |
|---|---|
| *0* | on success. |
| *-1* | on error. |

Definition at line 73 of file gpu_malloc.c.

Here is the caller graph for this function:


#### 6.3.2.4 gpu_malloc_t* gpu_malloc_init ( int *_max_segment,* size_t *_unit_size* )

Creates a new device allocator.

**Parameters**

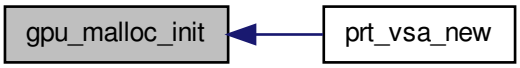
| | |
|---|---|
| *_max_segment* | – The maximum number of segments. |
| *_unit_size* | – The size of each segment. |

**Returns**

A new allocator. NULL on error.

Definition at line 24 of file gpu_malloc.c.

Here is the caller graph for this function:

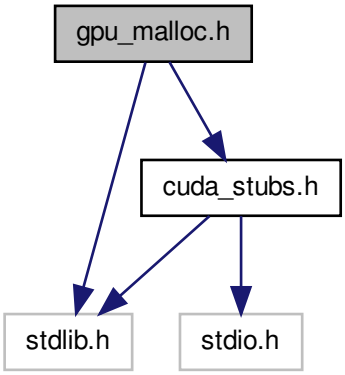



## 6.4 gpu_malloc.h File Reference

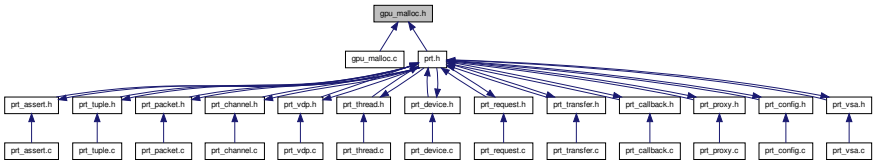
Simple device memory allocator.

```
#include <stdlib.h>
#include "cuda_stubs.h"
```

Include dependency graph for gpu_malloc.h:



This graph shows which files directly or indirectly include this file:




## Data Structures

- struct segment
- struct gpu_malloc_s

## Typedefs

- typedef struct segment **segment_t**
- typedef struct gpu_malloc_s **gpu_malloc_t**

## Functions

- gpu_malloc_t ∗ gpu_malloc_init (int max_segment, size_t unit_size)

  *Creates a new device allocator.*
- int gpu_malloc_fini (gpu_malloc_t ∗gdata)

  *Destroys a device allocator.*
- void ∗ gpu_malloc (gpu_malloc_t ∗gdata, size_t size)

  *Allocates device memory.*
- int gpu_free (gpu_malloc_t ∗gdata, void ∗ptr)

  *Frees device memory.*

### 6.4.1 Detailed Description

Simple device memory allocator.

**Author**

Aurelien Bouteiller
Thomas Herault
George Bosilca

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file gpu_malloc.h.

### 6.4.2 Function Documentation

#### 6.4.2.1 int gpu_free ( gpu_malloc_t ∗ *gdata*, void ∗ *add* )

Frees device memory.

**Parameters**

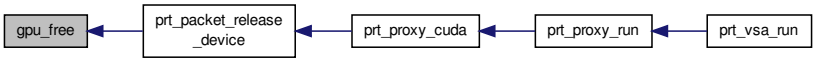
| | |
|---|---|
| *gdata* | – The allocator to use. |
| *add* | – The pointer to the memory to free. |

**Return values**

| | |
|---|---|
| *0* | on success. |
| *-1* | on error. |

Definition at line 143 of file gpu_malloc.c.

Here is the caller graph for this function:




#### 6.4.2.2 void* gpu_malloc ( gpu_malloc_t * *gdata*, size_t *size* )

Allocates device memory.

**Parameters**

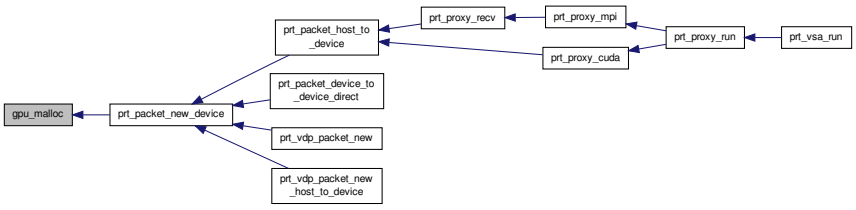
| | |
|---|---|
| *gdata* | – The allocator to use. |
| *size* | – The size in bytes to allocate. |

**Returns**

A pointer to the allocated memory. NULL on error.

Definition at line 106 of file gpu_malloc.c.

Here is the caller graph for this function:




#### 6.4.2.3 int gpu_malloc_fini ( gpu_malloc_t * *gdata* )

Destroys a device allocator.

**Parameters**

| | |
|---|---|
| *gdata* | – The allocator to destroy. |

**Return values**

| | |
|---|---|
| *0* | on success. |
| *-1* | on error. |

Definition at line 73 of file gpu_malloc.c.

Here is the caller graph for this function:


#### 6.4.2.4 gpu_malloc_t* gpu_malloc_init ( int *_max_segment*, size_t *_unit_size* )

Creates a new device allocator.

**Parameters**

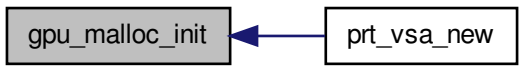
| | |
|---|---|
| *_max_segment* | – The maximum number of segments. |
| *_unit_size* | – The size of each segment. |

**Returns**

A new allocator. NULL on error.

Definition at line 24 of file gpu_malloc.c.

Here is the caller graph for this function:

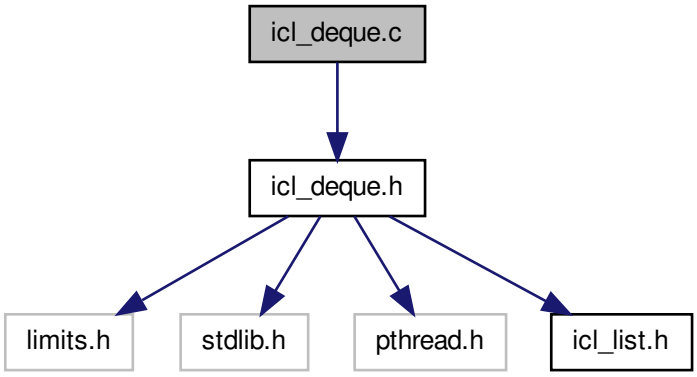



## 6.5 icl_deque.c File Reference

Thread-safe double-ended queue.

```
#include "icl_deque.h"
```

Include dependency graph for icl_deque.c:




## Functions

- icl_deque_t * icl_deque_new ()

  *Creates a new deque.*
- int icl_deque_destroy (icl_deque_t *deque, void(*free_func)(void *))

  *Destroys a deque.*
- icl_node_t * icl_deque_first (icl_deque_t *deque)

  *Returns the first node in a deque.*
- icl_node_t * icl_deque_next (icl_deque_t *deque, icl_node_t *node)

  *Returns next node in a deque.*
- icl_node_t * icl_deque_append (icl_deque_t *deque, void *data)

  *Inserts a node at the end of a deque.*
- icl_node_t * icl_deque_prepend (icl_deque_t *deque, void *data)

  *Inserts a node at the front of a deque.*
- int icl_deque_delete (icl_deque_t *deque, icl_node_t *node, void(*free_func)(void *))

  *Deletes a node from a deque.*
- int icl_deque_size (icl_deque_t *deque)

  *Returns the size of a deque.*

### 6.5.1 Detailed Description

Thread-safe double-ended queue.

**Author**

Jakub Kurzak

Implemented by protecting access to icl_list using spinlocks. Also, unlike icl_list, icl_deque keeps track of its size.

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file icl_deque.c.

### 6.5.2 Function Documentation

#### 6.5.2.1 icl_node_t* icl_deque_append ( icl_deque_t * *deque*, void * *data* )

Inserts a node at the end of a deque.

**Parameters**

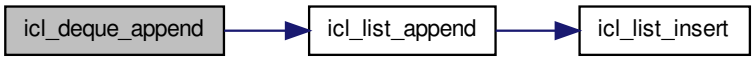
| | |
|---|---|
| *deque* | – The deque to append to. |
| *data* | – The data to append. |

**Returns**

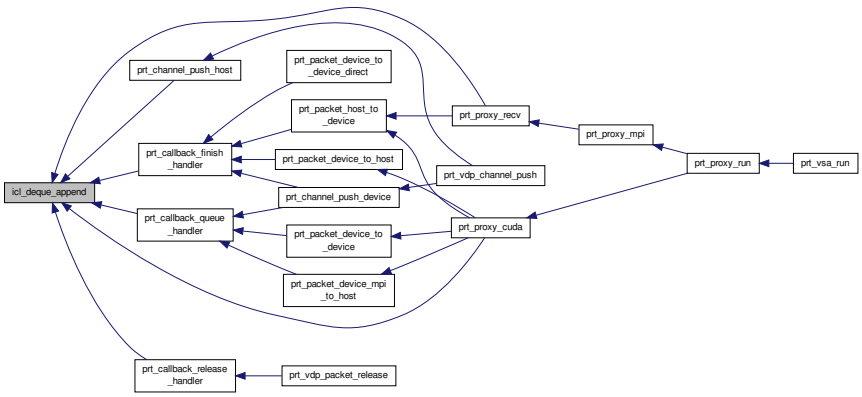
The new node. NULL on error.

Definition at line 117 of file icl_deque.c.

Here is the call graph for this function:




Here is the caller graph for this function:



#### 6.5.2.2 int icl_deque_delete ( icl_deque_t * *deque*, icl_node_t * *node*, void(*)(void *) *free_func* )

Deletes a node from a deque.

**Parameters**

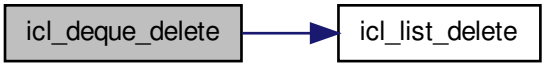
| | |
|---|---|
| *deque* | – The deque to delete from. |
| *node* | – The node to delete. |
| *free_func* | – The function that frees the node's data. |

**Return values**

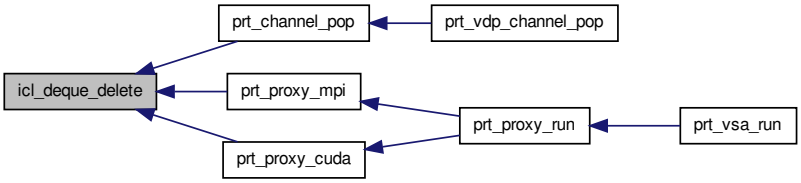
| | |
|---|---|
| *0* | on success. |
| *-1* | on failure. |

Definition at line 163 of file icl_deque.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.5.2.3 int icl_deque_destroy ( icl_deque_t * *deque*, void(*)(void *) *free_func* )

Destroys a deque.

**Parameters**

| | |
|---|---|
| *deque* | – The deque to destroy. |

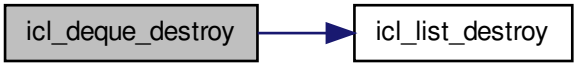
| | |
|---|---|
| *free_func* | – The function that frees the node's data. |

**Return values**

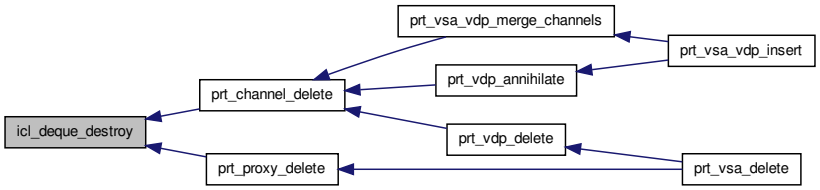
| | |
|---|---|
| *0* | on success. |
| *-1* | on failure. |

Definition at line 52 of file icl_deque.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.5.2.4 icl_node_t* icl_deque_first ( icl_deque_t * *deque* )

Returns the first node in a deque.

**Parameters**

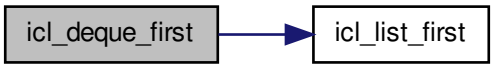
| | |
|---|---|
| *deque* | – The deque to fetch from. |

**Returns**

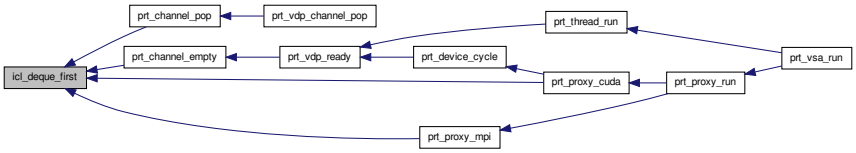
The node at the front of the deque. NULL if empty or error.

Definition at line 75 of file icl_deque.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.5.2.5 icl_deque_t* icl_deque_new ( )

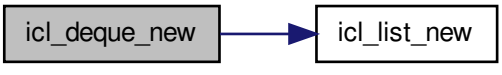
Creates a new deque.

**Returns**

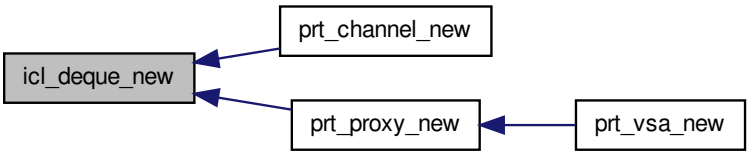
A new deque. NULL on error.

Definition at line 22 of file icl_deque.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.5.2.6 icl_node_t* icl_deque_next ( icl_deque_t * *deque*, icl_node_t * *node* )

Returns next node in a deque.

**Parameters**

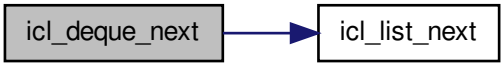
| | |
|---|---|
| *deque* | – The deque to fetch from. |
| *node* | – The node current node. |

**Returns**

The next node. NULL if empty or error.

Definition at line 96 of file icl_deque.c.

Here is the call graph for this function:




#### 6.5.2.7 icl_node_t* icl_deque_prepend ( icl_deque_t * *deque*, void * *data* )

Inserts a node at the front of a deque.

**Parameters**

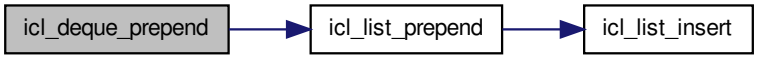
| | |
|---|---|
| *deque* | – The deque to prepend to. |
| *data* | – The data to prepent. |

**Returns**

The new node. NULL on error.

Definition at line 139 of file icl_deque.c.

Here is the call graph for this function:




#### 6.5.2.8 int icl_deque_size ( icl_deque_t * *deque* )

Returns the size of a deque.

**Parameters**

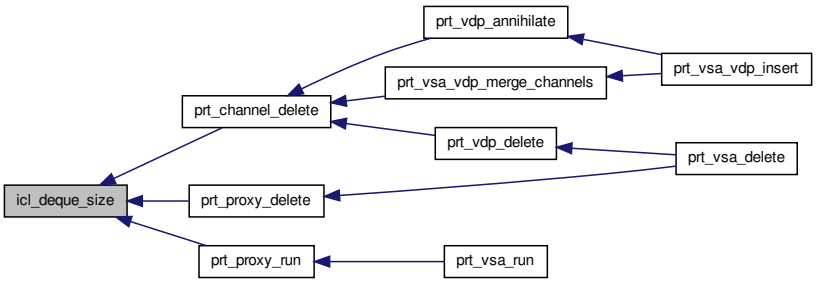
| | |
|---|---|
| *deque* | – The deque to get size of. |

**Returns**

– The size of the deque. -1 on error.

Definition at line 189 of file icl_deque.c.

Here is the caller graph for this function:

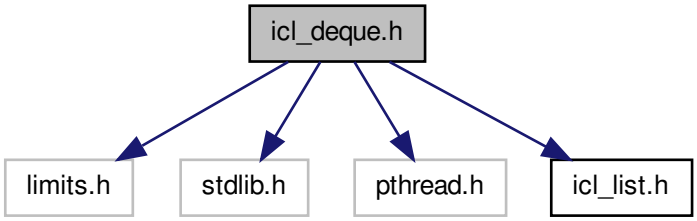



## 6.6 icl_deque.h File Reference

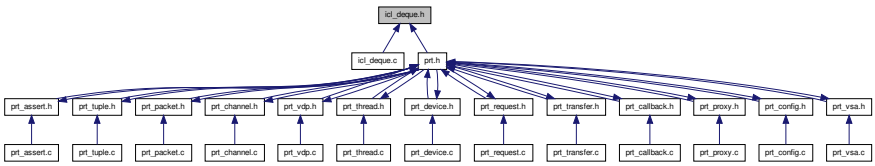
Thread-safe double-ended queue.

```
#include <limits.h>
#include <stdlib.h>
#include <pthread.h>
#include "icl_list.h"
```

Include dependency graph for icl_deque.h:




This graph shows which files directly or indirectly include this file:




## Data Structures

- struct icl_deque_s

## Typedefs

- typedef icl_list_t **icl_node_t**
- typedef struct icl_deque_s **icl_deque_t**

## Functions

- icl_deque_t * icl_deque_new ()

  *Creates a new deque.*
- int icl_deque_destroy (icl_deque_t *deque, void(*free_func)(void *))

  *Destroys a deque.*
- icl_node_t * icl_deque_first (icl_deque_t *deque)

  *Returns the first node in a deque.*
- icl_node_t * icl_deque_next (icl_deque_t *deque, icl_node_t *node)

*Returns next node in a deque.*

- icl_node_t * icl_deque_append (icl_deque_t *deque, void *data)

  *Inserts a node at the end of a deque.*

- icl_node_t * icl_deque_prepend (icl_deque_t *deque, void *data)

  *Inserts a node at the front of a deque.*

- int icl_deque_delete (icl_deque_t *deque, icl_node_t *node, void(*free_func)(void *))

  *Deletes a node from a deque.*

- int icl_deque_size (icl_deque_t *deque)

  *Returns the size of a deque.*

### 6.6.1 Detailed Description

Thread-safe double-ended queue.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file icl_deque.h.

### 6.6.2 Function Documentation

#### 6.6.2.1 icl_node_t* icl_deque_append ( icl_deque_t * *deque*, void * *data* )

Inserts a node at the end of a deque.

**Parameters**

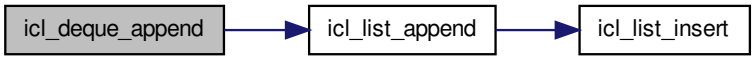
| | |
|---|---|
| *deque* | – The deque to append to. |
| *data* | – The data to append. |

**Returns**

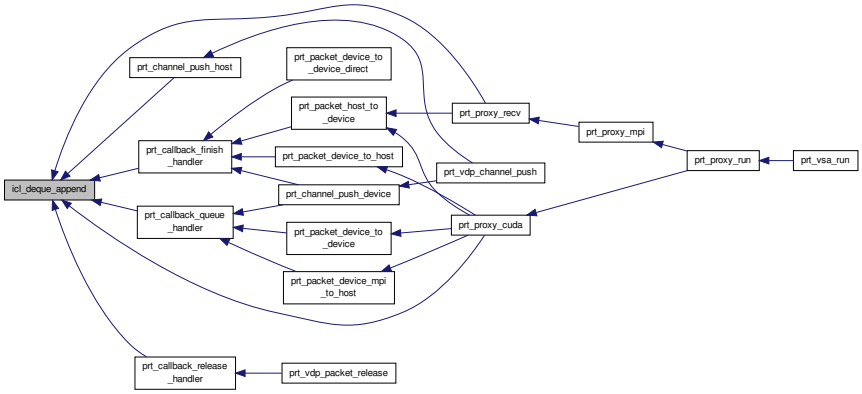
The new node. NULL on error.

Definition at line 117 of file icl_deque.c.

Here is the call graph for this function:



Here is the caller graph for this function:




### 6.6.2.2 int icl_deque_delete ( icl_deque_t * *deque*, icl_node_t * *node*, void(*)(void *) *free_func* )

Deletes a node from a deque.

**Parameters**

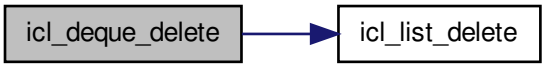
| | |
|---|---|
| *deque* | – The deque to delete from. |
| *node* | – The node to delete. |
| *free_func* | – The function that frees the node's data. |

**Return values**

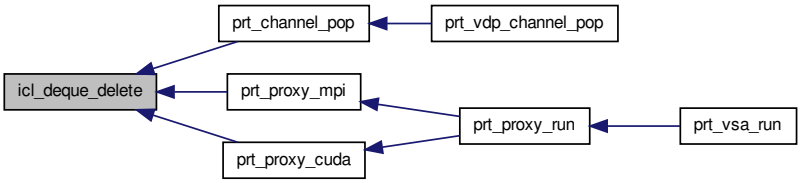
| | |
|---|---|
| *0* | on success. |
| *-1* | on failure. |

Definition at line 163 of file icl_deque.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.6.2.3 int icl_deque_destroy ( icl_deque_t * *deque*, void(*)(void *) *free_func* )

Destroys a deque.

**Parameters**

| | |
|---|---|
| *deque* | – The deque to destroy. |
| *free_func* | – The function that frees the node's data. |

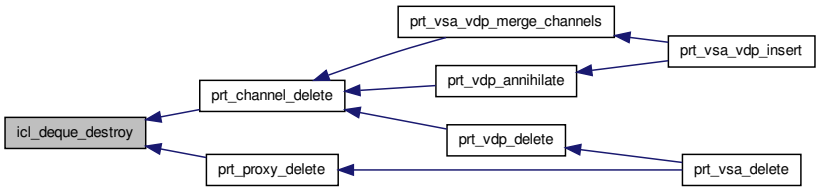
**Return values**

| | |
|---|---|
| *0* | on success. |
| *-1* | on failure. |

Definition at line 52 of file icl_deque.c.

Here is the call graph for this function:


Here is the caller graph for this function:



#### 6.6.2.4 icl_node_t* icl_deque_first ( icl_deque_t * *deque* )

Returns the first node in a deque.

**Parameters**

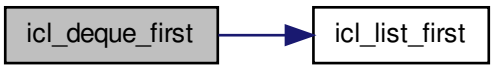
| | |
|---|---|
| *deque* | – The deque to fetch from. |

**Returns**

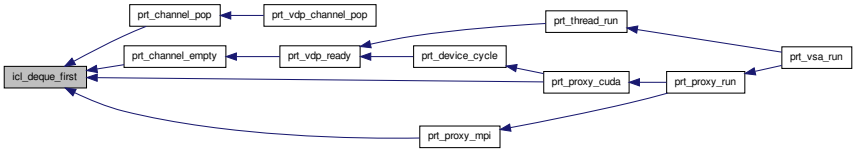
The node at the front of the deque. NULL if empty or error.

Definition at line 75 of file icl_deque.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.6.2.5 icl_deque_t* icl_deque_new ( )

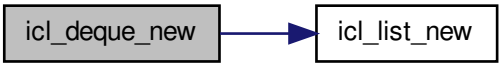
Creates a new deque.

**Returns**

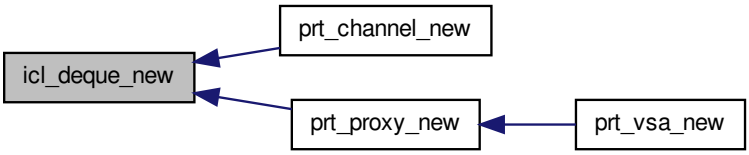
A new deque. NULL on error.

Definition at line 22 of file icl_deque.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.6.2.6 icl_node_t* icl_deque_next ( icl_deque_t * *deque*, icl_node_t * *node* )

Returns next node in a deque.

**Parameters**

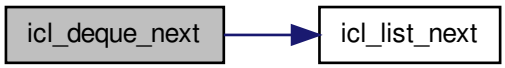
| | |
|---|---|
| *deque* | – The deque to fetch from. |
| *node* | – The node current node. |

**Returns**

The next node. NULL if empty or error.

Definition at line 96 of file icl_deque.c.

Here is the call graph for this function:




#### 6.6.2.7 icl_node_t* icl_deque_prepend ( icl_deque_t * *deque*, void * *data* )

Inserts a node at the front of a deque.

**Parameters**

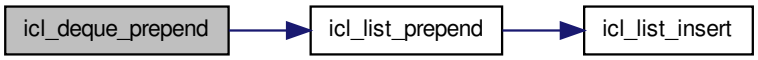
| | |
|---|---|
| *deque* | – The deque to prepend to. |
| *data* | – The data to prepent. |

**Returns**

The new node. NULL on error.

Definition at line 139 of file icl_deque.c.

Here is the call graph for this function:




#### 6.6.2.8 int icl_deque_size ( icl_deque_t * *deque* )

Returns the size of a deque.

**Parameters**

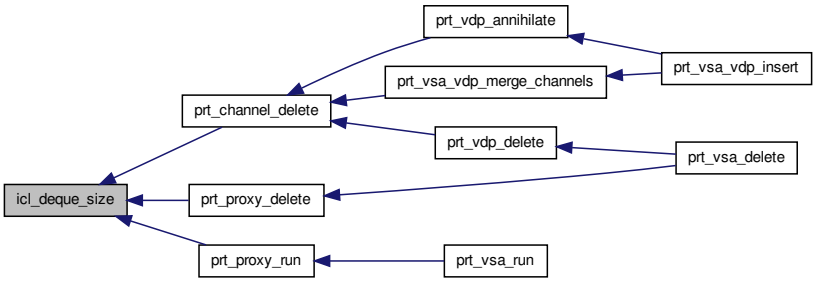
| | |
|---|---|
| *deque* | – The deque to get size of. |

**Returns**

– The size of the deque. -1 on error.

Definition at line 189 of file icl_deque.c.

Here is the caller graph for this function:

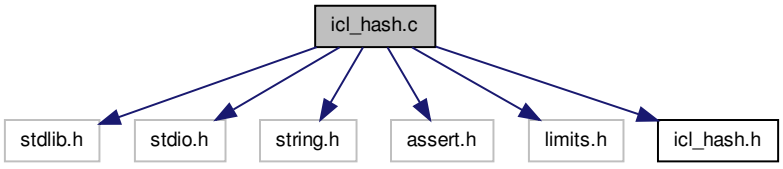



## 6.7 icl_hash.c File Reference

Dependency-free hash table.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <limits.h>
#include "icl_hash.h"
```

Include dependency graph for icl_hash.c:




### Macros

- #define **BITS_IN_int** (sizeof(int) ∗ CHAR_BIT)

- #define **THREE_QUARTERS** ((int)((BITS_IN_int * 3) / 4))
- #define **ONE_EIGHTH** ((int)(BITS_IN_int / 8))
- #define **HIGH_BITS** (~((unsigned int)(~0) >> ONE_EIGHTH))

### Functions

- icl_hash_t * icl_hash_create (int nbuckets, unsigned int(*hash_function)(void *), int(*hash_key_compare)(void *, void *))

  *Creates a new hash table.*
- void * icl_hash_find (icl_hash_t *ht, void *key)

  *Searches for an entry in a hash table.*
- icl_entry_t * icl_hash_insert (icl_hash_t *ht, void *key, void *data)

  *Inserts an item into a hash table.*
- icl_entry_t * icl_hash_update_insert (icl_hash_t *ht, void *key, void *data, void **olddata)

  *Replaces an entry in a hash table with a given entry.*
- int icl_hash_delete (icl_hash_t *ht, void *key, void(*free_key)(void *), void(*free_data)(void *))

  *Frees one hash table entry located by a key. Key and data are freed using functions.*
- int icl_hash_destroy (icl_hash_t *ht, void(*free_key)(void *), void(*free_data)(void *))

  *Destroys a hash table. Keys and data are freed using functions.*
- int icl_hash_dump (FILE *stream, icl_hash_t *ht)

  *Dumps the hash table's contents to the given file pointer.*

### 6.7.1 Detailed Description

Dependency-free hash table.

**Author**

Keith Seymour

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file icl_hash.c.

### 6.7.2 Function Documentation

#### 6.7.2.1 icl_hash_t* icl_hash_create ( int *nbuckets*, unsigned int(*)(void *) *hash_function*, int(*)(void *, void *) *hash_key_compare* )

Creates a new hash table.

**Parameters**

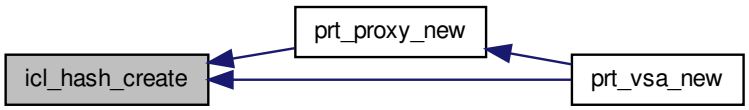
| | |
|---|---|
| *nbuckets* | – The number of buckets to create. |
| *hash_function* | – The pointer to the hashing function. |
| *hash_key_compare* | – The pointer to the hash key comparison function. |

**Returns**

A pointer to new hash table.

Definition at line 71 of file icl_hash.c.

Here is the caller graph for this function:




#### 6.7.2.2 int icl_hash_delete ( icl_hash_t * *ht*, void * *key*, void(*)(void *) *free_key*, void(*)(void *) *free_data* )

Frees one hash table entry located by a key. Key and data are freed using functions.

**Parameters**

| | |
|---|---|
| *ht* | – The hash table. |
| *key* | – The key of the item to be deleted. |
| *free_key* | – The pointer to the function that frees the key. |
| *free_data* | – The pointer to the function that frees the data. |

**Return values**

| | |
|---|---|
| *0* | on success. |
| *-1* | on failure. |

Definition at line 234 of file icl_hash.c.

#### 6.7.2.3 int icl_hash_destroy ( icl_hash_t * *ht*, void(*)(void *) *free_key*, void(*)(void *) *free_data* )

Destroys a hash table. Keys and data are freed using functions.

**Parameters**

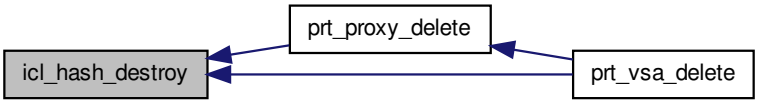
| | |
|---|---|
| *ht* | – The hash table to destroy. |
| *free_key* | – The pointer to function that frees the keys. |
| *free_data* | – The pointer to function that frees the data. |

**Return values**

| | |
|---|---|
| *0* | on success. |
| *-1* | on failure. |

Definition at line 282 of file icl_hash.c.

Here is the caller graph for this function:




#### 6.7.2.4 int icl_hash_dump ( FILE * *stream*, icl_hash_t * *ht* )

Dumps the hash table's contents to the given file pointer.

**Parameters**

| | |
|---|---|
| *stream* | – The file to dump the hash table to. |
| *ht* | – The hash table to be dumped. |

**Return values**

| | |
|---|---|
| *0* | on success. |
| *-1* | on failure. |

Definition at line 323 of file icl_hash.c.

#### 6.7.2.5 void* icl_hash_find ( icl_hash_t * *ht*, void * *key* )

Searches for an entry in a hash table.

**Parameters**

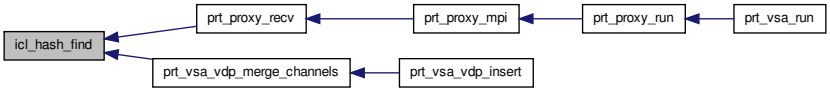
| | |
|---|---|
| *ht* | – The hash table to be searched. |
| *key* | – The key of the item to search for. |

**Returns**

A pointer to the data corresponding to the key. NULL if the key is not found.

Definition at line 109 of file icl_hash.c.

Here is the caller graph for this function:



#### 6.7.2.6 icl_entry_t* icl_hash_insert ( icl_hash_t * *ht,* void * *key,* void * *data* )

Inserts an item into a hash table.

**Parameters**

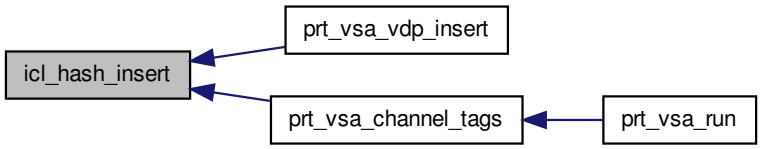
| | |
|---|---|
| *ht* | – The hash table. |
| *key* | – The key of the new item. |
| *data* | – The pointer to the new item's data. |

**Returns**

A pointer to the new item. NULL on error.

Definition at line 135 of file icl_hash.c.

Here is the caller graph for this function:




#### 6.7.2.7 icl_entry_t* icl_hash_update_insert ( icl_hash_t * *ht*, void * *key*, void * *data*, void ** *olddata* )

Replaces an entry in a hash table with a given entry.

**Parameters**

| | |
|---|---|
| *ht* | – The hash table. |
| *key* | – The key of the new item. |
| *data* | – The pointer to the new item's data. |
| *olddata* | – The pointer to the old item's data (set upon return). |

**Returns**

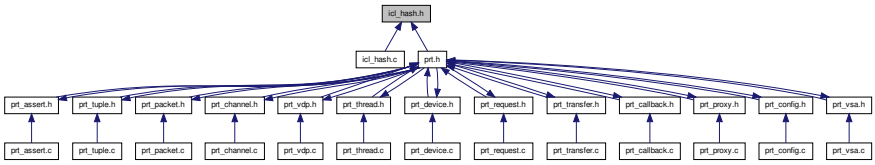
A pointer to the new item. NULL on error.

Definition at line 174 of file icl_hash.c.

## 6.8 icl_hash.h File Reference

Dependency-free hash table.

This graph shows which files directly or indirectly include this file:




## Data Structures

- struct icl_entry_s
- struct icl_hash_s

## Macros

- #define **icl_hash_foreach**(ht, tmpint, tmpent, kp, dp)

## Typedefs

- typedef struct icl_entry_s **icl_entry_t**
- typedef struct icl_hash_s **icl_hash_t**

## Functions

- icl_hash_t * icl_hash_create (int nbuckets, unsigned int(*hash_function)(void *), int(*hash_key_compare)(void *, void *))

  *Creates a new hash table.*
- void * icl_hash_find (icl_hash_t *, void *)

  *Searches for an entry in a hash table.*
- icl_entry_t * icl_hash_insert (icl_hash_t *, void *, void *)

  *Inserts an item into a hash table.*
- icl_entry_t * icl_hash_update_insert (icl_hash_t *, void *, void *, void **)

  *Replaces an entry in a hash table with a given entry.*
- int icl_hash_destroy (icl_hash_t *, void(*)(void *), void(*)(void *))

  *Destroys a hash table. Keys and data are freed using functions.*
- int icl_hash_dump (FILE *, icl_hash_t *)

  *Dumps the hash table's contents to the given file pointer.*
- int icl_hash_delete (icl_hash_t *, void *, void(*)(void *), void(*)(void *))

  *Frees one hash table entry located by a key. Key and data are freed using functions.*

### 6.8.1 Detailed Description

Dependency-free hash table.

**Author**

Keith Seymour

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file icl_hash.h.

### 6.8.2 Macro Definition Documentation

#### 6.8.2.1 #define icl_hash_foreach( *ht, tmpint, tmpent, kp, dp* )

**Value:**

```
for (tmpint=0;tmpint<ht->nbuckets; tmpint++)      \
        for (tmpent=ht->buckets[tmpint];             \
             tmpent!=NULL&&((kp=tmpent->key)!=NULL)&&((dp=tmpent->data)!=NULL); \
             tmpent=tmpent->next)
```

Definition at line 43 of file icl_hash.h.

### 6.8.3 Function Documentation

#### 6.8.3.1 icl_hash_t* icl_hash_create ( int *nbuckets*, unsigned int(*)(void *) *hash_function*, int(*)(void *, void *) *hash_key_compare* )

Creates a new hash table.

**Parameters**

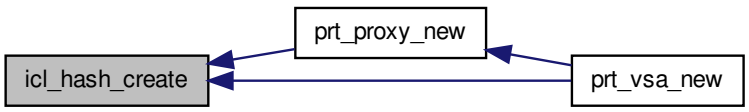
| | |
|---|---|
| *nbuckets* | – The number of buckets to create. |
| *hash_function* | – The pointer to the hashing function. |
| *hash_key_compare* | – The pointer to the hash key comparison function. |

**Returns**

A pointer to new hash table.

Definition at line 71 of file icl_hash.c.

Here is the caller graph for this function:



#### 6.8.3.2 int icl_hash_delete ( icl_hash_t * *ht,* void * *key,* void(*)(void *) *free_key,* void(*)(void *) *free_data* )

Frees one hash table entry located by a key. Key and data are freed using functions.

**Parameters**

| | |
|---|---|
| *ht* | – The hash table. |
| *key* | – The key of the item to be deleted. |
| *free_key* | – The pointer to the function that frees the key. |
| *free_data* | – The pointer to the function that frees the data. |

**Return values**

| | |
|---|---|
| *0* | on success. |
| *-1* | on failure. |

Definition at line 234 of file icl_hash.c.

#### 6.8.3.3 int icl_hash_destroy ( icl_hash_t * *ht*, void(*)(void *) *free_key*, void(*)(void *) *free_data* )

Destroys a hash table. Keys and data are freed using functions.

**Parameters**

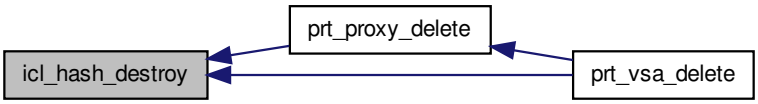
| | |
|---|---|
| *ht* | – The hash table to destroy. |
| *free_key* | – The pointer to function that frees the keys. |
| *free_data* | – The pointer to function that frees the data. |

**Return values**

| | |
|---|---|
| *0* | on success. |
| *-1* | on failure. |

Definition at line 282 of file icl_hash.c.

Here is the caller graph for this function:




#### 6.8.3.4 int icl_hash_dump ( FILE * *stream*, icl_hash_t * *ht* )

Dumps the hash table's contents to the given file pointer.

**Parameters**

| | |
|---|---|
| *stream* | – The file to dump the hash table to. |
| *ht* | – The hash table to be dumped. |

**Return values**

| | |
|---|---|
| *0* | on success. |
| *-1* | on failure. |

Definition at line 323 of file icl_hash.c.

#### 6.8.3.5 void* icl_hash_find ( icl_hash_t * *ht,* void * *key* )

Searches for an entry in a hash table.

**Parameters**

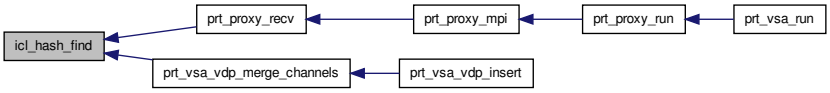
| | |
|---|---|
| *ht* | – The hash table to be searched. |
| *key* | – The key of the item to search for. |

**Returns**

A pointer to the data corresponding to the key. NULL if the key is not found.

Definition at line 109 of file icl_hash.c.

Here is the caller graph for this function:




#### 6.8.3.6 icl_entry_t* icl_hash_insert ( icl_hash_t * *ht,* void * *key,* void * *data* )

Inserts an item into a hash table.

**Parameters**

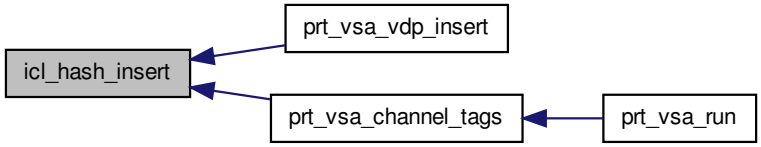
| | |
|---|---|
| *ht* | – The hash table. |
| *key* | – The key of the new item. |
| *data* | – The pointer to the new item's data. |

**Returns**

A pointer to the new item. NULL on error.

Definition at line 135 of file icl_hash.c.

Here is the caller graph for this function:




#### 6.8.3.7 icl_entry_t* icl_hash_update_insert ( icl_hash_t * *ht,* void * *key,* void * *data,* void ** *olddata* )

Replaces an entry in a hash table with a given entry.

**Parameters**

| | |
|---|---|
| *ht* | – The hash table. |
| *key* | – The key of the new item. |
| *data* | – The pointer to the new item's data. |
| *olddata* | – The pointer to the old item's data (set upon return). |

**Returns**

A pointer to the new item. NULL on error.

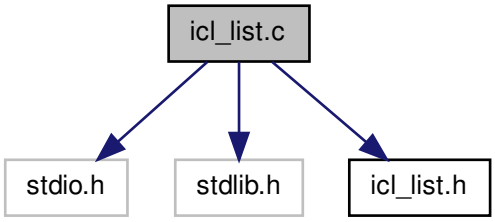
Definition at line 174 of file icl_hash.c.

## 6.9 icl_list.c File Reference

Dependency-free linked list.

```
#include <stdio.h>
#include <stdlib.h>
#include "icl_list.h"
```

Include dependency graph for icl_list.c:




## Functions

- icl_list_t * icl_list_new ()

  *Creates a new linked list.*
- icl_list_t * icl_list_insert (icl_list_t *head, icl_list_t *pos, void *data)

  *Inserts a new node after the specified node.*
- int icl_list_delete (icl_list_t *head, icl_list_t *pos, void(*free_function)(void *))

  *Deletes the specified node.*
- icl_list_t * icl_list_search (icl_list_t *head, void *data, int(*compare)(void *, void *))

  *Finds a data item in a linked list.*
- icl_list_t * icl_list_isort (icl_list_t *head, void *data, int(*compare)(void *, void *))

  *Inserts data into a sorted list. Does not support direct comparison of pointers.*
- int icl_list_destroy (icl_list_t *head, void(*free_function)(void *))

  *Destroys a linked list.*
- int icl_list_size (icl_list_t *head)

  *Returns the number of items in a linked list.*
- icl_list_t * icl_list_first (icl_list_t *head)

  *Returns the first item in a linked list.*
- icl_list_t * icl_list_last (icl_list_t *head)

  *Returns the last item in a linked list.*
- icl_list_t * icl_list_next (icl_list_t *head, icl_list_t *pos)

  *Returns the node following the specified node.*
- icl_list_t * icl_list_prev (icl_list_t *head, icl_list_t *pos)

  *Returns the node preceding the specified node.*
- icl_list_t * icl_list_concat (icl_list_t *head1, icl_list_t *head2)

  *Concatenates two linked lists.*
- icl_list_t * icl_list_prepend (icl_list_t *head, void *data)

  *Inserts a node at the beginning of a list.*
- icl_list_t * icl_list_append (icl_list_t *head, void *data)

  *Inserts a node at the end of a list.*

### 6.9.1 Detailed Description

Dependency-free linked list.

**Author**

Keith Seymour

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file icl_list.c.

### 6.9.2 Function Documentation

#### 6.9.2.1 icl_list_t* icl_list_append ( icl_list_t * *head*, void * *data* )

Inserts a node at the end of a list.

**Parameters**

| | |
|---|---|
| *head* | – The linked list. |
| *data* | – The data to be inserted. |

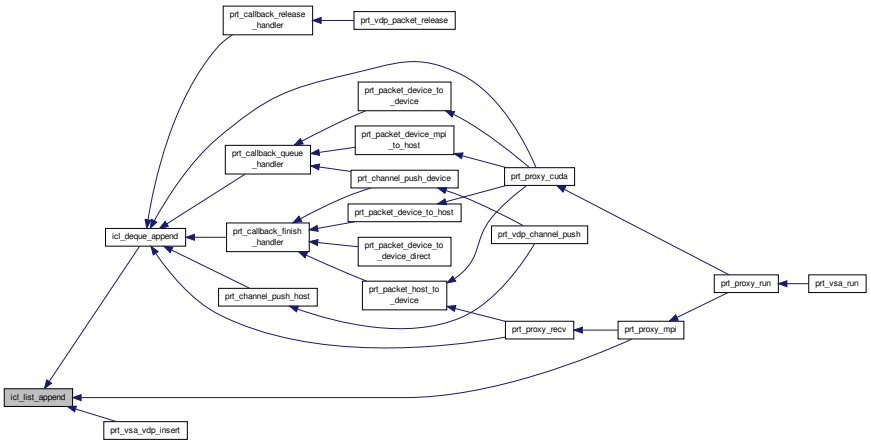
**Returns**

A pointer to the new node. NULL on error.

Definition at line 326 of file icl_list.c.

Here is the call graph for this function:

Here is the caller graph for this function:




#### 6.9.2.2 icl_list_t* icl_list_concat ( icl_list_t * *head1,* icl_list_t * *head2* )

Concatenates two linked lists.

**Parameters**

| | |
|---|---|
| *head1* | – The first linked list. |
| *head2* | – The second linked list. |

**Returns**

A pointer to the new linked list, which consists of <head1,head2>. NULL on error.

Definition at line 290 of file icl_list.c.

#### 6.9.2.3 int icl_list_delete ( icl_list_t * *head,* icl_list_t * *pos,* void(*)(void *) *free_function* )

Deletes the specified node.

**Parameters**

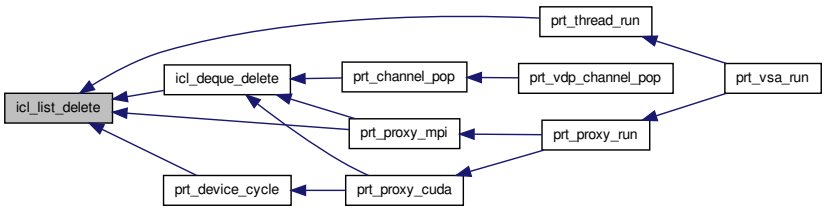
| | |
|---|---|
| *head* | – The linked list containing the node to be deleted. |
| *pos* | – The node to be deleted. |
| *free_function* | – The function that frees the node's data. |

**Return values**

| | |
|---|---|
| *0* | on success. |
| *-1* | on failure. |

Definition at line 82 of file icl_list.c.

Here is the caller graph for this function:




#### 6.9.2.4 int icl_list_destroy ( icl_list_t * *head*, void(*)(void *) *free_function* )

Destroys a linked list.

**Parameters**

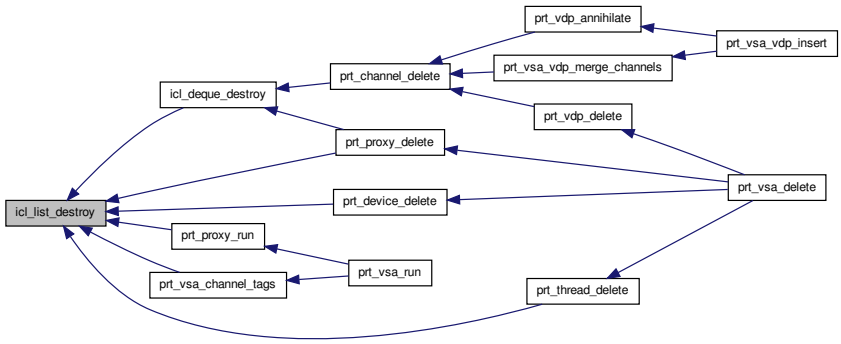
| | |
|---|---|
| *head* | – The linked list to be destroyed. |
| *free_function* | – The function that frees the node's data. |

**Return values**

| | |
|---|---|
| *0* | on success. |
| *-1* | on failure. |

Definition at line 173 of file icl_list.c.

Here is the caller graph for this function:




#### 6.9.2.5 icl_list_t* icl_list_first ( icl_list_t * *head* )

Returns the first item in a linked list.

**Parameters**

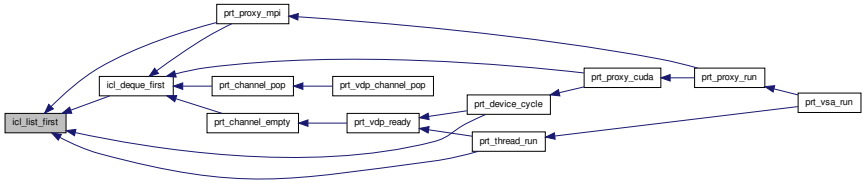
| | |
|---|---|
| *head* | – The linked list. |

**Returns**

A pointer to the first item. NULL on error.

Definition at line 221 of file icl_list.c.

Here is the caller graph for this function:




### 6.9.2.6 icl_list_t* icl_list_insert ( icl_list_t * *head*, icl_list_t * *pos*, void * *data* )

Inserts a new node after the specified node.

**Parameters**

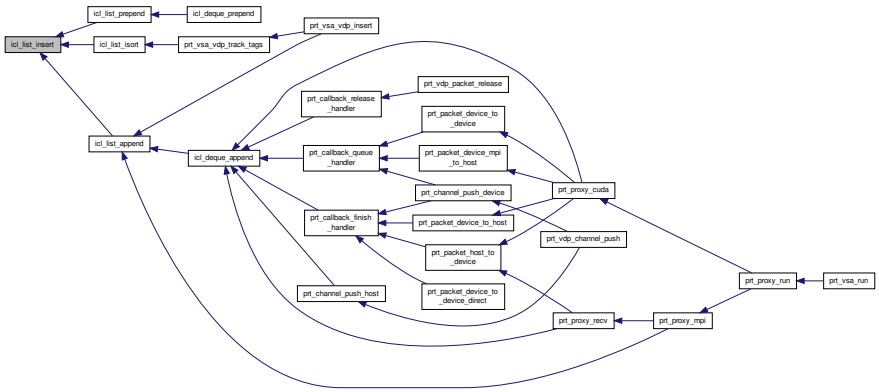
| | |
|---|---|
| *head* | – The linked list. |
| *pos* | – The insertion position (the node to append to). |
| *data* | – The pointer to the data to be inserted. |

**Returns**

A pointer to the new node. NULL on error.

Definition at line 47 of file icl_list.c.

Here is the caller graph for this function:



#### 6.9.2.7 icl_list_t* icl_list_isort ( icl_list_t * *head*, void * *data*, int(*)(void *, void *) *compare* )

Inserts data into a sorted list. Does not support direct comparison of pointers.

**Parameters**

| | |
|---|---|
| *head* | – The linked list. |
| *data* | – The data to be inserted. |
| *compare* | – The function that compares the data items. |

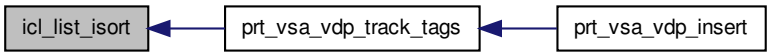
**Returns**

A pointer to the new node. NULL on error.

Definition at line 144 of file icl_list.c.

Here is the call graph for this function:


Here is the caller graph for this function:




#### 6.9.2.8 icl_list_t* icl_list_last ( icl_list_t * *head* )

Returns the last item in a linked list.

**Parameters**

| | |
|---|---|
| *head* | – The linked list. |

**Returns**

A pointer to the last item. NULL on error.

Definition at line 237 of file icl_list.c.

#### 6.9.2.9 icl_list_t* icl_list_new ( )

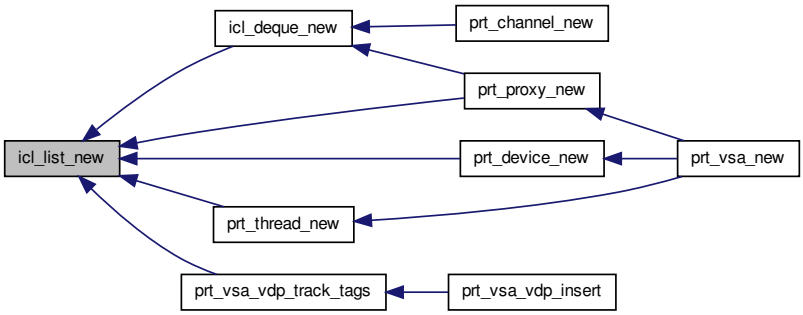
Creates a new linked list.

**Returns**

A new linked list. NULL on error.

Definition at line 22 of file icl_list.c.

Here is the caller graph for this function:




#### 6.9.2.10 icl_list_t* icl_list_next ( icl_list_t * *head*, icl_list_t * *pos* )

Returns the node following the specified node.

**Parameters**

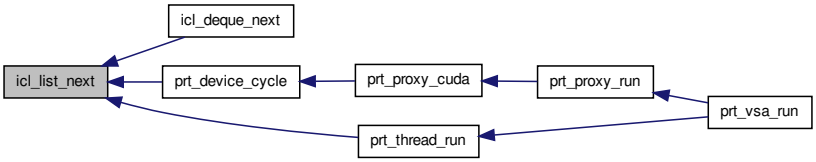
| | |
|---|---|
| *head* | – The list containing the specified node. |
| *pos* | – The node whose successor should be returned. |

**Returns**

A pointer to the next node. NULL on error.

Definition at line 254 of file icl_list.c.

Here is the caller graph for this function:



#### 6.9.2.11 icl_list_t* icl_list_prepend ( icl_list_t * *head,* void * *data* )

Inserts a node at the beginning of a list.

**Parameters**

| | |
|---|---|
| *head* | – The linked list. |
| *data* | – The data to be inserted. |

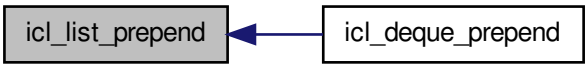
**Returns**

A pointer to the new node. NULL on error.

Definition at line 312 of file icl_list.c.

Here is the call graph for this function:


Here is the caller graph for this function:




#### 6.9.2.12 icl_list_t* icl_list_prev ( icl_list_t * *head*, icl_list_t * *pos* )

Returns the node preceding the specified node.

**Parameters**

| | |
|---|---|
| *head* | – The list containing the specified node. |
| *pos* | – The node whose predecessor should be returned. |

**Returns**

A pointer to the previous node. NULL on error.

Definition at line 271 of file icl_list.c.

#### 6.9.2.13 icl_list_t* icl_list_search ( icl_list_t * *head*, void * *data*, int(*)(void *, void *) *compare* )

Finds a data item in a linked list.

**Parameters**

| | |
|---|---|
| *head* | – The linked list. |
| *data* | – The data to be found. |
| *compare* | – The function that compares the data items. |

**Returns**

A pointer to the node, if found. Otherwise NULL.

Definition at line 114 of file icl_list.c.

#### 6.9.2.14 int icl_list_size ( icl_list_t * *head* )

Returns the number of items in a linked list.

**Parameters**

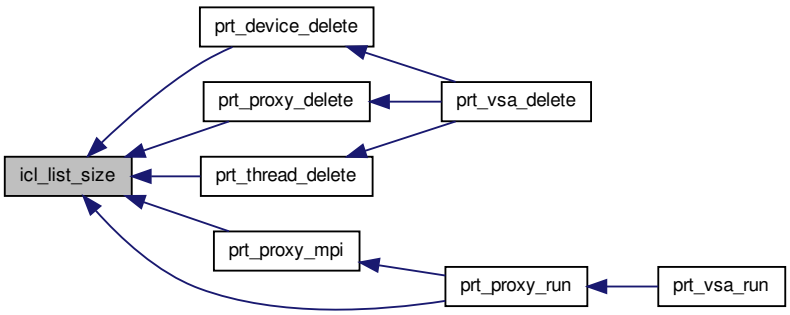
| | |
|---|---|
| *head* | – The linked list. |

**Returns**

The number of items in the list. -1 on error.

Definition at line 200 of file icl_list.c.

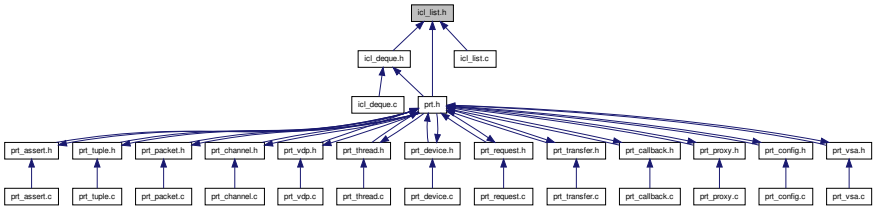
Here is the caller graph for this function:




## 6.10 icl_list.h File Reference

Dependency-free linked list.

This graph shows which files directly or indirectly include this file:




## Data Structures

- struct icl_list_s

## Macros

- #define **icl_list_foreach**(list, ptr) for (ptr = icl_list_first(list); ptr != NULL; ptr = icl_list_next(list, ptr))

## Typedefs

- typedef struct icl_list_s **icl_list_t**

## Functions

- icl_list_t * icl_list_new ()

  *Creates a new linked list.*
- icl_list_t * icl_list_insert (icl_list_t *, icl_list_t *, void *)

  *Inserts a new node after the specified node.*
- icl_list_t * icl_list_search (icl_list_t *, void *, int(*)(void *, void *))

  *Finds a data item in a linked list.*
- icl_list_t * icl_list_isort (icl_list_t *head, void *data, int(*)(void *, void *))

  *Inserts data into a sorted list. Does not support direct comparison of pointers.*
- icl_list_t * icl_list_first (icl_list_t *)

  *Returns the first item in a linked list.*
- icl_list_t * icl_list_last (icl_list_t *)

  *Returns the last item in a linked list.*
- icl_list_t * icl_list_next (icl_list_t *, icl_list_t *)

  *Returns the node following the specified node.*
- icl_list_t * icl_list_prev (icl_list_t *, icl_list_t *)

  *Returns the node preceding the specified node.*
- icl_list_t * icl_list_concat (icl_list_t *, icl_list_t *)

  *Concatenates two linked lists.*
- icl_list_t * icl_list_prepend (icl_list_t *, void *)

  *Inserts a node at the beginning of a list.*
- icl_list_t * icl_list_append (icl_list_t *, void *)

*Inserts a node at the end of a list.*

- int icl_list_delete (icl_list_t *, icl_list_t *, void(*)(void *))

  *Deletes the specified node.*

- int icl_list_destroy (icl_list_t *, void(*)(void *))

  *Destroys a linked list.*

- int icl_list_size (icl_list_t *)

  *Returns the number of items in a linked list.*

### 6.10.1 Detailed Description

Dependency-free linked list.

**Author**

Keith Seymour

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file icl_list.h.

### 6.10.2 Function Documentation

#### 6.10.2.1 icl_list_t* icl_list_append ( icl_list_t * *head*, void * *data* )

Inserts a node at the end of a list.

**Parameters**

| | |
|---|---|
| *head* | – The linked list. |
| *data* | – The data to be inserted. |

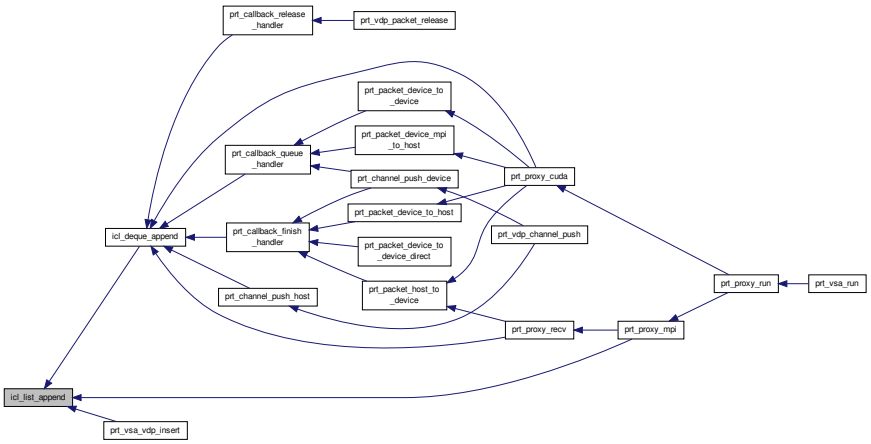
**Returns**

A pointer to the new node. NULL on error.

Definition at line 326 of file icl_list.c.

Here is the call graph for this function:

Here is the caller graph for this function:




#### 6.10.2.2 icl_list_t* icl_list_concat ( icl_list_t * *head1,* icl_list_t * *head2* )

Concatenates two linked lists.

**Parameters**

| | |
|---|---|
| *head1* | – The first linked list. |
| *head2* | – The second linked list. |

**Returns**

A pointer to the new linked list, which consists of <head1,head2>. NULL on error.

Definition at line 290 of file icl_list.c.

#### 6.10.2.3 int icl_list_delete ( icl_list_t * *head,* icl_list_t * *pos,* void(*)(void *) *free_function* )

Deletes the specified node.

**Parameters**

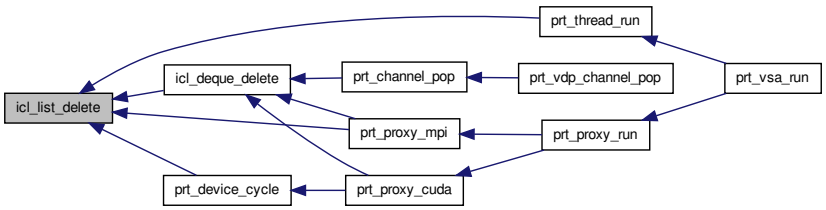
| | |
|---|---|
| *head* | – The linked list containing the node to be deleted. |
| *pos* | – The node to be deleted. |
| *free_function* | – The function that frees the node's data. |

**Return values**

| | |
|---|---|
| *0* | on success. |
| *-1* | on failure. |

Definition at line 82 of file icl_list.c.

Here is the caller graph for this function:




#### 6.10.2.4 int icl_list_destroy ( icl_list_t * *head*, void(*)(void *) *free_function* )

Destroys a linked list.

**Parameters**

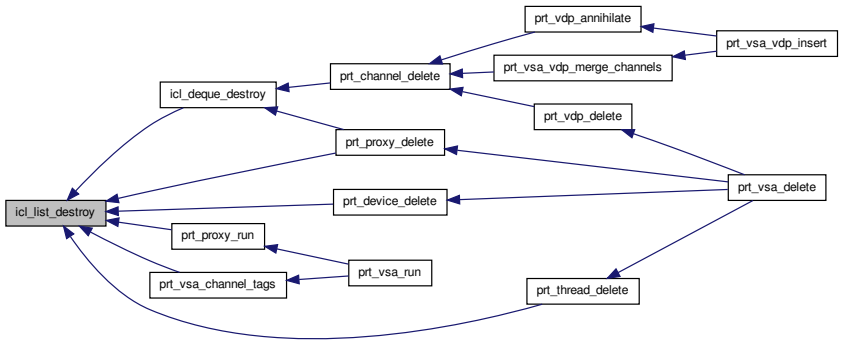
| | |
|---|---|
| *head* | – The linked list to be destroyed. |
| *free_function* | – The function that frees the node's data. |

**Return values**

| | |
|---|---|
| *0* | on success. |
| *-1* | on failure. |

Definition at line 173 of file icl_list.c.

Here is the caller graph for this function:




#### 6.10.2.5 icl_list_t* icl_list_first ( icl_list_t * *head* )

Returns the first item in a linked list.

**Parameters**

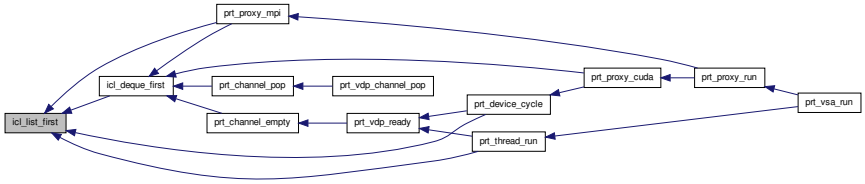
| | |
|---|---|
| *head* | – The linked list. |

**Returns**

A pointer to the first item. NULL on error.

Definition at line 221 of file icl_list.c.

Here is the caller graph for this function:




### 6.10.2.6 icl_list_t* icl_list_insert ( icl_list_t * *head*, icl_list_t * *pos*, void * *data* )

Inserts a new node after the specified node.

**Parameters**

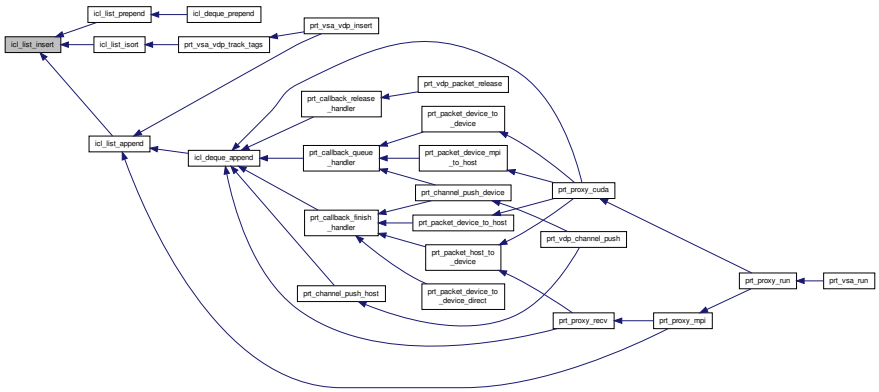
| | |
|---|---|
| *head* | – The linked list. |
| *pos* | – The insertion position (the node to append to). |
| *data* | – The pointer to the data to be inserted. |

**Returns**

A pointer to the new node. NULL on error.

Definition at line 47 of file icl_list.c.

Here is the caller graph for this function:



#### 6.10.2.7 icl_list_t* icl_list_isort ( icl_list_t * *head,* void * *data,* int(*)(void *, void *) *compare* )

Inserts data into a sorted list. Does not support direct comparison of pointers.

**Parameters**

| | |
|---|---|
| *head* | – The linked list. |
| *data* | – The data to be inserted. |
| *compare* | – The function that compares the data items. |

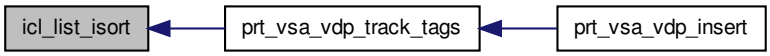
**Returns**

A pointer to the new node. NULL on error.

Definition at line 144 of file icl_list.c.

Here is the call graph for this function:


Here is the caller graph for this function:




#### 6.10.2.8 icl_list_t* icl_list_last ( icl_list_t * *head* )

Returns the last item in a linked list.

**Parameters**

| | |
|---|---|
| *head* | – The linked list. |

**Returns**

A pointer to the last item. NULL on error.

Definition at line 237 of file icl_list.c.

#### 6.10.2.9 icl_list_t* icl_list_new ( )

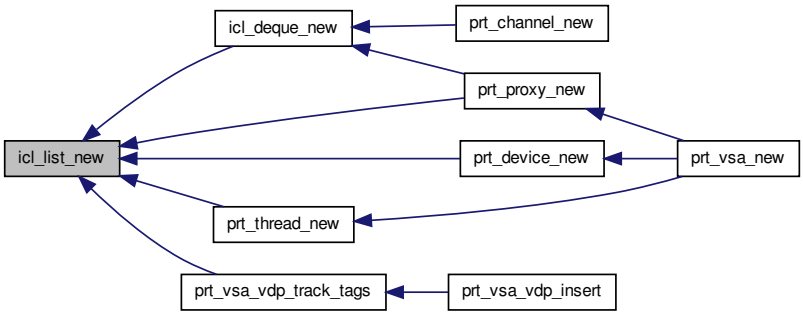
Creates a new linked list.

**Returns**

A new linked list. NULL on error.

Definition at line 22 of file icl_list.c.

Here is the caller graph for this function:




#### 6.10.2.10 icl_list_t* icl_list_next ( icl_list_t * *head*, icl_list_t * *pos* )

Returns the node following the specified node.

**Parameters**

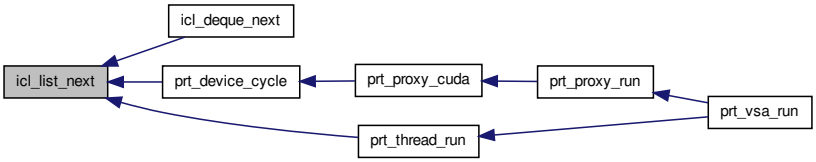
| | |
|---|---|
| *head* | – The list containing the specified node. |
| *pos* | – The node whose successor should be returned. |

**Returns**

A pointer to the next node. NULL on error.

Definition at line 254 of file icl_list.c.

Here is the caller graph for this function:



#### 6.10.2.11 icl_list_t* icl_list_prepend ( icl_list_t * *head,* void * *data* )

Inserts a node at the beginning of a list.

**Parameters**

| | |
|---|---|
| *head* | – The linked list. |
| *data* | – The data to be inserted. |

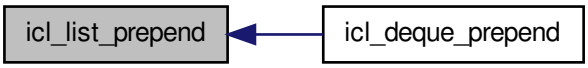
**Returns**

A pointer to the new node. NULL on error.

Definition at line 312 of file icl_list.c.

Here is the call graph for this function:


Here is the caller graph for this function:




#### 6.10.2.12 icl_list_t* icl_list_prev ( icl_list_t * *head*, icl_list_t * *pos* )

Returns the node preceding the specified node.

**Parameters**

| | |
|---|---|
| *head* | – The list containing the specified node. |
| *pos* | – The node whose predecessor should be returned. |

**Returns**

A pointer to the previous node. NULL on error.

Definition at line 271 of file icl_list.c.

#### 6.10.2.13 icl_list_t* icl_list_search ( icl_list_t * *head*, void * *data*, int(*)(void *, void *) *compare* )

Finds a data item in a linked list.

**Parameters**

| | |
|---|---|
| *head* | – The linked list. |
| *data* | – The data to be found. |
| *compare* | – The function that compares the data items. |

**Returns**

A pointer to the node, if found. Otherwise NULL.

Definition at line 114 of file icl_list.c.

#### 6.10.2.14 int icl_list_size ( icl_list_t * *head* )

Returns the number of items in a linked list.

**Parameters**

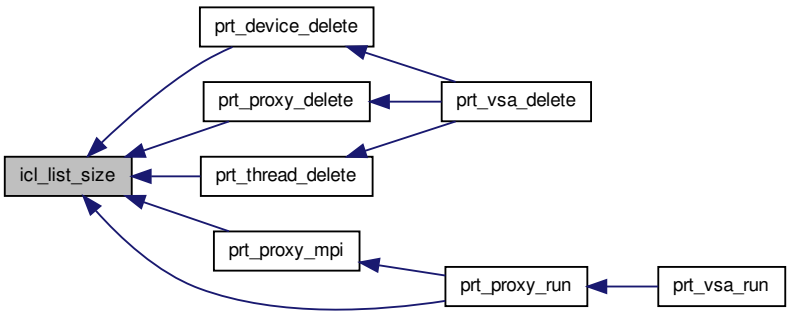
| | |
|---|---|
| *head* | – The linked list. |

**Returns**

The number of items in the list. -1 on error.

Definition at line 200 of file icl_list.c.

Here is the caller graph for this function:

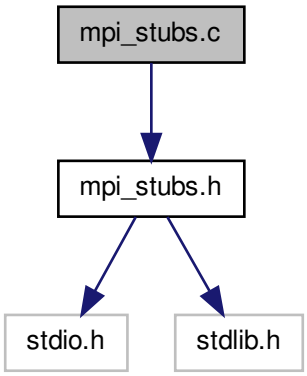



## 6.11 mpi_stubs.c File Reference

Stubs for a no-MPI build.

```
#include "mpi_stubs.h"
```

Include dependency graph for mpi_stubs.c:




## Functions

- int **MPI_Initialized** (int *flag)
- int **MPI_Comm_rank** (MPI_Comm comm, int *rank)
- int **MPI_Comm_size** (MPI_Comm comm, int *size)
- int **MPI_Barrier** (MPI_Comm comm)
- int **MPI_Cancel** (MPI_Request *request)
- int **MPI_Abort** (MPI_Comm comm, int errorcode)
- int **MPI_Test** (MPI_Request *request, int *flag, MPI_Status *status)
- int **MPI_Get_count** (const MPI_Status *status, MPI_Datatype datatype, int *count)
- int **MPI_Send** (const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
- int **MPI_Recv** (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)
- int **MPI_Irecv** (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)
- int **MPI_Isend** (const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)
- int **MPI_Reduce** (const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

### 6.11.1 Detailed Description

Stubs for a no-MPI build.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

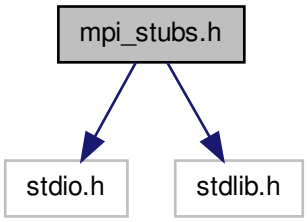
Definition in file mpi_stubs.c.

## 6.12 mpi_stubs.h File Reference

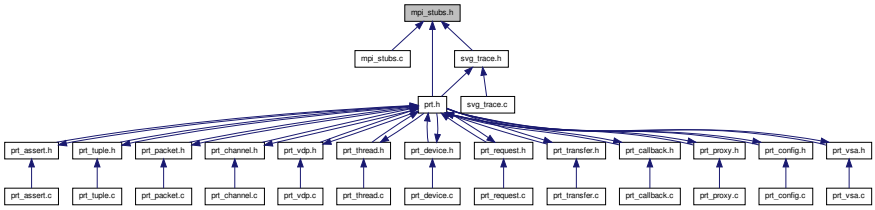
Stubs for a no-MPI build.

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

Include dependency graph for mpi_stubs.h:




This graph shows which files directly or indirectly include this file:




### Data Structures

- struct MPI_Status
- struct MPI_Request

### Macros

- #define **MPI_STATUS_IGNORE** NULL

### Typedefs

- typedef int **MPI_Op**
- typedef int **MPI_Comm**
- typedef int **MPI_Datatype**

### Enumerations

- enum {
  **MPI_BYTE**, **MPI_INT**, **MPI_DOUBLE**, **MPI_ANY_SOURCE**,
  **MPI_ANY_TAG**, **MPI_MAX**, **MPI_SUCCESS**, **MPI_COMM_WORLD** }

### Functions

- int **MPI_Initialized** (int *flag)
- int **MPI_Barrier** (MPI_Comm comm)
- int **MPI_Cancel** (MPI_Request *request)
- int **MPI_Abort** (MPI_Comm comm, int errorcode)
- int **MPI_Comm_rank** (MPI_Comm comm, int *rank)
- int **MPI_Comm_size** (MPI_Comm comm, int *size)
- int **MPI_Test** (MPI_Request *request, int *flag, MPI_Status *status)
- int **MPI_Get_count** (const MPI_Status *status, MPI_Datatype datatype, int *count)
- int **MPI_Send** (const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
- int **MPI_Recv** (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)
- int **MPI_Irecv** (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)
- int **MPI_Isend** (const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_-Request *request)
- int **MPI_Reduce** (const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

#### 6.12.1 Detailed Description

Stubs for a no-MPI build.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file mpi_stubs.h.

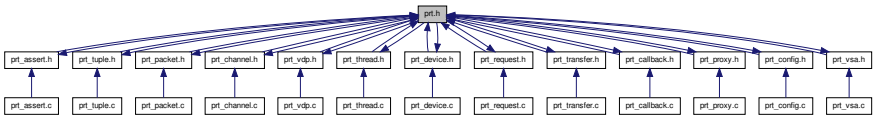
## 6.13 prt.h File Reference

PULSAR Runtime (PRT)

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <string.h>
#include <stdarg.h>
#include <limits.h>
#include <malloc.h>
#include <sched.h>
#include <pthread.h>
#include "mpi_stubs.h"
#include "cuda_stubs.h"
#include "gpu_malloc.h"
#include "icl_list.h"
#include "icl_hash.h"
#include "icl_deque.h"
#include "svg_trace.h"
#include "prt_assert.h"
#include "prt_tuple.h"
#include "prt_packet.h"
#include "prt_channel.h"
#include "prt_vdp.h"
#include "prt_thread.h"
#include "prt_device.h"
#include "prt_request.h"
#include "prt_transfer.h"
#include "prt_callback.h"
#include "prt_proxy.h"
#include "prt_config.h"
#include "prt_vsa.h"
```

Include dependency graph for prt.h:

This graph shows which files directly or indirectly include this file:




## Data Structures

- struct prt_mapping_s

  *Mapping of VDPs to hardware.*

### Typedefs

- typedef enum prt_location_e prt_location_t

  *Locations of VDPs and packets.*
- typedef enum prt_direction_e prt_direction_t

  *Directions of local transfers.*
- typedef struct prt_mapping_s prt_mapping_t

  *Mapping of VDPs to hardware.*

### Enumerations

- enum prt_location_e { **PRT_LOCATION_HOST**, **PRT_LOCATION_DEVICE** }

  *Locations of VDPs and packets.*
- enum prt_direction_e {
  **PRT_HOST_TO_DEVICE**, **PRT_DEVICE_TO_HOST**, **PRT_DEVICE_TO_DEVICE**, **PRT_DEVICE_MPI_TO_HOST**,
  **PRT_DEVICE_MPI_FROM_HOST**, **PRT_DEVICE_PACKET_RELEASE** }

  *Directions of local transfers.*

#### 6.13.1 Detailed Description

PULSAR Runtime (PRT)

**Author**

Jakub Kurzak

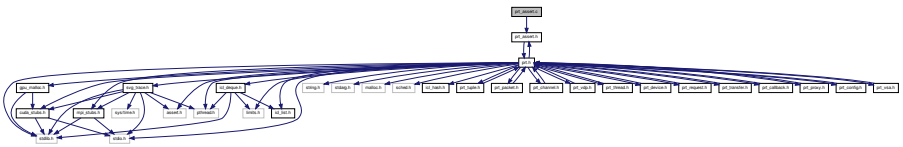
Definition in file prt.h.

## 6.14 prt_assert.c File Reference

PRT exception handling.

```
#include "prt_assert.h"
```

Include dependency graph for prt_assert.c:



### Functions

- void prt_assert_line_file (int cond, const char *msg, int line, char *file)

  *Checks an assertion and exits on error. Prints an error message.*
- void prt_error_line_file (const char *msg, int line, char *file)

*Prints an error message and exits.*

- void prt_warning_line_file (const char *msg, int line, char *file)

  *Prints a warning and continues.*

### 6.14.1 Detailed Description

PRT exception handling.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_assert.c.

### 6.14.2 Function Documentation

#### 6.14.2.1 void prt_assert_line_file ( int *cond*, const char * *msg*, int *line*, char * *file* )

Checks an assertion and exits on error. Prints an error message.

**Parameters**

| | |
|---|---|
| *cond* | – The condition. |
| *msg* | – The error message. |
| *line* | – The line number. |
| *file* | – The name of the source file. |

Definition at line 23 of file prt_assert.c.

Here is the call graph for this function:


#### 6.14.2.2 void prt_error_line_file ( const char * *msg*, int *line*, char * *file* )

Prints an error message and exits.

**Parameters**

| | |
|---|---|
| *msg* | – The error message. |
| *line* | – The line number. |
| *file* | – The name of the source file. |

Definition at line 37 of file prt_assert.c.

Here is the caller graph for this function:


#### 6.14.2.3 void prt_warning_line_file ( const char * *msg*, int *line*, char * *file* )

Prints a warning and continues.

**Parameters**

| | |
|---|---|
| *msg* | – The warning message. |
| *line* | – The line number. |
| *file* | – The name of the source file. |

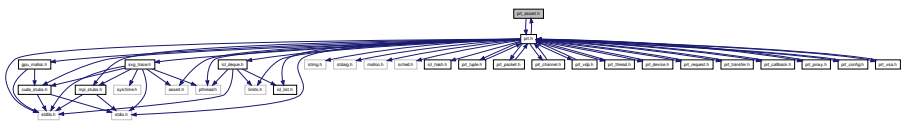
Definition at line 52 of file prt_assert.c.

## 6.15 prt_assert.h File Reference

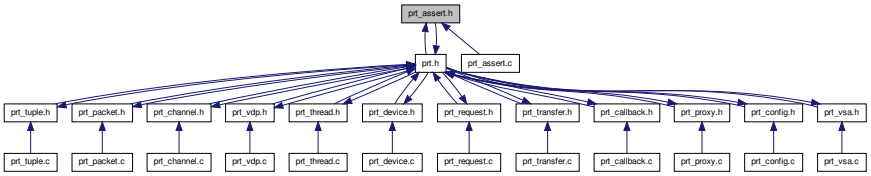
PRT exception handling.

```
#include "prt.h"
```

Include dependency graph for prt_assert.h:



This graph shows which files directly or indirectly include this file:




## Macros

- #define **prt_error**(msg) prt_error_line_file(msg, __LINE__, __FILE__)
- #define **prt_warning**(msg) prt_warning_line_file(msg, __LINE__, __FILE__)
- #define **prt_assert**(cond, msg) prt_assert_line_file(cond, msg, __LINE__, __FILE__)

## Functions

- void prt_error_line_file (const char *msg, int line, char *file)

  *Prints an error message and exits.*
- void prt_warning_line_file (const char *msg, int line, char *file)

  *Prints a warning and continues.*
- void prt_assert_line_file (int cond, const char *msg, int line, char *file)

  *Checks an assertion and exits on error. Prints an error message.*

### 6.15.1 Detailed Description

PRT exception handling.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_assert.h.

### 6.15.2 Function Documentation

#### 6.15.2.1 void prt_assert_line_file ( int *cond*, const char * *msg*, int *line*, char * *file* )

Checks an assertion and exits on error. Prints an error message.

**Parameters**

| | |
|---|---|
| *cond* | – The condition. |
| *msg* | – The error message. |
| *line* | – The line number. |
| *file* | – The name of the source file. |

Definition at line 23 of file prt_assert.c.

Here is the call graph for this function:


#### 6.15.2.2 void prt_error_line_file ( const char * *msg*, int *line*, char * *file* )

Prints an error message and exits.

**Parameters**

| | |
|---|---|
| *msg* | – The error message. |
| *line* | – The line number. |
| *file* | – The name of the source file. |

Definition at line 37 of file prt_assert.c.

Here is the caller graph for this function:


#### 6.15.2.3 void prt_warning_line_file ( const char * *msg*, int *line*, char * *file* )

Prints a warning and continues.

**Parameters**

| | |
|---|---|
| *msg* | – The warning message. |
| *line* | – The line number. |
| *file* | – The name of the source file. |

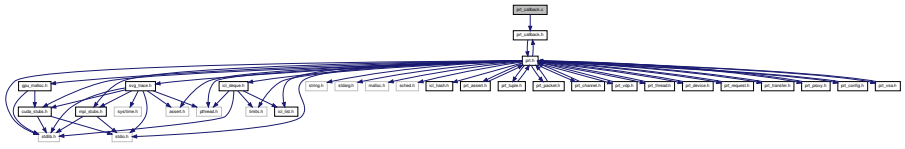
Definition at line 52 of file prt_assert.c.

## 6.16 prt_callback.c File Reference

PRT callback.

```
#include "prt_callback.h"
```

Include dependency graph for prt_callback.c:



### Functions

- prt_callback_finish_t * prt_callback_finish_new (struct prt_packet_s *src_packet, struct prt_packet_s *dst_packet, struct prt_channel_s *channel)

  *Creates a new callback data structure. This is for the callback that completes a local transfer.*
- void prt_callback_finish_delete (prt_callback_finish_t *callback)

  *Destroys a callback data structure. This is for the callback that completes a local transfer.*
- void CUDART_CB prt_callback_finish_handler (cudaStream_t stream, cudaError_t status, void *clbck)

  *Finishes a local transfer. Puts the packet in the channel after a local transfer finishes. Services host-to-device and device-to-host transfers.*
- prt_callback_queue_t * prt_callback_queue_new (struct prt_packet_s *old_packet, struct prt_packet_s *src_packet, struct prt_channel_s *channel, prt_direction_t direction, int agent)

  *Creates a new callback data structure. This is for the callback that queues a local transfer.*
- void prt_callback_queue_delete (prt_callback_queue_t *callback)

  *Destroys a callback data structure. This is for the callback that queues a local transfer.*
- void CUDART_CB prt_callback_queue_handler (cudaStream_t stream, cudaError_t status, void *clbck)

  *Queues a local transfer request. Services device-to-device requensts and MPI requests from a device.*
- prt_callback_release_t * prt_callback_release_new (struct prt_vdp_s *vdp, struct prt_packet_s *packet)

  *Creates a new callback data structure. This is for the callback that releases a device packet.*
- void prt_callback_release_delete (prt_callback_release_t *callback)

  *Destroys a callback data structure. This is for the callback that releases a device packet.*
- void CUDART_CB prt_callback_release_handler (cudaStream_t stream, cudaError_t status, void *clbck)

  *Releases a device packet.*

### 6.16.1 Detailed Description

PRT callback.

**Author**

Jakub Kurzak



Definition in file prt_callback.c.

### 6.16.2 Function Documentation

#### 6.16.2.1 void prt_callback_finish_delete ( prt_callback_finish_t * *callback* )

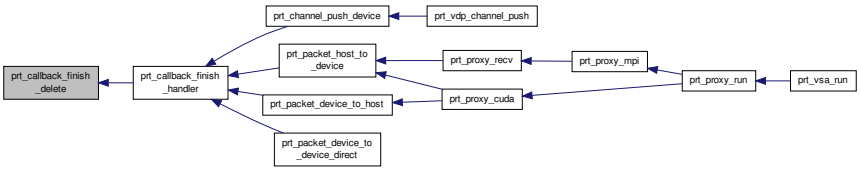
Destroys a callback data structure. This is for the callback that completes a local transfer.

**Parameters**

| | |
|---|---|
| *callback* | – The callback data structure to destroy. |

Definition at line 45 of file prt_callback.c.

Here is the caller graph for this function:




#### 6.16.2.2 void CUDART_CB prt_callback_finish_handler ( cudaStream_t *stream*, cudaError_t *status*, void * *clbck* )

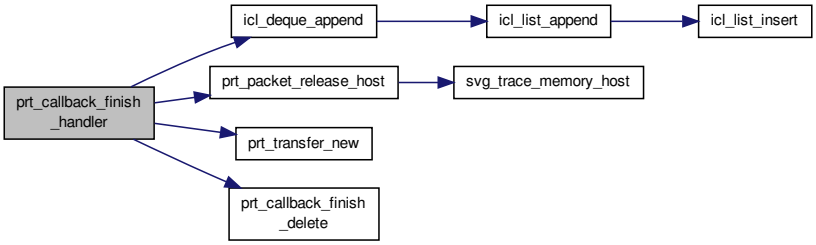
Finishes a local transfer. Puts the packet in the channel after a local transfer finishes. Services host-to-device and device-to-host transfers.

**Parameters**

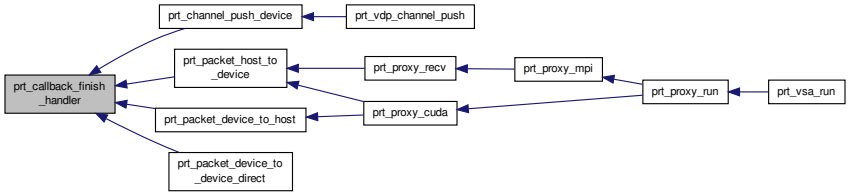
| | |
|---|---|
| *stream* | – The callback's stream. |
| *status* | – The stream's status. |
| *clbck* | – The callback data. |

Definition at line 60 of file prt_callback.c.

Here is the call graph for this function:




Here is the caller graph for this function:




### 6.16.2.3 prt_callback_finish_t* prt_callback_finish_new ( struct prt_packet_s * *src_packet,* struct prt_packet_s * *dst_packet,* struct prt_channel_s * *channel* )

Creates a new callback data structure. This is for the callback that completes a local transfer.

**Parameters**

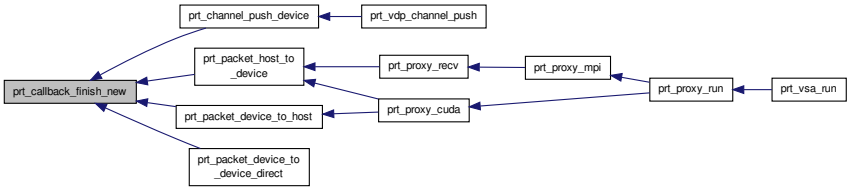
| | |
|---|---|
| *src_packet* | – The packet to release when the transfer completes. |
| *dst_packet* | – The packet to place in the channel when the transfer completes. |
| *channel* | – The channel to insert the packet into. |

**Returns**

A new callback data structure.

Definition at line 24 of file prt_callback.c.

Here is the caller graph for this function:




### 6.16.2.4 void prt_callback_queue_delete ( prt_callback_queue_t * *callback* )

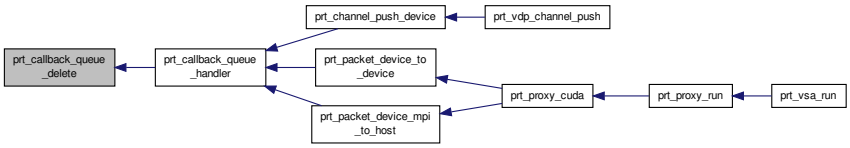
Destroys a callback data structure. This is for the callback that queues a local transfer.

**Parameters**

| | |
|---|---|
| *callback* | – The callback data structure to destroy. |

Definition at line 129 of file prt_callback.c.

Here is the caller graph for this function:




### 6.16.2.5 void CUDART_CB prt_callback_queue_handler ( cudaStream_t *stream*, cudaError_t *status*, void * *clbck* )

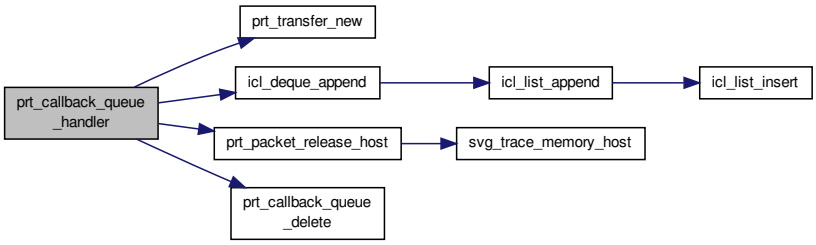
Queues a local transfer request. Services device-to-device requensts and MPI requests from a device.

**Parameters**

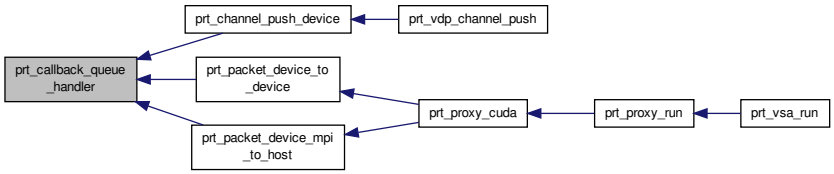
| | |
|---|---|
| *stream* | – The callback's stream. |
| *status* | – The stream's status. |
| *clbck* | – The callback data. |

Definition at line 143 of file prt_callback.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.16.2.6 prt_callback_queue_t* prt_callback_queue_new ( struct prt_packet_s * *old_packet,* struct prt_packet_s * *src_packet,* struct prt_channel_s * *channel,* prt_direction_t *direction,* int *agent* )

Creates a new callback data structure. This is for the callback that queues a local transfer.

**Parameters**

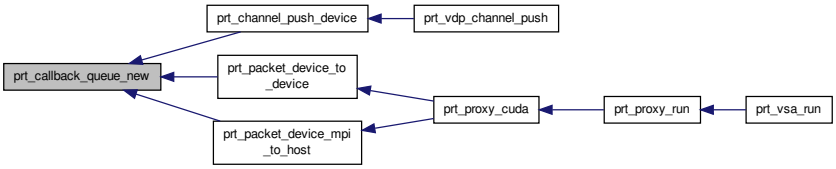
| | |
|---|---|
| *old_packet* | – The packet to release when the transfer completes. |
| *src_packet* | – The packet to use for the followup transfer request. |
| *channel* | – The channel to use for the followup transfer request. |
| *direction* | – The direction of the followup transfer request. |

**Returns**

A new callback data structure.

Definition at line 104 of file prt_callback.c.

Here is the caller graph for this function:




### 6.16.2.7 void prt_callback_release_delete ( prt_callback_release_t * *callback* )

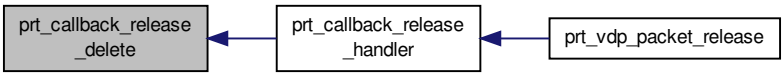
Destroys a callback data structure. This is for the callback that releases a device packet.

**Parameters**

| | |
|---|---|
| *callback* | – The callback data structure to be destroyed. |

Definition at line 210 of file prt_callback.c.

Here is the caller graph for this function:




### 6.16.2.8 void CUDART_CB prt_callback_release_handler ( cudaStream_t *stream*, cudaError_t *status*, void * *clbck* )

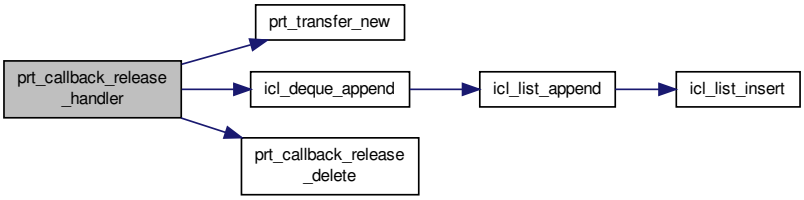
Releases a device packet.

**Parameters**

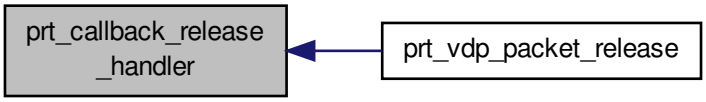
| | |
|---|---|
| *stream* | – The callback's stream. |
| *status* | – The stream's status. |
| *clbck* | – The callback data. |

Definition at line 223 of file prt_callback.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.16.2.9 prt_callback_release_t* prt_callback_release_new ( struct prt_vdp_s * *vdp*, struct prt_packet_s * *packet* )

Creates a new callback data structure. This is for the callback that releases a device packet.

**Parameters**

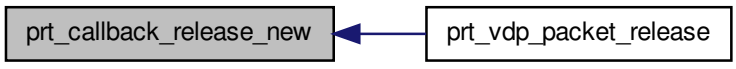
| | |
|---|---|
| *vdp* | – The VDP releasing the packet. |
| *packet* | – The packet to release. |

**Returns**

A new callback data structure.

Definition at line 191 of file prt_callback.c.

Here is the caller graph for this function:

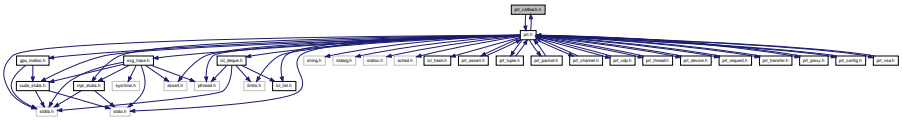


## 6.17 prt_callback.h File Reference

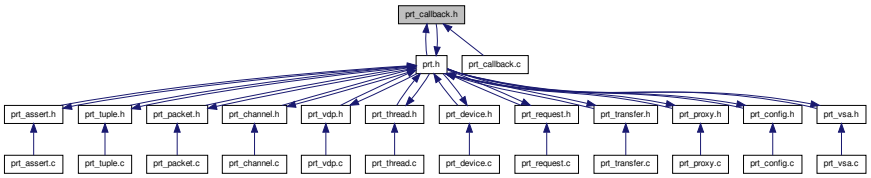
PRT callback.

```
#include "prt.h"
```

Include dependency graph for prt_callback.h:



This graph shows which files directly or indirectly include this file:




### Data Structures

- struct prt_callback_finish_s

  *Callback data for finishing a local communication.*
- struct prt_callback_queue_s

  *Callback data for queueing a local communication.*
- struct prt_callback_release_s

  *Callback data for releasing a device packet.*

### Typedefs

- typedef struct prt_callback_finish_s prt_callback_finish_t

  *Callback data for finishing a local communication.*
- typedef struct prt_callback_queue_s prt_callback_queue_t

  *Callback data for queueing a local communication.*
- typedef struct prt_callback_release_s prt_callback_release_t

  *Callback data for releasing a device packet.*

### Functions

- prt_callback_finish_t * prt_callback_finish_new (struct prt_packet_s *src_packet, struct prt_packet_s *dst_packet, struct prt_channel_s *channel)

*Creates a new callback data structure. This is for the callback that completes a local transfer.*

- void prt_callback_finish_delete (prt_callback_finish_t *clbck)

  *Destroys a callback data structure. This is for the callback that completes a local transfer.*
- void CUDART_CB prt_callback_finish_handler (cudaStream_t stream, cudaError_t status, void *dat)

  *Finishes a local transfer. Puts the packet in the channel after a local transfer finishes. Services host-to-device and device-to-host transfers.*
- prt_callback_queue_t * prt_callback_queue_new (struct prt_packet_s *old_packet, struct prt_packet_s *src_packet, struct prt_channel_s *channel, prt_direction_t direction, int agent)

  *Creates a new callback data structure. This is for the callback that queues a local transfer.*
- void prt_callback_queue_delete (prt_callback_queue_t *clbck)

  *Destroys a callback data structure. This is for the callback that queues a local transfer.*
- void CUDART_CB prt_callback_queue_handler (cudaStream_t stream, cudaError_t status, void *dat)

  *Queues a local transfer request. Services device-to-device requensts and MPI requests from a device.*
- prt_callback_release_t * prt_callback_release_new (struct prt_vdp_s *vdp, struct prt_packet_s *packet)

  *Creates a new callback data structure. This is for the callback that releases a device packet.*
- void prt_callback_release_delete (prt_callback_release_t *callback)

  *Destroys a callback data structure. This is for the callback that releases a device packet.*
- void CUDART_CB prt_callback_release_handler (cudaStream_t stream, cudaError_t status, void *clbck)

  *Releases a device packet.*

### 6.17.1 Detailed Description

PRT callback.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_callback.h.

### 6.17.2 Function Documentation

#### 6.17.2.1 void prt_callback_finish_delete ( prt_callback_finish_t * *callback* )

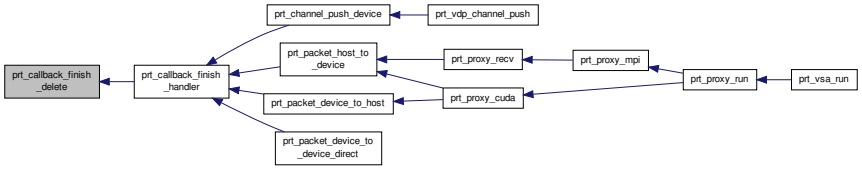
Destroys a callback data structure. This is for the callback that completes a local transfer.

**Parameters**

| | |
|---|---|
| *callback* | – The callback data structure to destroy. |

Definition at line 45 of file prt_callback.c.

Here is the caller graph for this function:



#### 6.17.2.2 void CUDART_CB prt_callback_finish_handler ( cudaStream_t *stream*, cudaError_t *status*, void * *clbck* )

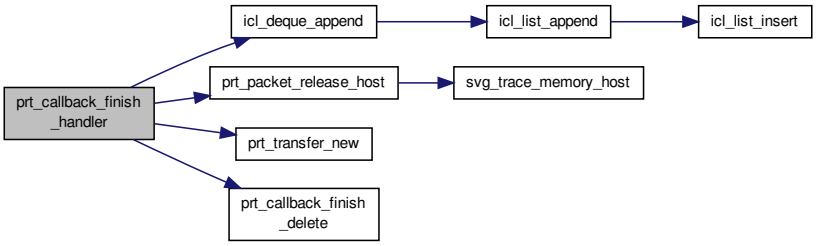
Finishes a local transfer. Puts the packet in the channel after a local transfer finishes. Services host-to-device and device-to-host transfers.

**Parameters**

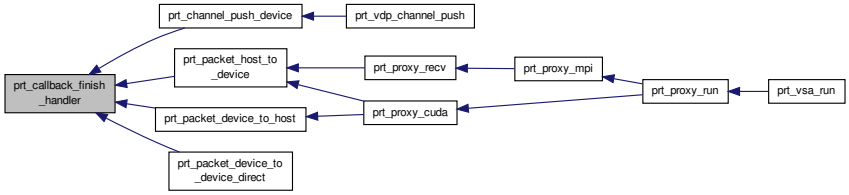
| | |
|---|---|
| *stream* | – The callback's stream. |
| *status* | – The stream's status. |
| *clbck* | – The callback data. |

Definition at line 60 of file prt_callback.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.17.2.3 prt_callback_finish_t* prt_callback_finish_new ( struct prt_packet_s * *src_packet*, struct prt_packet_s * *dst_packet*, struct prt_channel_s * *channel* )

Creates a new callback data structure. This is for the callback that completes a local transfer.

**Parameters**

| | |
|---|---|
| *src_packet* | – The packet to release when the transfer completes. |
| *dst_packet* | – The packet to place in the channel when the transfer completes. |

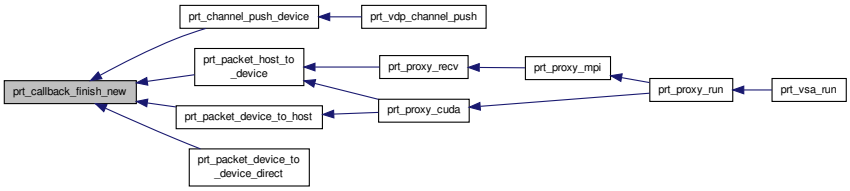
| | |
|---|---|
| *channel* | – The channel to insert the packet into. |

**Returns**

A new callback data structure.

Definition at line 24 of file prt_callback.c.

Here is the caller graph for this function:




### 6.17.2.4 void prt_callback_queue_delete ( prt_callback_queue_t * *callback* )

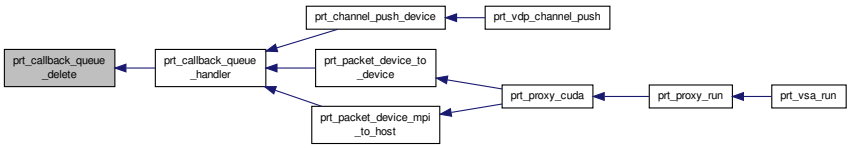
Destroys a callback data structure. This is for the callback that queues a local transfer.

**Parameters**

| | |
|---|---|
| *callback* | – The callback data structure to destroy. |

Definition at line 129 of file prt_callback.c.

Here is the caller graph for this function:




### 6.17.2.5 void CUDART_CB prt_callback_queue_handler ( cudaStream_t *stream,* cudaError_t *status,* void * *clbck* )

Queues a local transfer request. Services device-to-device requensts and MPI requests from a device.

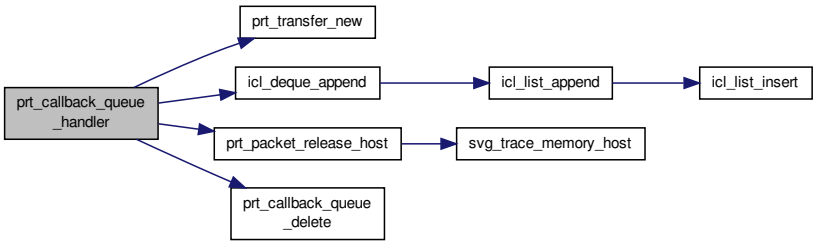
**Parameters**

| | |
|---|---|
| *stream* | – The callback's stream. |

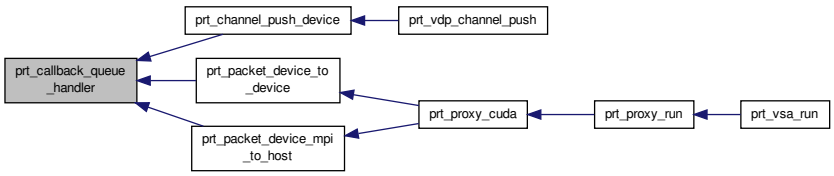
| | |
|---|---|
| *status* | – The stream's status. |
| *clbck* | – The callback data. |

Definition at line 143 of file prt_callback.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.17.2.6 prt_callback_queue_t* prt_callback_queue_new ( struct prt_packet_s * *old_packet,* struct prt_packet_s * *src_packet,* struct prt_channel_s * *channel,* prt_direction_t *direction,* int *agent* )

Creates a new callback data structure. This is for the callback that queues a local transfer.

**Parameters**

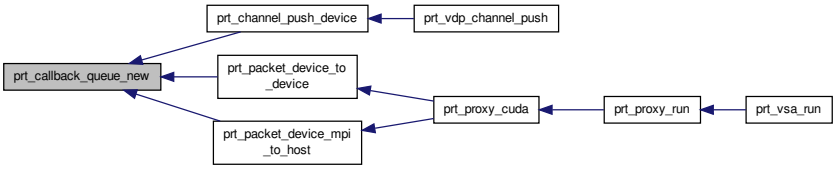
| | |
|---|---|
| *old_packet* | – The packet to release when the transfer completes. |
| *src_packet* | – The packet to use for the followup transfer request. |
| *channel* | – The channel to use for the followup transfer request. |
| *direction* | – The direction of the followup transfer request. |

**Returns**

A new callback data structure.

Definition at line 104 of file prt_callback.c.

Here is the caller graph for this function:




### 6.17.2.7 void prt_callback_release_delete ( prt_callback_release_t * *callback* )

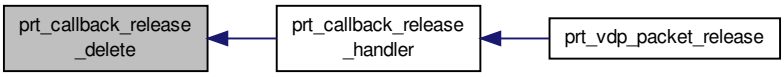
Destroys a callback data structure. This is for the callback that releases a device packet.

**Parameters**

| | |
|---|---|
| *callback* | – The callback data structure to be destroyed. |

Definition at line 210 of file prt_callback.c.

Here is the caller graph for this function:




### 6.17.2.8 void CUDART_CB prt_callback_release_handler ( cudaStream_t *stream*, cudaError_t *status*, void * *clbck* )

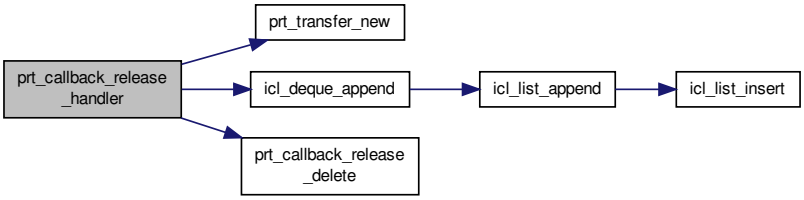
Releases a device packet.

**Parameters**

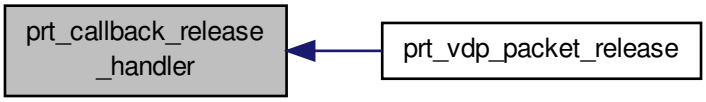
| | |
|---|---|
| *stream* | – The callback's stream. |
| *status* | – The stream's status. |
| *clbck* | – The callback data. |

Definition at line 223 of file prt_callback.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.17.2.9 prt_callback_release_t* prt_callback_release_new ( struct prt_vdp_s * *vdp*, struct prt_packet_s * *packet* )

Creates a new callback data structure. This is for the callback that releases a device packet.

**Parameters**

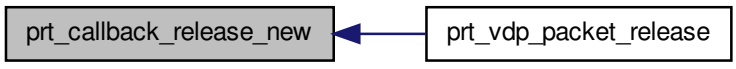
| | |
|---|---|
| *vdp* | – The VDP releasing the packet. |
| *packet* | – The packet to release. |

**Returns**

A new callback data structure.

Definition at line 191 of file prt_callback.c.

Here is the caller graph for this function:

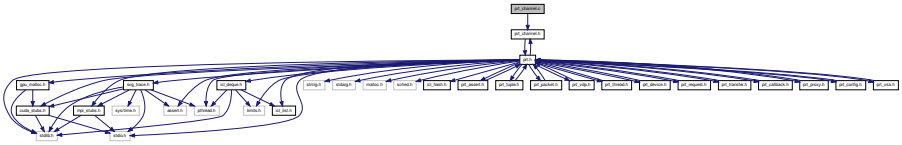


## 6.18 prt_channel.c File Reference

PRT data channel.

```
#include "prt_channel.h"
```

Include dependency graph for prt_channel.c:



### Functions

- prt_channel_t * prt_channel_new (size_t size, int *src_tuple, int src_slot, int *dst_tuple, int dst_slot)

  *Creates a new channel. Channel size cannot be larger than INT_MAX, because all data typea are packed inside messages of type MPI_BYTE.*
- void prt_channel_delete (prt_channel_t *channel)

  *Destroys a channel.*
- void prt_channel_push_host (prt_vdp_t *vdp, prt_channel_t *channel, prt_packet_t *packet)

  *Sends a packet from a host VDP.*
- void prt_channel_push_device (prt_vdp_t *vdp, prt_channel_t *channel, prt_packet_t *packet)

  *Sends a packet from a device VDP. Puts a callback in the VDP's stream. When reached, the callback puts the transfer in the channel's stream.*
- prt_packet_t * prt_channel_pop (prt_channel_t *channel)

  *Fetches a packef from a channel. Does not decrement the number of active references. The packet leaves the channel, but enters the VDP.*
- int prt_channel_empty (prt_channel_t *channel)

  *Checks if a channel is empty.*
- int prt_channel_compare (void *channel1, void *channel2)

  *Compares two channels.*
- void prt_channel_off (prt_channel_t *channel)

  *Deactivates a channel. Newly created channels are active. Inactive channels are excluded from readiness checks.*
- void prt_channel_on (prt_channel_t *channel)

  *Activates a channel. Newly created channels are active. Inactive channels are excluded from readiness checks.*

### 6.18.1 Detailed Description

PRT data channel.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_channel.c.

### 6.18.2 Function Documentation

#### 6.18.2.1 int prt_channel_compare ( void * *channel1*, void * *channel2* )

Compares two channels.

**Parameters**

| | |
|---|---|
| *channel1* | – The first channel. |
| *channel2* | – The second channel. |

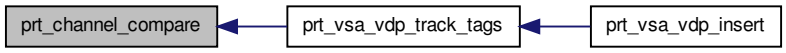
**Return values**

| | |
|---|---|
| *-1* | channel1 is less than channel2. |
| *0* | channel1 is equal to channel2. |
| *1* | channel1 is greater than channel2. |

Definition at line 264 of file prt_channel.c.

Here is the call graph for this function:


Here is the caller graph for this function:




#### 6.18.2.2 void prt_channel_delete ( prt_channel_t * *channel* )

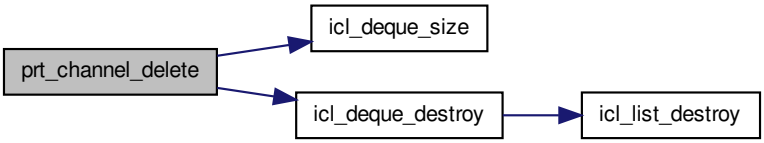
Destroys a channel.

**Parameters**

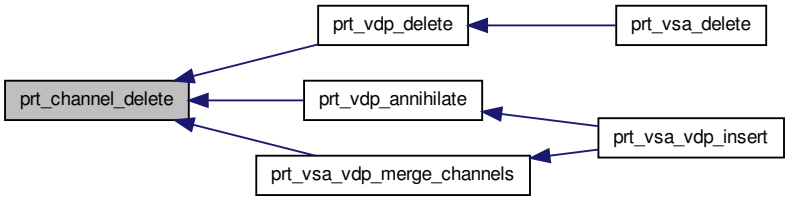
| | |
|---|---|
| *channel* | – The channel to destroy. |

Definition at line 70 of file prt_channel.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.18.2.3 int prt_channel_empty ( prt_channel_t * *channel* )

Checks if a channel is empty.

**Parameters**

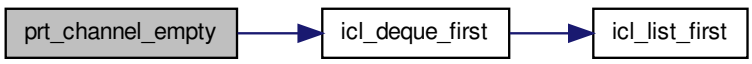
| | |
|---|---|
| *channel* | – The channel to check. |

**Return values**

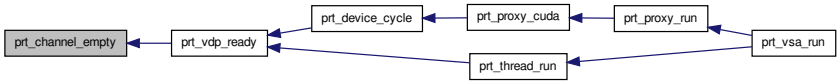
| | |
|---|---|
| *1* | if the channel is empty. |
| *0* | if the channel is not empty. |

Definition at line 243 of file prt_channel.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.18.2.4 void prt_channel_off ( prt_channel_t * *channel* )

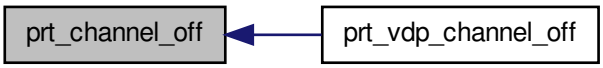
Deactivates a channel. Newly created channels are active. Inactive channels are excluded from readiness checks.

**Parameters**

| | |
|---|---|
| *channel* | – The channel to deactivate. |

Definition at line 292 of file prt_channel.c.

Here is the caller graph for this function:




#### 6.18.2.5 void prt_channel_on ( prt_channel_t * *channel* )

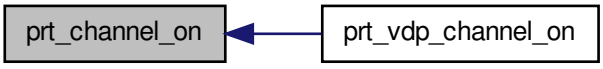
Activates a channel. Newly created channels are active. Inactive channels are excluded from readiness checks.

**Parameters**

| | |
|---|---|
| *channel* | - The channel to activate. |

Definition at line 306 of file prt_channel.c.

Here is the caller graph for this function:



#### 6.18.2.6 prt_packet_t* prt_channel_pop ( prt_channel_t * *channel* )

Fetches a packef from a channel. Does not decrement the number of active references. The packet leaves the channel, but enters the VDP.

**Parameters**

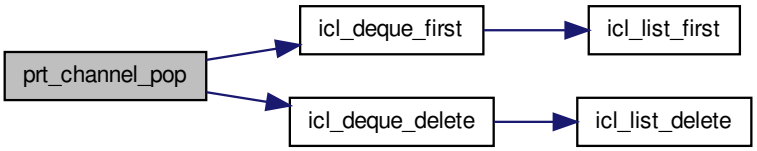
| | |
|---|---|
| *channel* | – The channel to fetch the packet from. |

**Returns**

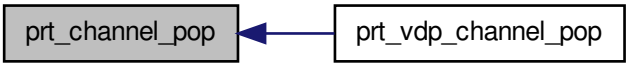
A data packet.

Definition at line 219 of file prt_channel.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.18.2.7 void prt_channel_push_device ( prt_vdp_t * *vdp,* prt_channel_t * *channel,* prt_packet_t * *packet* )

Sends a packet from a device VDP. Puts a callback in the VDP's stream. When reached, the callback puts the transfer in the channel's stream.

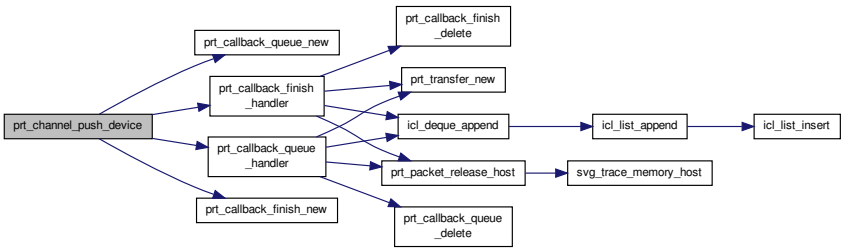
There is no need to set the device here. This function is called by prt_vdp_channel_push, which is called from VDP code, where the device is already set.

**Parameters**

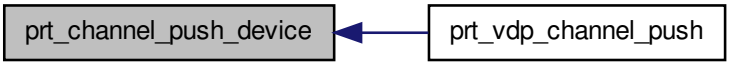
| | |
|---|---|
| *vdp* | – The device VDP sending the packet. |
| *channel* | – The channel to send the packet to. |
| *packet* | – The packet to send. |

Definition at line 151 of file prt_channel.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.18.2.8 void prt_channel_push_host ( prt_vdp_t * *vdp*, prt_channel_t * *channel*, prt_packet_t * *packet* )

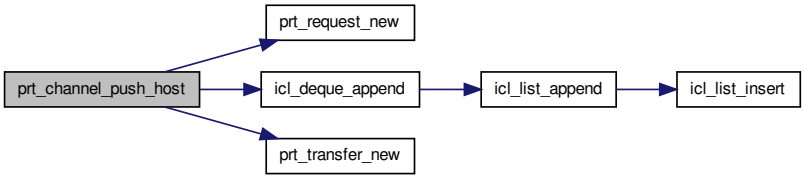
Sends a packet from a host VDP.

**Parameters**

| | |
|---|---|
| *vdp* | – The host VDP sending the packet. |
| *channel* | – The channel to send the packet to. |
| *packet* | – The packet to send. |

Definition at line 104 of file prt_channel.c.

Here is the call graph for this function:



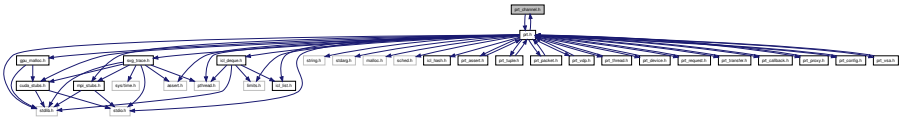

Here is the caller graph for this function:


## 6.19 prt_channel.h File Reference

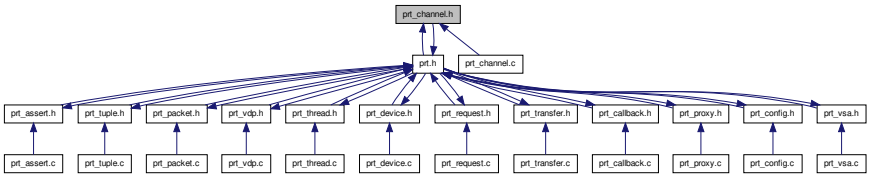
PRT data channel.

```
#include "prt.h"
```

Include dependency graph for prt_channel.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct prt_channel_s

  *VDP's data channel. Implements a data link between a pair of VDPs. Identifies the source and destination VDPs by tuples. Contains a thread-safe list of data packets.*

## Typedefs

- typedef struct prt_channel_s prt_channel_t

  *VDP's data channel. Implements a data link between a pair of VDPs. Identifies the source and destination VDPs by tuples. Contains a thread-safe list of data packets.*

- typedef enum prt_channel_direction_e prt_channel_direction_t

  *VDP's data channel direction. Identifies the direction of a VDP channel during insertion.*

## Enumerations

- enum prt_channel_direction_e { **PRT_INPUT_CHANNEL**, **PRT_OUTPUT_CHANNEL** }

  *VDP's data channel direction. Identifies the direction of a VDP channel during insertion.*

## Functions

- prt_channel_t * prt_channel_new (size_t size, int *src_tuple, int src_slot, int *dst_tuple, int dst_slot)

  *Creates a new channel. Channel size cannot be larger than INT_MAX, because all data typea are packed inside messages of type MPI_BYTE.*

- void prt_channel_delete (prt_channel_t *channel)

  *Destroys a channel.*

- void prt_channel_push_host (struct prt_vdp_s *vdp, prt_channel_t *channel, struct prt_packet_s *packet)

  *Sends a packet from a host VDP.*

- void prt_channel_push_device (struct prt_vdp_s *vdp, prt_channel_t *channel, struct prt_packet_s *packet)

  *Sends a packet from a device VDP. Puts a callback in the VDP's stream. When reached, the callback puts the transfer in the channel's stream.*

- struct prt_packet_s * prt_channel_pop (prt_channel_t *channel)

  *Fetches a packef from a channel. Does not decrement the number of active references. The packet leaves the channel, but enters the VDP.*

- int prt_channel_empty (prt_channel_t *channel)

  *Checks if a channel is empty.*

- int prt_channel_compare (void *channel1, void *channel2)

  *Compares two channels.*

- void prt_channel_off (prt_channel_t *channel)

  *Deactivates a channel. Newly created channels are active. Inactive channels are excluded from readiness checks.*

- void prt_channel_on (prt_channel_t *channel)

  *Activates a channel. Newly created channels are active. Inactive channels are excluded from readiness checks.*

### 6.19.1 Detailed Description

PRT data channel.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_channel.h.

### 6.19.2 Typedef Documentation

#### 6.19.2.1 typedef struct prt_channel_s prt_channel_t

VDP's data channel. Implements a data link between a pair of VDPs. Identifies the source and destination VDPs by tuples. Contains a thread-safe list of data packets.

The in_stream is used when the recipient device pulls: host->device, device->device (second stage). The out_stream is used when the sender device pushes: device->host, device->device (first stage).

### 6.19.3 Function Documentation

#### 6.19.3.1 int prt_channel_compare ( void * *channel1*, void * *channel2* )

Compares two channels.

**Parameters**

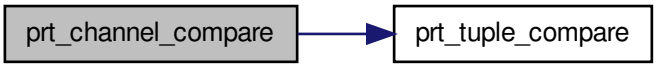
| | |
|---|---|
| *channel1* | – The first channel. |
| *channel2* | – The second channel. |

**Return values**

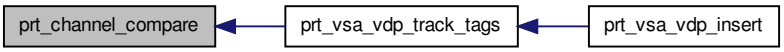
| | |
|---|---|
| *-1* | channel1 is less than channel2. |
| *0* | channel1 is equal to channel2. |
| *1* | channel1 is greater than channel2. |

Definition at line 264 of file prt_channel.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.19.3.2 void prt_channel_delete ( prt_channel_t * *channel* )

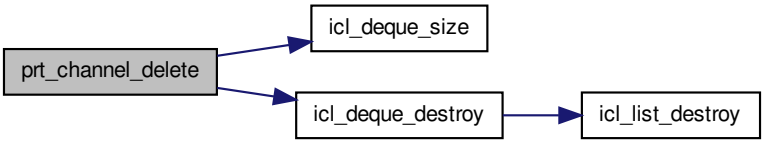
Destroys a channel.

**Parameters**

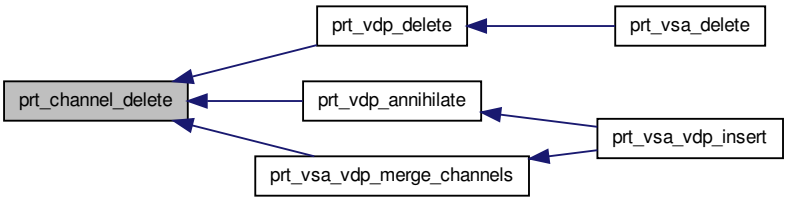
| | |
|---|---|
| *channel* | – The channel to destroy. |

Definition at line 70 of file prt_channel.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.19.3.3 int prt_channel_empty ( prt_channel_t * *channel* )

Checks if a channel is empty.

**Parameters**

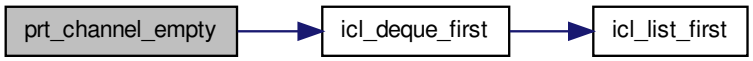
| | |
|---|---|
| *channel* | – The channel to check. |

**Return values**

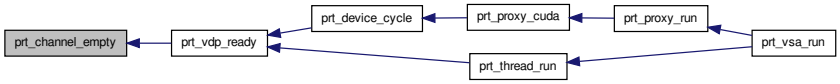
| | |
|---|---|
| *1* | if the channel is empty. |
| *0* | if the channel is not empty. |

Definition at line 243 of file prt_channel.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.19.3.4 void prt_channel_off ( prt_channel_t * *channel* )

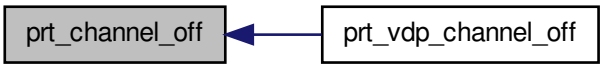
Deactivates a channel. Newly created channels are active. Inactive channels are excluded from readiness checks.

**Parameters**

| | |
|---|---|
| *channel* | – The channel to deactivate. |

Definition at line 292 of file prt_channel.c.

Here is the caller graph for this function:



#### 6.19.3.5 void prt_channel_on ( prt_channel_t * *channel* )

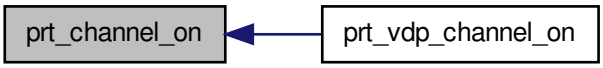
Activates a channel. Newly created channels are active. Inactive channels are excluded from readiness checks.

**Parameters**

| | |
|---|---|
| *channel* | - The channel to activate. |

Definition at line 306 of file prt_channel.c.

Here is the caller graph for this function:




#### 6.19.3.6 struct prt_packet_s* prt_channel_pop ( prt_channel_t * *channel* )

Fetches a packef from a channel. Does not decrement the number of active references. The packet leaves the channel, but enters the VDP.

**Parameters**

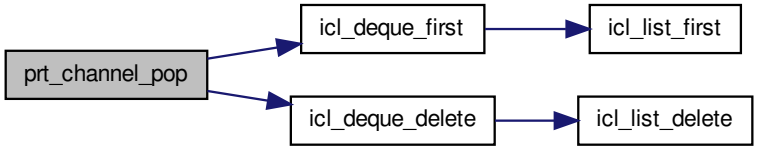
| | |
|---|---|
| *channel* | – The channel to fetch the packet from. |

**Returns**

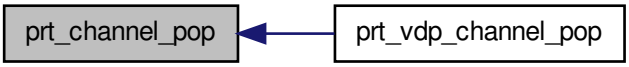
A data packet.

Definition at line 219 of file prt_channel.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.19.3.7 void prt_channel_push_device ( prt_vdp_t * *vdp,* prt_channel_t * *channel,* prt_packet_t * *packet* )

Sends a packet from a device VDP. Puts a callback in the VDP's stream. When reached, the callback puts the transfer in the channel's stream.

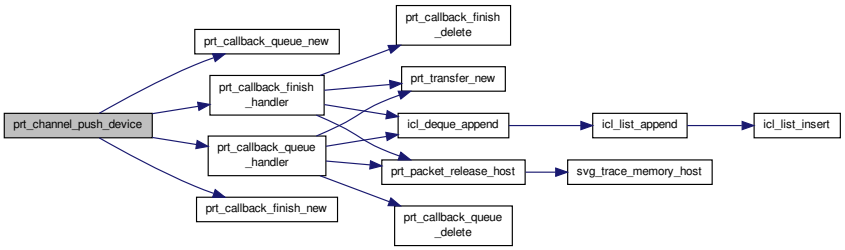
There is no need to set the device here. This function is called by prt_vdp_channel_push, which is called from VDP code, where the device is already set.

**Parameters**

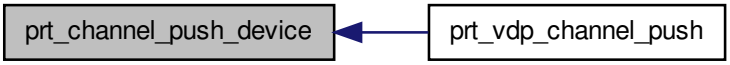
| | |
|---|---|
| *vdp* | – The device VDP sending the packet. |
| *channel* | – The channel to send the packet to. |
| *packet* | – The packet to send. |

Definition at line 151 of file prt_channel.c.

Here is the call graph for this function:




Here is the caller graph for this function:



#### 6.19.3.8 void prt_channel_push_host ( prt_vdp_t * *vdp*, prt_channel_t * *channel*, prt_packet_t * *packet* )

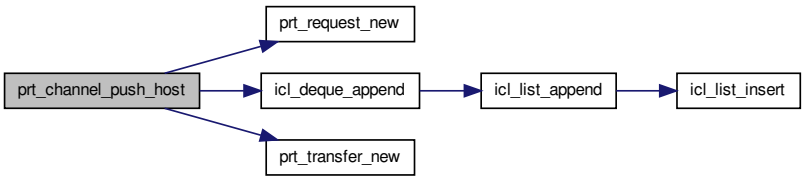
Sends a packet from a host VDP.

**Parameters**

| | |
|---|---|
| *vdp* | – The host VDP sending the packet. |
| *channel* | – The channel to send the packet to. |
| *packet* | – The packet to send. |

Definition at line 104 of file prt_channel.c.

Here is the call graph for this function:



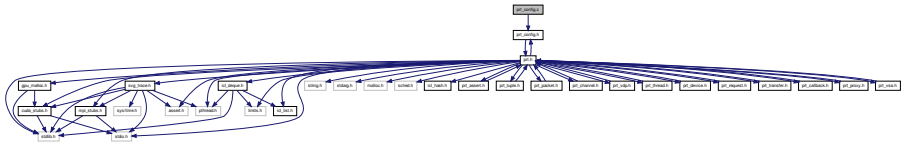

Here is the caller graph for this function:


## 6.20 prt_config.c File Reference

PRT configuration.

```
#include "prt_config.h"
```

Include dependency graph for prt_config.c:



## Functions

- prt_config_t * prt_config_new ()

  *Creates a new configuration object.*
- void prt_config_delete (prt_config_t *config)

  *Destroys a configuration object.*

### 6.20.1 Detailed Description

PRT configuration.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_config.c.

### 6.20.2 Function Documentation

#### 6.20.2.1 void prt_config_delete ( prt_config_t * *config* )

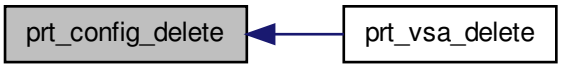
Destroys a configuration object.

**Parameters**

| | |
|---|---|
| *config* | – The configuration object to destroy. |

Definition at line 39 of file prt_config.c.

Here is the caller graph for this function:




#### 6.20.2.2 prt_config_t* prt_config_new ( )

Creates a new configuration object.

**Returns**

New configuration object with default values.

Definition at line 19 of file prt_config.c.

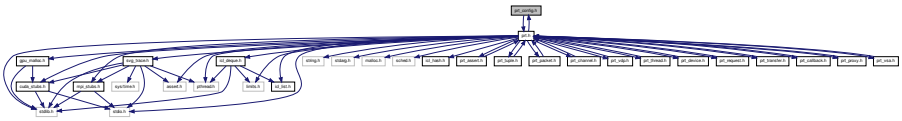
Here is the caller graph for this function:


## 6.21 prt_config.h File Reference

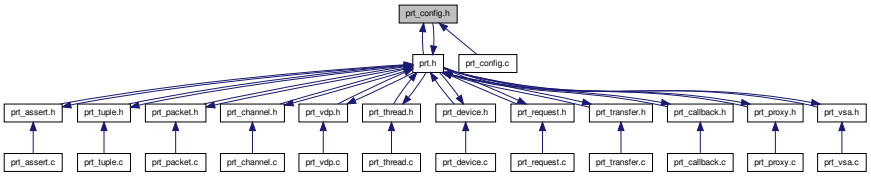
PRT configuration.

```
#include "prt.h"
```

Include dependency graph for prt_config.h:



This graph shows which files directly or indirectly include this file:




### Data Structures

- struct prt_config_s

  *PRT configuration.*

### Typedefs

- typedef enum prt_config_param_e prt_config_param_t

*PRT configuration parameters.*

- typedef enum prt_config_value_e prt_config_value_t

  *Values for PRT configuration parameters.*

- typedef struct prt_config_s prt_config_t

  *PRT configuration.*

## Enumerations

- enum prt_config_param_e { **PRT_VDP_SCHEDULING**, **PRT_SVG_TRACING** }

  *PRT configuration parameters.*

- enum prt_config_value_e { **PRT_VDP_SCHEDULING_AGGRESSIVE**, **PRT_VDP_SCHEDULING_LAZY**, **PRT_SVG_TRACING_ON**, **PRT_SVG_TRACING_OFF** }

  *Values for PRT configuration parameters.*

## Functions

- prt_config_t * prt_config_new ()

  *Creates a new configuration object.*

- void prt_config_delete (prt_config_t *config)

  *Destroys a configuration object.*

### 6.21.1 Detailed Description

PRT configuration.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_config.h.

### 6.21.2 Function Documentation

#### 6.21.2.1 void prt_config_delete ( prt_config_t * *config* )

Destroys a configuration object.

**Parameters**

| | |
|---|---|
| *config* | – The configuration object to destroy. |

Definition at line 39 of file prt_config.c.

Here is the caller graph for this function:


#### 6.21.2.2 prt_config_t* prt_config_new ( )

Creates a new configuration object.

**Returns**

New configuration object with default values.

Definition at line 19 of file prt_config.c.

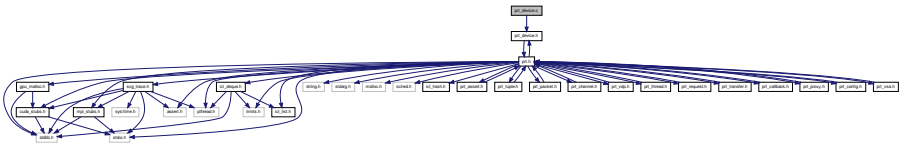
Here is the caller graph for this function:


## 6.22 prt_device.c File Reference

PRT device.

```
#include "prt_device.h"
```

Include dependency graph for prt_device.c:



### Functions

- prt_device_t * prt_device_new (int rank, int accelerator, int agent_rank)

*Creates a new device.*

- void prt_device_delete (prt_device_t *device)

  *Destroys a device.*

- void prt_device_cycle (prt_device_t *device)

  *Implements device processing cycle.*

### 6.22.1 Detailed Description

PRT device.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_device.c.

### 6.22.2 Function Documentation

#### 6.22.2.1 void prt_device_cycle ( prt_device_t * *device* )

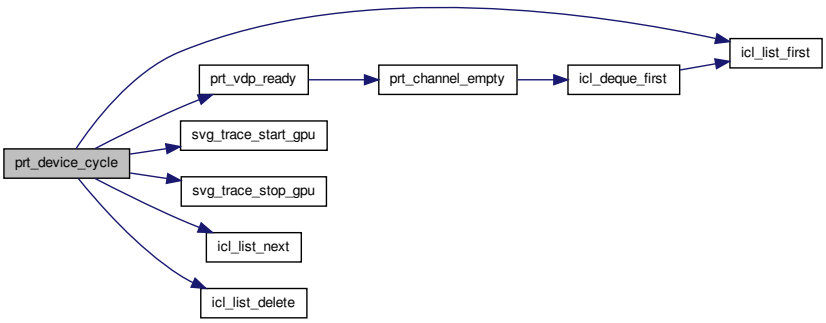
Implements device processing cycle.

**Parameters**

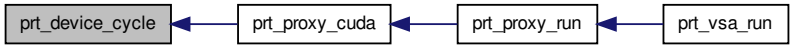
| | |
|---|---|
| *device* | – The device to cycle. |

Definition at line 67 of file prt_device.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.22.2.2 void prt_device_delete ( prt_device_t * *device* )

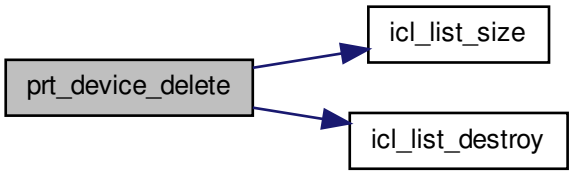
Destroys a device.

**Parameters**

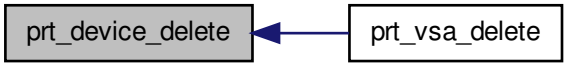
| | |
|---|---|
| *device* | – The device to destroy. |

Definition at line 49 of file prt_device.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.22.2.3 prt_device_t* prt_device_new ( int *rank,* int *accelerator,* int *agent_rank* )

Creates a new device.

**Parameters**

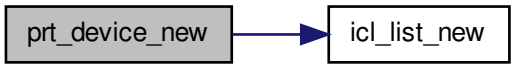
| | |
|---|---|
| *rank* | – The local rank of the device. |
| *accelerator* | – The global rank of the device. |
| *agent_rank* | – The rank of the communication agent. |

**Returns**

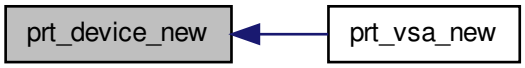
A new device object.

Definition at line 23 of file prt_device.c.

Here is the call graph for this function:




Here is the caller graph for this function:

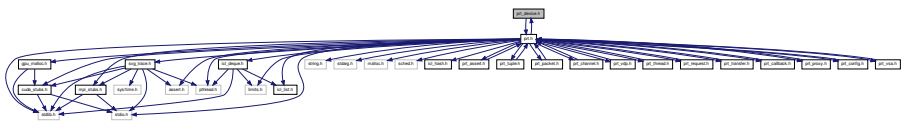



## 6.23 prt_device.h File Reference

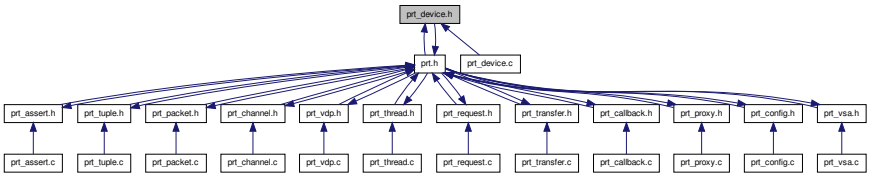
PRT device.

```
#include "prt.h"
```

Include dependency graph for prt_device.h:



This graph shows which files directly or indirectly include this file:




## Data Structures

- struct prt_device_s

  *VSA's accelerator device. Represents a hardware accelerator. Currently synonymous with an Nvidia GPU.*

## Typedefs

- typedef struct prt_device_s prt_device_t

  *VSA's accelerator device. Represents a hardware accelerator. Currently synonymous with an Nvidia GPU.*

## Functions

- prt_device_t ∗ prt_device_new (int rank, int accelerator, int agent_rank)

  *Creates a new device.*
- void prt_device_delete (prt_device_t ∗device)

  *Destroys a device.*
- void prt_device_cycle (prt_device_t ∗device)

  *Implements device processing cycle.*

### 6.23.1 Detailed Description

PRT device.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_device.h.

### 6.23.2 Typedef Documentation

#### 6.23.2.1 typedef struct prt_device_s prt_device_t

VSA's accelerator device. Represents a hardware accelerator. Currently synonymous with an Nvidia GPU.

"finished" is a one-directional synchronization variable. Therefore declered volatile, but no need for atomic access.

### 6.23.3 Function Documentation

#### 6.23.3.1 void prt_device_cycle ( prt_device_t * *device* )

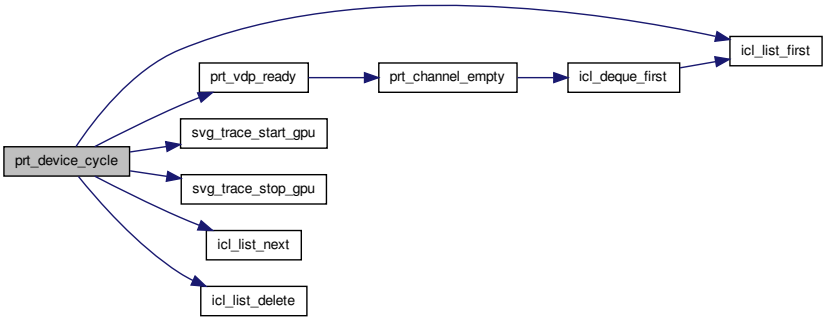
Implements device processing cycle.

**Parameters**

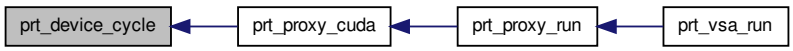
| | |
|---|---|
| *device* | – The device to cycle. |

Definition at line 67 of file prt_device.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.23.3.2 void prt_device_delete ( prt_device_t * *device* )

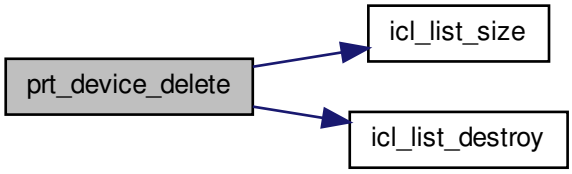
Destroys a device.

**Parameters**

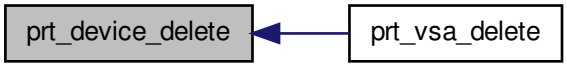
| | |
|---|---|
| *device* | – The device to destroy. |

Definition at line 49 of file prt_device.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.23.3.3 prt_device_t* prt_device_new ( int *rank,* int *accelerator,* int *agent_rank* )

Creates a new device.

**Parameters**

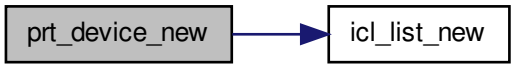
| | |
|---|---|
| *rank* | – The local rank of the device. |
| *accelerator* | – The global rank of the device. |
| *agent_rank* | – The rank of the communication agent. |

**Returns**

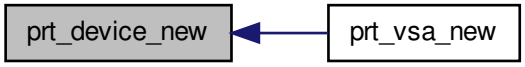
A new device object.

Definition at line 23 of file prt_device.c.

Here is the call graph for this function:




Here is the caller graph for this function:

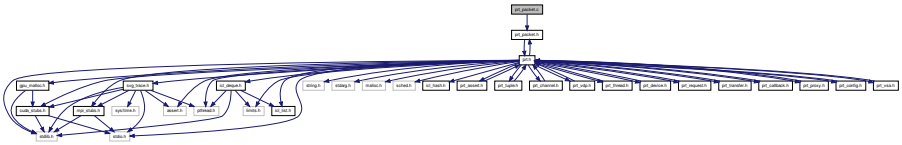



## 6.24 prt_packet.c File Reference

PRT data packet.

```
#include "prt_packet.h"
```

Include dependency graph for prt_packet.c:



### Functions

- prt_packet_t * prt_packet_new_host (size_t size, void *data)

  *Creates a new packet in host memory. Allocates the size amount of data if a NULL pointer is passed.*
- prt_packet_t * prt_packet_new_device (size_t size, void *data, prt_vdp_t *vdp)

  *Creates a new packet in device memory. Allocates the size amount of data if a NULL pointer is passed.*
- void prt_packet_resize_host (prt_packet_t *packet, size_t size)

*Resizes a packet in host memory. Used to resize placeholder packets for incoming MPI messages, wich initially are allocated with the maximum packet size.*

- void prt_packet_release_host (prt_packet_t *packet)

  *Releases a packet located in host memory. Decrements the number of active references. Destroys the packet when the last reference is removed.*

- void prt_packet_release_device (prt_packet_t *packet)

  *Releases a packet located in device memory. Decrements the number of active references. Destroys the packet when the last reference is removed.*

- void prt_packet_host_to_device (prt_packet_t *src_packet, prt_channel_t *channel)

  *Transfers a packet from the host to a device.*

- void prt_packet_device_to_host (prt_packet_t *src_packet, prt_channel_t *channel)

  *Transfers a packet from the host to a device.*

- void prt_packet_device_to_device (prt_packet_t *src_packet, prt_channel_t *channel)

  *Transfers a packet from a device to another device.*

- void prt_packet_device_to_device_direct (prt_packet_t *src_packet, prt_channel_t *channel)

  *Transfers a packet from a device to another device. Uses a direct copy, without involving the host.*

- void prt_packet_device_mpi_to_host (prt_packet_t *src_packet, prt_channel_t *channel, int agent)

  *Initiates an MPI transfer from a device. Sends a packet from a device to the host. Then requests an MPI transfer from the host.*

### 6.24.1 Detailed Description

PRT data packet.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_packet.c.

### 6.24.2 Function Documentation

#### 6.24.2.1 void prt_packet_device_mpi_to_host ( prt_packet_t * *src_packet*, prt_channel_t * *channel*, int *agent* )

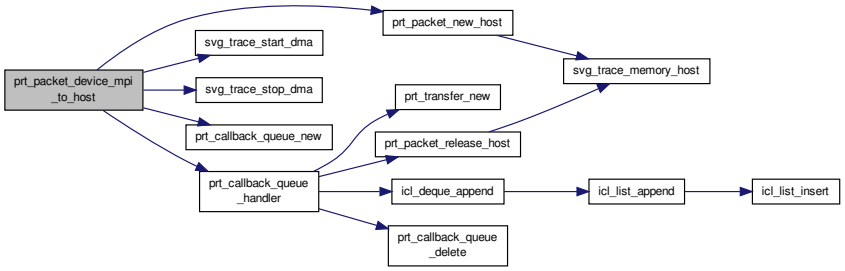
Initiates an MPI transfer from a device. Sends a packet from a device to the host. Then requests an MPI transfer from the host.

**Parameters**

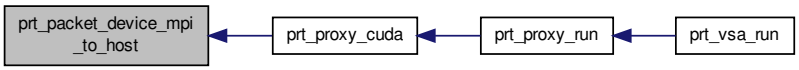
| | |
|---|---|
| *src_packet* | – The packet to transfer. |
| *channel* | – The destination channel. |

Definition at line 306 of file prt_packet.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.24.2.2 void prt_packet_device_to_device ( prt_packet_t * *src_packet,* prt_channel_t * *channel* )

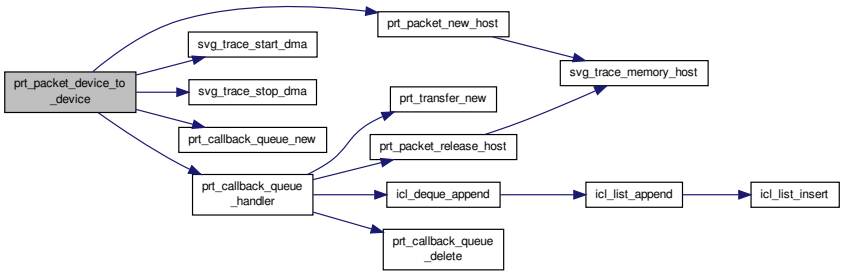
Transfers a packet from a device to another device.

**Parameters**

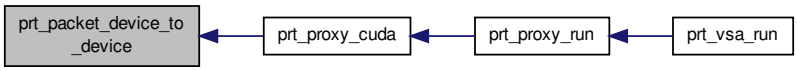
| | |
|---|---|
| *src_packet* | – The packet to transfer. |
| *channel* | – The destination channel. |

Definition at line 238 of file prt_packet.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.24.2.3 void prt_packet_device_to_device_direct ( prt_packet_t * *src_packet,* prt_channel_t * *channel* )

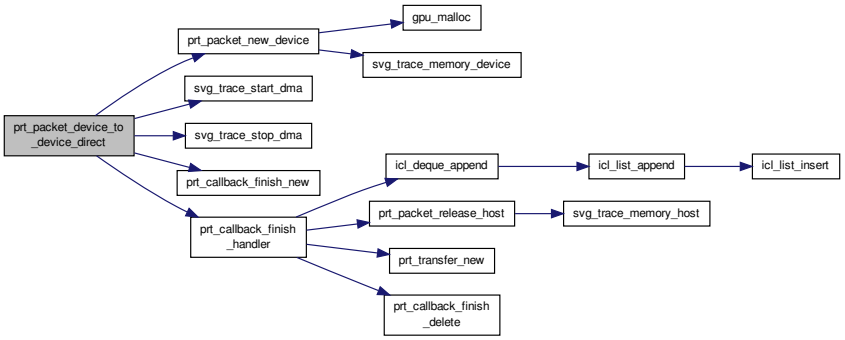
Transfers a packet from a device to another device. Uses a direct copy, without involving the host.

**Parameters**

| | |
|---|---|
| *src_packet* | – The packet to transfer. |
| *channel* | – The destination channel. |

Definition at line 270 of file prt_packet.c.

Here is the call graph for this function:




#### 6.24.2.4 void prt_packet_device_to_host ( prt_packet_t * *src_packet,* prt_channel_t * *channel* )

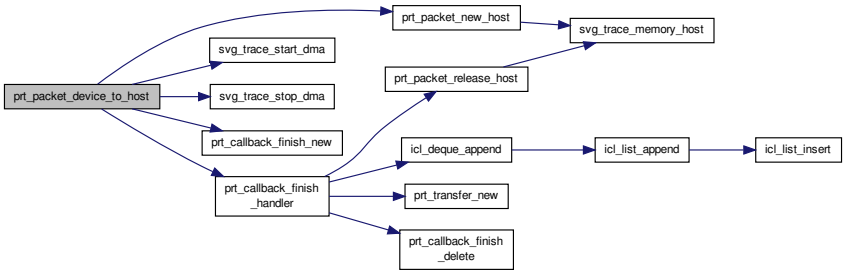
Transfers a packet from the host to a device.

**Parameters**

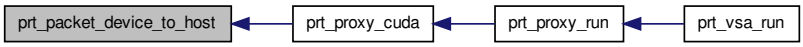
| | |
|---|---|
| *packet* | – The packet to transfer. |
| *channel* | – The destination channel. |
| *kind* | – The direction of the transfer. |

Definition at line 207 of file prt_packet.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.24.2.5 void prt_packet_host_to_device ( prt_packet_t * *src_packet,* prt_channel_t * *channel* )

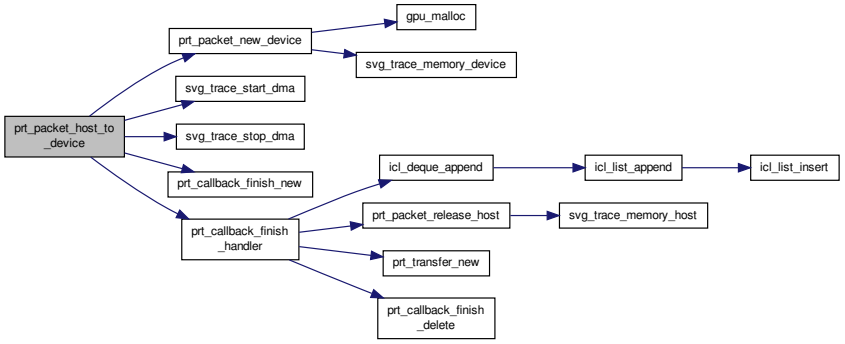
Transfers a packet from the host to a device.

**Parameters**

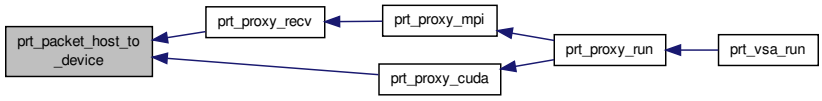
| | |
|---|---|
| *packet* | – The packet to transfer. |
| *channel* | – The destination channel. |
| *kind* | – The direction of the transfer. |

Definition at line 174 of file prt_packet.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.24.2.6 prt_packet_t* prt_packet_new_device ( size_t *size*, void * *data*, prt_vdp_t * *vdp* )

Creates a new packet in device memory. Allocates the size amount of data if a NULL pointer is passed.

Registers increased memory usage in both cases. This way the ending balance is expected to be zero. Packet release does not care (know) how the data was allocated.

**Parameters**

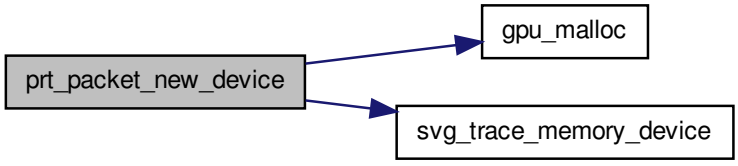
| | |
|---|---|
| *size* | – The size of the packet's data. |
| *data* | – The pointer to the packet's data. |
| *vdp* | – The VDP creating the packet. |

**Returns**

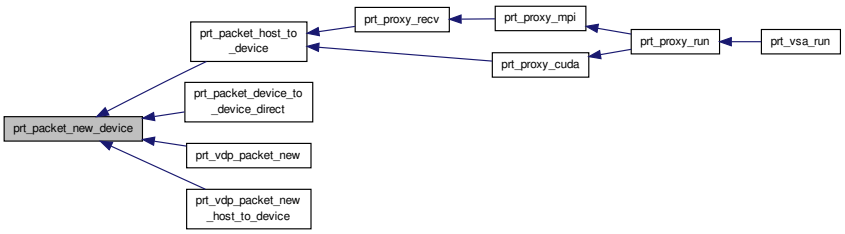
A new packet.

Definition at line 68 of file prt_packet.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.24.2.7 prt_packet_t* prt_packet_new_host ( size_t *size*, void * *data* )

Creates a new packet in host memory. Allocates the size amount of data if a NULL pointer is passed.

Registers increased memory usage in both cases. This way the ending balance is expected to be zero. Packet release does not care (know) how the data was allocated.

**Parameters**

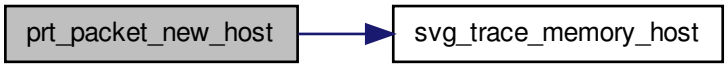
| | |
|---|---|
| *size* | – The size of packet's data. |
| *data* | – The pointer to the packet's data. |

**Returns**

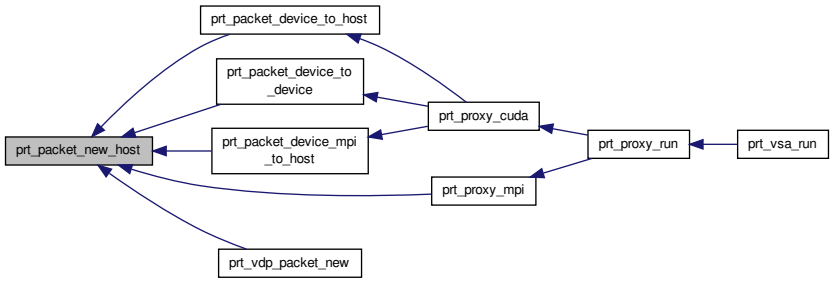
A new packet.

Definition at line 27 of file prt_packet.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.24.2.8 void prt_packet_release_device ( prt_packet_t * *packet* )

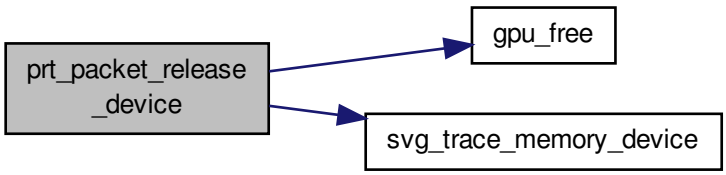
Releases a packet located in device memory. Decrements the number of active references. Destroys the packet when the last reference is removed.

**Parameters**

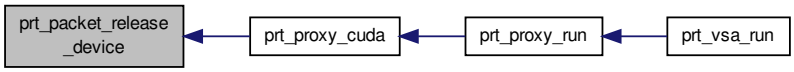
| | |
|---|---|
| *packet* | – The device packet to release. |

Definition at line 148 of file prt_packet.c.

Here is the call graph for this function:




Here is the caller graph for this function:



#### 6.24.2.9 void prt_packet_release_host ( prt_packet_t * *packet* )

Releases a packet located in host memory. Decrements the number of active references. Destroys the packet when the last reference is removed.

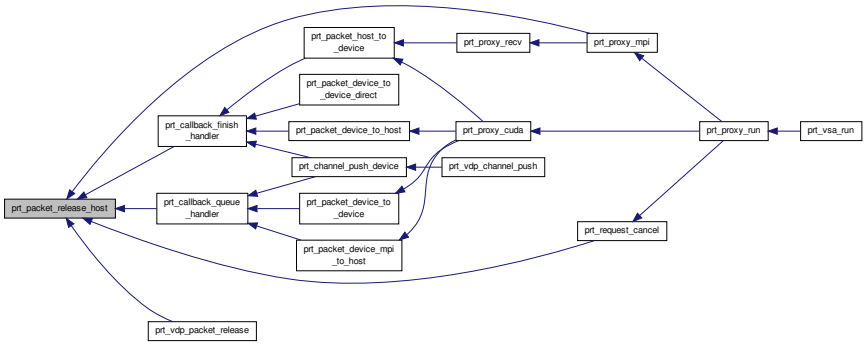
**Parameters**

| | |
|---|---|
| *packet* | – The host packet to release. |

Definition at line 127 of file prt_packet.c.

Here is the call graph for this function:


Here is the caller graph for this function:




#### 6.24.2.10 void prt_packet_resize_host ( prt_packet_t * *packet*, size_t *size* )

Resizes a packet in host memory. Used to resize placeholder packets for incoming MPI messages, wich initially are allocated with the maximum packet size.

**Parameters**

| | |
|---|---|
| *packet* | – The packet to resize. |
| *size* | – The new size in bytes. |

Definition at line 105 of file prt_packet.c.

Here is the call graph for this function:

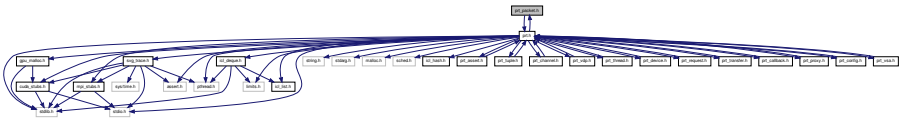

Here is the caller graph for this function:


## 6.25 prt_packet.h File Reference

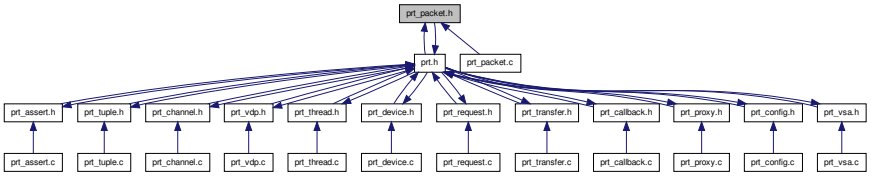
PRT data packet.

```
#include "prt.h"
```

Include dependency graph for prt_packet.h:



This graph shows which files directly or indirectly include this file:




### Data Structures

- struct prt_packet_s

  *VDP's data packet A packet of data transferred through VDP's channels.*

### Typedefs

- typedef struct prt_packet_s prt_packet_t

  *VDP's data packet A packet of data transferred through VDP's channels.*

### Functions

- prt_packet_t * prt_packet_new_host (size_t size, void *data)

  *Creates a new packet in host memory. Allocates the size amount of data if a NULL pointer is passed.*
- prt_packet_t * prt_packet_new_device (size_t size, void *data, struct prt_vdp_s *vdp)

  *Creates a new packet in device memory. Allocates the size amount of data if a NULL pointer is passed.*
- void prt_packet_resize_host (prt_packet_t *packet, size_t size)

  *Resizes a packet in host memory. Used to resize placeholder packets for incoming MPI messages, wich initially are allocated with the maximum packet size.*
- void prt_packet_release_host (prt_packet_t *packet)

  *Releases a packet located in host memory. Decrements the number of active references. Destroys the packet when the last reference is removed.*
- void prt_packet_release_device (prt_packet_t *packet)

  *Releases a packet located in device memory. Decrements the number of active references. Destroys the packet when the last reference is removed.*
- void prt_packet_host_to_device (prt_packet_t *src_packet, struct prt_channel_s *channel)

  *Transfers a packet from the host to a device.*
- void prt_packet_device_to_host (prt_packet_t *src_packet, struct prt_channel_s *channel)

  *Transfers a packet from the host to a device.*
- void prt_packet_device_to_device (prt_packet_t *src_packet, struct prt_channel_s *channel)

  *Transfers a packet from a device to another device.*
- void prt_packet_device_to_device_direct (prt_packet_t *src_packet, struct prt_channel_s *channel)

  *Transfers a packet from a device to another device. Uses a direct copy, without involving the host.*
- void prt_packet_device_mpi_to_host (prt_packet_t *src_packet, struct prt_channel_s *channel, int agent)

  *Initiates an MPI transfer from a device. Sends a packet from a device to the host. Then requests an MPI transfer from the host.*

## 6.25.1 Detailed Description

PRT data packet.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_packet.h.

## 6.25.2 Typedef Documentation

### 6.25.2.1 typedef struct prt_packet_s prt_packet_t

VDP's data packet A packet of data transferred through VDP's channels.

"num_refs" is a multi-access synchronization variable. Therefore, declared as volatile and accessed with atomics.

### 6.25.3 Function Documentation

#### 6.25.3.1 void prt_packet_device_mpi_to_host ( prt_packet_t * *src_packet,* prt_channel_t * *channel,* int *agent* )

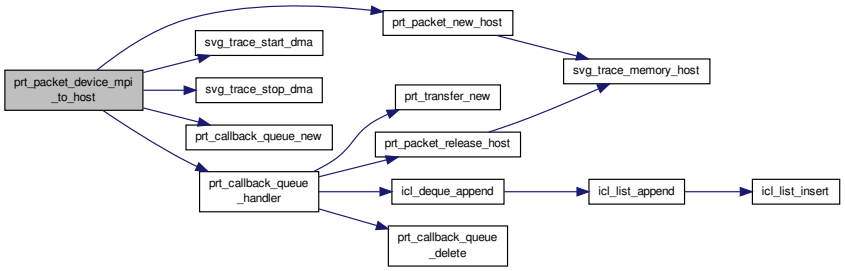
Initiates an MPI transfer from a device. Sends a packet from a device to the host. Then requests an MPI transfer from the host.

**Parameters**

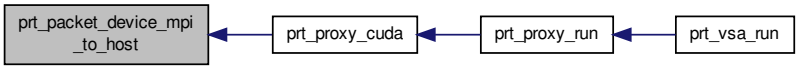
| | |
|---|---|
| *src_packet* | – The packet to transfer. |
| *channel* | – The destination channel. |

Definition at line 306 of file prt_packet.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.25.3.2 void prt_packet_device_to_device ( prt_packet_t * *src_packet,* prt_channel_t * *channel* )

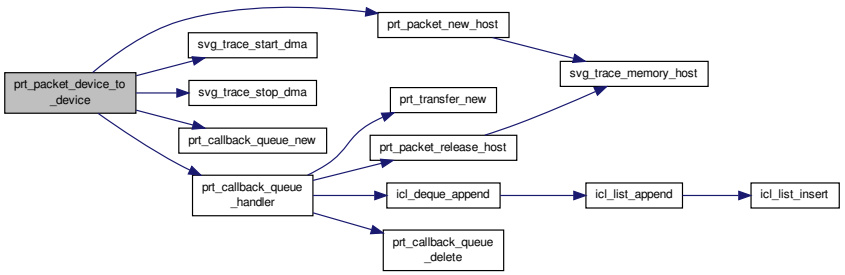
Transfers a packet from a device to another device.

**Parameters**

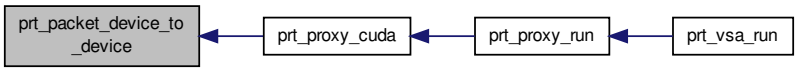
| | |
|---|---|
| *src_packet* | – The packet to transfer. |
| *channel* | – The destination channel. |

Definition at line 238 of file prt_packet.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.25.3.3 void prt_packet_device_to_device_direct ( prt_packet_t * *src_packet,* prt_channel_t * *channel* )

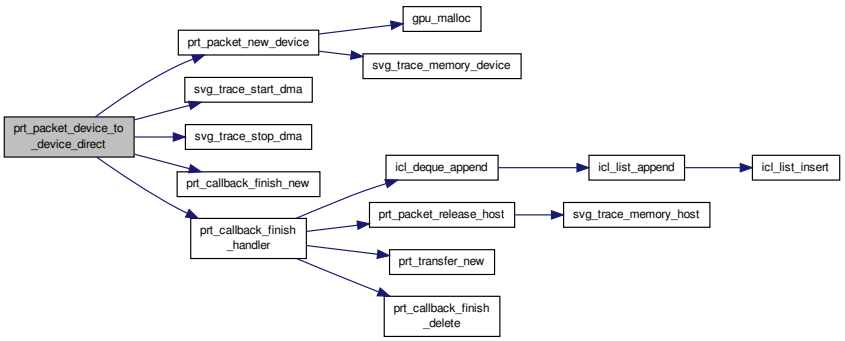
Transfers a packet from a device to another device. Uses a direct copy, without involving the host.

**Parameters**

| | |
|---|---|
| *src_packet* | – The packet to transfer. |
| *channel* | – The destination channel. |

Definition at line 270 of file prt_packet.c.

Here is the call graph for this function:



#### 6.25.3.4 void prt_packet_device_to_host ( prt_packet_t * *src_packet,* prt_channel_t * *channel* )

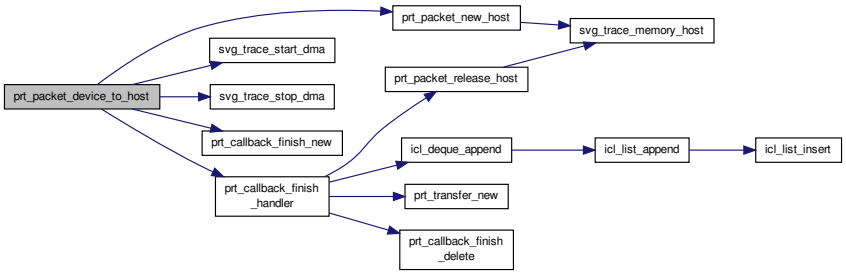
Transfers a packet from the host to a device.

**Parameters**

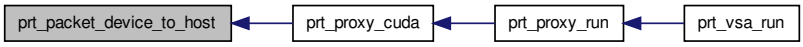
| | |
|---|---|
| *packet* | – The packet to transfer. |
| *channel* | – The destination channel. |
| *kind* | – The direction of the transfer. |

Definition at line 207 of file prt_packet.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.25.3.5 void prt_packet_host_to_device ( prt_packet_t * *src_packet,* prt_channel_t * *channel* )

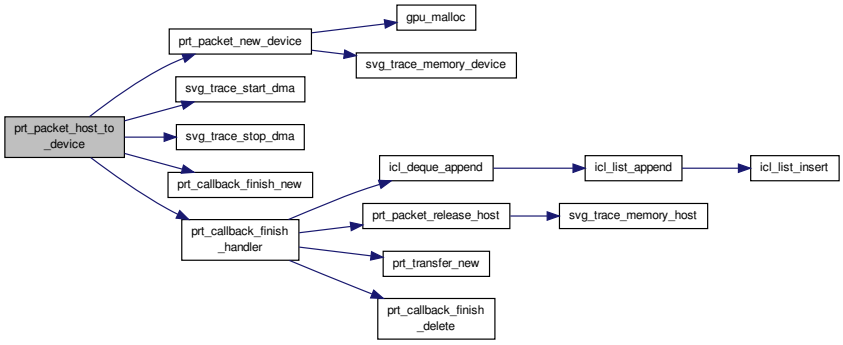
Transfers a packet from the host to a device.

**Parameters**

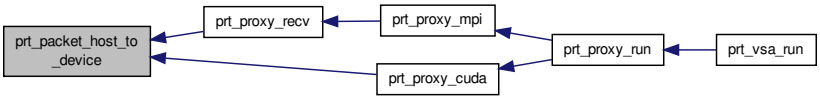
| | |
|---|---|
| *packet* | – The packet to transfer. |
| *channel* | – The destination channel. |
| *kind* | – The direction of the transfer. |

Definition at line 174 of file prt_packet.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.25.3.6 prt_packet_t* prt_packet_new_device ( size_t *size*, void * *data*, prt_vdp_t * *vdp* )

Creates a new packet in device memory. Allocates the size amount of data if a NULL pointer is passed.

Registers increased memory usage in both cases. This way the ending balance is expected to be zero. Packet release does not care (know) how the data was allocated.

**Parameters**

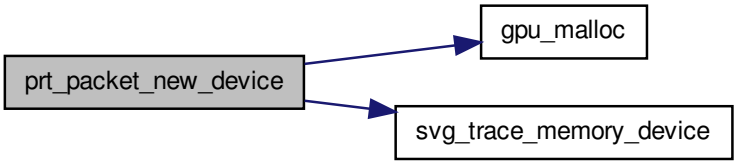
| | |
|---|---|
| *size* | – The size of the packet's data. |
| *data* | – The pointer to the packet's data. |
| *vdp* | – The VDP creating the packet. |

**Returns**

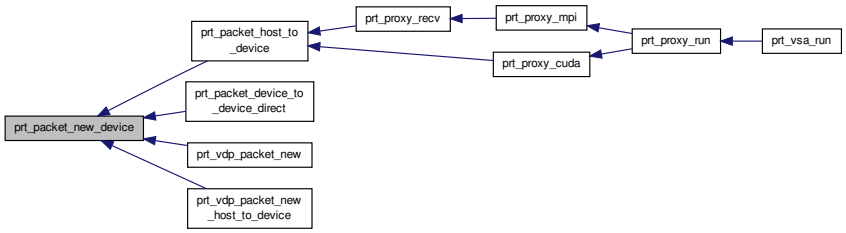
A new packet.

Definition at line 68 of file prt_packet.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.25.3.7 prt_packet_t* prt_packet_new_host ( size_t *size*, void * *data* )

Creates a new packet in host memory. Allocates the size amount of data if a NULL pointer is passed.

Registers increased memory usage in both cases. This way the ending balance is expected to be zero. Packet release does not care (know) how the data was allocated.

**Parameters**

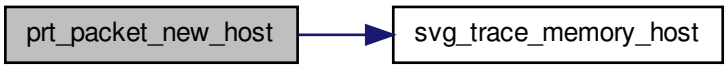
| | |
|---|---|
| *size* | – The size of packet's data. |
| *data* | – The pointer to the packet's data. |

**Returns**

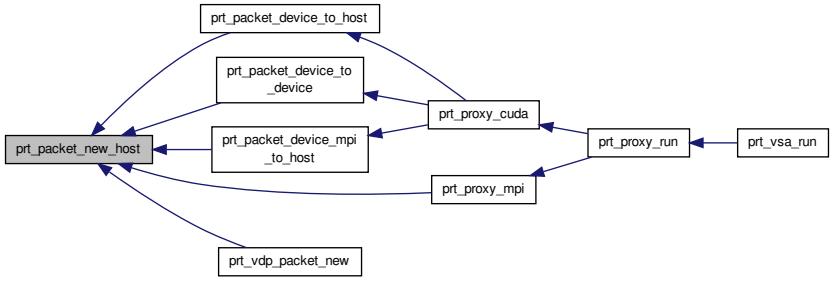
A new packet.

Definition at line 27 of file prt_packet.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.25.3.8 void prt_packet_release_device ( prt_packet_t * *packet* )

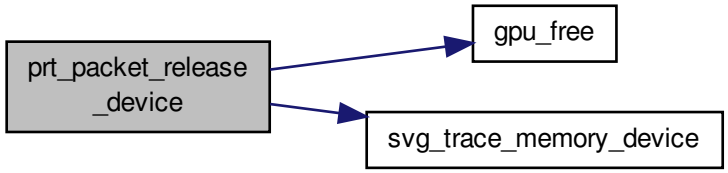
Releases a packet located in device memory. Decrements the number of active references. Destroys the packet when the last reference is removed.

**Parameters**

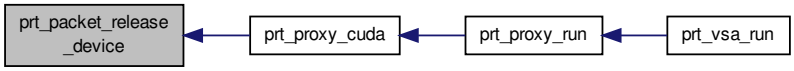
| | |
|---|---|
| *packet* | – The device packet to release. |

Definition at line 148 of file prt_packet.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.25.3.9 void prt_packet_release_host ( prt_packet_t * *packet* )

Releases a packet located in host memory. Decrements the number of active references. Destroys the packet when the last reference is removed.

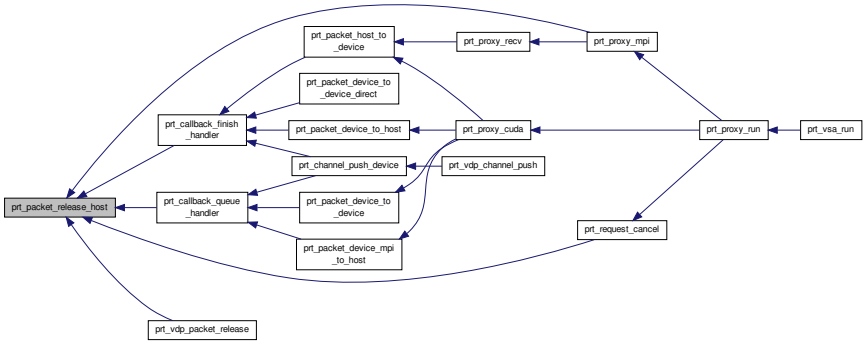
**Parameters**

| | |
|---|---|
| *packet* | – The host packet to release. |

Definition at line 127 of file prt_packet.c.

Here is the call graph for this function:


Here is the caller graph for this function:



#### 6.25.3.10 void prt_packet_resize_host ( prt_packet_t * *packet*, size_t *size* )

Resizes a packet in host memory. Used to resize placeholder packets for incoming MPI messages, wich initially are allocated with the maximum packet size.

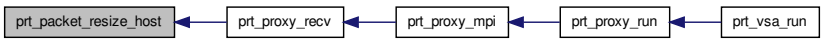
**Parameters**

| | |
|---|---|
| *packet* | – The packet to resize. |
| *size* | – The new size in bytes. |

Definition at line 105 of file prt_packet.c.

Here is the call graph for this function:


Here is the caller graph for this function:

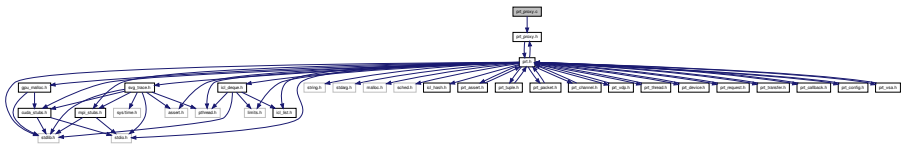



## 6.26 prt_proxy.c File Reference

PRT communication proxy.

```
#include "prt_proxy.h"
```

Include dependency graph for prt_proxy.c:



### Functions

- prt_proxy_t ∗ prt_proxy_new (int num_agents)

  *Creates a proxy.*
- void prt_proxy_delete (prt_proxy_t ∗proxy)

  *Destroys a proxy. Checks if all the lists are empty at the time of destruction. Not destroying the list of receives (destroyed at the end of the proxy's cycle).*
- void prt_proxy_max_channel_size (prt_proxy_t ∗proxy, prt_channel_t ∗channel)

  *Looks for maximum channel/packet size.*
- void prt_proxy_recv (prt_proxy_t ∗proxy, prt_request_t ∗request)

*Receives to a channel.*

- void prt_proxy_mpi (prt_proxy_t *proxy)

  *Implements the proxy's MPI cycle. Services all MPI requests.*

- void prt_proxy_cuda (prt_proxy_t *proxy)

  *Implements the proxy's CUDA cycle. Services all local transfer requests. Runs all device code.*

- double prt_proxy_run (prt_proxy_t *proxy)

  *Implements the proxy's production cycle. First, barriers with all MPI processes. Then, barriers with all local worker threads and starts measuring time. When finished, barriers with all local worker threads. Then, barriers with all MPI processes and stops the timer.*

### 6.26.1 Detailed Description

PRT communication proxy.

**Author**

Jakub Kurzak

The proxy executes all MPI communication and all CUDA code. In the case of multiple CUDA devices, the proxy services all the devices. The proxy implements device-to-device communications as staged, device-to-host + host-to-device communications. If supported, direct device-to-device communication is also possible, using the prt_packet_device_to_device_direct function (currently not used). The proxy also implements MPI transfers involving devices as staged, device-to-host + MPI communications.

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_proxy.c.

### 6.26.2 Function Documentation

#### 6.26.2.1 void prt_proxy_cuda ( prt_proxy_t * *proxy* )

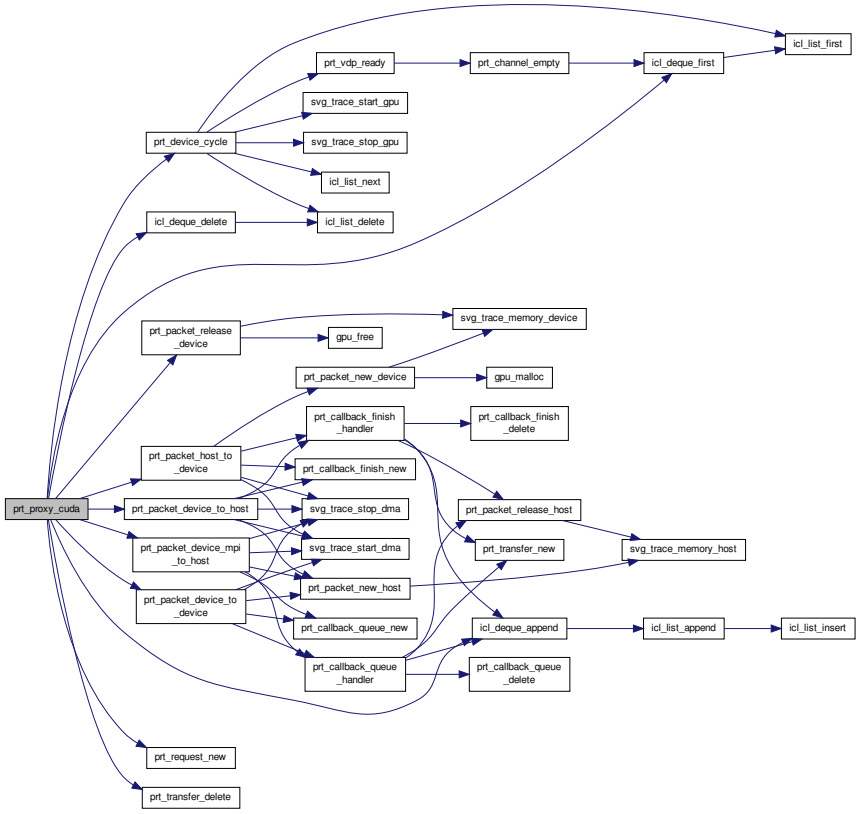
Implements the proxy's CUDA cycle. Services all local transfer requests. Runs all device code.

**Parameters**

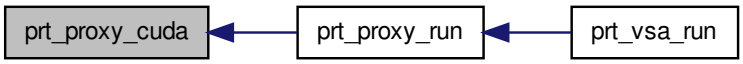
| | |
|---|---|
| *proxy* | – The proxy to cycle CUDA. |

Definition at line 256 of file prt_proxy.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.26.2.2 void prt_proxy_delete ( prt_proxy_t * *proxy* )

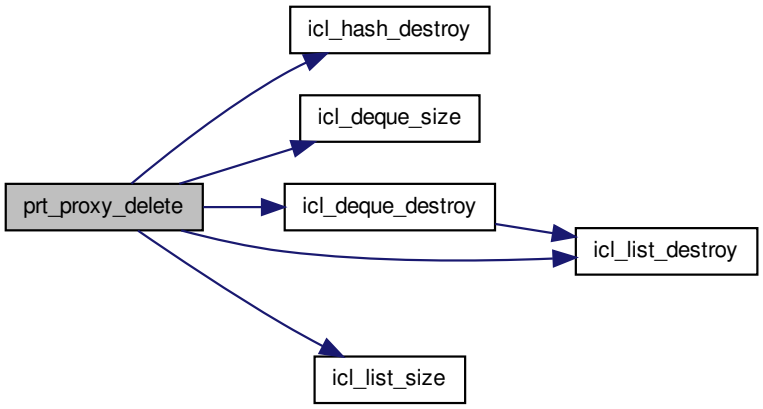
Destroys a proxy. Checks if all the lists are empty at the time of destruction. Not destroying the list of receives (destroyed at the end of the proxy's cycle).

**Parameters**

| | |
|---|---|
| *proxy* | – The proxy to destroy. |

Definition at line 86 of file prt_proxy.c.

Here is the call graph for this function:




Here is the caller graph for this function:


#### 6.26.2.3 void prt_proxy_max_channel_size ( prt_proxy_t * *proxy*, prt_channel_t * *channel* )

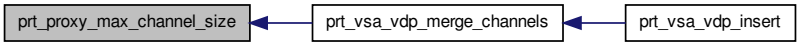
Looks for maximum channel/packet size.

**Parameters**

| | |
|---|---|
| *proxy* | – The proxy registering the size. |
| *channel* | – The channel to register the size of. |

Definition at line 132 of file prt_proxy.c.

Here is the caller graph for this function:




#### 6.26.2.4 void prt_proxy_mpi ( prt_proxy_t * *proxy* )

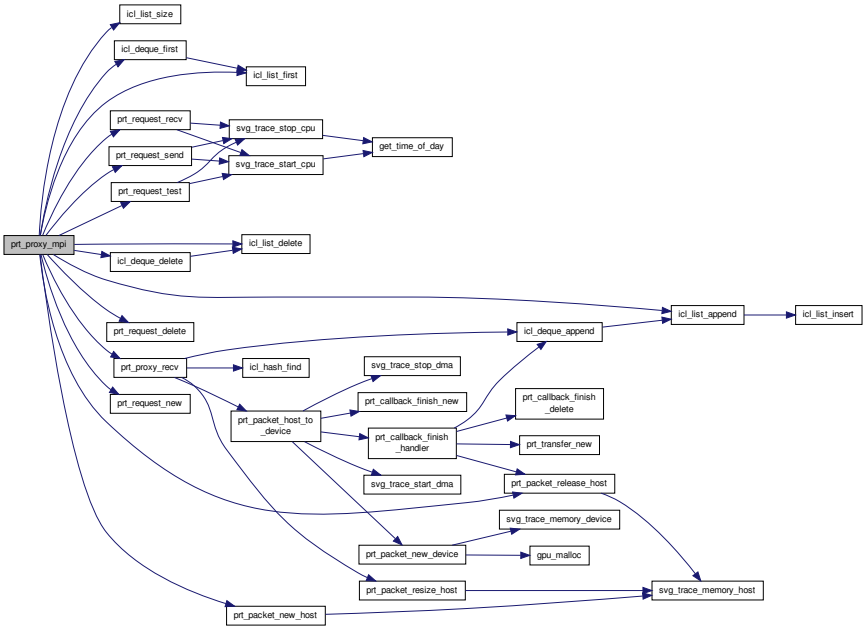
Implements the proxy's MPI cycle. Services all MPI requests.

**Parameters**

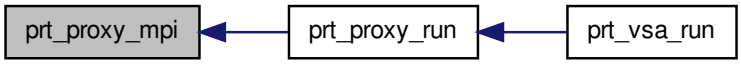
| | |
|---|---|
| *proxy* | – The proxy to cycle MPI. |

Definition at line 187 of file prt_proxy.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.26.2.5 prt_proxy_t* prt_proxy_new ( int *num_agents* )

Creates a proxy.

**Parameters**

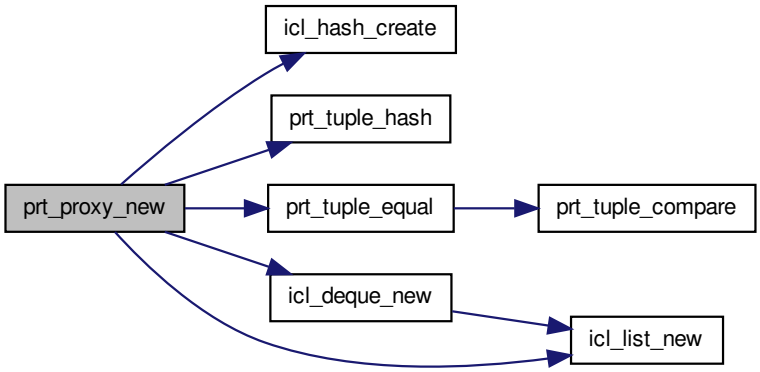
| | |
|---|---|
| *num_agents* | – The number of local agents (threads + devices). |

**Returns**

A new proxy.

Definition at line 30 of file prt_proxy.c.

Here is the call graph for this function:



Here is the caller graph for this function:


#### 6.26.2.6 void prt_proxy_recv ( prt_proxy_t * *proxy*, prt_request_t * *request* )

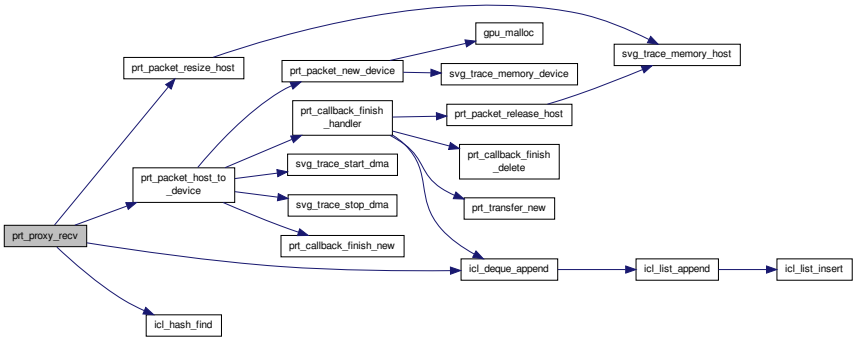
Receives to a channel.

**Parameters**

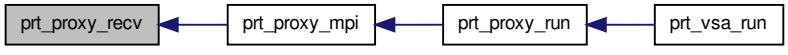
| | |
|---|---|
| *proxy* | – The proxy to receive the request. |
| *request* | – The receive request to process. |

Definition at line 150 of file prt_proxy.c.

Here is the call graph for this function:




Here is the caller graph for this function:



#### 6.26.2.7 double prt_proxy_run ( prt_proxy_t * *proxy* )

Implements the proxy's production cycle. First, barriers with all MPI processes. Then, barriers with all local worker threads and starts measuring time. When finished, barriers with all local worker threads. Then, barriers with all MPI processes and stops the timer.

**Parameters**

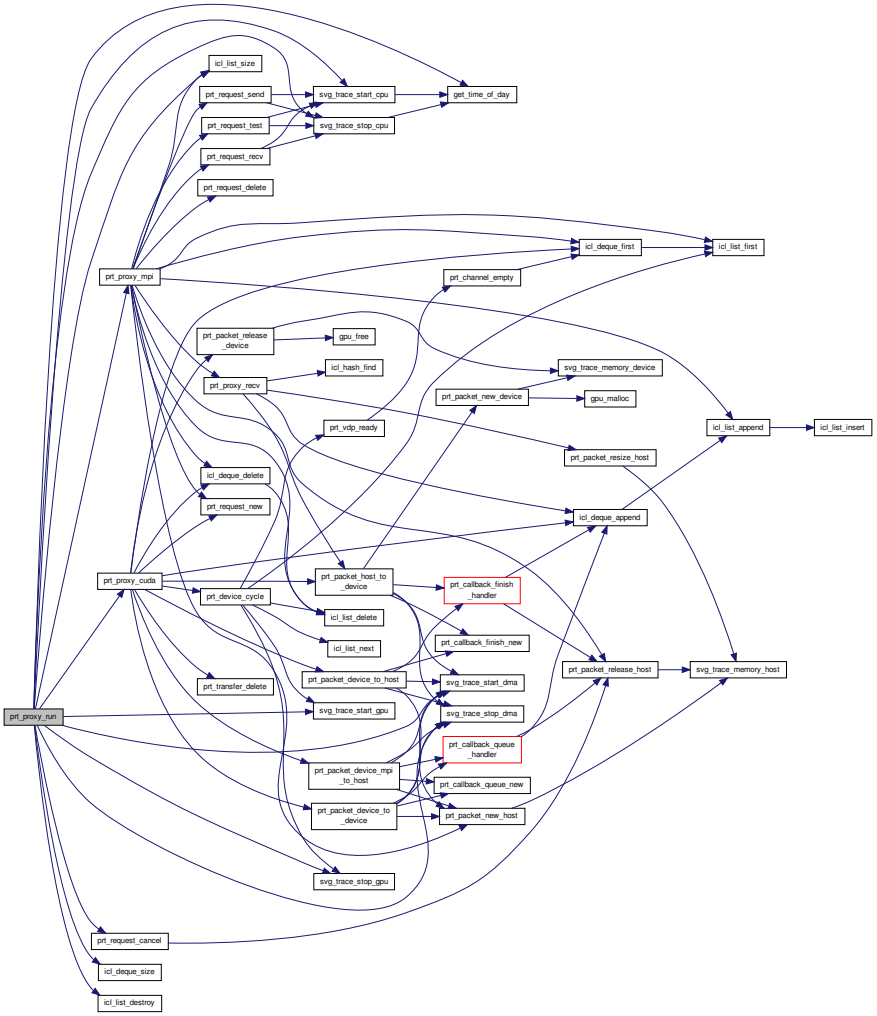
| | |
|---|---|
| *proxy* | – The proxy to run. |

**Returns**

The execution time.

Definition at line 319 of file prt_proxy.c.

Here is the call graph for this function:



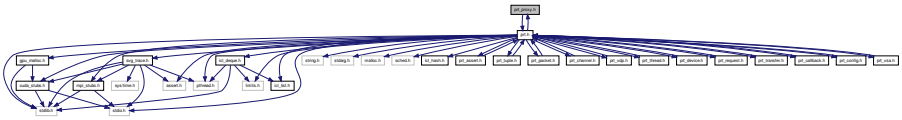
Here is the caller graph for this function:


## 6.27 prt_proxy.h File Reference

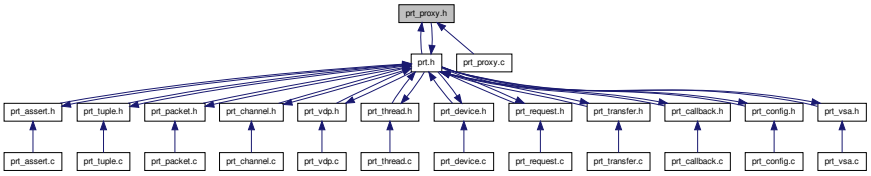
PRT communication proxy.

```
#include "prt.h"
```

Include dependency graph for prt_proxy.h:



This graph shows which files directly or indirectly include this file:




### Data Structures

- struct prt_proxy_s

  *VSA's proxy.*

### Macros

- #define PRT_PROXY_MAX_TAGS_PER_NODE 10003

  *Maximum tags per node. Size of the proxy's hash table for tags. It should be a prime number.*
- #define PRT_PROXY_MAX_SENDS_PER_AGENT 1

  *Maximum numbers of outstanding MPI send requests per agent.*
- #define PRT_PROXY_MAX_RECVS_PER_AGENT 1

  *Maximum numbers of outstanding MPI receive requests per agent.*

### Typedefs

- typedef struct prt_proxy_s prt_proxy_t

  *VSA's proxy.*

### Functions

- prt_proxy_t * prt_proxy_new (int num_agents)

  *Creates a proxy.*
- void prt_proxy_delete (prt_proxy_t *proxy)

  *Destroys a proxy. Checks if all the lists are empty at the time of destruction. Not destroying the list of receives (destroyed at the end of the proxy's cycle).*
- void prt_proxy_max_channel_size (prt_proxy_t *proxy, struct prt_channel_s *channel)

  *Looks for maximum channel/packet size.*
- void prt_proxy_recv (prt_proxy_t *proxy, struct prt_request_s *request)

  *Receives to a channel.*
- void prt_proxy_mpi (prt_proxy_t *proxy)

  *Implements the proxy's MPI cycle. Services all MPI requests.*
- void prt_proxy_cuda (prt_proxy_t *proxy)

  *Implements the proxy's CUDA cycle. Services all local transfer requests. Runs all device code.*
- double prt_proxy_run (prt_proxy_t *proxy)

  *Implements the proxy's production cycle. First, barriers with all MPI processes. Then, barriers with all local worker threads and starts measuring time. When finished, barriers with all local worker threads. Then, barriers with all MPI processes and stops the timer.*

## 6.27.1 Detailed Description

PRT communication proxy.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_proxy.h.

## 6.27.2 Typedef Documentation

### 6.27.2.1 typedef struct prt_proxy_s prt_proxy_t

VSA's proxy.

The reason for the num_callbacks counter is the following: Empty transfers queue does not mean there is nothing pending. Communication requests may be sitting in a stream waiting to be queued.

## 6.27.3 Function Documentation

### 6.27.3.1 void prt_proxy_cuda ( prt_proxy_t * *proxy* )

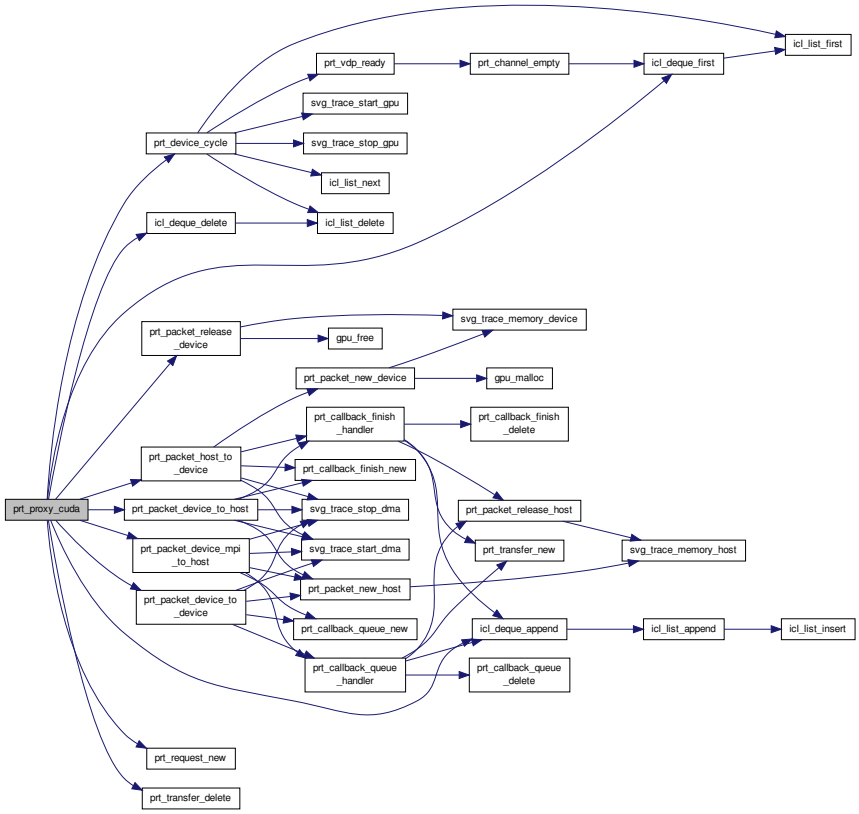
Implements the proxy's CUDA cycle. Services all local transfer requests. Runs all device code.

**Parameters**

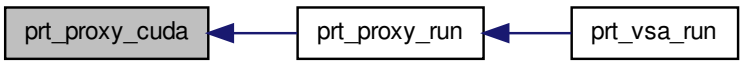
| | |
|---|---|
| *proxy* | – The proxy to cycle CUDA. |

Definition at line 256 of file prt_proxy.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.27.3.2 void prt_proxy_delete ( prt_proxy_t * *proxy* )

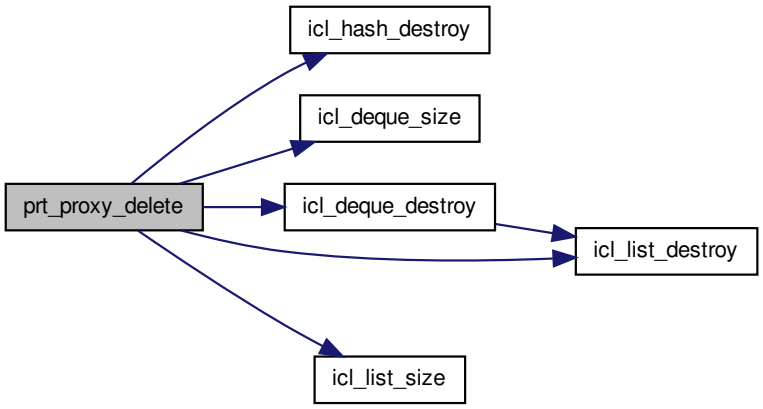
Destroys a proxy. Checks if all the lists are empty at the time of destruction. Not destroying the list of receives (destroyed at the end of the proxy's cycle).

**Parameters**

| | |
|---|---|
| *proxy* | – The proxy to destroy. |

Definition at line 86 of file prt_proxy.c.

Here is the call graph for this function:




Here is the caller graph for this function:


#### 6.27.3.3 void prt_proxy_max_channel_size ( prt_proxy_t * *proxy*, prt_channel_t * *channel* )

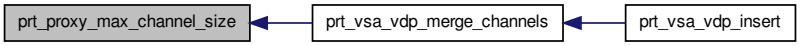
Looks for maximum channel/packet size.

**Parameters**

| | |
|---|---|
| *proxy* | – The proxy registering the size. |
| *channel* | – The channel to register the size of. |

Definition at line 132 of file prt_proxy.c.

Here is the caller graph for this function:




#### 6.27.3.4 void prt_proxy_mpi ( prt_proxy_t * *proxy* )

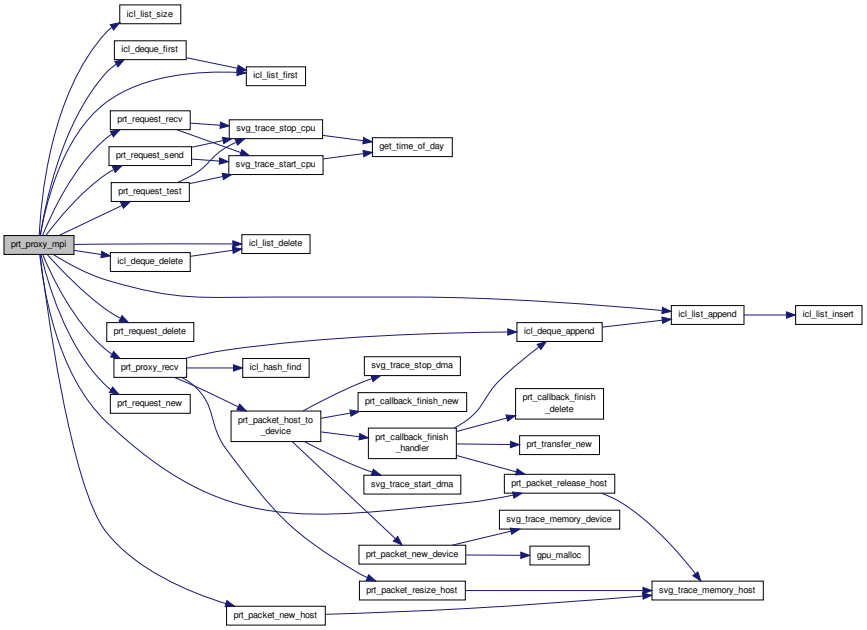
Implements the proxy's MPI cycle. Services all MPI requests.

**Parameters**

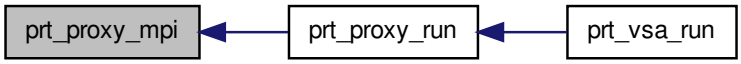
| | |
|---|---|
| *proxy* | – The proxy to cycle MPI. |

Definition at line 187 of file prt_proxy.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.27.3.5 prt_proxy_t* prt_proxy_new ( int *num_agents* )

Creates a proxy.

**Parameters**

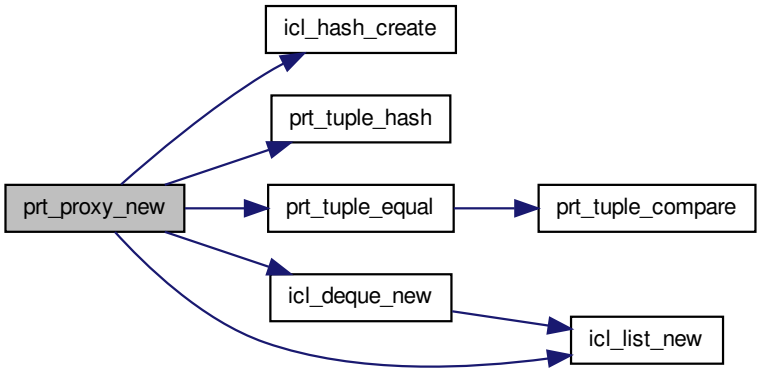
| | |
|---|---|
| *num_agents* | – The number of local agents (threads + devices). |

**Returns**

A new proxy.

Definition at line 30 of file prt_proxy.c.

Here is the call graph for this function:



Here is the caller graph for this function:


#### 6.27.3.6 void prt_proxy_recv ( prt_proxy_t * *proxy*, prt_request_t * *request* )

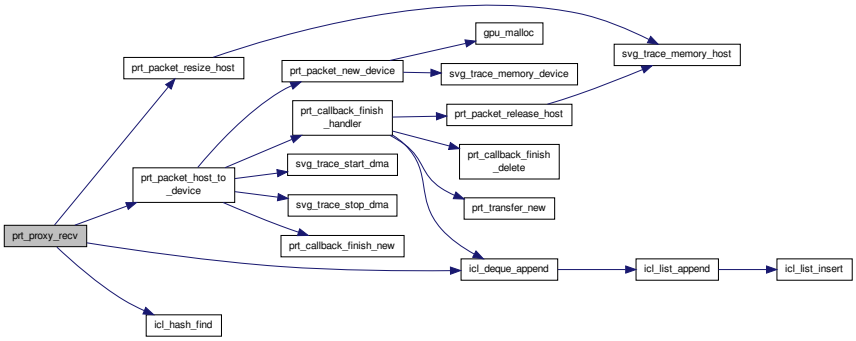
Receives to a channel.

**Parameters**

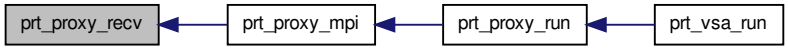
| | |
|---|---|
| *proxy* | – The proxy to receive the request. |
| *request* | – The receive request to process. |

Definition at line 150 of file prt_proxy.c.

Here is the call graph for this function:




Here is the caller graph for this function:



#### 6.27.3.7 double prt_proxy_run ( prt_proxy_t * *proxy* )

Implements the proxy's production cycle. First, barriers with all MPI processes. Then, barriers with all local worker threads and starts measuring time. When finished, barriers with all local worker threads. Then, barriers with all MPI processes and stops the timer.

**Parameters**

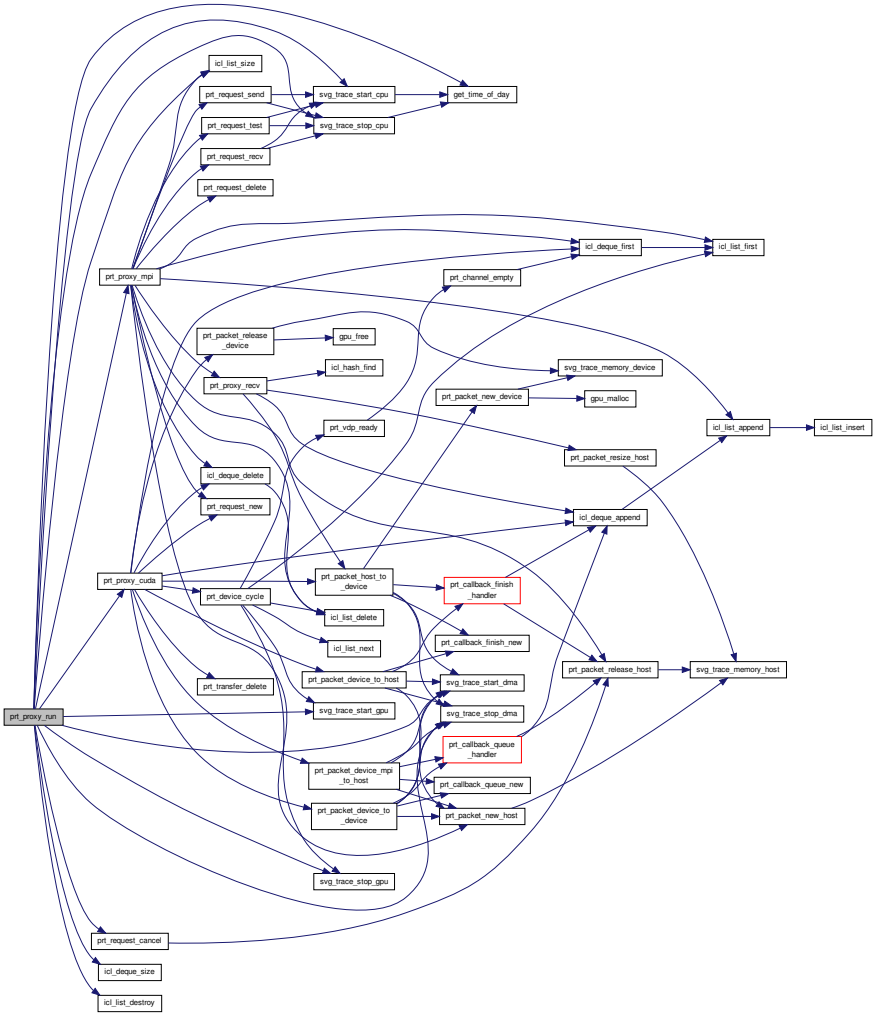
| | |
|---|---|
| *proxy* | – The proxy to run. |

**Returns**

The execution time.

Definition at line 319 of file prt_proxy.c.

Here is the call graph for this function:



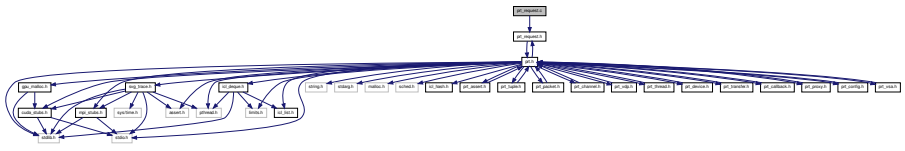
Here is the caller graph for this function:


## 6.28 prt_request.c File Reference

PRT communication request.

```
#include "prt_request.h"
```

Include dependency graph for prt_request.c:



### Functions

- prt_request_t * prt_request_new (prt_packet_t *packet, size_t size, int peer, int tag)

  *Creates a new request.*
- void prt_request_delete (prt_request_t *request)

  *Destroys a request.*
- void prt_request_send (prt_request_t *request)

  *Posts a send request.*
- void prt_request_recv (prt_request_t *request)

  *Posts a receive request.*
- int prt_request_test (prt_request_t *request)

  *Tests a request. Traces only completed requests.*
- void prt_request_cancel (prt_request_t *request)

  *Cancels a request. Cancels them MPI request, releases the packet, frees the request object.*

### 6.28.1 Detailed Description

PRT communication request.

**Author**

Jakub Kurzak



Definition in file prt_request.c.

### 6.28.2 Function Documentation

#### 6.28.2.1 void prt_request_cancel ( prt_request_t * *request* )

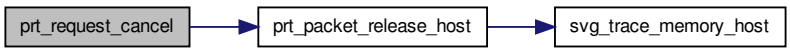
Cancels a request. Cancels them MPI request, releases the packet, frees the request object.

**Parameters**

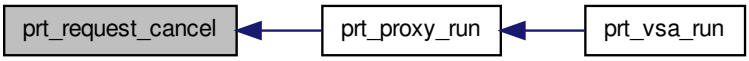
| | |
|---|---|
| *request* | – The request to cancel. |

Definition at line 126 of file prt_request.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.28.2.2 void prt_request_delete ( prt_request_t * *request* )

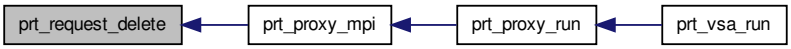
Destroys a request.

**Parameters**

| | |
|---|---|
| *request* | – The request to destroy. |

Definition at line 43 of file prt_request.c.

Here is the caller graph for this function:




#### 6.28.2.3 prt_request_t* prt_request_new ( prt_packet_t * *packet*, size_t *size*, int *peer*, int *tag* )

Creates a new request.

**Parameters**

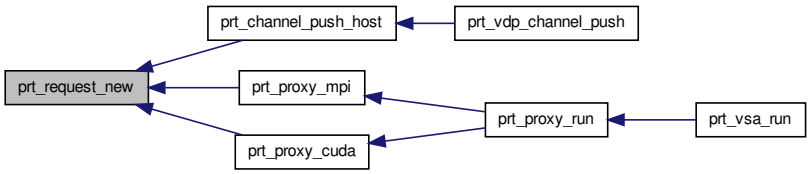
| | |
|---|---|
| *packet* | – The packet to create the request for. |
| *count* | – The number of data items. |
| *datatype* | – The type of data items. |
| *peer* | – The peer communicating node. |
| *tag* | – The MPI tag of the message. |

**Returns**

A new request.

Definition at line 25 of file prt_request.c.

Here is the caller graph for this function:




#### 6.28.2.4 void prt_request_recv ( prt_request_t * *request* )

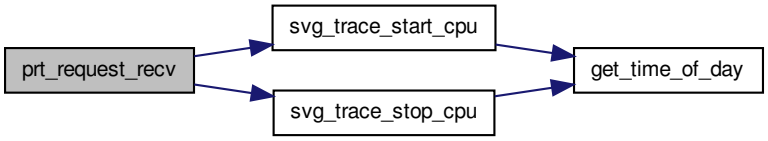
Posts a receive request.

**Parameters**

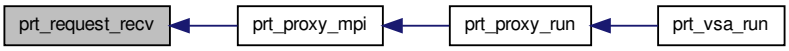
| | |
|---|---|
| *request* | – The receive request to post. |

Definition at line 77 of file prt_request.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.28.2.5 void prt_request_send ( prt_request_t * *request* )

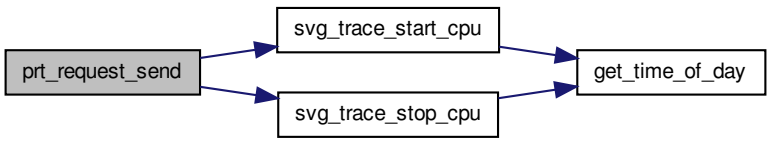
Posts a send request.

**Parameters**

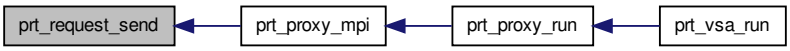
| | |
|---|---|
| *request* | – The send request to post. |

Definition at line 54 of file prt_request.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.28.2.6 int prt_request_test ( prt_request_t * *request* )

Tests a request. Traces only completed requests.

**Parameters**

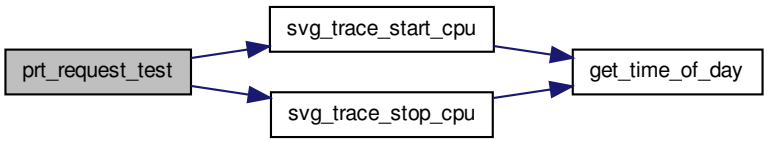
| | |
|---|---|
| *request* | – The request to test. |

**Return values**

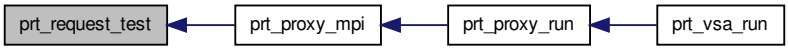
| | |
|---|---|
| *1* | if operation completed. |
| *0* | if operation is pending. |

Definition at line 104 of file prt_request.c.

Here is the call graph for this function:




Here is the caller graph for this function:

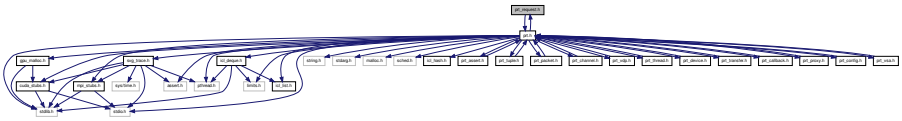



## 6.29 prt_request.h File Reference

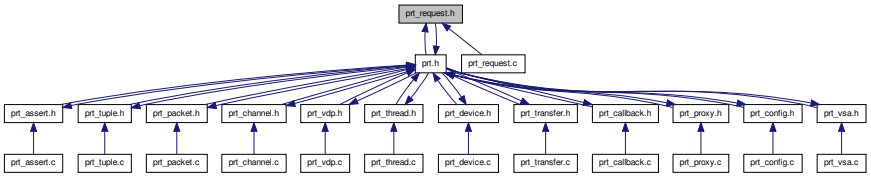
PRT communication request.

```
#include "prt.h"
```

Include dependency graph for prt_request.h:



This graph shows which files directly or indirectly include this file:




## Data Structures

- struct prt_request_s

  *MPI communication request for a packet. Contains a packet, some info, MPI request and MPI status.*

## Typedefs

- typedef struct prt_request_s prt_request_t

  *MPI communication request for a packet. Contains a packet, some info, MPI request and MPI status.*

## Functions

- prt_request_t * prt_request_new (struct prt_packet_s *packet, size_t size, int peer, int tag)

  *Creates a new request.*
- void prt_request_delete (prt_request_t *request)

  *Destroys a request.*
- void prt_request_send (prt_request_t *request)

  *Posts a send request.*
- void prt_request_recv (prt_request_t *request)

  *Posts a receive request.*
- int prt_request_test (prt_request_t *request)

  *Tests a request. Traces only completed requests.*
- void prt_request_cancel (prt_request_t *request)

  *Cancels a request. Cancels them MPI request, releases the packet, frees the request object.*

### 6.29.1 Detailed Description

PRT communication request.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_request.h.

### 6.29.2 Function Documentation

#### 6.29.2.1 void prt_request_cancel ( prt_request_t * *request* )

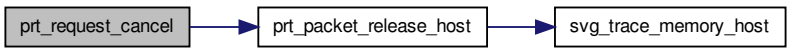
Cancels a request. Cancels them MPI request, releases the packet, frees the request object.

**Parameters**

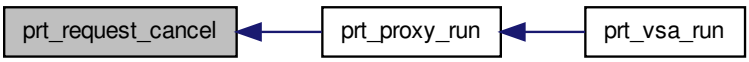
| | |
|---|---|
| *request* | – The request to cancel. |

Definition at line 126 of file prt_request.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.29.2.2 void prt_request_delete ( prt_request_t * *request* )

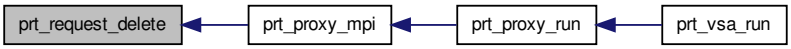
Destroys a request.

**Parameters**

| | |
|---|---|
| *request* | – The request to destroy. |

Definition at line 43 of file prt_request.c.

Here is the caller graph for this function:




#### 6.29.2.3 prt_request_t* prt_request_new ( prt_packet_t * *packet*, size_t *size*, int *peer*, int *tag* )

Creates a new request.

**Parameters**

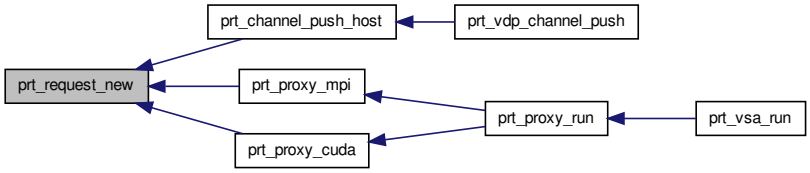
| | |
|---|---|
| *packet* | – The packet to create the request for. |
| *count* | – The number of data items. |
| *datatype* | – The type of data items. |
| *peer* | – The peer communicating node. |
| *tag* | – The MPI tag of the message. |

**Returns**

A new request.

Definition at line 25 of file prt_request.c.

Here is the caller graph for this function:




#### 6.29.2.4 void prt_request_recv ( prt_request_t * *request* )

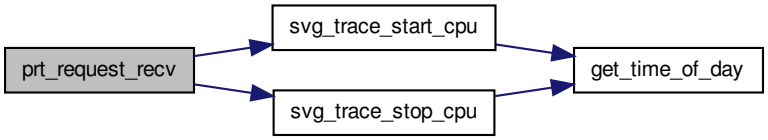
Posts a receive request.

**Parameters**

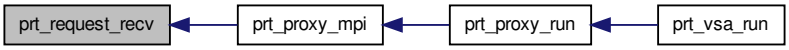
| | |
|---|---|
| *request* | – The receive request to post. |

Definition at line 77 of file prt_request.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.29.2.5 void prt_request_send ( prt_request_t * *request* )

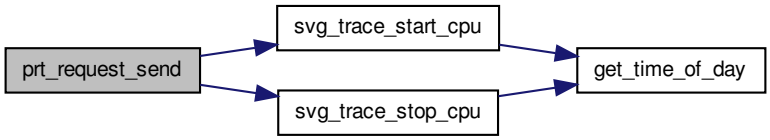
Posts a send request.

**Parameters**

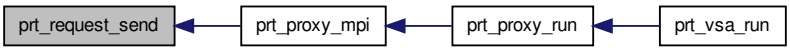
| | |
|---|---|
| *request* | – The send request to post. |

Definition at line 54 of file prt_request.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.29.2.6 int prt_request_test ( prt_request_t * *request* )

Tests a request. Traces only completed requests.

**Parameters**

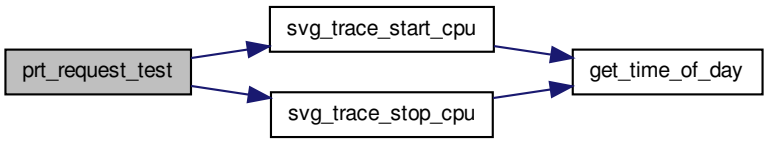
| | |
|---|---|
| *request* | – The request to test. |

**Return values**

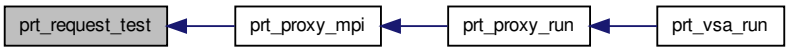
| | |
|---|---|
| *1* | if operation completed. |
| *0* | if operation is pending. |

Definition at line 104 of file prt_request.c.

Here is the call graph for this function:




Here is the caller graph for this function:

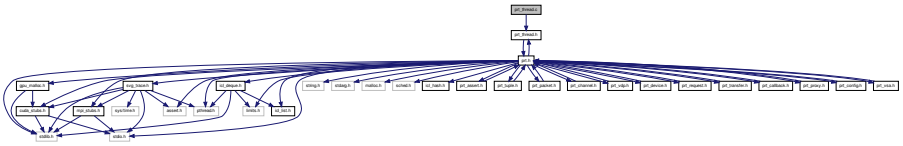



## 6.30 prt_thread.c File Reference

PRT thread.

```
#include "prt_thread.h"
```

Include dependency graph for prt_thread.c:



## Functions

- prt_thread_t * prt_thread_new (int rank, int core, int agent_rank)

  *Creates a new thread object.*

- void prt_thread_delete (prt_thread_t *thread)

  *Destroys a thread.*

- void * prt_thread_run (void *thrd)

  *Implements the thread's processing cycle. If set, calls the thread warmup function. Barriers all threads. If the communication proxy is active, it participates in the barrier. Cycles through VDPs. Fires the ones that are ready. Removes the ones which burned out. Quits when the list of VDPs becomes empty. Saves the execution time.*

### 6.30.1 Detailed Description

PRT thread.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_thread.c.

### 6.30.2 Function Documentation

#### 6.30.2.1 void prt_thread_delete ( prt_thread_t * *thread* )

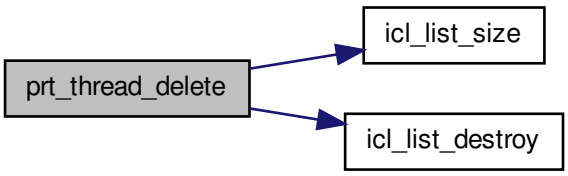
Destroys a thread.

**Parameters**

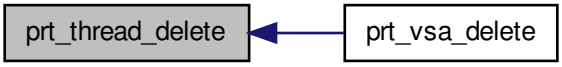
| | |
|---|---|
| *thread* | – The thread to destroy. |

Definition at line 49 of file prt_thread.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.30.2.2 prt_thread_t* prt_thread_new ( int *rank,* int *core,* int *agent_rank* )

Creates a new thread object.

**Parameters**

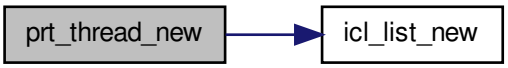
| | |
|---|---|
| *rank* | – The local rank of the thread. |
| *core* | – The global rank of the thread. |
| *agent_rank* | - The rank of the communication agent. |

**Returns**

A new thread object.

Definition at line 23 of file prt_thread.c.

Here is the call graph for this function:




Here is the caller graph for this function:


#### 6.30.2.3 void* prt_thread_run ( void * *thrd* )

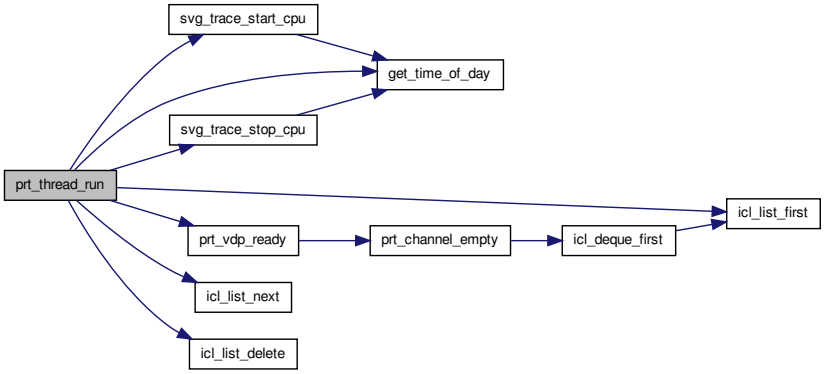
Implements the thread's processing cycle. If set, calls the thread warmup function. Barriers all threads. If the communication proxy is active, it participates in the barrier. Cycles through VDPs. Fires the ones that are ready. Removes the ones which burned out. Quits when the list of VDPs becomes empty. Saves the execution time.

**Parameters**

| | |
|---|---|
| *thrd* | – The pointer to the thread object. |

Definition at line 75 of file prt_thread.c.

Here is the call graph for this function:



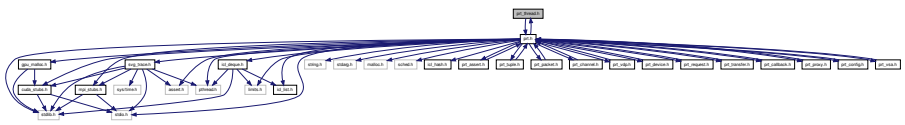

Here is the caller graph for this function:


## 6.31 prt_thread.h File Reference

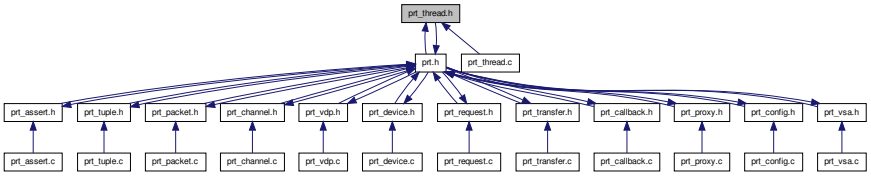
PRT thread.

```
#include "prt.h"
```

Include dependency graph for prt_thread.h:



This graph shows which files directly or indirectly include this file:




## Data Structures

- struct prt_thread_s

  *VSA's worker thread. Represents a single CPU core or a collection of cores.*

## Typedefs

- typedef struct prt_thread_s prt_thread_t

  *VSA's worker thread. Represents a single CPU core or a collection of cores.*

## Functions

- prt_thread_t * prt_thread_new (int rank, int core, int agent_rank)

  *Creates a new thread object.*
- void prt_thread_delete (prt_thread_t *thread)

  *Destroys a thread.*
- void * prt_thread_run (void *thrd)

  *Implements the thread's processing cycle. If set, calls the thread warmup function. Barriers all threads. If the communication proxy is active, it participates in the barrier. Cycles through VDPs. Fires the ones that are ready. Removes the ones which burned out. Quits when the list of VDPs becomes empty. Saves the execution time.*

### 6.31.1 Detailed Description

PRT thread.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_thread.h.

### 6.31.2 Typedef Documentation

#### 6.31.2.1 typedef struct prt_thread_s prt_thread_t

VSA's worker thread. Represents a single CPU core or a collection of cores.

"finished" is a one-directional synchronization variable. Therefore declered volatile, but no need for atomic access.

### 6.31.3 Function Documentation

#### 6.31.3.1 void prt_thread_delete ( prt_thread_t * thread )

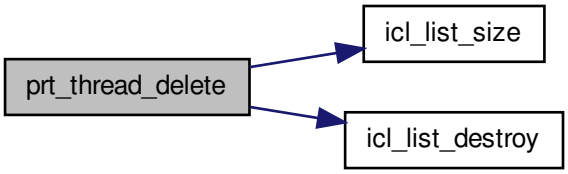
Destroys a thread.

**Parameters**

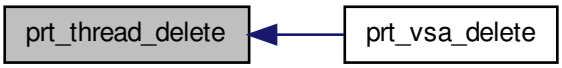
| | |
|---|---|
| *thread* | – The thread to destroy. |

Definition at line 49 of file prt_thread.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.31.3.2 prt_thread_t* prt_thread_new ( int *rank*, int *core*, int *agent_rank* )

Creates a new thread object.

**Parameters**

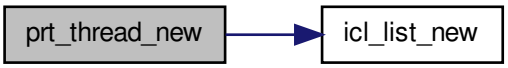
| | |
|---|---|
| *rank* | – The local rank of the thread. |
| *core* | – The global rank of the thread. |
| *agent_rank* | - The rank of the communication agent. |

**Returns**

A new thread object.

Definition at line 23 of file prt_thread.c.

Here is the call graph for this function:




Here is the caller graph for this function:


#### 6.31.3.3 void* prt_thread_run ( void * *thrd* )

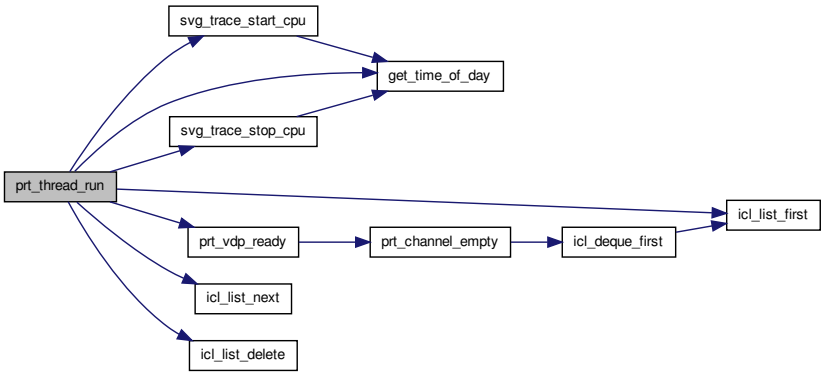
Implements the thread's processing cycle. If set, calls the thread warmup function. Barriers all threads. If the communication proxy is active, it participates in the barrier. Cycles through VDPs. Fires the ones that are ready. Removes the ones which burned out. Quits when the list of VDPs becomes empty. Saves the execution time.

**Parameters**

| | |
|---|---|
| *thrd* | – The pointer to the thread object. |

Definition at line 75 of file prt_thread.c.

Here is the call graph for this function:



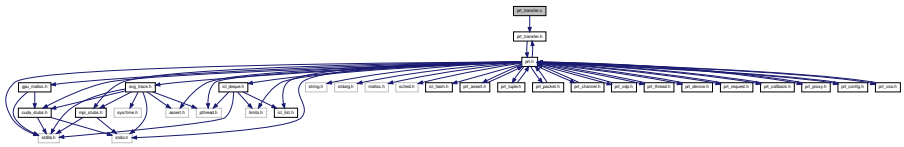

Here is the caller graph for this function:


## 6.32 prt_transfer.c File Reference

PRT local transfer.

```
#include "prt_transfer.h"
```

Include dependency graph for prt_transfer.c:



### Functions

- prt_transfer_t * prt_transfer_new (struct prt_packet_s *packet, struct prt_channel_s *channel, enum prt_direction_e direction, int agent)

  *Creates a new local transfer object.*

- void prt_transfer_delete (prt_transfer_t *transfer)

  *Destroys a local transfer object.*

### 6.32.1 Detailed Description

PRT local transfer.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_transfer.c.

### 6.32.2 Function Documentation

#### 6.32.2.1 void prt_transfer_delete ( prt_transfer_t * *transfer* )

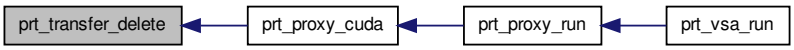
Destroys a local transfer object.

**Parameters**

| | |
|---|---|
| – | The local transfer object to destroy. |

Definition at line 46 of file prt_transfer.c.

Here is the caller graph for this function:




#### 6.32.2.2 prt_transfer_t* prt_transfer_new ( struct prt_packet_s * *packet*, struct prt_channel_s * *channel*, enum prt_direction_e *direction*, int *agent* )

Creates a new local transfer object.

**Parameters**

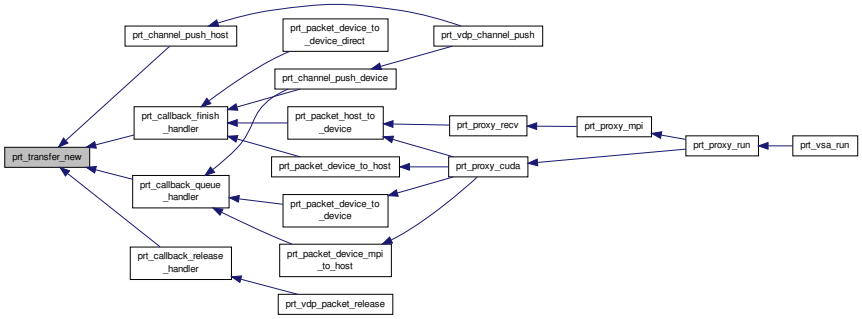
| | |
|---|---|
| *packet* | – The packet to transfer. |
| *channel* | – The channel to push to. |
| *direction* | – The direction of the transfer. |
| *agent* | – The number of the communication agent to use. |

### Returns

A new local transfer object.

Definition at line 24 of file prt_transfer.c.

Here is the caller graph for this function:

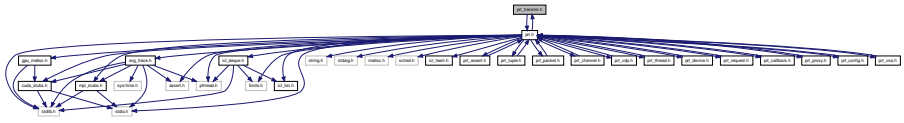



## 6.33 prt_transfer.h File Reference

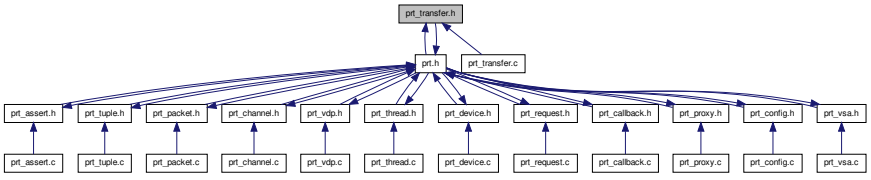
PRT local transfer.

```
#include "prt.h"
```

Include dependency graph for prt_transfer.h:



This graph shows which files directly or indirectly include this file:




### Data Structures

- struct prt_transfer_s

  *Local transfer object.*

### Typedefs

- typedef struct prt_transfer_s prt_transfer_t

  *Local transfer object.*

### Functions

- prt_transfer_t * **prt_transfer_new** (struct prt_packet_s *packet, struct prt_channel_s *channel, prt_direction_t direction, int agent)
- void prt_transfer_delete (prt_transfer_t *transfer)

  *Destroys a local transfer object.*

#### 6.33.1 Detailed Description

PRT local transfer.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_transfer.h.

#### 6.33.2 Function Documentation

##### 6.33.2.1 void prt_transfer_delete ( prt_transfer_t * *transfer* )

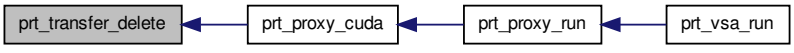
Destroys a local transfer object.

**Parameters**

| | |
|---|---|
| *–* | The local transfer object to destroy. |

Definition at line 46 of file prt_transfer.c.

Here is the caller graph for this function:

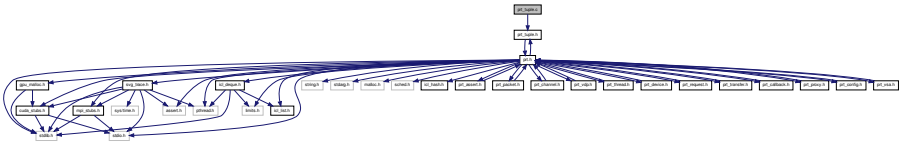



## 6.34 prt_tuple.c File Reference

PRT tuple.

```
#include "prt_tuple.h"
```

Include dependency graph for prt_tuple.c:



## Functions

- int * prt_tuple_new (int len,...)

  *Creates a new tuple. Allocates memory for the tuple plus the termination symbol (INT_MAX). Fills out the tuple with the integers on the list. There is also a set of macros, prt_tuple_new1/2/3/4/5/6, where the length of the tuple is indicated by the number in the name. Because this is such a tiny function, and is mostly intended to be accessed through macros, skipping error checks for input parameters.*

- int prt_tuple_len (int *tuple)

  *Returns the length of a tuple.*

- int * prt_tuple_cat (int *first_tuple,...)

  *Concatenates a list of tuples. Concatenates a variable-length, NULL-terminated, list of tuples.*

- void prt_tuple_delete (int *tuple)

  *Destroys a tuple.*

- int * prt_tuple_copy (int *in_tuple)

  *Copies a tuple.*

- int prt_tuple_compare (void *tuple_a, void *tuple_b)

  *Compares two tuples.*

- int prt_tuple_equal (void *tuple_a, void *tuple_b)

  *Checks if two tuples are identical. Check if tuples are identical in length and content.*

- void prt_tuple_print (int *tuple)

  *Prints a tuple.*

- unsigned int prt_tuple_hash (void *tuple)

  *Hashes a tuple. This function is required by the VSA's tuples hash table. It computes the lenght in characters and calls a string hashing function.*

### 6.34.1 Detailed Description

PRT tuple.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_tuple.c.

### 6.34.2 Function Documentation

#### 6.34.2.1 int* prt_tuple_cat ( int * *first_tuple, ...* )

Concatenates a list of tuples. Concatenates a variable-length, NULL-terminated, list of tuples.

**Parameters**

| | |
|---|---|
| *first_tuple* | – The first tuple in the sequence. |
| ... | – A list of more tuples. |

**Returns**

The aggregate tuple.

Definition at line 70 of file prt_tuple.c.

Here is the call graph for this function:


#### 6.34.2.2 int prt_tuple_compare ( void * *tuple_a*, void * *tuple_b* )

Compares two tuples.

**Parameters**

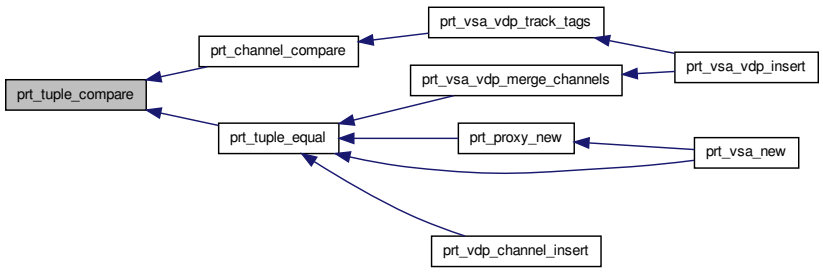
| | |
|---|---|
| *tuple_a* | – The first tuple. |
| *tuple_b* | – The second tuple. |

**Return values**

| | |
|---|---|
| *-1* | if tuple_a is less than tuple_b. |
| *0* | if tuple_a is equal to tuple_b. |
| *1* | if tuple_a is greater than tuple_b. |

Definition at line 141 of file prt_tuple.c.

Here is the caller graph for this function:



#### 6.34.2.3 int* prt_tuple_copy ( int * *in_tuple* )

Copies a tuple.

**Parameters**

| | |
|---|---|
| *in_tuple* | – The tuple to copy. |

**Returns**

A new copy of the tuple.

Definition at line 115 of file prt_tuple.c.

#### 6.34.2.4 void prt_tuple_delete ( int * *tuple* )

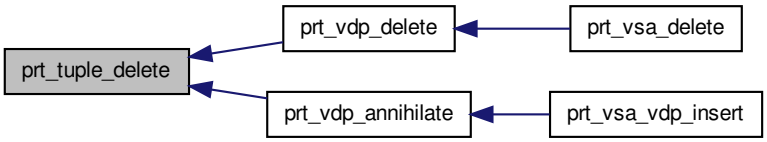
Destroys a tuple.

**Parameters**

| | |
|---|---|
| *tuple* | – The tuple to destroy. |

Definition at line 101 of file prt_tuple.c.

Here is the caller graph for this function:




#### 6.34.2.5 int prt_tuple_equal ( void * *tuple_a*, void * *tuple_b* )

Checks if two tuples are identical. Check if tuples are identical in length and content.

**Parameters**

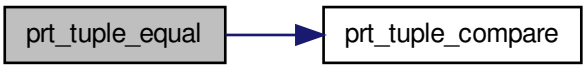
| | |
|---|---|
| *tuple_a* | – The first tuple. |
| *tuple_b* | – The second tuple. |

**Return values**

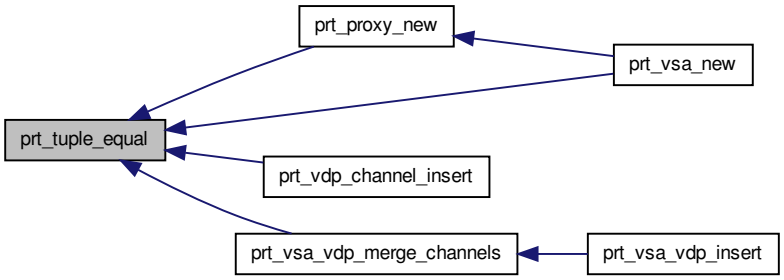
| | |
|---|---|
| *0* | if tuples are different. |
| *1* | if tuples are identical. |

Definition at line 167 of file prt_tuple.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.34.2.6 unsigned int prt_tuple_hash ( void * *tuple* )

Hashes a tuple. This function is required by the VSA's tuples hash table. It computes the lenght in characters and calls a string hashing function.

**Parameters**

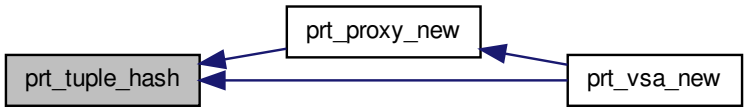
| | |
|---|---|
| *tuple* | – The tuple to hash. |

**Returns**

hash

Definition at line 194 of file prt_tuple.c.

Here is the caller graph for this function:




#### 6.34.2.7 int prt_tuple_len ( int * *tuple* )

Returns the length of a tuple.

**Parameters**

| | |
|---|---|
| *tuple* | – The tuple to return the length of. |

**Returns**

The length of the tuple without the terminating symbol.

Definition at line 53 of file prt_tuple.c.

Here is the caller graph for this function:


#### 6.34.2.8 void prt_tuple_print ( int * *tuple* )

Prints a tuple.

**Parameters**

| | |
|---|---|
| *tuple* | – The tuple to print. |

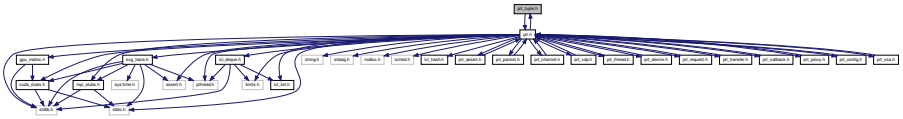
Definition at line 178 of file prt_tuple.c.

## 6.35 prt_tuple.h File Reference

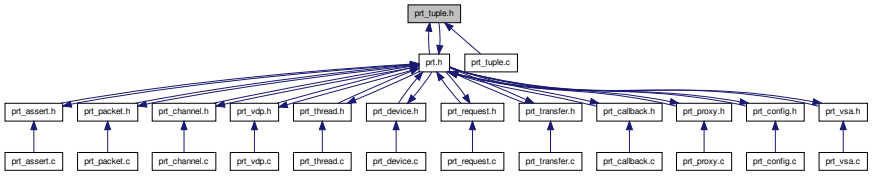
PRT tuple.

```
#include "prt.h"
```

Include dependency graph for prt_tuple.h:



This graph shows which files directly or indirectly include this file:




### Macros

- #define **prt_tuple_new1**(a) prt_tuple_new(1,a)
- #define **prt_tuple_new2**(a, b) prt_tuple_new(2,a,b)
- #define **prt_tuple_new3**(a, b, c) prt_tuple_new(3,a,b,c)
- #define **prt_tuple_new4**(a, b, c, d) prt_tuple_new(4,a,b,c,d)
- #define **prt_tuple_new5**(a, b, c, d, e) prt_tuple_new(5,a,b,c,d,e)
- #define **prt_tuple_new6**(a, b, c, d, e, f) prt_tuple_new(6,a,b,c,d,e,f)

### Functions

- int * prt_tuple_new (int len,...)

  *Creates a new tuple. Allocates memory for the tuple plus the termination symbol (INT_MAX). Fills out the tuple with the integers on the list. There is also a set of macros, prt_tuple_new1/2/3/4/5/6, where the length of the tuple is indicated by the number in the name. Because this is such a tiny function, and is mostly intended to be accessed through macros, skipping error checks for input parameters.*
- int prt_tuple_len (int *tuple)

  *Returns the length of a tuple.*
- int * prt_tuple_cat (int *first_tuple,...)

  *Concatenates a list of tuples. Concatenates a variable-length, NULL-terminated, list of tuples.*

- void prt_tuple_delete (int *tuple)

  *Destroys a tuple.*

- int * prt_tuple_copy (int *in_tuple)

  *Copies a tuple.*

- int prt_tuple_compare (void *tuple_a, void *tuple_b)

  *Compares two tuples.*

- int prt_tuple_equal (void *tuple_a, void *tuple_b)

  *Checks if two tuples are identical. Check if tuples are identical in length and content.*

- void prt_tuple_print (int *tuple)

  *Prints a tuple.*

- unsigned int prt_tuple_hash (void *tuple)

  *Hashes a tuple. This function is required by the VSA's tuples hash table. It computes the lenght in characters and calls a string hashing function.*

### 6.35.1 Detailed Description

PRT tuple.

**Author**

Jakub Kurzak

Tuples uniquely identify VDPs in a VSA. Tuple is an array of integers terminated with INT_MAX. For all practical purposes a tuple behaves like a string.

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_tuple.h.

### 6.35.2 Function Documentation

#### 6.35.2.1 int* prt_tuple_cat ( int * *first_tuple*, ... )

Concatenates a list of tuples. Concatenates a variable-length, NULL-terminated, list of tuples.

**Parameters**

| | |
|---|---|
| *first_tuple* | – The first tuple in the sequence. |
| ... | – A list of more tuples. |

**Returns**

The aggregate tuple.

Definition at line 70 of file prt_tuple.c.

Here is the call graph for this function:


#### 6.35.2.2 int prt_tuple_compare ( void * *tuple_a*, void * *tuple_b* )

Compares two tuples.

**Parameters**

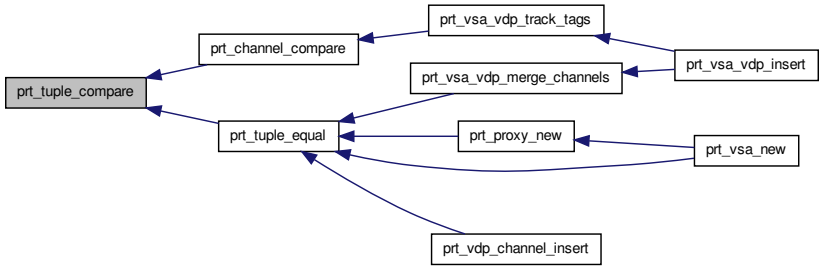
| | |
|---|---|
| *tuple_a* | – The first tuple. |
| *tuple_b* | – The second tuple. |

**Return values**

| | |
|---|---|
| *-1* | if tuple_a is less than tuple_b. |
| *0* | if tuple_a is equal to tuple_b. |
| *1* | if tuple_a is greater than tuple_b. |

Definition at line 141 of file prt_tuple.c.

Here is the caller graph for this function:




#### 6.35.2.3 int* prt_tuple_copy ( int * *in_tuple* )

Copies a tuple.

**Parameters**

| | |
|---|---|
| *in_tuple* | – The tuple to copy. |

**Returns**

A new copy of the tuple.

Definition at line 115 of file prt_tuple.c.

#### 6.35.2.4 void prt_tuple_delete ( int * *tuple* )

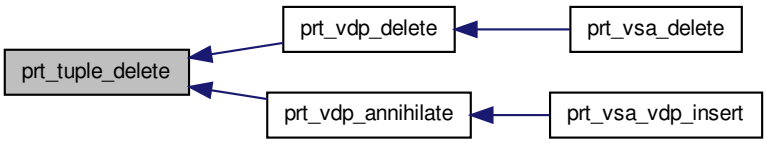
Destroys a tuple.

**Parameters**

| | |
|---|---|
| *tuple* | – The tuple to destroy. |

Definition at line 101 of file prt_tuple.c.

Here is the caller graph for this function:




#### 6.35.2.5 int prt_tuple_equal ( void * *tuple_a*, void * *tuple_b* )

Checks if two tuples are identical. Check if tuples are identical in length and content.

**Parameters**

| | |
|---|---|
| *tuple_a* | – The first tuple. |
| *tuple_b* | – The second tuple. |

**Return values**

| | |
|---|---|
| *0* | if tuples are different. |
| *1* | if tuples are identical. |

Definition at line 167 of file prt_tuple.c.

#### 6.35.2.6 unsigned int prt_tuple_hash ( void * *tuple* )

Hashes a tuple. This function is required by the VSA's tuples hash table. It computes the lenght in characters and calls a string hashing function.

**Parameters**

| | |
|---|---|
| *tuple* | – The tuple to hash. |

**Returns**

hash

Definition at line 194 of file prt_tuple.c.

#### 6.35.2.7 int prt_tuple_len ( int * *tuple* )

Returns the length of a tuple.

**Parameters**

| | |
|---|---|
| *tuple* | – The tuple to return the length of. |

**Returns**

The length of the tuple without the terminating symbol.

Definition at line 53 of file prt_tuple.c.

Here is the caller graph for this function:


#### 6.35.2.8 void prt_tuple_print ( int * *tuple* )

Prints a tuple.

**Parameters**

| | |
|---|---|
| *tuple* | – The tuple to print. |

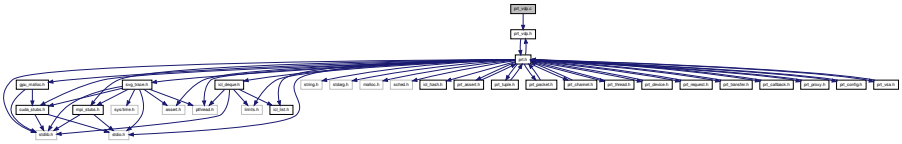
Definition at line 178 of file prt_tuple.c.

## 6.36 prt_vdp.c File Reference

Virtual Data Processor.

```
#include "prt_vdp.h"
```

Include dependency graph for prt_vdp.c:



## Functions

- prt_vdp_t * prt_vdp_new (int *tuple, int counter, prt_vdp_function_t function, size_t local_store_size, int num_inputs, int num_outputs, int color)

  *Creates a new VDP.*

- void prt_vdp_delete (prt_vdp_t *vdp)

  *Destroys a VDP. Used for destruction of local VDPs. Destroys all input channels. Destroys all dangling output channels. Local output channels are destroyed as input channels of other local VDPs.*

- void prt_vdp_annihilate (prt_vdp_t *vdp)

  *Annihilates a VDP. Used for complete annihilation of VDPs that don't belong in the node. Destroys all input channels. Destroys all output channels.*

- void prt_vdp_channel_insert (prt_vdp_t *vdp, prt_channel_t *channel, prt_channel_direction_t direction, int slot)

  *Inserts a new channel into a VDP.*

- prt_packet_t * prt_vdp_packet_new (prt_vdp_t *vdp, size_t size, void *data)

  *Creates a new packet. Allocates the size amount of data if a NULL pointer is passed. The size cannot be larger than INT_MAX, because all data typea are packed inside messages of type MPI_BYTE. Calls host constructor or device constructor depending on the VDP's location.*

- prt_packet_t * prt_vdp_packet_new_host_to_device (prt_vdp_t *vdp, size_t size, void *data)

  *Creates a new packet and queues a host-to-device transfer. The size cannot be larger than INT_MAX, because all data typea are packed inside messages of type MPI_BYTE. Expects a non-NULL pointer to the data in host memory. Right now, device memory is allocated immediately. Potentially, it could also be done in the VDP's stream.*

- void prt_vdp_packet_release (prt_vdp_t *vdp, prt_packet_t *packet)

  *Releases a packet. Decrements the number of active references. Destroys the packet when the number of references goes down to zero. For device packets, puts a callback in the VDP's stream.*

- void prt_vdp_channel_push (prt_vdp_t *vdp, int channel_num, prt_packet_t *packet)

  *Pushes a packet in a channel.*

- prt_packet_t * prt_vdp_channel_pop (prt_vdp_t *vdp, int channel_num)

  *Fetches a packet from a channel.*

- void prt_vdp_channel_off (prt_vdp_t *vdp, int channel_num)

  *Deactivates a channel.*

- void prt_vdp_channel_on (prt_vdp_t *vdp, int channel_num)

  *Activates a channel.*

- int prt_vdp_ready (prt_vdp_t *vdp)

  *Checks if a VDP is ready to fire. Only checks established channels. (NULL channels don't prevent firing.) Inactive channels don't precent firing.*

### 6.36.1 Detailed Description

Virtual Data Processor.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_vdp.c.

### 6.36.2 Function Documentation

#### 6.36.2.1 void prt_vdp_annihilate ( prt_vdp_t * *vdp* )

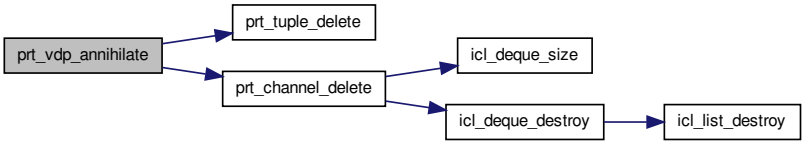
Annihilates a VDP. Used for complete annihilation of VDPs that don't belong in the node. Destroys all input channels. Destroys all output channels.

**Parameters**

| | |
|---|---|
| *vdp* | – The VDP to annihilate. |

Definition at line 152 of file prt_vdp.c.

Here is the call graph for this function:




Here is the caller graph for this function:


#### 6.36.2.2 void prt_vdp_delete ( prt_vdp_t * *vdp* )

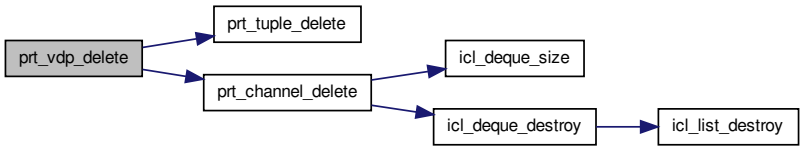
Destroys a VDP. Used for destruction of local VDPs. Destroys all input channels. Destroys all dangling output channels. Local output channels are destroyed as input channels of other local VDPs.

**Parameters**

| | |
|---|---|
| *vdp* | – The VDP to destroy. |

Definition at line 87 of file prt_vdp.c.

Here is the call graph for this function:




Here is the caller graph for this function:


#### 6.36.2.3 int prt_vdp_ready ( prt_vdp_t * *vdp* )

Checks if a VDP is ready to fire. Only checks established channels. (NULL channels don't prevent firing.) Inactive channels don't precent firing.

**Parameters**

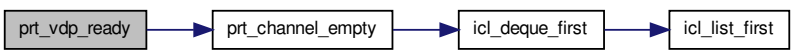
| | |
|---|---|
| *vdp* | – The VDP to check. |

**Return values**

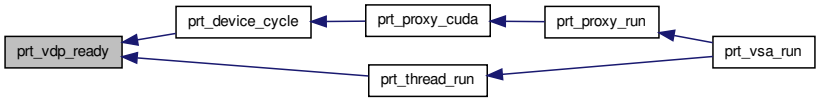
| | |
|---|---|
| *1* | if ready. |
| *0* | if not ready. |

Definition at line 456 of file prt_vdp.c.

Here is the call graph for this function:



Here is the caller graph for this function:

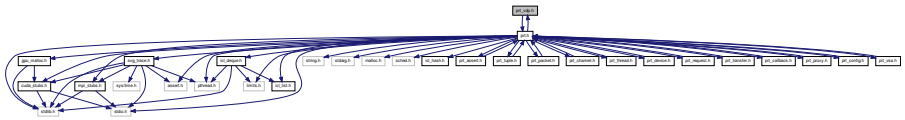



## 6.37 prt_vdp.h File Reference

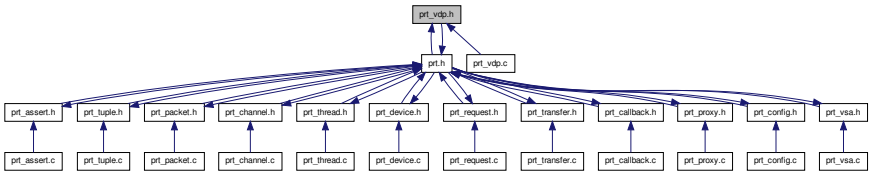
Virtual Data Processor.

```
#include "prt.h"
```

Include dependency graph for prt_vdp.h:



This graph shows which files directly or indirectly include this file:




### Data Structures

- struct prt_vdp_s

  *Virtual Data Processor (VDP). Is uniquely identified by a tuple. Fires for a predefined number of cycles. Has a fixed number of input and output channels. Has a persistent local store. Has access to read-only global store.*

### Typedefs

- typedef void(∗ prt_vdp_function_t )(struct prt_vdp_s ∗)

  *VDP's function pointer. Defines the type of the pointer to the VDP's function.*

- typedef struct prt_vdp_s prt_vdp_t

  *Virtual Data Processor (VDP). Is uniquely identified by a tuple. Fires for a predefined number of cycles. Has a fixed number of input and output channels. Has a persistent local store. Has access to read-only global store.*

## Functions

- prt_vdp_t * prt_vdp_new (int *tuple, int counter, prt_vdp_function_t function, size_t local_store_size, int num_inputs, int num_outputs, int color)

  *Creates a new VDP.*

- void prt_vdp_delete (prt_vdp_t *vdp)

  *Destroys a VDP. Used for destruction of local VDPs. Destroys all input channels. Destroys all dangling output channels. Local output channels are destroyed as input channels of other local VDPs.*

- void prt_vdp_annihilate (prt_vdp_t *vdp)

  *Annihilates a VDP. Used for complete annihilation of VDPs that don't belong in the node. Destroys all input channels. Destroys all output channels.*

- void **prt_vdp_channel_insert** (prt_vdp_t *vdp, struct prt_channel_s *channel, enum prt_channel_direction_e direction, int slot)

- struct prt_packet_s * prt_vdp_packet_new (prt_vdp_t *vdp, size_t size, void *data)

  *Creates a new packet. Allocates the size amount of data if a NULL pointer is passed. The size cannot be larger than INT_MAX, because all data typea are packed inside messages of type MPI_BYTE. Calls host constructor or device constructor depending on the VDP's location.*

- struct prt_packet_s * prt_vdp_packet_new_host_to_device (prt_vdp_t *vdp, size_t size, void *data)

  *Creates a new packet and queues a host-to-device transfer. The size cannot be larger than INT_MAX, because all data typea are packed inside messages of type MPI_BYTE. Expects a non-NULL pointer to the data in host memory. Right now, device memory is allocated immediately. Potentially, it could also be done in the VDP's stream.*

- void prt_vdp_packet_release (prt_vdp_t *vdp, struct prt_packet_s *packet)

  *Releases a packet. Decrements the number of active references. Destroys the packet when the number of references goes down to zero. For device packets, puts a callback in the VDP's stream.*

- void prt_vdp_channel_push (prt_vdp_t *vdp, int channel_num, struct prt_packet_s *packet)

  *Pushes a packet in a channel.*

- struct prt_packet_s * prt_vdp_channel_pop (prt_vdp_t *vdp, int channel_num)

  *Fetches a packet from a channel.*

- void prt_vdp_channel_off (prt_vdp_t *vdp, int channel_num)

  *Deactivates a channel.*

- void prt_vdp_channel_on (prt_vdp_t *vdp, int channel_num)

  *Activates a channel.*

- int prt_vdp_ready (prt_vdp_t *vdp)

  *Checks if a VDP is ready to fire. Only checks established channels. (NULL channels don't prevent firing.) Inactive channels don't precent firing.*

### 6.37.1 Detailed Description

Virtual Data Processor.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_vdp.h.

### 6.37.2 Function Documentation

#### 6.37.2.1 void prt_vdp_annihilate ( prt_vdp_t * *vdp* )

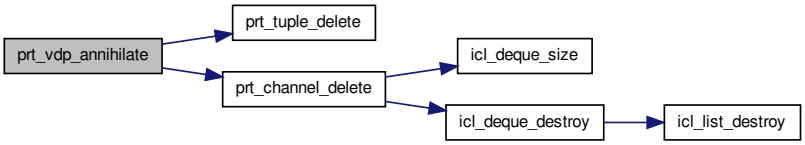
Annihilates a VDP. Used for complete annihilation of VDPs that don't belong in the node. Destroys all input channels. Destroys all output channels.

**Parameters**

| | |
|---|---|
| *vdp* | – The VDP to annihilate. |

Definition at line 152 of file prt_vdp.c.

Here is the call graph for this function:




Here is the caller graph for this function:


#### 6.37.2.2 void prt_vdp_delete ( prt_vdp_t * *vdp* )

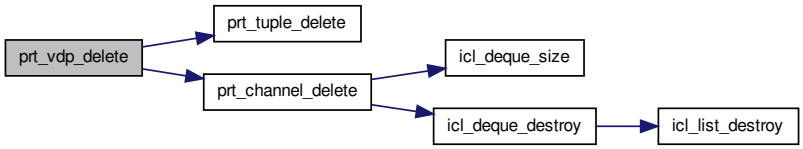
Destroys a VDP. Used for destruction of local VDPs. Destroys all input channels. Destroys all dangling output channels. Local output channels are destroyed as input channels of other local VDPs.

**Parameters**

| | |
|---|---|
| *vdp* | – The VDP to destroy. |

Definition at line 87 of file prt_vdp.c.

Here is the call graph for this function:



Here is the caller graph for this function:


#### 6.37.2.3 int prt_vdp_ready ( prt_vdp_t * vdp )

Checks if a VDP is ready to fire. Only checks established channels. (NULL channels don't prevent firing.) Inactive channels don't precent firing.

**Parameters**

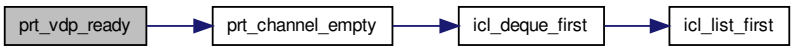
| | |
|---|---|
| *vdp* | – The VDP to check. |

**Return values**

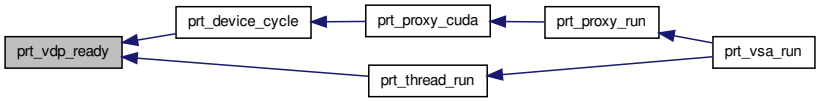
| | |
|---|---|
| *1* | if ready. |
| *0* | if not ready. |

Definition at line 456 of file prt_vdp.c.

Here is the call graph for this function:




Here is the caller graph for this function:

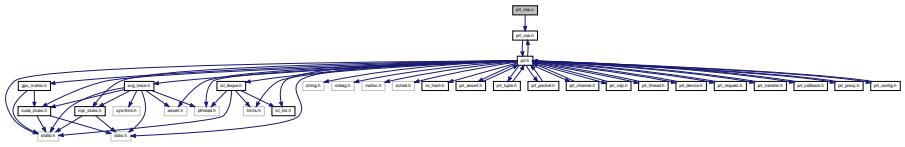



## 6.38 prt_vsa.c File Reference

Virtual Systolic Array.

```
#include "prt_vsa.h"
```

Include dependency graph for prt_vsa.c:



## Functions

- int prt_tuple_equal (void *tuple_a, void *tuple_b)

  *Checks if two tuples are identical. Check if tuples are identical in length and content.*
- unsigned int prt_tuple_hash (void *tuple)

  *Hashes a tuple. This function is required by the VSA's tuples hash table. It computes the lenght in characters and calls a string hashing function.*
- prt_vsa_t * prt_vsa_new (int num_threads, int num_devices, void *global_store, struct prt_mapping_s(*vdp_mapping)(int *, void *, int, int))

  *Creates a new VSA.*
- void prt_vsa_delete (prt_vsa_t *vsa)

  *Destroys a VSA.*
- void prt_vsa_vdp_insert (prt_vsa_t *vsa, prt_vdp_t *vdp)

  *Inserts a VDP in a VSA. Destroys VDPs that do not belong to this node. Puts the VDP in the list of VDPs of the owner thread or device. Connects corresponding input and output channels of intra-node VDPs. Builds the list of channel connections to other nodes. For a device VDP, creates a CUDA stream with the cudaStreamNonBlocking flag. This indicates no synchronization with the default stream (stream 0). Stream 0 is not used anywere in PRT.*
- void prt_vsa_vdp_merge_channels (prt_vsa_t *vsa, prt_vdp_t *vdp)

  *Connects corresponding input and output channels of intra-node VDPs. An input channel always overrides an output channel. This way the on/off switch of the input channel is preserved.*
- void prt_vsa_vdp_track_tags (prt_vsa_t *vsa, prt_vdp_t *vdp)

  *Builds the list of channel connections to other nodes.*
- void prt_vsa_channel_tags (prt_vsa_t *vsa)

  *Assigns channel tags. Builds the node-tag lookup. Destroys channel lists.*
- void prt_vsa_channel_streams (prt_vsa_t *vsa)

  *Creates channel streams.*
- double prt_vsa_run (prt_vsa_t *vsa)

  *Implements the VSA's production cycle. Launches worker threads. Sends the master thread in the proxy production cycle. Joins the worker threads.*
- void prt_vsa_config_set (prt_vsa_t *vsa, prt_config_param_t param, prt_config_value_t value)

  *Sets a VSA configuration parameter.*
- void prt_vsa_thread_warmup_func_set (prt_vsa_t *vsa, void(*func)())

  *Sets a thread warmup function. If set, the thread warmup function is called by each thread right after launching and before threads are barriered and the timer is started. Allows for excluding the time for initialization procedures of libraries, such as loading of dynamic libraries, internal memory allocations, possibly time consuming pinned memory allocations, etc. A NULL function pointer can be passed to remove the thread warmup function.*
- void prt_vsa_device_warmup_func_set (prt_vsa_t *vsa, void(*func)())

  *Sets a device warmup function. If set, the device warmup function is called by each device right after launching and before devices are barriered and the timer is started. Allows for excluding the time for initialization procedures of libraries, such as loading of dynamic libraries, internal memory allocations, possibly time consuming pinned memory allocations, etc. A NULL function pointer can be passed to remove the device warmup function.*

- void prt_vsa_devices_warmup (prt_vsa_t *vsa)

  *Calls the warmup function for all devices and synchronizes.*

### 6.38.1 Detailed Description

Virtual Systolic Array.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_vsa.c.

### 6.38.2 Function Documentation

#### 6.38.2.1 int prt_tuple_equal ( void * *tuple_a*, void * *tuple_b* )

Checks if two tuples are identical. Check if tuples are identical in length and content.

**Parameters**

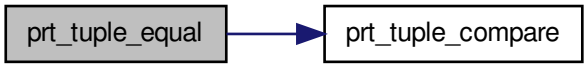
| | |
|---|---|
| *tuple_a* | – The first tuple. |
| *tuple_b* | – The second tuple. |

**Return values**

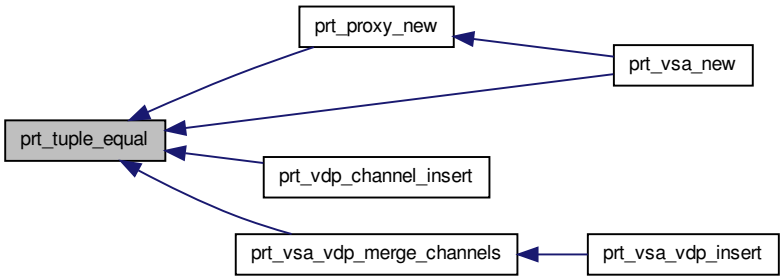
| | |
|---|---|
| *0* | if tuples are different. |
| *1* | if tuples are identical. |

Definition at line 167 of file prt_tuple.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.38.2.2 unsigned int prt_tuple_hash ( void * *tuple* )

Hashes a tuple. This function is required by the VSA's tuples hash table. It computes the lenght in characters and calls a string hashing function.

**Parameters**

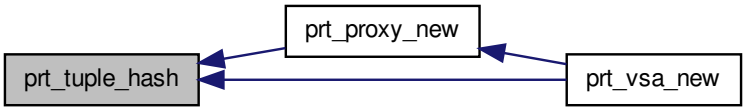
| | |
|---|---|
| *tuple* | – The tuple to hash. |

**Returns**

hash

Definition at line 194 of file prt_tuple.c.

Here is the caller graph for this function:




#### 6.38.2.3 void prt_vsa_channel_streams ( prt_vsa_t * *vsa* )

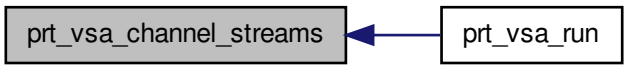
Creates channel streams.

**Parameters**

| | |
|---|---|
| *vsa* | – The VSA to create streams for. |

Definition at line 469 of file prt_vsa.c.

Here is the caller graph for this function:




#### 6.38.2.4 void prt_vsa_channel_tags ( prt_vsa_t * *vsa* )

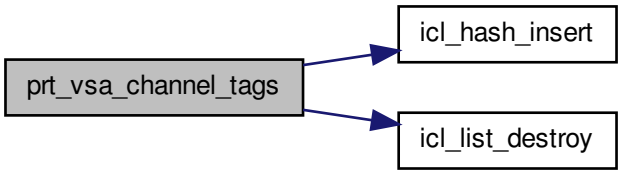
Assigns channel tags. Builds the node-tag lookup. Destroys channel lists.

**Parameters**

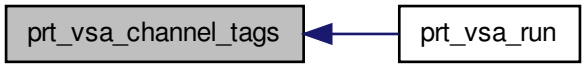
| | |
|---|---|
| *vsa* | – The VSA to find the tags for. |

Definition at line 432 of file prt_vsa.c.

Here is the call graph for this function:




Here is the caller graph for this function:



#### 6.38.2.5 void prt_vsa_devices_warmup ( prt_vsa_t * *vsa* )

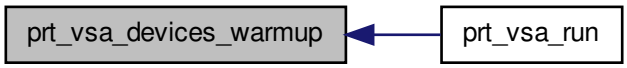
Calls the warmup function for all devices and synchronizes.

**Parameters**

| | |
|---|---|
| *vsa* | – The VSA to warmup devices for. |

Definition at line 694 of file prt_vsa.c.

Here is the caller graph for this function:




#### 6.38.2.6 void prt_vsa_vdp_merge_channels ( prt_vsa_t * *vsa*, prt_vdp_t * *vdp* )

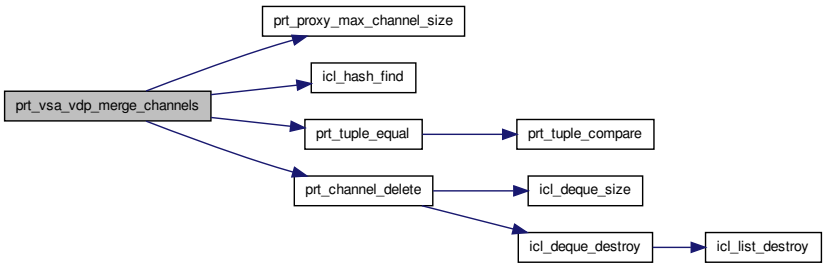
Connects corresponding input and output channels of intra-node VDPs. An input channel always overrides an output channel. This way the on/off switch of the input channel is preserved.

**Parameters**

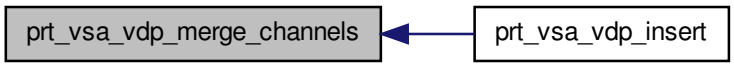
| | |
|---|---|
| *vsa* | – The VSA to merge channels within. |
| *vdp* | – The VDP to merge channels for. |

Definition at line 290 of file prt_vsa.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.38.2.7 void prt_vsa_vdp_track_tags ( prt_vsa_t * *vsa*, prt_vdp_t * *vdp* )

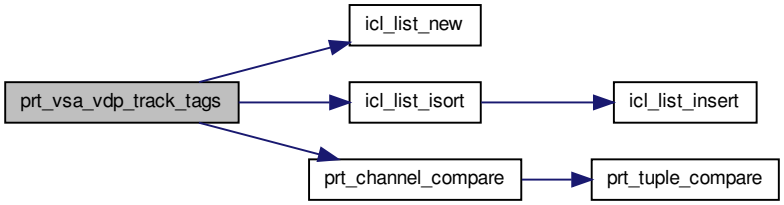
Builds the list of channel connections to other nodes.

**Parameters**

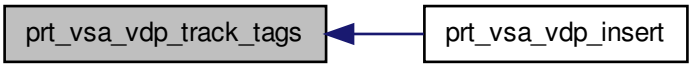
| | |
|---|---|
| *vsa* | – The VSA to track the tags within. |
| *vdp* | – The VDP to track the tags for. |

Definition at line 351 of file prt_vsa.c.

Here is the call graph for this function:




Here is the caller graph for this function:

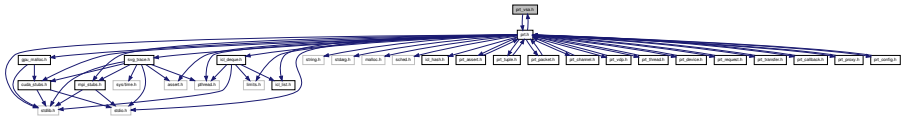


## 6.39 prt_vsa.h File Reference

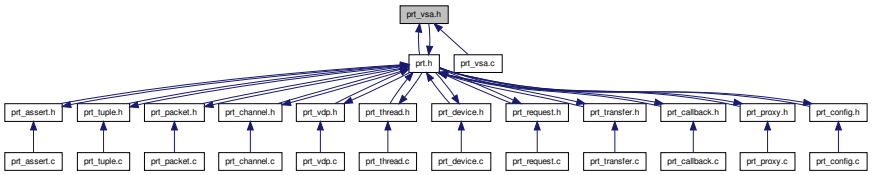
Virtual Systolic Array.

```
#include "prt.h"
```

Include dependency graph for prt_vsa.h:



This graph shows which files directly or indirectly include this file:




### Data Structures

- struct prt_vsa_s

  *Virtual Systolic Array (VSA) VSA contains global informationa about the system, a local communication proxy, an array of local worker threads, and an array of local accelerator devices.*

### Macros

- #define PRT_VSA_MAX_VDPS_PER_NODE 10003

  *The maximum number of VDPs per node. The size of the VSA's hash table of VDPs. Should be a prime number.*

- #define PRT_VSA_GPU_ALLOC_UNIT_SIZE 131072

  *The size of segments allocated by the GPU memory allocator. Setting the unit size to 128 KB, which is 64 x 256 x sizeof(double).*

### Typedefs

- typedef struct prt_mapping_s(∗ prt_vdp_mapping_t )(int ∗, void ∗, int, int)

  *The function pointer for the VDP mapping function.*

- typedef struct prt_vsa_s prt_vsa_t

  *Virtual Systolic Array (VSA) VSA contains global informationa about the system, a local communication proxy, an array of local worker threads, and an array of local accelerator devices.*

## Functions

- prt_vsa_t * prt_vsa_new (int num_threads, int num_devices, void *global_store, struct prt_mapping_s(*vdp_mapping)(int *, void *, int, int))

  *Creates a new VSA.*

- void prt_vsa_delete (prt_vsa_t *vsa)

  *Destroys a VSA.*

- void prt_vsa_vdp_insert (prt_vsa_t *vsa, struct prt_vdp_s *vdp)

  *Inserts a VDP in a VSA. Destroys VDPs that do not belong to this node. Puts the VDP in the list of VDPs of the owner thread or device. Connects corresponding input and output channels of intra-node VDPs. Builds the list of channel connections to other nodes. For a device VDP, creates a CUDA stream with the cudaStreamNonBlocking flag. This indicates no synchronization with the default stream (stream 0). Stream 0 is not used anywere in PRT.*

- void prt_vsa_vdp_merge_channels (prt_vsa_t *vsa, struct prt_vdp_s *vdp)

  *Connects corresponding input and output channels of intra-node VDPs. An input channel always overrides an output channel. This way the on/off switch of the input channel is preserved.*

- void prt_vsa_vdp_track_tags (prt_vsa_t *vsa, struct prt_vdp_s *vdp)

  *Builds the list of channel connections to other nodes.*

- void prt_vsa_channel_tags (prt_vsa_t *vsa)

  *Assigns channel tags. Builds the node-tag lookup. Destroys channel lists.*

- void prt_vsa_channel_streams (prt_vsa_t *vsa)

  *Creates channel streams.*

- double prt_vsa_run (prt_vsa_t *vsa)

  *Implements the VSA's production cycle. Launches worker threads. Sends the master thread in the proxy production cycle. Joins the worker threads.*

- void **prt_vsa_config_set** (prt_vsa_t *vsa, enum prt_config_param_e param, enum prt_config_value_e value)
- void prt_vsa_thread_warmup_func_set (prt_vsa_t *vsa, void(*func)())

  *Sets a thread warmup function. If set, the thread warmup function is called by each thread right after launching and before threads are barriered and the timer is started. Allows for excluding the time for initialization procedures of libraries, such as loading of dynamic libraries, internal memory allocations, possibly time consuming pinned memory allocations, etc. A NULL function pointer can be passed to remove the thread warmup function.*

- void prt_vsa_device_warmup_func_set (prt_vsa_t *vsa, void(*func)())

  *Sets a device warmup function. If set, the device warmup function is called by each device right after launching and before devices are barriered and the timer is started. Allows for excluding the time for initialization procedures of libraries, such as loading of dynamic libraries, internal memory allocations, possibly time consuming pinned memory allocations, etc. A NULL function pointer can be passed to remove the device warmup function.*

- void prt_vsa_devices_warmup (prt_vsa_t *vsa)

  *Calls the warmup function for all devices and synchronizes.*

### 6.39.1 Detailed Description

Virtual Systolic Array.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file prt_vsa.h.

### 6.39.2 Function Documentation

#### 6.39.2.1 void prt_vsa_channel_streams ( prt_vsa_t * *vsa* )

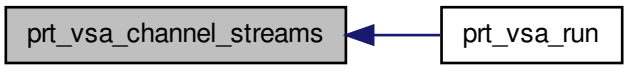
Creates channel streams.

**Parameters**

| | |
|---|---|
| *vsa* | – The VSA to create streams for. |

Definition at line 469 of file prt_vsa.c.

Here is the caller graph for this function:




#### 6.39.2.2 void prt_vsa_channel_tags ( prt_vsa_t * *vsa* )

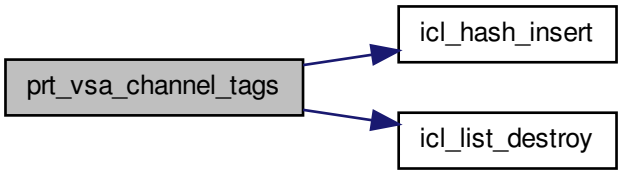
Assigns channel tags. Builds the node-tag lookup. Destroys channel lists.

**Parameters**

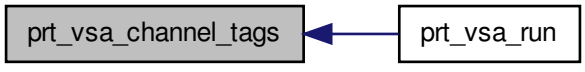
| | |
|---|---|
| *vsa* | – The VSA to find the tags for. |

Definition at line 432 of file prt_vsa.c.

Here is the call graph for this function:




Here is the caller graph for this function:



#### 6.39.2.3 void prt_vsa_devices_warmup ( prt_vsa_t * *vsa* )

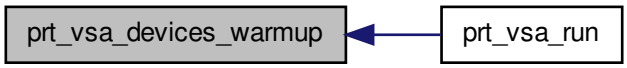
Calls the warmup function for all devices and synchronizes.

**Parameters**

| | |
|---|---|
| *vsa* | – The VSA to warmup devices for. |

Definition at line 694 of file prt_vsa.c.

Here is the caller graph for this function:




#### 6.39.2.4 void prt_vsa_vdp_merge_channels ( prt_vsa_t * *vsa*, prt_vdp_t * *vdp* )

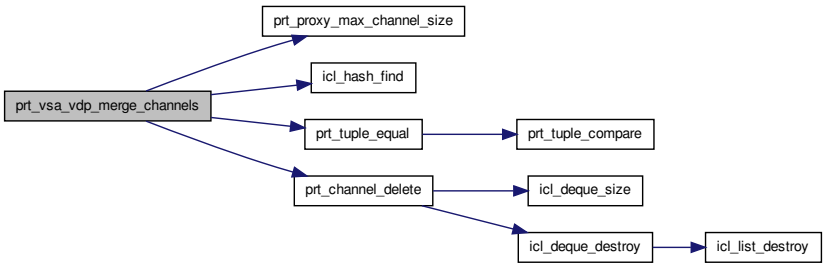
Connects corresponding input and output channels of intra-node VDPs. An input channel always overrides an output channel. This way the on/off switch of the input channel is preserved.

**Parameters**

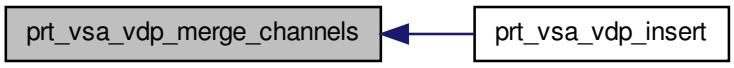
| | |
|---|---|
| *vsa* | – The VSA to merge channels within. |
| *vdp* | – The VDP to merge channels for. |

Definition at line 290 of file prt_vsa.c.

Here is the call graph for this function:



Here is the caller graph for this function:




#### 6.39.2.5 void prt_vsa_vdp_track_tags ( prt_vsa_t * *vsa*, prt_vdp_t * *vdp* )

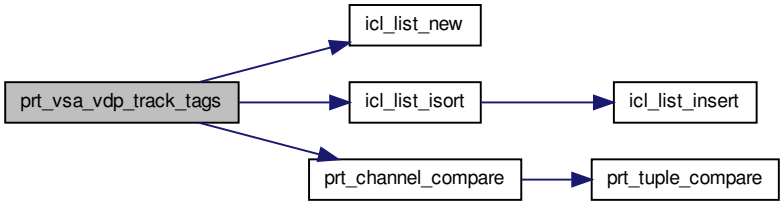
Builds the list of channel connections to other nodes.

**Parameters**

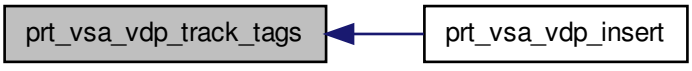
| | |
|---|---|
| *vsa* | – The VSA to track the tags within. |
| *vdp* | – The VDP to track the tags for. |

Definition at line 351 of file prt_vsa.c.

Here is the call graph for this function:




Here is the caller graph for this function:

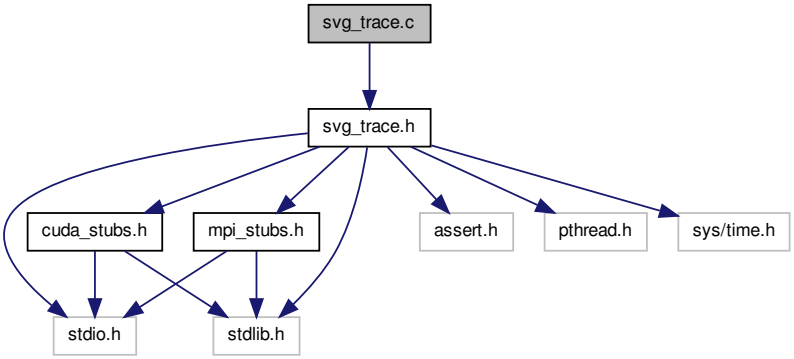


## 6.40 svg_trace.c File Reference

SVG tracing.

```
#include "svg_trace.h"
```

Include dependency graph for svg_trace.c:




### Functions

- double get_time_of_day ()

  *Returns current time.*
- void svg_trace_init (int num_cores, int num_devices)

  *Initializes tracing.*
- void svg_trace_start_cpu (int thread_rank)

  *Starts tracing of a CPU event.*
- void svg_trace_stop_cpu (int thread_rank, int color)

  *Stops tracing a CPU event.*
- void svg_trace_start_gpu (cudaStream_t stream)

  *Starts tracing a GPU event.*
- void svg_trace_stop_gpu (cudaStream_t stream, int color)

  *Stops tracing a GPU event.*
- void svg_trace_start_dma (cudaStream_t stream)

  *Starts tracing a DMA event.*
- void svg_trace_stop_dma (cudaStream_t stream, int color)

  *Stops tracing a DMA event.*
- void svg_trace_memory_host (long delta)

  *Registers host memory usage. The operation has to be atomic, because it can be invoked by a callback. Because there are two variables to keep track of, the level and the maximum, doing it with atomics is not worth it. Using a spinlock instead.*
- void svg_trace_memory_device (long delta)

  *Register device memory usage.*
- void svg_trace_finish (int num_cores, int num_devices)

  *Finishes tracing. Collects traces from all nodes. Writes the combined trace to an SVG file.*

### 6.40.1 Detailed Description

SVG tracing.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file svg_trace.c.

### 6.40.2 Function Documentation

#### 6.40.2.1 double get_time_of_day ( )

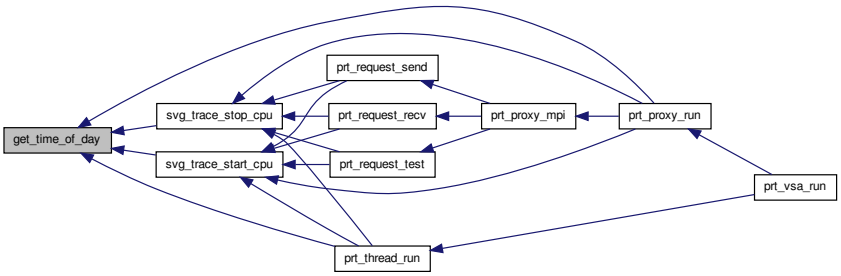
Returns current time.

**Returns**

Current Unix time in seconds as a double-precision number.

Definition at line 66 of file svg_trace.c.

Here is the caller graph for this function:




#### 6.40.2.2 void svg_trace_init ( int *num_cores*, int *num_devices* )

Initializes tracing.

**Parameters**

| | |
|---|---|
| *num_cores* | – The number of cores. |
| *num_devices* | – The number of devices. |

Definition at line 81 of file svg_trace.c.

Here is the caller graph for this function:


#### 6.40.2.3 void svg_trace_memory_device ( long *delta* )

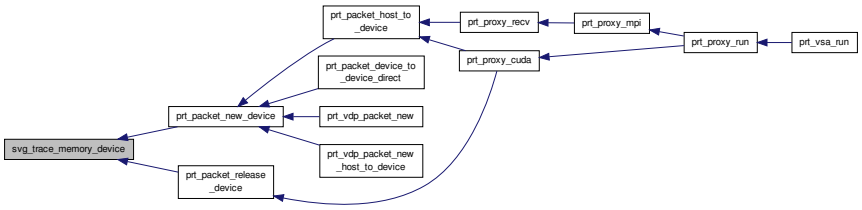
Register device memory usage.

**Parameters**

| | |
|---|---|
| *delta* | – The change of host memory usage in bytes. |

Definition at line 243 of file svg_trace.c.

Here is the caller graph for this function:




#### 6.40.2.4 void svg_trace_memory_host ( long *delta* )

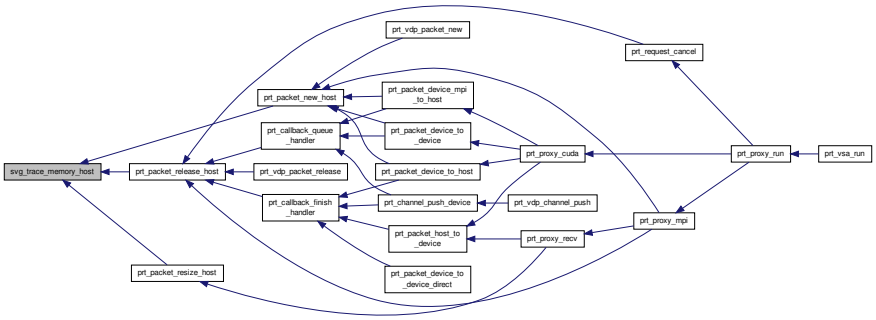
Registers host memory usage. The operation has to be atomic, because it can be invoked by a callback. Because there are two variables to keep track of, the level and the maximum, doing it with atomics is not worth it. Using a spinlock instead.

**Parameters**

| | |
|---|---|
| *delta* | – The change of host memory usage in bytes. |

Definition at line 228 of file svg_trace.c.

Here is the caller graph for this function:




#### 6.40.2.5 void svg_trace_start_cpu ( int *thread_rank* )

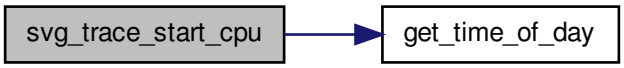
Starts tracing of a CPU event.

**Parameters**

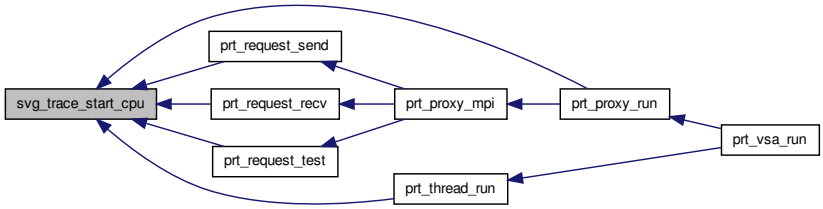
| | |
|---|---|
| *thread_rank* | – The rank of the thread. |

Definition at line 125 of file svg_trace.c.

Here is the call graph for this function:




Here is the caller graph for this function:



#### 6.40.2.6 void svg_trace_start_dma ( cudaStream_t *stream* )

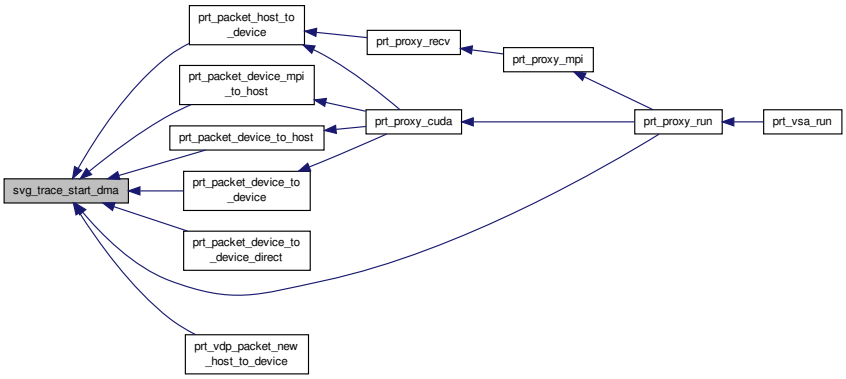
Starts tracing a DMA event.

**Parameters**

| | |
|---|---|
| *stream* | – The stream of the event. |

Definition at line 189 of file svg_trace.c.

Here is the caller graph for this function:




### 6.40.2.7 void svg_trace_start_gpu ( cudaStream_t *stream* )

Starts tracing a GPU event.

**Parameters**

| | |
|---|---|
| *stream* | – The stream of the event. |

Definition at line 153 of file svg_trace.c.

Here is the caller graph for this function:


### 6.40.2.8 void svg_trace_stop_cpu ( int *thread_rank,* int *color* )

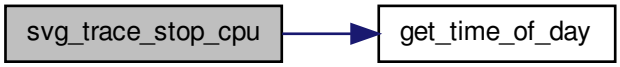
Stops tracing a CPU event.

**Parameters**

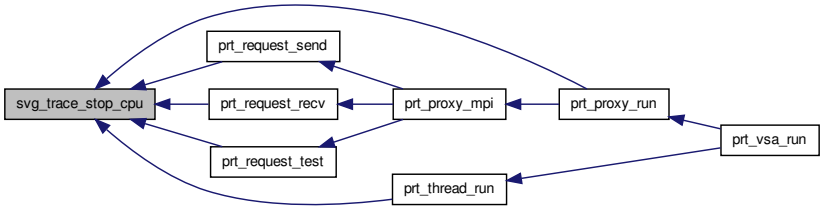
| | |
|---|---|
| *thread_rank* | – The rank of the thread. |
| *color* | – The RGB color of the SVG box. |

Definition at line 138 of file svg_trace.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.40.2.9 void svg_trace_stop_dma ( cudaStream_t *stream*, int *color* )

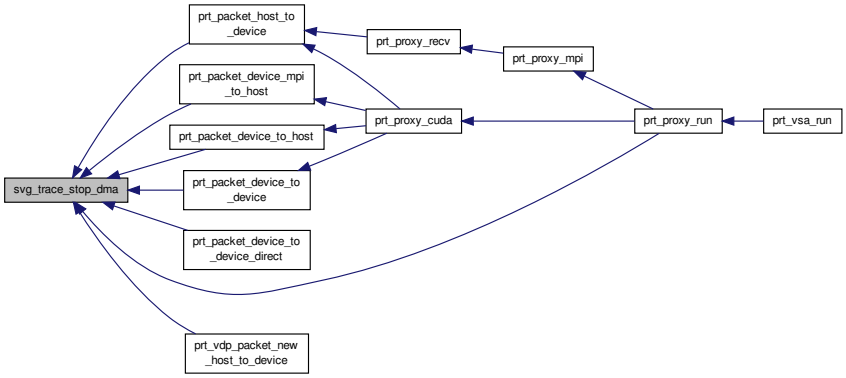
Stops tracing a DMA event.

**Parameters**

| | |
|---|---|
| *stream* | – The stream of the event. |
| *color* | – The RGB color of the SVG box. |

Definition at line 206 of file svg_trace.c.

Here is the caller graph for this function:




#### 6.40.2.10 void svg_trace_stop_gpu ( cudaStream_t *stream*, int *color* )

Stops tracing a GPU event.

**Parameters**

| | |
|---|---|
| *stream* | – The stream of the event. |
| *color* | – The RGB color of the SVG box. |

Definition at line 170 of file svg_trace.c.

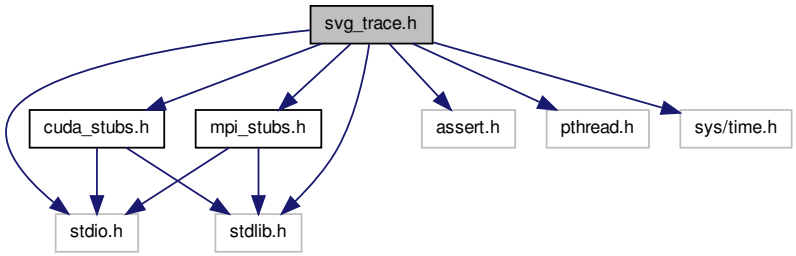
Here is the caller graph for this function:


## 6.41 svg_trace.h File Reference

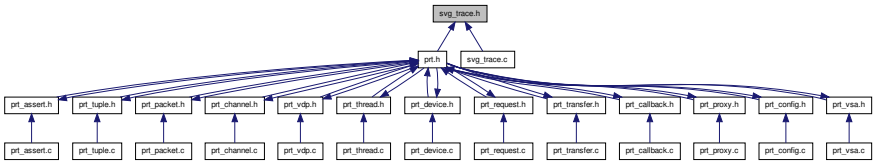
SVG tracing.

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <pthread.h>
#include <sys/time.h>
#include "mpi_stubs.h"
#include "cuda_stubs.h"
```

Include dependency graph for svg_trace.h:




This graph shows which files directly or indirectly include this file:




## Macros

- #define **SVG_TRACE_MAX_CORES** 64
- #define **SVG_TRACE_MAX_DEVICES** 16
- #define **SVG_TRACE_MAX_EVENTS** 65536
- #define **SVG_TRACE_MAX_MEM_SNAPSHOTS** 65536
- #define **SVG_TRACE_FILE_NAME_SIZE** 64

## Enumerations

- enum {
  **Pink** = 0xFFC0CB, **LightPink** = 0xFFB6C1, **HotPink** = 0xFF69B4, **DeepPink** = 0xFF1493,
  **PaleVioletRed** = 0xDB7093, **MediumVioletRed** = 0xC71585, **LightSalmon** = 0xFFA07A, **Salmon** = 0xFA8072,
  **DarkSalmon** = 0xE9967A, **LightCoral** = 0xF08080, **IndianRed** = 0xCD5C5C, **Crimson** = 0xDC143C,
  **FireBrick** = 0xB22222, **DarkRed** = 0x8B0000, **Red** = 0xFF0000, **OrangeRed** = 0xFF4500,
  **Tomato** = 0xFF6347, **Coral** = 0xFF7F50, **DarkOrange** = 0xFF8C00, **Orange** = 0xFFA500,
  **Gold** = 0xFFD700, **Yellow** = 0xFFFF00, **LightYellow** = 0xFFFFE0, **LemonChiffon** = 0xFFFACD,
  **LightGoldenrodYellow** = 0xFAFAD2, **PapayaWhip** = 0xFFEFD5, **Moccasin** = 0xFFE4B5, **PeachPuff** = 0xFFD-

AB9,
**PaleGoldenrod** = 0xEEE8AA, **Khaki** = 0xF0E68C, **DarkKhaki** = 0xBDB76B, **Cornsilk** = 0xFFF8DC,
**BlanchedAlmond** = 0xFFEBCD, **Bisque** = 0xFFE4C4, **NavajoWhite** = 0xFFDEAD, **Wheat** = 0xF5DEB3,
**BurlyWood** = 0xDEB887, **Tan** = 0xD2B48C, **RosyBrown** = 0xBC8F8F, **SandyBrown** = 0xF4A460,
**Goldenrod** = 0xDAA520, **DarkGoldenrod** = 0xB8860B, **Peru** = 0xCD853F, **Chocolate** = 0xD2691E,
**SaddleBrown** = 0x8B4513, **Sienna** = 0xA0522D, **Brown** = 0xA52A2A, **Maroon** = 0x800000,
**DarkOliveGreen** = 0x556B2F, **Olive** = 0x808000, **OliveDrab** = 0x6B8E23, **YellowGreen** = 0x9ACD32,
**LimeGreen** = 0x32CD32, **Lime** = 0x00FF00, **LawnGreen** = 0x7CFC00, **Chartreuse** = 0x7FFF00,
**GreenYellow** = 0xADFF2F, **SpringGreen** = 0x00FF7F, **MediumSpringGreen** = 0x00FA9A, **LightGreen** = 0x90-EE90,
**PaleGreen** = 0x98FB98, **DarkSeaGreen** = 0x8FBC8F, **MediumSeaGreen** = 0x3CB371, **SeaGreen** = 0x2E8B57,
**ForestGreen** = 0x228B22, **Green** = 0x008000, **DarkGreen** = 0x006400, **MediumAquamarine** = 0x66CDAA,
**Aqua** = 0x00FFFF, **Cyan** = 0x00FFFF, **LightCyan** = 0xE0FFFF, **PaleTurquoise** = 0xAFEEEE,
**Aquamarine** = 0x7FFFD4, **Turquoise** = 0x40E0D0, **MediumTurquoise** = 0x48D1CC, **DarkTurquoise** = 0x00C-ED1,
**LightSeaGreen** = 0x20B2AA, **CadetBlue** = 0x5F9EA0, **DarkCyan** = 0x008B8B, **Teal** = 0x008080,
**LightSteelBlue** = 0xB0C4DE, **PowderBlue** = 0xB0E0E6, **LightBlue** = 0xADD8E6, **SkyBlue** = 0x87CEEB,
**LightSkyBlue** = 0x87CEFA, **DeepSkyBlue** = 0x00BFFF, **DodgerBlue** = 0x1E90FF, **CornflowerBlue** = 0x6495-ED,
**SteelBlue** = 0x4682B4, **RoyalBlue** = 0x4169E1, **Blue** = 0x0000FF, **MediumBlue** = 0x0000CD,
**DarkBlue** = 0x00008B, **Navy** = 0x000080, **MidnightBlue** = 0x191970, **Lavender** = 0xE6E6FA,
**Thistle** = 0xD8BFD8, **Plum** = 0xDDA0DD, **Violet** = 0xEE82EE, **Orchid** = 0xDA70D6,
**Fuchsia** = 0xFF00FF, **Magenta** = 0xFF00FF, **MediumOrchid** = 0xBA55D3, **MediumPurple** = 0x9370DB,
**BlueViolet** = 0x8A2BE2, **DarkViolet** = 0x9400D3, **DarkOrchid** = 0x9932CC, **DarkMagenta** = 0x8B008B,
**Purple** = 0x800080, **Indigo** = 0x4B0082, **DarkSlateBlue** = 0x483D8B, **SlateBlue** = 0x6A5ACD,
**MediumSlateBlue** = 0x7B68EE, **White** = 0xFFFFFF, **Snow** = 0xFFFAFA, **Honeydew** = 0xF0FFF0,
**MintCream** = 0xF5FFFA, **Azure** = 0xF0FFFF, **AliceBlue** = 0xF0F8FF, **GhostWhite** = 0xF8F8FF,
**WhiteSmoke** = 0xF5F5F5, **Seashell** = 0xFFF5EE, **Beige** = 0xF5F5DC, **OldLace** = 0xFDF5E6,
**FloralWhite** = 0xFFFAF0, **Ivory** = 0xFFFFF0, **AntiqueWhite** = 0xFAEBD7, **Linen** = 0xFAF0E6,
**LavenderBlush** = 0xFFF0F5, **MistyRose** = 0xFFE4E1, **Gainsboro** = 0xDCDCDC, **LightGray** = 0xD3D3D3,
**Silver** = 0xC0C0C0, **DarkGray** = 0xA9A9A9, **Gray** = 0x808080, **DimGray** = 0x696969,
**LightSlateGray** = 0x778899, **SlateGray** = 0x708090, **DarkSlateGray** = 0x2F4F4F, **Black** = 0x000000 }

*Web colors.* http://en.wikipedia.org/wiki/Web_colors.

## Functions

- double get_time_of_day ()

  *Returns current time.*
- void svg_trace_init (int num_cores, int num_devices)

  *Initializes tracing.*
- void svg_trace_start_cpu (int thread_rank)

  *Starts tracing of a CPU event.*
- void svg_trace_stop_cpu (int thread_rank, int color)

  *Stops tracing a CPU event.*
- void svg_trace_start_gpu (cudaStream_t stream)

  *Starts tracing a GPU event.*
- void svg_trace_stop_gpu (cudaStream_t stream, int color)

  *Stops tracing a GPU event.*
- void svg_trace_start_dma (cudaStream_t stream)

  *Starts tracing a DMA event.*
- void svg_trace_stop_dma (cudaStream_t stream, int color)

*Stops tracing a DMA event.*

- void svg_trace_memory_host (long delta)

  *Registers host memory usage. The operation has to be atomic, because it can be invoked by a callback. Because there are two variables to keep track of, the level and the maximum, doing it with atomics is not worth it. Using a spinlock instead.*

- void svg_trace_memory_device (long delta)

  *Register device memory usage.*

- void svg_trace_finish (int num_cores, int num_devices)

  *Finishes tracing. Collects traces from all nodes. Writes the combined trace to an SVG file.*

### 6.41.1 Detailed Description

SVG tracing.

**Author**

Jakub Kurzak

PULSAR Runtime http://icl.utk.edu/pulsar/ Copyright (C) 2012-2015 University of Tennessee.

Definition in file svg_trace.h.

### 6.41.2 Function Documentation

#### 6.41.2.1 double get_time_of_day ( )

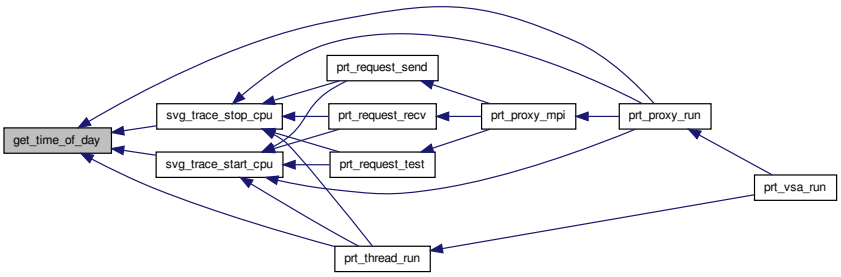
Returns current time.

**Returns**

Current Unix time in seconds as a double-precision number.

Definition at line 66 of file svg_trace.c.

Here is the caller graph for this function:




#### 6.41.2.2 void svg_trace_init ( int *num_cores*, int *num_devices* )

Initializes tracing.

**Parameters**

| | |
|---|---|
| *num_cores* | – The number of cores. |
| *num_devices* | – The number of devices. |

Definition at line 81 of file svg_trace.c.

Here is the caller graph for this function:


#### 6.41.2.3 void svg_trace_memory_device ( long *delta* )

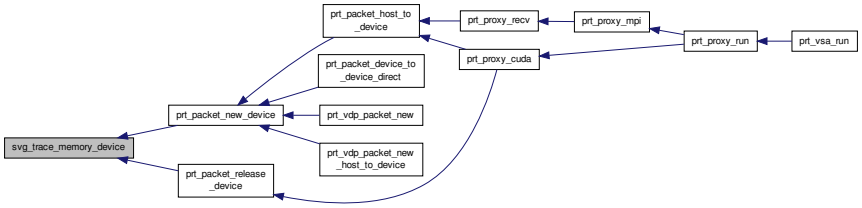
Register device memory usage.

**Parameters**

| | |
|---|---|
| *delta* | – The change of host memory usage in bytes. |

Definition at line 243 of file svg_trace.c.

Here is the caller graph for this function:




#### 6.41.2.4 void svg_trace_memory_host ( long *delta* )

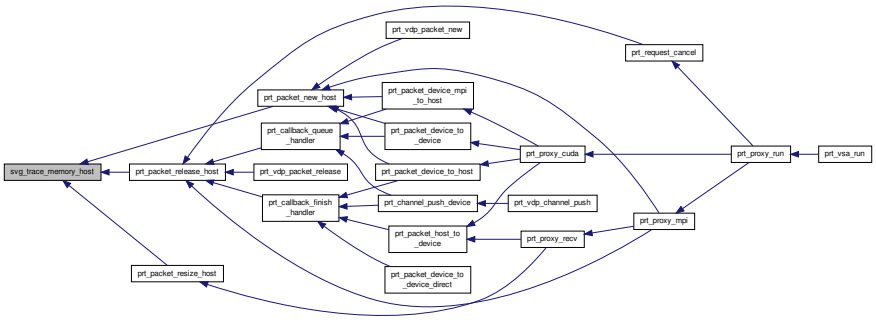
Registers host memory usage. The operation has to be atomic, because it can be invoked by a callback. Because there are two variables to keep track of, the level and the maximum, doing it with atomics is not worth it. Using a spinlock instead.

**Parameters**

| | |
|---|---|
| *delta* | – The change of host memory usage in bytes. |

Definition at line 228 of file svg_trace.c.

Here is the caller graph for this function:




#### 6.41.2.5 void svg_trace_start_cpu ( int *thread_rank* )

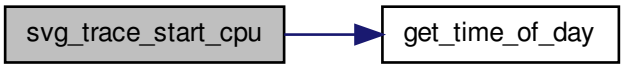
Starts tracing of a CPU event.

**Parameters**

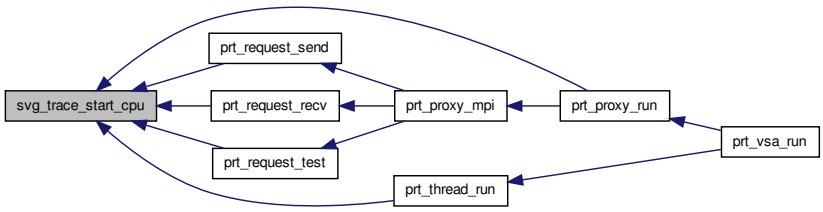
| | |
|---|---|
| *thread_rank* | – The rank of the thread. |

Definition at line 125 of file svg_trace.c.

Here is the call graph for this function:




Here is the caller graph for this function:



#### 6.41.2.6 void svg_trace_start_dma ( cudaStream_t *stream* )

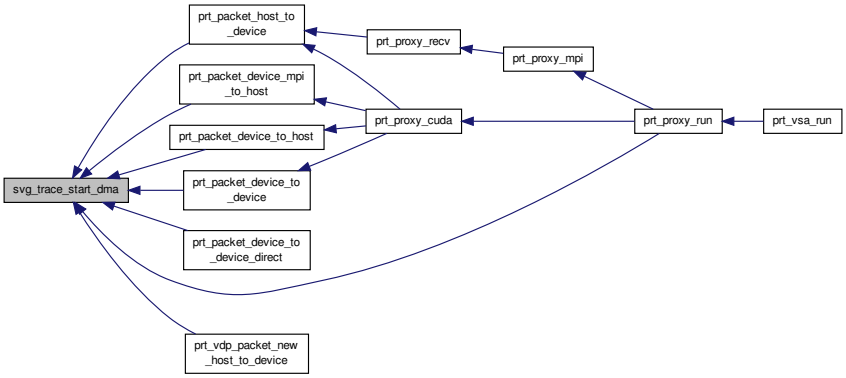
Starts tracing a DMA event.

**Parameters**

| | |
|---|---|
| *stream* | – The stream of the event. |

Definition at line 189 of file svg_trace.c.

Here is the caller graph for this function:




#### 6.41.2.7 void svg_trace_start_gpu ( cudaStream_t *stream* )

Starts tracing a GPU event.

**Parameters**

| | |
|---|---|
| *stream* | – The stream of the event. |

Definition at line 153 of file svg_trace.c.

Here is the caller graph for this function:


#### 6.41.2.8 void svg_trace_stop_cpu ( int *thread_rank*, int *color* )

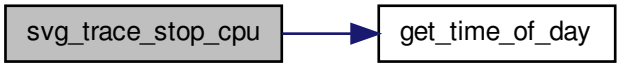
Stops tracing a CPU event.

**Parameters**

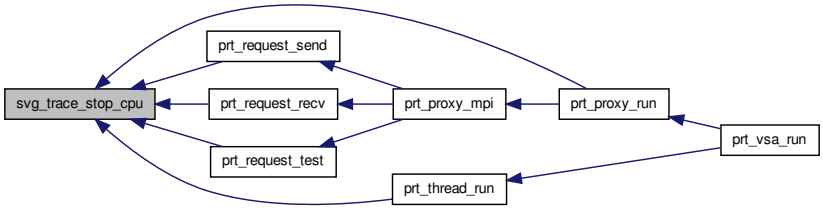
| | |
|---|---|
| *thread_rank* | – The rank of the thread. |
| *color* | – The RGB color of the SVG box. |

Definition at line 138 of file svg_trace.c.

Here is the call graph for this function:




Here is the caller graph for this function:




#### 6.41.2.9 void svg_trace_stop_dma ( cudaStream_t *stream*, int *color* )

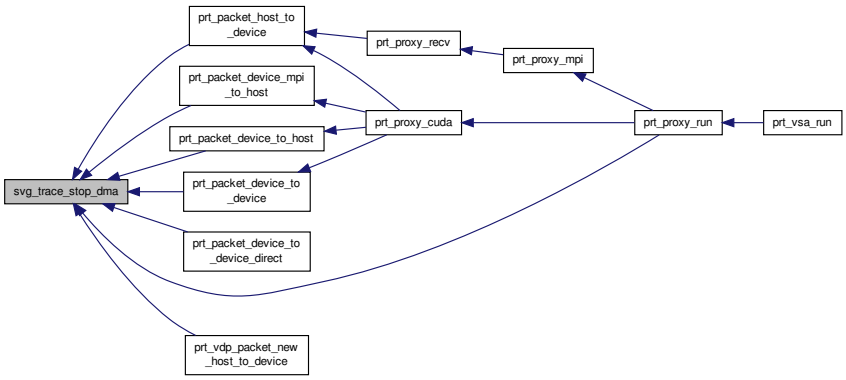
Stops tracing a DMA event.

**Parameters**

| | |
|---|---|
| *stream* | – The stream of the event. |
| *color* | – The RGB color of the SVG box. |

Definition at line 206 of file svg_trace.c.

Here is the caller graph for this function:




#### 6.41.2.10 void svg_trace_stop_gpu ( cudaStream_t *stream*, int *color* )

Stops tracing a GPU event.

**Parameters**

| | |
|---|---|
| *stream* | – The stream of the event. |
| *color* | – The RGB color of the SVG box. |

Definition at line 170 of file svg_trace.c.

Here is the caller graph for this function:

# Index


cuda_stubs.c, 39
cuda_stubs.h, 40

get_time_of_day
    svg_trace.c, 233
    svg_trace.h, 242
gpu_free
    gpu_malloc.c, 43
    gpu_malloc.h, 46
gpu_malloc
    gpu_malloc.c, 43
    gpu_malloc.h, 47
gpu_malloc.c, 42
    gpu_free, 43
    gpu_malloc, 43
    gpu_malloc_fini, 44
    gpu_malloc_init, 44
gpu_malloc.h, 45
    gpu_free, 46
    gpu_malloc, 47
    gpu_malloc_fini, 47
    gpu_malloc_init, 48
gpu_malloc_fini
    gpu_malloc.c, 44
    gpu_malloc.h, 47
gpu_malloc_init
    gpu_malloc.c, 44
    gpu_malloc.h, 48
gpu_malloc_s, 21

icl_deque.c, 48
    icl_deque_append, 50
    icl_deque_delete, 50
    icl_deque_destroy, 51
    icl_deque_first, 52
    icl_deque_new, 53
    icl_deque_next, 54
    icl_deque_prepend, 54
    icl_deque_size, 55
icl_deque.h, 55
    icl_deque_append, 57
    icl_deque_delete, 58
    icl_deque_destroy, 59
    icl_deque_first, 60
    icl_deque_new, 60
    icl_deque_next, 61
    icl_deque_prepend, 62
    icl_deque_size, 62
icl_deque_append
    icl_deque.c, 50
    icl_deque.h, 57
icl_deque_delete
    icl_deque.c, 50
    icl_deque.h, 58
icl_deque_destroy
    icl_deque.c, 51
    icl_deque.h, 59
icl_deque_first
    icl_deque.c, 52
    icl_deque.h, 60
icl_deque_new
    icl_deque.c, 53
    icl_deque.h, 60
icl_deque_next
    icl_deque.c, 54
    icl_deque.h, 61
icl_deque_prepend
    icl_deque.c, 54
    icl_deque.h, 62
icl_deque_s, 22
icl_deque_size
    icl_deque.c, 55
    icl_deque.h, 62
icl_entry_s, 22
icl_hash.c, 63
    icl_hash_create, 64
    icl_hash_delete, 65
    icl_hash_destroy, 65
    icl_hash_dump, 66
    icl_hash_find, 66
    icl_hash_insert, 66
    icl_hash_update_insert, 68
icl_hash.h, 68
    icl_hash_create, 70
    icl_hash_delete, 70
    icl_hash_destroy, 72
    icl_hash_dump, 72
    icl_hash_find, 73
    icl_hash_foreach, 70
    icl_hash_insert, 73
    icl_hash_update_insert, 74

icl_hash_create
    icl_hash.c, 64
    icl_hash.h, 70
icl_hash_delete
    icl_hash.c, 65
    icl_hash.h, 70
icl_hash_destroy
    icl_hash.c, 65
    icl_hash.h, 72
icl_hash_dump
    icl_hash.c, 66
    icl_hash.h, 72
icl_hash_find
    icl_hash.c, 66
    icl_hash.h, 73
icl_hash_foreach
    icl_hash.h, 70
icl_hash_insert
    icl_hash.c, 66
    icl_hash.h, 73
icl_hash_s, 23
icl_hash_update_insert
    icl_hash.c, 68
    icl_hash.h, 74
icl_list.c, 74
    icl_list_append, 76
    icl_list_concat, 77
    icl_list_delete, 77
    icl_list_destroy, 78
    icl_list_first, 78
    icl_list_insert, 79
    icl_list_isort, 79
    icl_list_last, 80
    icl_list_new, 80
    icl_list_next, 81
    icl_list_prepend, 81
    icl_list_prev, 83
    icl_list_search, 83
    icl_list_size, 84
icl_list.h, 84
    icl_list_append, 86
    icl_list_concat, 87
    icl_list_delete, 87
    icl_list_destroy, 88
    icl_list_first, 88
    icl_list_insert, 89
    icl_list_isort, 89
    icl_list_last, 90
    icl_list_new, 90
    icl_list_next, 91
    icl_list_prepend, 91
    icl_list_prev, 93
    icl_list_search, 93
    icl_list_size, 94
icl_list_append
    icl_list.c, 76
    icl_list.h, 86
icl_list_concat
    icl_list.c, 77
    icl_list.h, 87
icl_list_delete
    icl_list.c, 77
    icl_list.h, 87
icl_list_destroy
    icl_list.c, 78
    icl_list.h, 88
icl_list_first
    icl_list.c, 78
    icl_list.h, 88
icl_list_insert
    icl_list.c, 79
    icl_list.h, 89
icl_list_isort
    icl_list.c, 79
    icl_list.h, 89
icl_list_last
    icl_list.c, 80
    icl_list.h, 90
icl_list_new
    icl_list.c, 80
    icl_list.h, 90
icl_list_next
    icl_list.c, 81
    icl_list.h, 91
icl_list_prepend
    icl_list.c, 81
    icl_list.h, 91
icl_list_prev
    icl_list.c, 83
    icl_list.h, 93
icl_list_s, 24
icl_list_search
    icl_list.c, 83
    icl_list.h, 93
icl_list_size
    icl_list.c, 84
    icl_list.h, 94

MPI_Request, 24
MPI_Status, 24
mpi_stubs.c, 94
mpi_stubs.h, 96

PRT API - accelerator interface, 19
    prt_vdp_packet_new_host_to_device, 19
    prt_vsa_device_warmup_func_set, 19
PRT API - auxiliary interface, 17
    prt_vdp_channel_off, 17
    prt_vdp_channel_on, 17

prt_vsa_config_set, 18
prt_vsa_thread_warmup_func_set, 18
PRT API - core interface, 7
prt_channel_new, 8
prt_tuple_new, 8
prt_vdp_channel_insert, 9
prt_vdp_channel_pop, 9
prt_vdp_channel_push, 10
prt_vdp_new, 10
prt_vdp_packet_new, 11
prt_vdp_packet_release, 11
prt_vsa_delete, 12
prt_vsa_new, 12
prt_vsa_run, 14
prt_vsa_vdp_insert, 15
prt.h, 97
prt_assert.c, 99
prt_assert_line_file, 100
prt_error_line_file, 100
prt_warning_line_file, 101
prt_assert.h, 101
prt_assert_line_file, 102
prt_error_line_file, 103
prt_warning_line_file, 103
prt_assert_line_file
prt_assert.c, 100
prt_assert.h, 102
prt_callback.c, 104
prt_callback_finish_delete, 105
prt_callback_finish_handler, 105
prt_callback_finish_new, 106
prt_callback_queue_delete, 107
prt_callback_queue_handler, 107
prt_callback_queue_new, 108
prt_callback_release_delete, 109
prt_callback_release_handler, 109
prt_callback_release_new, 110
prt_callback.h, 111
prt_callback_finish_delete, 112
prt_callback_finish_handler, 113
prt_callback_finish_new, 113
prt_callback_queue_delete, 114
prt_callback_queue_handler, 114
prt_callback_queue_new, 115
prt_callback_release_delete, 116
prt_callback_release_handler, 116
prt_callback_release_new, 117
prt_callback_finish_delete
prt_callback.c, 105
prt_callback.h, 112
prt_callback_finish_handler
prt_callback.c, 105
prt_callback.h, 113
prt_callback_finish_new
prt_callback.c, 106
prt_callback.h, 113
prt_callback_finish_s, 25
prt_callback_queue_delete
prt_callback.c, 107
prt_callback.h, 114
prt_callback_queue_handler
prt_callback.c, 107
prt_callback.h, 114
prt_callback_queue_new
prt_callback.c, 108
prt_callback.h, 115
prt_callback_queue_s, 26
prt_callback_release_delete
prt_callback.c, 109
prt_callback.h, 116
prt_callback_release_handler
prt_callback.c, 109
prt_callback.h, 116
prt_callback_release_new
prt_callback.c, 110
prt_callback.h, 117
prt_callback_release_s, 26
prt_channel.c, 118
prt_channel_compare, 119
prt_channel_delete, 120
prt_channel_empty, 121
prt_channel_off, 122
prt_channel_on, 122
prt_channel_pop, 122
prt_channel_push_device, 123
prt_channel_push_host, 124
prt_channel.h, 125
prt_channel_compare, 127
prt_channel_delete, 128
prt_channel_empty, 128
prt_channel_off, 129
prt_channel_on, 129
prt_channel_pop, 131
prt_channel_push_device, 132
prt_channel_push_host, 132
prt_channel_t, 127
prt_channel_compare
prt_channel.c, 119
prt_channel.h, 127
prt_channel_delete
prt_channel.c, 120
prt_channel.h, 128
prt_channel_empty
prt_channel.c, 121
prt_channel.h, 128
prt_channel_new
PRT API - core interface, 8
prt_channel_off

prt_channel.c, 122
prt_channel.h, 129
prt_channel_on
prt_channel.c, 122
prt_channel.h, 129
prt_channel_pop
prt_channel.c, 122
prt_channel.h, 131
prt_channel_push_device
prt_channel.c, 123
prt_channel.h, 132
prt_channel_push_host
prt_channel.c, 124
prt_channel.h, 132
prt_channel_s, 27
prt_channel_t
prt_channel.h, 127
prt_config.c, 133
prt_config_delete, 134
prt_config_new, 134
prt_config.h, 135
prt_config_delete, 136
prt_config_new, 137
prt_config_delete
prt_config.c, 134
prt_config.h, 136
prt_config_new
prt_config.c, 134
prt_config.h, 137
prt_config_s, 29
prt_device.c, 137
prt_device_cycle, 138
prt_device_delete, 139
prt_device_new, 139
prt_device.h, 140
prt_device_cycle, 142
prt_device_delete, 142
prt_device_new, 143
prt_device_t, 141
prt_device_cycle
prt_device.c, 138
prt_device.h, 142
prt_device_delete
prt_device.c, 139
prt_device.h, 142
prt_device_new
prt_device.c, 139
prt_device.h, 143
prt_device_s, 29
prt_device_t
prt_device.h, 141
prt_error_line_file
prt_assert.c, 100
prt_assert.h, 103
prt_mapping_s, 30
prt_packet.c, 144
prt_packet_device_mpi_to_host, 145
prt_packet_device_to_device, 146
prt_packet_device_to_device_direct, 147
prt_packet_device_to_host, 147
prt_packet_host_to_device, 148
prt_packet_new_device, 149
prt_packet_new_host, 150
prt_packet_release_device, 151
prt_packet_release_host, 151
prt_packet_resize_host, 153
prt_packet.h, 154
prt_packet_device_mpi_to_host, 156
prt_packet_device_to_device, 156
prt_packet_device_to_device_direct, 157
prt_packet_device_to_host, 157
prt_packet_host_to_device, 158
prt_packet_new_device, 159
prt_packet_new_host, 160
prt_packet_release_device, 161
prt_packet_release_host, 162
prt_packet_resize_host, 162
prt_packet_t, 155
prt_packet_device_mpi_to_host
prt_packet.c, 145
prt_packet.h, 156
prt_packet_device_to_device
prt_packet.c, 146
prt_packet.h, 156
prt_packet_device_to_device_direct
prt_packet.c, 147
prt_packet.h, 157
prt_packet_device_to_host
prt_packet.c, 147
prt_packet.h, 157
prt_packet_host_to_device
prt_packet.c, 148
prt_packet.h, 158
prt_packet_new_device
prt_packet.c, 149
prt_packet.h, 159
prt_packet_new_host
prt_packet.c, 150
prt_packet.h, 160
prt_packet_release_device
prt_packet.c, 151
prt_packet.h, 161
prt_packet_release_host
prt_packet.c, 151
prt_packet.h, 162
prt_packet_resize_host
prt_packet.c, 153
prt_packet.h, 162

prt_packet_s, 30
prt_packet_t
    prt_packet.h, 155
prt_proxy.c, 164
    prt_proxy_cuda, 165
    prt_proxy_delete, 166
    prt_proxy_max_channel_size, 167
    prt_proxy_mpi, 168
    prt_proxy_new, 169
    prt_proxy_recv, 170
    prt_proxy_run, 170
prt_proxy.h, 172
    prt_proxy_cuda, 173
    prt_proxy_delete, 174
    prt_proxy_max_channel_size, 175
    prt_proxy_mpi, 176
    prt_proxy_new, 177
    prt_proxy_recv, 178
    prt_proxy_run, 178
    prt_proxy_t, 173
prt_proxy_cuda
    prt_proxy.c, 165
    prt_proxy.h, 173
prt_proxy_delete
    prt_proxy.c, 166
    prt_proxy.h, 174
prt_proxy_max_channel_size
    prt_proxy.c, 167
    prt_proxy.h, 175
prt_proxy_mpi
    prt_proxy.c, 168
    prt_proxy.h, 176
prt_proxy_new
    prt_proxy.c, 169
    prt_proxy.h, 177
prt_proxy_recv
    prt_proxy.c, 170
    prt_proxy.h, 178
prt_proxy_run
    prt_proxy.c, 170
    prt_proxy.h, 178
prt_proxy_s, 31
prt_proxy_t
    prt_proxy.h, 173
prt_request.c, 180
    prt_request_cancel, 181
    prt_request_delete, 181
    prt_request_new, 182
    prt_request_recv, 182
    prt_request_send, 183
    prt_request_test, 184
prt_request.h, 184
    prt_request_cancel, 186
    prt_request_delete, 186
    prt_request_new, 187
    prt_request_recv, 187
    prt_request_send, 188
    prt_request_test, 189
prt_request_cancel
    prt_request.c, 181
    prt_request.h, 186
prt_request_delete
    prt_request.c, 181
    prt_request.h, 186
prt_request_new
    prt_request.c, 182
    prt_request.h, 187
prt_request_recv
    prt_request.c, 182
    prt_request.h, 187
prt_request_s, 32
prt_request_send
    prt_request.c, 183
    prt_request.h, 188
prt_request_test
    prt_request.c, 184
    prt_request.h, 189
prt_thread.c, 189
    prt_thread_delete, 190
    prt_thread_new, 191
    prt_thread_run, 192
prt_thread.h, 193
    prt_thread_delete, 195
    prt_thread_new, 195
    prt_thread_run, 196
    prt_thread_t, 194
prt_thread_delete
    prt_thread.c, 190
    prt_thread.h, 195
prt_thread_new
    prt_thread.c, 191
    prt_thread.h, 195
prt_thread_run
    prt_thread.c, 192
    prt_thread.h, 196
prt_thread_s, 34
prt_thread_t
    prt_thread.h, 194
prt_transfer.c, 197
    prt_transfer_delete, 198
    prt_transfer_new, 198
prt_transfer.h, 199
    prt_transfer_delete, 200
prt_transfer_delete
    prt_transfer.c, 198
    prt_transfer.h, 200
prt_transfer_new
    prt_transfer.c, 198

prt_transfer_s, 34
prt_tuple.c, 200
    prt_tuple_cat, 202
    prt_tuple_compare, 203
    prt_tuple_copy, 203
    prt_tuple_delete, 204
    prt_tuple_equal, 204
    prt_tuple_hash, 205
    prt_tuple_len, 206
    prt_tuple_print, 206
prt_tuple.h, 207
    prt_tuple_cat, 208
    prt_tuple_compare, 209
    prt_tuple_copy, 209
    prt_tuple_delete, 210
    prt_tuple_equal, 210
    prt_tuple_hash, 210
    prt_tuple_len, 211
    prt_tuple_print, 211
prt_tuple_cat
    prt_tuple.c, 202
    prt_tuple.h, 208
prt_tuple_compare
    prt_tuple.c, 203
    prt_tuple.h, 209
prt_tuple_copy
    prt_tuple.c, 203
    prt_tuple.h, 209
prt_tuple_delete
    prt_tuple.c, 204
    prt_tuple.h, 210
prt_tuple_equal
    prt_tuple.c, 204
    prt_tuple.h, 210
    prt_vsa.c, 221
prt_tuple_hash
    prt_tuple.c, 205
    prt_tuple.h, 210
    prt_vsa.c, 222
prt_tuple_len
    prt_tuple.c, 206
    prt_tuple.h, 211
prt_tuple_new
    PRT API - core interface, 8
prt_tuple_print
    prt_tuple.c, 206
    prt_tuple.h, 211
prt_vdp.c, 211
    prt_vdp_annihilate, 213
    prt_vdp_delete, 213
    prt_vdp_ready, 214
prt_vdp.h, 215
    prt_vdp_annihilate, 217
    prt_vdp_delete, 218
    prt_vdp_ready, 219
prt_vdp_annihilate
    prt_vdp.c, 213
    prt_vdp.h, 217
prt_vdp_channel_insert
    PRT API - core interface, 9
prt_vdp_channel_off
    PRT API - auxiliary interface, 17
prt_vdp_channel_on
    PRT API - auxiliary interface, 17
prt_vdp_channel_pop
    PRT API - core interface, 9
prt_vdp_channel_push
    PRT API - core interface, 10
prt_vdp_delete
    prt_vdp.c, 213
    prt_vdp.h, 218
prt_vdp_new
    PRT API - core interface, 10
prt_vdp_packet_new
    PRT API - core interface, 11
prt_vdp_packet_new_host_to_device
    PRT API - accelerator interface, 19
prt_vdp_packet_release
    PRT API - core interface, 11
prt_vdp_ready
    prt_vdp.c, 214
    prt_vdp.h, 219
prt_vdp_s, 35
prt_vsa.c, 219
    prt_tuple_equal, 221
    prt_tuple_hash, 222
    prt_vsa_channel_streams, 222
    prt_vsa_channel_tags, 223
    prt_vsa_devices_warmup, 224
    prt_vsa_vdp_merge_channels, 224
    prt_vsa_vdp_track_tags, 225
prt_vsa.h, 226
    prt_vsa_channel_streams, 228
    prt_vsa_channel_tags, 229
    prt_vsa_devices_warmup, 230
    prt_vsa_vdp_merge_channels, 230
    prt_vsa_vdp_track_tags, 231
prt_vsa_channel_streams
    prt_vsa.c, 222
    prt_vsa.h, 228
prt_vsa_channel_tags
    prt_vsa.c, 223
    prt_vsa.h, 229
prt_vsa_config_set
    PRT API - auxiliary interface, 18
prt_vsa_delete
    PRT API - core interface, 12
prt_vsa_device_warmup_func_set

PRT API - accelerator interface, 19
prt_vsa_devices_warmup
prt_vsa.c, 224
prt_vsa.h, 230
prt_vsa_new
PRT API - core interface, 12
prt_vsa_run
PRT API - core interface, 14
prt_vsa_s, 37
prt_vsa_thread_warmup_func_set
PRT API - auxiliary interface, 18
prt_vsa_vdp_insert
PRT API - core interface, 15
prt_vsa_vdp_merge_channels
prt_vsa.c, 224
prt_vsa.h, 230
prt_vsa_vdp_track_tags
prt_vsa.c, 225
prt_vsa.h, 231
prt_warning_line_file
prt_assert.c, 101
prt_assert.h, 103

segment, 38
svg_trace.c, 232
get_time_of_day, 233
svg_trace_init, 233
svg_trace_memory_device, 234
svg_trace_memory_host, 234
svg_trace_start_cpu, 235
svg_trace_start_dma, 235
svg_trace_start_gpu, 237
svg_trace_stop_cpu, 237
svg_trace_stop_dma, 238
svg_trace_stop_gpu, 239
svg_trace.h, 239
get_time_of_day, 242
svg_trace_init, 242
svg_trace_memory_device, 243
svg_trace_memory_host, 243
svg_trace_start_cpu, 244
svg_trace_start_dma, 245
svg_trace_start_gpu, 245
svg_trace_stop_cpu, 245
svg_trace_stop_dma, 246
svg_trace_stop_gpu, 247
svg_trace_init
svg_trace.c, 233
svg_trace.h, 242
svg_trace_memory_device
svg_trace.c, 234
svg_trace.h, 243
svg_trace_memory_host
svg_trace.c, 234
svg_trace.h, 243
svg_trace_start_cpu
svg_trace.c, 235
svg_trace.h, 244
svg_trace_start_dma
svg_trace.c, 235
svg_trace.h, 245
svg_trace_start_gpu
svg_trace.c, 237
svg_trace.h, 245
svg_trace_stop_cpu
svg_trace.c, 237
svg_trace.h, 245
svg_trace_stop_dma
svg_trace.c, 238
svg_trace.h, 246
svg_trace_stop_gpu
svg_trace.c, 239
svg_trace.h, 247