

PULSAR

2.0.0

Generated by Doxygen 1.8.4

Thu Dec 4 2014 14:16:56

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	67
6.8	icl_hash.h File Reference	68
6.8.1	Detailed Description	69
6.8.2	Macro Definition Documentation	69
6.8.2.1	icl_hash_foreach	69
6.8.3	Function Documentation	69

6.8.3.1	icl_hash_create	69
6.8.3.2	icl_hash_delete	70
6.8.3.3	icl_hash_destroy	70
6.8.3.4	icl_hash_dump	71
6.8.3.5	icl_hash_find	71
6.8.3.6	icl_hash_insert	72
6.8.3.7	icl_hash_update_insert	72
6.9	icl_list.c File Reference	73
6.9.1	Detailed Description	74
6.9.2	Function Documentation	74
6.9.2.1	icl_list_append	74
6.9.2.2	icl_list_concat	75
6.9.2.3	icl_list_delete	75
6.9.2.4	icl_list_destroy	76
6.9.2.5	icl_list_first	77
6.9.2.6	icl_list_insert	77
6.9.2.7	icl_list_isort	78
6.9.2.8	icl_list_last	79
6.9.2.9	icl_list_new	79
6.9.2.10	icl_list_next	80
6.9.2.11	icl_list_prepend	80
6.9.2.12	icl_list_prev	81
6.9.2.13	icl_list_search	81
6.9.2.14	icl_list_size	82
6.10	icl_list.h File Reference	82
6.10.1	Detailed Description	84
6.10.2	Function Documentation	84
6.10.2.1	icl_list_append	84
6.10.2.2	icl_list_concat	85
6.10.2.3	icl_list_delete	85
6.10.2.4	icl_list_destroy	86
6.10.2.5	icl_list_first	87
6.10.2.6	icl_list_insert	88
6.10.2.7	icl_list_isort	89
6.10.2.8	icl_list_last	89
6.10.2.9	icl_list_new	90
6.10.2.10	icl_list_next	90

6.10.2.11	icl_list_prepend	91
6.10.2.12	icl_list_prev	92
6.10.2.13	icl_list_search	92
6.10.2.14	icl_list_size	93
6.11	mpi_stubs.c File Reference	93
6.11.1	Detailed Description	94
6.12	mpi_stubs.h File Reference	95
6.12.1	Detailed Description	96
6.13	prt.h File Reference	96
6.13.1	Detailed Description	98
6.14	prt_assert.c File Reference	98
6.14.1	Detailed Description	99
6.14.2	Function Documentation	99
6.14.2.1	prt_assert_line_file	99
6.14.2.2	prt_error_line_file	99
6.14.2.3	prt_warning_line_file	100
6.15	prt_assert.h File Reference	100
6.15.1	Detailed Description	101
6.15.2	Function Documentation	101
6.15.2.1	prt_assert_line_file	101
6.15.2.2	prt_error_line_file	102
6.15.2.3	prt_warning_line_file	102
6.16	prt_callback.c File Reference	103
6.16.1	Detailed Description	103
6.16.2	Function Documentation	104
6.16.2.1	prt_callback_finish_delete	104
6.16.2.2	prt_callback_finish_handler	104
6.16.2.3	prt_callback_finish_new	105
6.16.2.4	prt_callback_queue_delete	106
6.16.2.5	prt_callback_queue_handler	106
6.16.2.6	prt_callback_queue_new	107
6.16.2.7	prt_callback_release_delete	108
6.16.2.8	prt_callback_release_handler	108
6.16.2.9	prt_callback_release_new	109
6.17	prt_callback.h File Reference	110
6.17.1	Detailed Description	111
6.17.2	Function Documentation	111

6.17.2.1	prt_callback_finish_delete	111
6.17.2.2	prt_callback_finish_handler	112
6.17.2.3	prt_callback_finish_new	112
6.17.2.4	prt_callback_queue_delete	113
6.17.2.5	prt_callback_queue_handler	113
6.17.2.6	prt_callback_queue_new	114
6.17.2.7	prt_callback_release_delete	115
6.17.2.8	prt_callback_release_handler	115
6.17.2.9	prt_callback_release_new	116
6.18	prt_channel.c File Reference	117
6.18.1	Detailed Description	117
6.18.2	Function Documentation	118
6.18.2.1	prt_channel_compare	118
6.18.2.2	prt_channel_delete	119
6.18.2.3	prt_channel_empty	120
6.18.2.4	prt_channel_off	121
6.18.2.5	prt_channel_on	121
6.18.2.6	prt_channel_pop	122
6.18.2.7	prt_channel_push_device	122
6.18.2.8	prt_channel_push_host	123
6.19	prt_channel.h File Reference	124
6.19.1	Detailed Description	126
6.19.2	Typedef Documentation	126
6.19.2.1	prt_channel_t	126
6.19.3	Function Documentation	126
6.19.3.1	prt_channel_compare	126
6.19.3.2	prt_channel_delete	127
6.19.3.3	prt_channel_empty	127
6.19.3.4	prt_channel_off	128
6.19.3.5	prt_channel_on	129
6.19.3.6	prt_channel_pop	130
6.19.3.7	prt_channel_push_device	131
6.19.3.8	prt_channel_push_host	132
6.20	prt_config.c File Reference	132
6.20.1	Detailed Description	133
6.20.2	Function Documentation	133
6.20.2.1	prt_config_delete	133

6.20.2.2	prt_config_new	133
6.21	prt_config.h File Reference	134
6.21.1	Detailed Description	135
6.21.2	Function Documentation	135
6.21.2.1	prt_config_delete	135
6.21.2.2	prt_config_new	136
6.22	prt_device.c File Reference	136
6.22.1	Detailed Description	137
6.22.2	Function Documentation	137
6.22.2.1	prt_device_cycle	137
6.22.2.2	prt_device_delete	138
6.22.2.3	prt_device_new	138
6.23	prt_device.h File Reference	139
6.23.1	Detailed Description	140
6.23.2	Typedef Documentation	140
6.23.2.1	prt_device_t	140
6.23.3	Function Documentation	141
6.23.3.1	prt_device_cycle	141
6.23.3.2	prt_device_delete	141
6.23.3.3	prt_device_new	142
6.24	prt_packet.c File Reference	143
6.24.1	Detailed Description	144
6.24.2	Function Documentation	144
6.24.2.1	prt_packet_device_mpi_to_host	144
6.24.2.2	prt_packet_device_to_device	145
6.24.2.3	prt_packet_device_to_device_direct	146
6.24.2.4	prt_packet_device_to_host	146
6.24.2.5	prt_packet_host_to_device	147
6.24.2.6	prt_packet_new_device	148
6.24.2.7	prt_packet_new_host	149
6.24.2.8	prt_packet_release_device	150
6.24.2.9	prt_packet_release_host	151
6.24.2.10	prt_packet_resize_host	152
6.25	prt_packet.h File Reference	153
6.25.1	Detailed Description	154
6.25.2	Typedef Documentation	154
6.25.2.1	prt_packet_t	154

6.25.3	Function Documentation	155
6.25.3.1	prt_packet_device_mpi_to_host	155
6.25.3.2	prt_packet_device_to_device	155
6.25.3.3	prt_packet_device_to_device_direct	156
6.25.3.4	prt_packet_device_to_host	157
6.25.3.5	prt_packet_host_to_device	157
6.25.3.6	prt_packet_new_device	158
6.25.3.7	prt_packet_new_host	159
6.25.3.8	prt_packet_release_device	160
6.25.3.9	prt_packet_release_host	161
6.25.3.10	prt_packet_resize_host	162
6.26	prt_proxy.c File Reference	163
6.26.1	Detailed Description	164
6.26.2	Function Documentation	164
6.26.2.1	prt_proxy_cuda	164
6.26.2.2	prt_proxy_delete	165
6.26.2.3	prt_proxy_max_channel_size	166
6.26.2.4	prt_proxy_mpi	167
6.26.2.5	prt_proxy_new	168
6.26.2.6	prt_proxy_recv	169
6.26.2.7	prt_proxy_run	170
6.27	prt_proxy.h File Reference	171
6.27.1	Detailed Description	172
6.27.2	Typedef Documentation	172
6.27.2.1	prt_proxy_t	172
6.27.3	Function Documentation	172
6.27.3.1	prt_proxy_cuda	172
6.27.3.2	prt_proxy_delete	173
6.27.3.3	prt_proxy_max_channel_size	174
6.27.3.4	prt_proxy_mpi	175
6.27.3.5	prt_proxy_new	176
6.27.3.6	prt_proxy_recv	177
6.27.3.7	prt_proxy_run	178
6.28	prt_request.c File Reference	179
6.28.1	Detailed Description	179
6.28.2	Function Documentation	180
6.28.2.1	prt_request_cancel	180

6.28.2.2	prt_request_delete	180
6.28.2.3	prt_request_new	181
6.28.2.4	prt_request_rcv	181
6.28.2.5	prt_request_send	182
6.28.2.6	prt_request_test	183
6.29	prt_request.h File Reference	183
6.29.1	Detailed Description	185
6.29.2	Function Documentation	185
6.29.2.1	prt_request_cancel	185
6.29.2.2	prt_request_delete	185
6.29.2.3	prt_request_new	186
6.29.2.4	prt_request_rcv	186
6.29.2.5	prt_request_send	187
6.29.2.6	prt_request_test	188
6.30	prt_thread.c File Reference	188
6.30.1	Detailed Description	189
6.30.2	Function Documentation	189
6.30.2.1	prt_thread_delete	189
6.30.2.2	prt_thread_new	190
6.30.2.3	prt_thread_run	191
6.31	prt_thread.h File Reference	192
6.31.1	Detailed Description	193
6.31.2	Typedef Documentation	193
6.31.2.1	prt_thread_t	193
6.31.3	Function Documentation	194
6.31.3.1	prt_thread_delete	194
6.31.3.2	prt_thread_new	194
6.31.3.3	prt_thread_run	195
6.32	prt_transfer.c File Reference	196
6.32.1	Detailed Description	197
6.32.2	Function Documentation	197
6.32.2.1	prt_transfer_delete	197
6.32.2.2	prt_transfer_new	197
6.33	prt_transfer.h File Reference	198
6.33.1	Detailed Description	199
6.33.2	Function Documentation	199
6.33.2.1	prt_transfer_delete	199

6.34	prt_tuple.c File Reference	199
6.34.1	Detailed Description	200
6.34.2	Function Documentation	201
6.34.2.1	prt_tuple_cat	201
6.34.2.2	prt_tuple_compare	202
6.34.2.3	prt_tuple_copy	203
6.34.2.4	prt_tuple_delete	203
6.34.2.5	prt_tuple_equal	203
6.34.2.6	prt_tuple_hash	204
6.34.2.7	prt_tuple_len	205
6.34.2.8	prt_tuple_print	205
6.35	prt_tuple.h File Reference	206
6.35.1	Detailed Description	207
6.35.2	Function Documentation	207
6.35.2.1	prt_tuple_cat	207
6.35.2.2	prt_tuple_compare	208
6.35.2.3	prt_tuple_copy	208
6.35.2.4	prt_tuple_delete	209
6.35.2.5	prt_tuple_equal	209
6.35.2.6	prt_tuple_hash	209
6.35.2.7	prt_tuple_len	210
6.35.2.8	prt_tuple_print	210
6.36	prt_vdp.c File Reference	210
6.36.1	Detailed Description	212
6.36.2	Function Documentation	212
6.36.2.1	prt_vdp_annihilate	212
6.36.2.2	prt_vdp_delete	212
6.36.2.3	prt_vdp_ready	213
6.37	prt_vdp.h File Reference	214
6.37.1	Detailed Description	215
6.37.2	Function Documentation	216
6.37.2.1	prt_vdp_annihilate	216
6.37.2.2	prt_vdp_delete	217
6.37.2.3	prt_vdp_ready	218
6.38	prt_vsa.c File Reference	218
6.38.1	Detailed Description	220
6.38.2	Function Documentation	220

6.38.2.1	prt_tuple_equal	220
6.38.2.2	prt_tuple_hash	221
6.38.2.3	prt_vsa_channel_streams	221
6.38.2.4	prt_vsa_channel_tags	222
6.38.2.5	prt_vsa_devices_warmup	223
6.38.2.6	prt_vsa_vdp_merge_channels	223
6.38.2.7	prt_vsa_vdp_track_tags	224
6.39	prt_vsa.h File Reference	225
6.39.1	Detailed Description	226
6.39.2	Function Documentation	227
6.39.2.1	prt_vsa_channel_streams	227
6.39.2.2	prt_vsa_channel_tags	228
6.39.2.3	prt_vsa_devices_warmup	229
6.39.2.4	prt_vsa_vdp_merge_channels	229
6.39.2.5	prt_vsa_vdp_track_tags	230
6.40	svg_trace.c File Reference	231
6.40.1	Detailed Description	232
6.40.2	Function Documentation	232
6.40.2.1	get_time_of_day	232
6.40.2.2	svg_trace_init	232
6.40.2.3	svg_trace_memory_device	233
6.40.2.4	svg_trace_memory_host	233
6.40.2.5	svg_trace_start_cpu	234
6.40.2.6	svg_trace_start_dma	235
6.40.2.7	svg_trace_start_gpu	236
6.40.2.8	svg_trace_stop_cpu	236
6.40.2.9	svg_trace_stop_dma	237
6.40.2.10	svg_trace_stop_gpu	238
6.41	svg_trace.h File Reference	238
6.41.1	Detailed Description	241
6.41.2	Function Documentation	241
6.41.2.1	get_time_of_day	241
6.41.2.2	svg_trace_init	241
6.41.2.3	svg_trace_memory_device	242
6.41.2.4	svg_trace_memory_host	242
6.41.2.5	svg_trace_start_cpu	243
6.41.2.6	svg_trace_start_dma	244

6.41.2.7	svg_trace_start_gpu	244
6.41.2.8	svg_trace_stop_cpu	244
6.41.2.9	svg_trace_stop_dma	245
6.41.2.10	svg_trace_stop_gpu	246

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	73
icl_list.h	Dependency-free linked list	82
make.inc	??
mpi_stubs.c	Stubs for a no-MPI build	93
mpi_stubs.h	Stubs for a no-MPI build	95
prt.h	PULSAR Runtime (PRT)	96
prt_assert.c	PRT exception handling	98
prt_assert.h	PRT exception handling	100
prt_callback.c	PRT callback	103

prt_callback.h	
PRT callback	110
prt_channel.c	
PRT data channel	117
prt_channel.h	
PRT data channel	124
prt_config.c	
PRT configuration	132
prt_config.h	
PRT configuration	134
prt_device.c	
PRT device	136
prt_device.h	
PRT device	139
prt_packet.c	
PRT data packet	143
prt_packet.h	
PRT data packet	153
prt_proxy.c	
PRT communication proxy	163
prt_proxy.h	
PRT communication proxy	171
prt_request.c	
PRT communication request	179
prt_request.h	
PRT communication request	183
prt_thread.c	
PRT thread	188
prt_thread.h	
PRT thread	192
prt_transfer.c	
PRT local transfer	196
prt_transfer.h	
PRT local transfer	198
prt_tuple.c	
PRT tuple	199
prt_tuple.h	
PRT tuple	206
prt_vdp.c	
Virtual Data Processor	210
prt_vdp.h	
Virtual Data Processor	214
prt_vsa.c	
Virtual Systolic Array	218
prt_vsa.h	
Virtual Systolic Array	225
svg_trace.c	
SVG tracing	231
svg_trace.h	
SVG tracing	238

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 anywhere 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

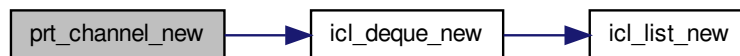
<i>size</i>	– The size of packets in bytes.
<i>src_tuple</i>	– The tuple of the source VDP.
<i>src_slot</i>	– The slot number in the source VDP.
<i>dst_tuple</i>	– The tuple of the destination VDP.
<i>dst_slot</i>	– The slot number in the destination VDP.

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

<i>len</i>	– The length of the tuple.
<i>...</i>	– 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*)

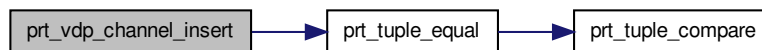
Inserts a new channel into a VDP.

Parameters

<i>vdp</i>	– The VDP to insert the channel into.
<i>channel</i>	– The channel to insert.
<i>direction</i>	– The direction of the channel.
<i>slot</i>	– 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

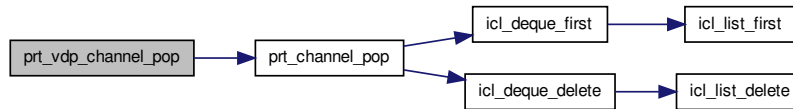
<i>vdp</i>	– The VDP fetching the packet.
<i>channel_num</i>	– The number of the channel to fetch from.

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)

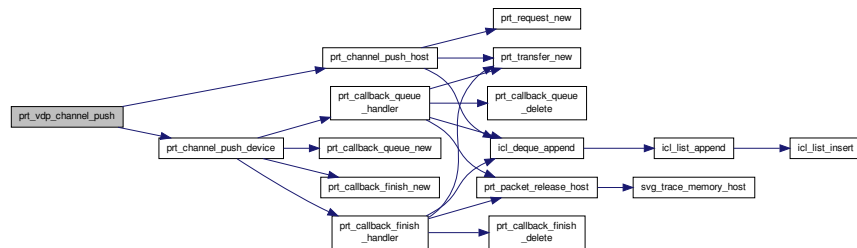
Pushes a packet in a channel.

Parameters

<i>vdp</i>	– The VDP pushing to the channel.
<i>channel_num</i>	– The number of the channel to push to.
<i>packet</i>	– 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

<i>tuple</i>	– A unique identifier of the VDP.
<i>counter</i>	– The number of times to fire the VDP.
<i>function</i>	– The function implementing the VDP's actions.
<i>local_store_size</i>	– The size of VDP's persistent local store in bytes.
<i>num_inputs</i>	– The number of input channels.
<i>num_outputs</i>	– The number of output channels.
<i>color</i>	– 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

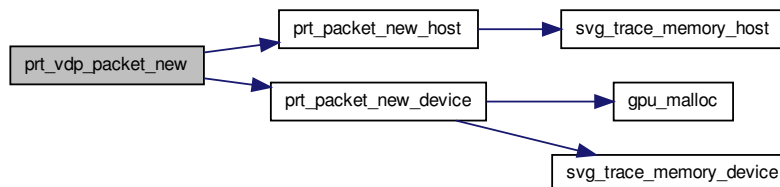
<i>vdp</i>	– The VDP creating the packet.
<i>size</i>	– The size of the packet in bytes.
<i>data</i>	– The data payload of the packet.

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)`

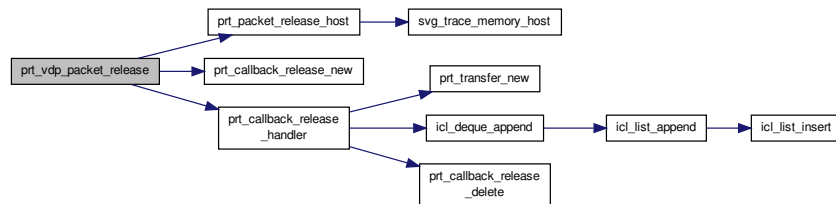
Releases a packet. Decrements the number of active references. Destroys the packet when the number of references goes down to zero. For device packets, puts a callback in the VDP's stream.

Parameters

<i>vdp</i>	– The VDP releasing the packet.
<i>packet</i>	– 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)

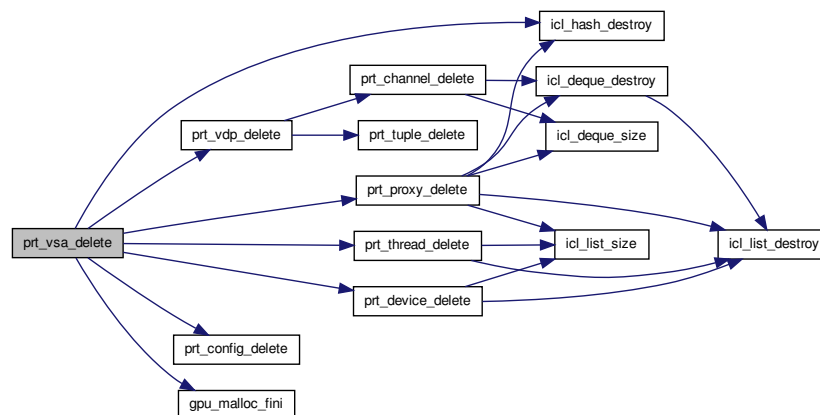
Destroys a VSA.

Parameters

<i>VSA</i>	– 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

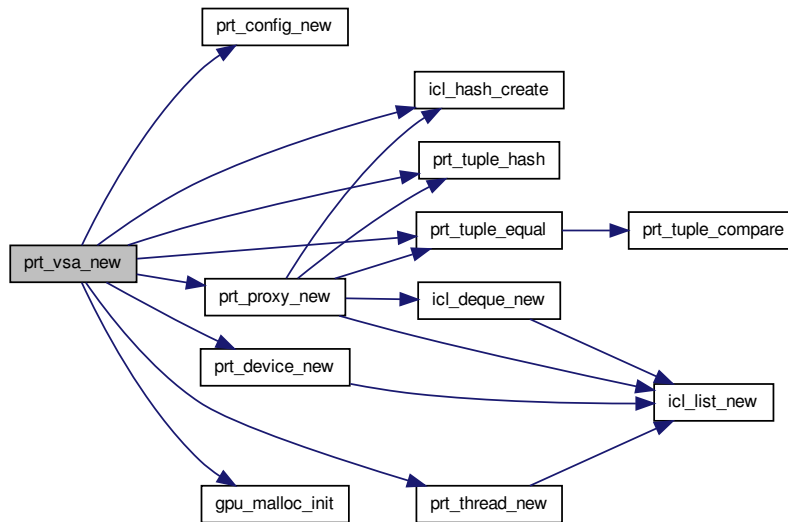
<i>num_threads</i>	– The number of local CPU threads.
<i>num_devices</i>	– The number of local GPU devices.
<i>global_store</i>	– VSA's global store, accessible to all VDPs.
<i>vdp_mapping</i>	– The function for mapping VDPs to cores and accelerators.

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

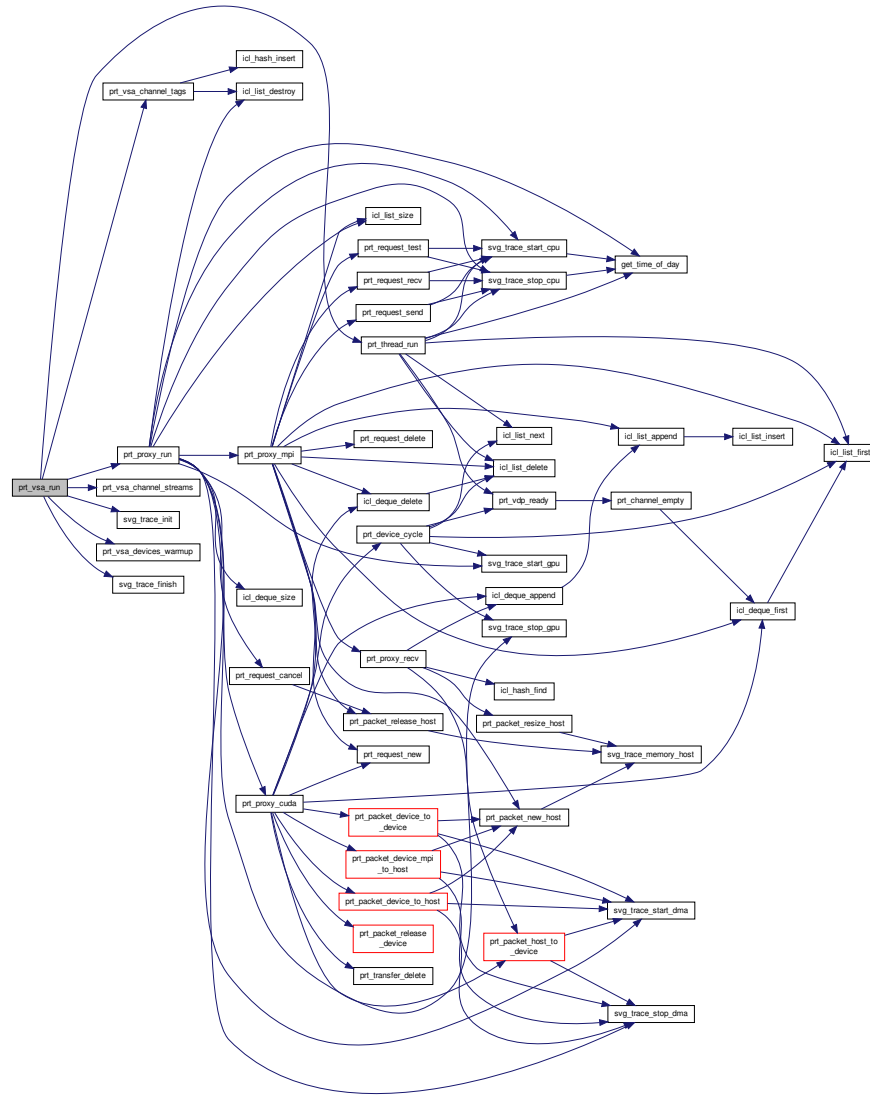
<i>vsa</i>	– The VSA to run.
------------	-------------------

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)

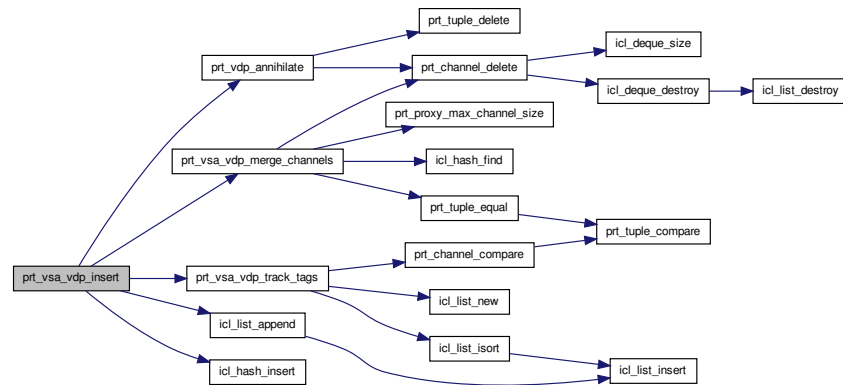
Inserts a VDP in a VSA. Destroys VDPs that do not belong to this node. Puts the VDP in the list of VDPs of the owner thread or device. Connects corresponding input and output channels of intra-node VDPs. Builds the list of channel connections to other nodes. For a device VDP, creates a CUDA stream with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

Parameters

<i>vsa</i>	– The VSA to insert into.
<i>vdp</i>	– 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

<code>vdp</code>	– The VDP deactivating the channel.
<code>channel_num</code>	– 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

<i>vdp</i>	– The VDP activating the channel.
<i>channel_num</i>	– 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

<i>vsa</i>	– The VSA to configure.
<i>param</i>	– The parameter to set.
<i>value</i>	– 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

<i>vsa</i>	– The VSA to set the function for.
<i>func</i>	– 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

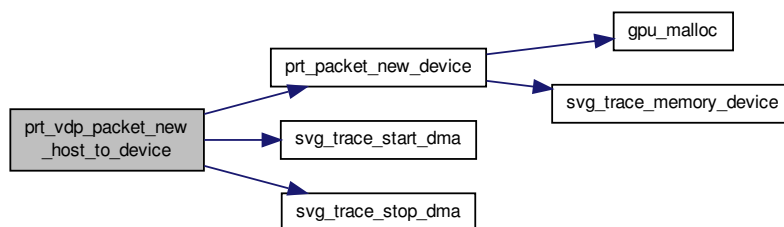
<i>vdp</i>	– The VDP creating the packet.
<i>size</i>	– Teh size of the packet in bytes.
<i>data</i>	– The data payload of the packet.

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

<i>vsa</i>	– The VSA to set the function for.
<i>func</i>	– The device (GPU) warmup function.

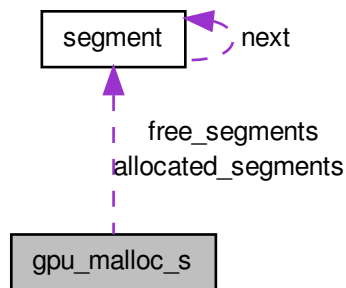
Definition at line 679 of file prt_vsa.c.

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

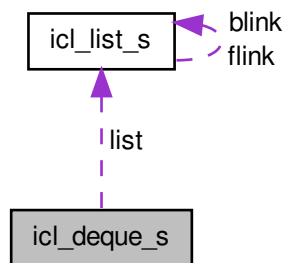
Definition at line 36 of file `gpu_malloc.h`.

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

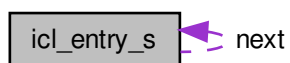
Definition at line 24 of file `icl_deque.h`.

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

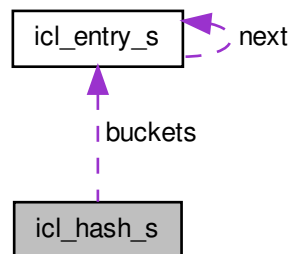
Definition at line 18 of file icl_hash.h.

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

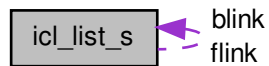
Definition at line 24 of file icl_hash.h.

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:

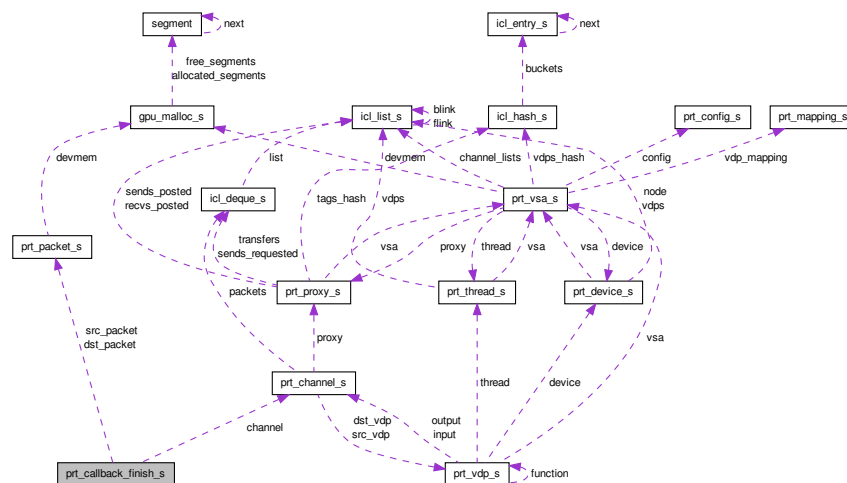
- `mpi_stubs.h`

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:

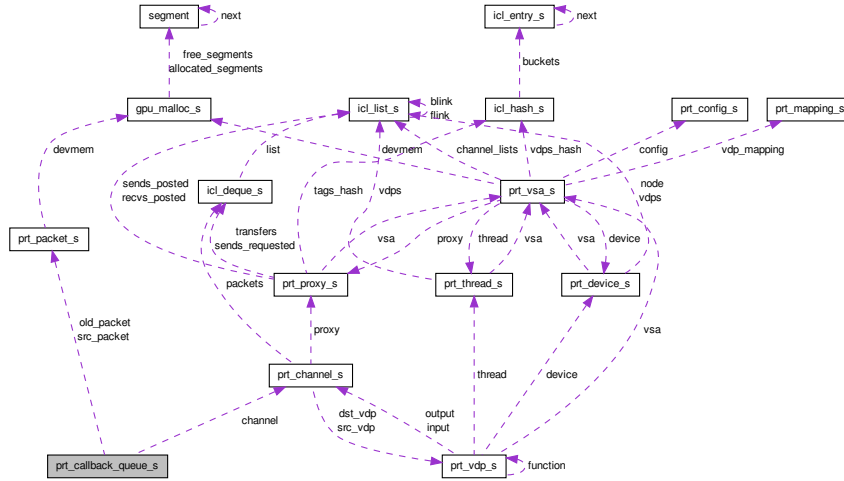
- `pvt_callback.h`

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:

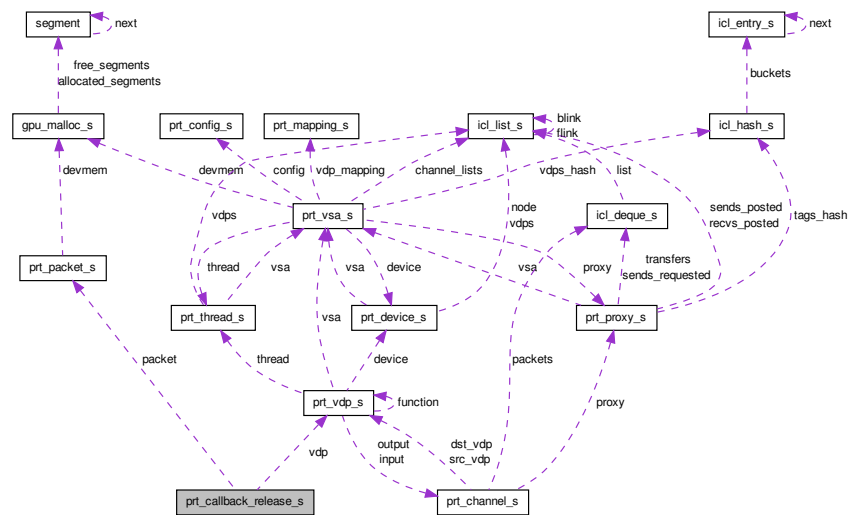
- `prt_callback.h`

5.10 prt_callback_release_s Struct Reference

Callback data for releasing a device packet.

```
#include <prt_callback.h>
```

```
graph LR
    segment[segment] -- next --> next[ ]
```



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

Callback data for releasing a device packet.

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

- `pvt_callback.h`

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.

Generated on Thu Dec 4 2014 14:16:56 for PULSAR by Doxygen

- [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:

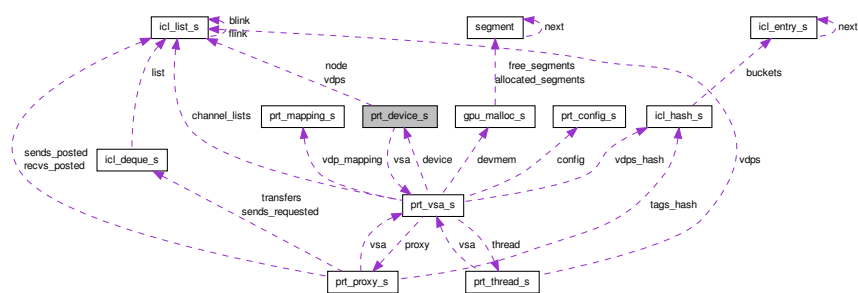
- [prt_config.h](#)

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` * **vdp**

- [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 declared 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:

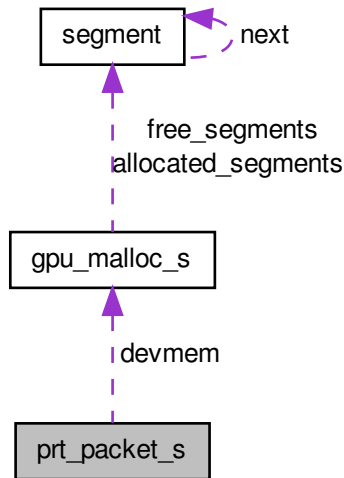
- [prt.h](#)

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:

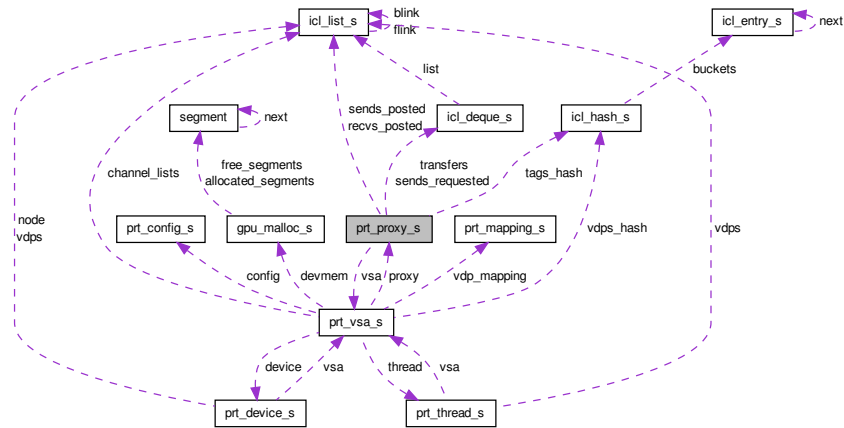
- [prt_packet.h](#)

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:

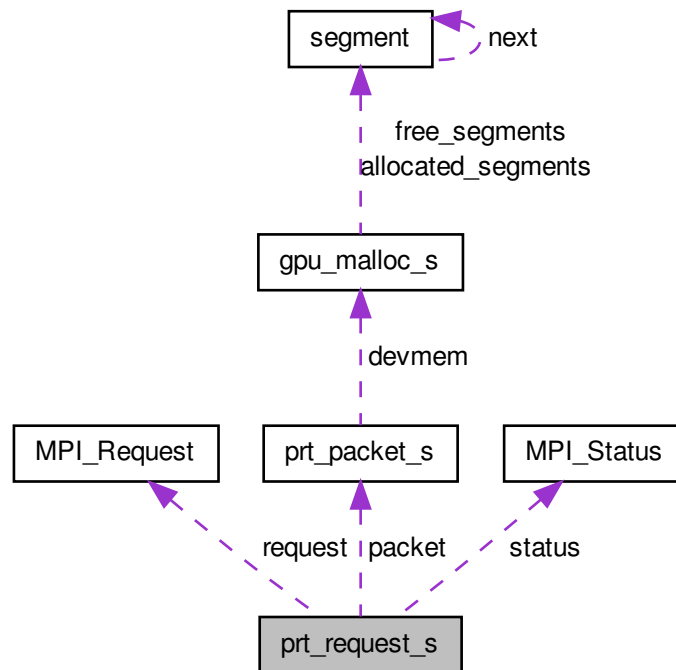
- [prt_proxy.h](#)

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:

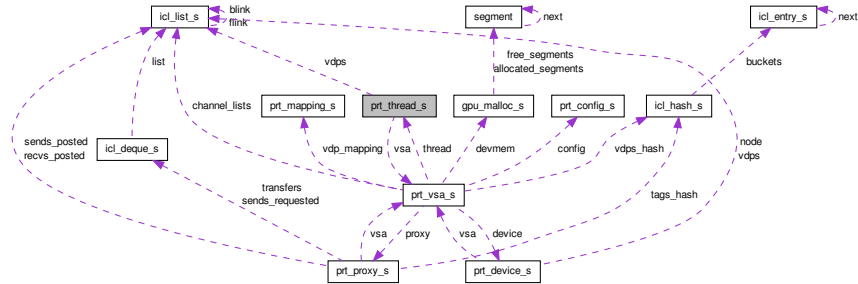
- `prt_request.h`

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 declared 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:

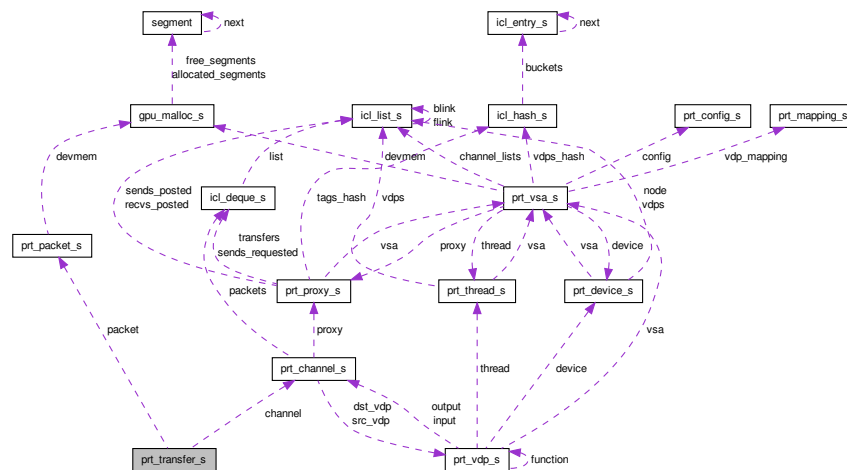
- `prt_thread.h`

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:

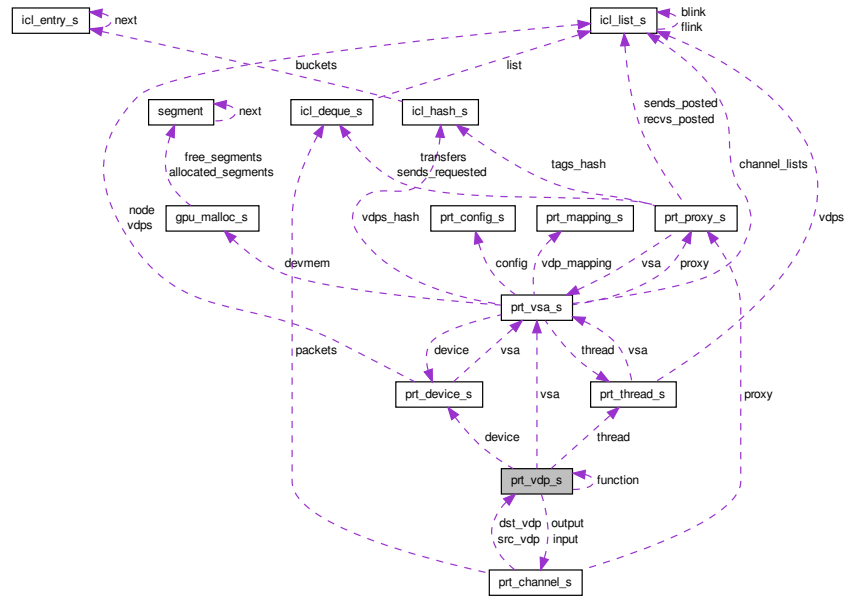
- `pvt_transfer.h`

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 <pvt_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:

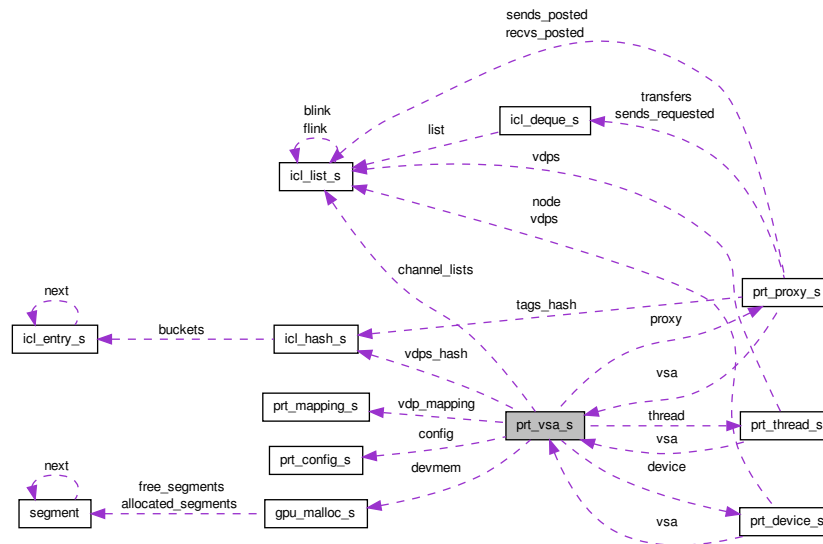
- [prt_vdp.h](#)

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.

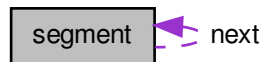
Definition at line 55 of file prt_vsa.h.

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

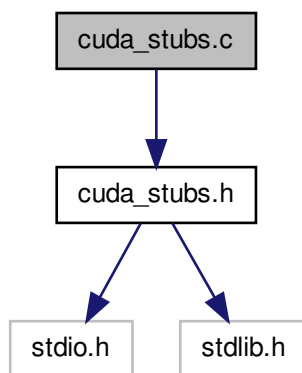
File Documentation

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 __cuda_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.

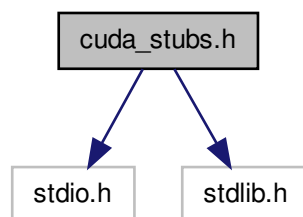
Definition in file [cuda_stubs.c](#).

6.2 cuda_stubs.h File Reference

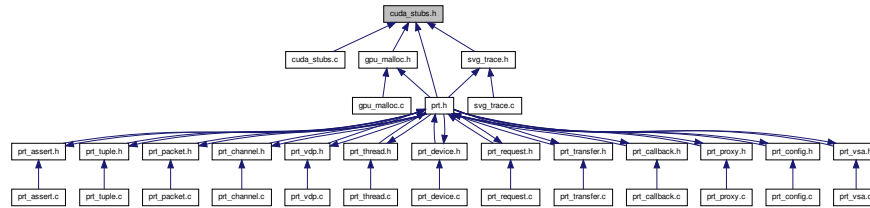
Stubs for a no-CUDA build.

```
#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.

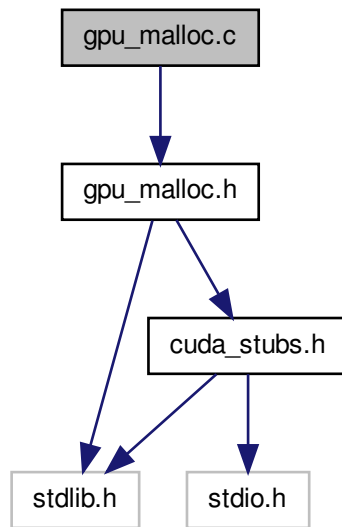
Definition in file [cuda_stubs.h](#).

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

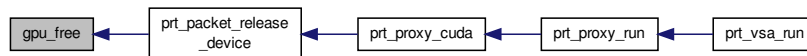
<i>gdata</i>	– The allocator to use.
<i>add</i>	– The pointer to the memory to free.

Return values

<i>0</i>	on success.
<i>-1</i>	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

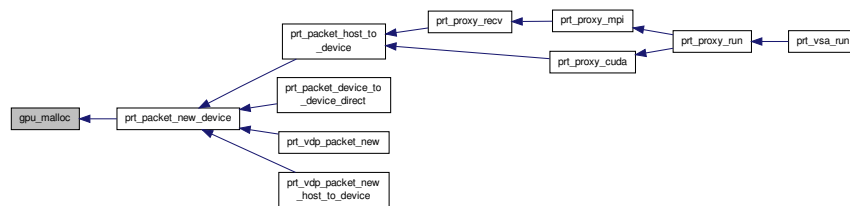
<i>gdata</i>	– The allocator to use.
<i>size</i>	– The size in bytes to allocate.

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

<i>gdata</i>	– The allocator to destroy.
--------------	-----------------------------

Return values

<i>0</i>	on success.
<i>-1</i>	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

<code>_max_segment</code>	– The maximum number of segments.
<code>_unit_size</code>	– 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:



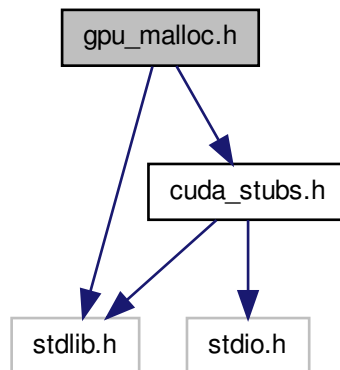
6.4 gpu_malloc.h File Reference

Simple device memory allocator.

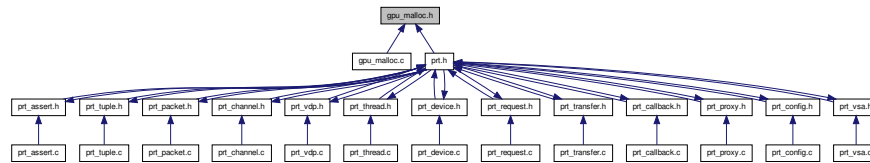
```
#include <stdlib.h>
```

```
#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

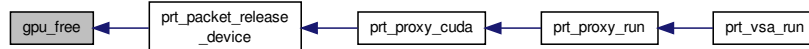
<i>gdata</i>	– The allocator to use.
<i>add</i>	– The pointer to the memory to free.

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

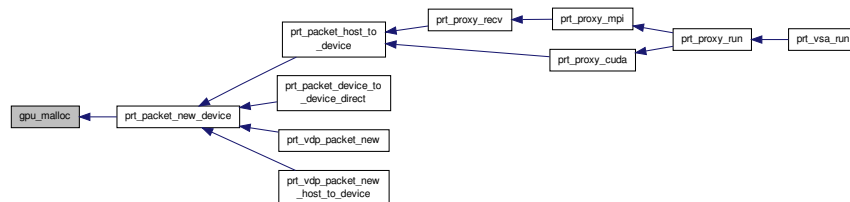
<i>gdata</i>	– The allocator to use.
<i>size</i>	– The size in bytes to allocate.

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

<i>gdata</i>	– 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

<i>_max_segment</i>	– The maximum number of segments.
<i>_unit_size</i>	– 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:

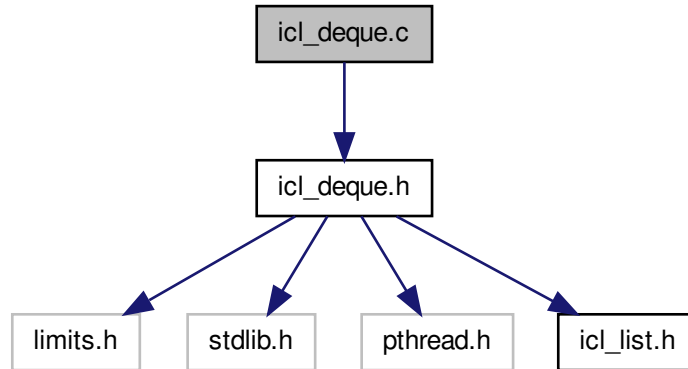


6.5 `icl_deque.c` File Reference

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

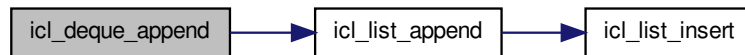
<i>deque</i>	– The deque to append to.
<i>data</i>	– The data to append.

Returns

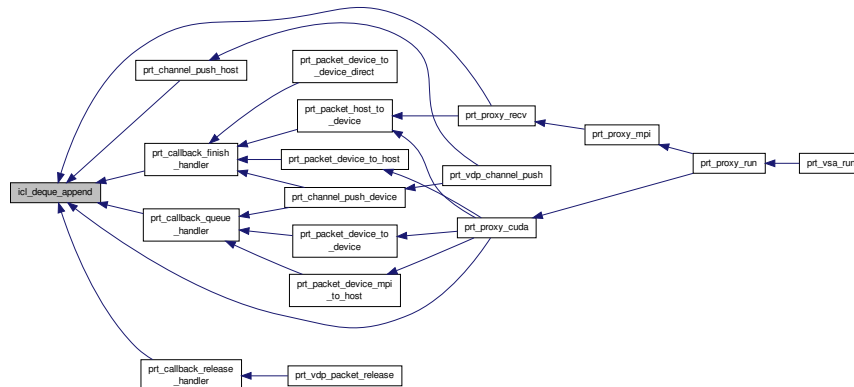
The new node. NULL on error.

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

<i>deque</i>	– The deque to delete from.
<i>node</i>	– The node to delete.
<i>free_func</i>	– The function that frees the node's data.

Return values

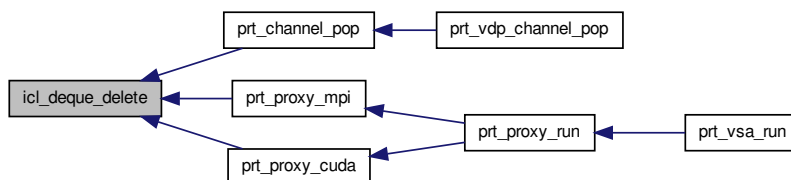
<i>0</i>	on success.
<i>-1</i>	on failure.

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

<i>deque</i>	– The deque to destroy.
--------------	-------------------------

<i>free_func</i>	– The function that frees the node's data.
------------------	--

Return values

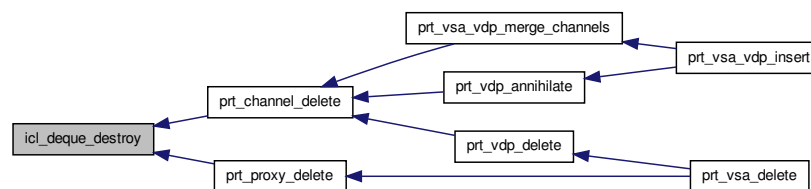
0	on success.
-1	on failure.

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

<i>deque</i>	– The deque to fetch from.
--------------	----------------------------

Returns

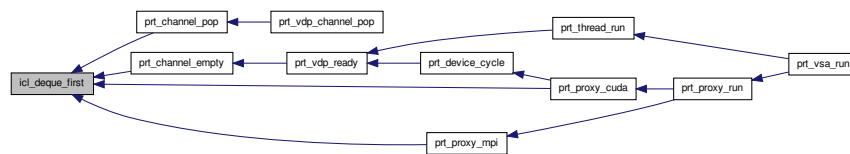
The node at the front of the deque. NULL if empty or error.

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

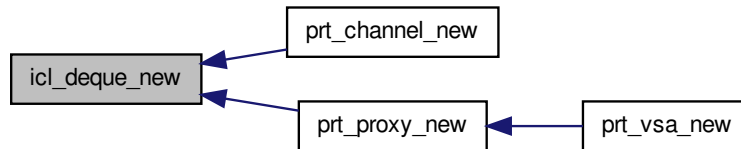
A new deque. NULL on error.

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

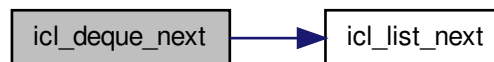
<i>deque</i>	– The deque to fetch from.
<i>node</i>	– The node current node.

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

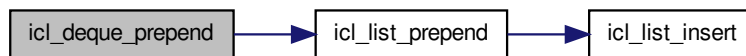
<i>deque</i>	– The deque to prepend to.
<i>data</i>	– The data to prepend.

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

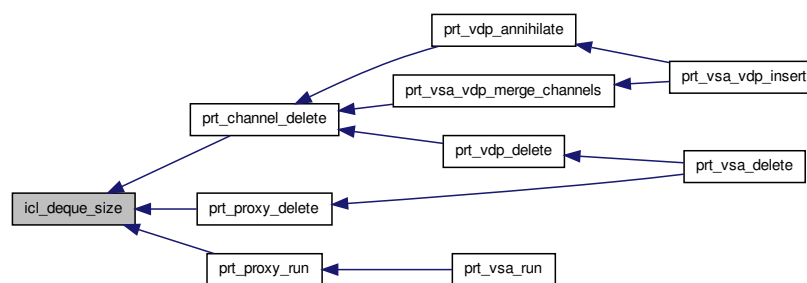
<i>deque</i>	– The deque to get size of.
--------------	-----------------------------

Returns

– The size of the deque. -1 on error.

Definition at line 189 of file icl_deque.c.

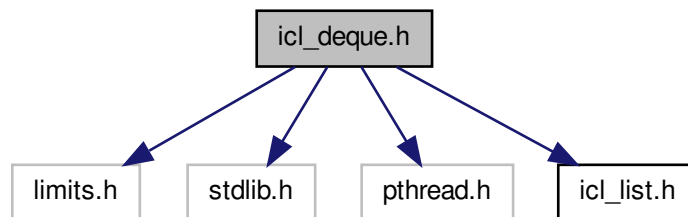
Here is the caller graph for this function:

**6.6 icl_deque.h File Reference**

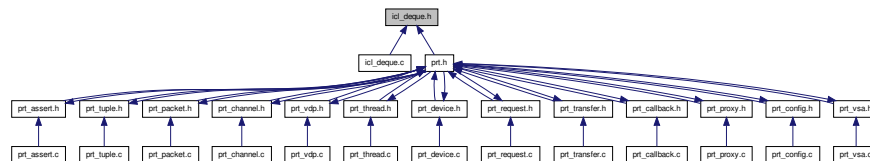
Thread-safe double-ended queue.

```
#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

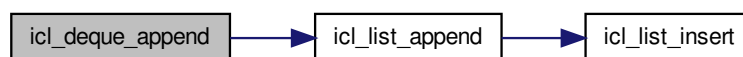
<i>deque</i>	– The deque to append to.
<i>data</i>	– The data to append.

Returns

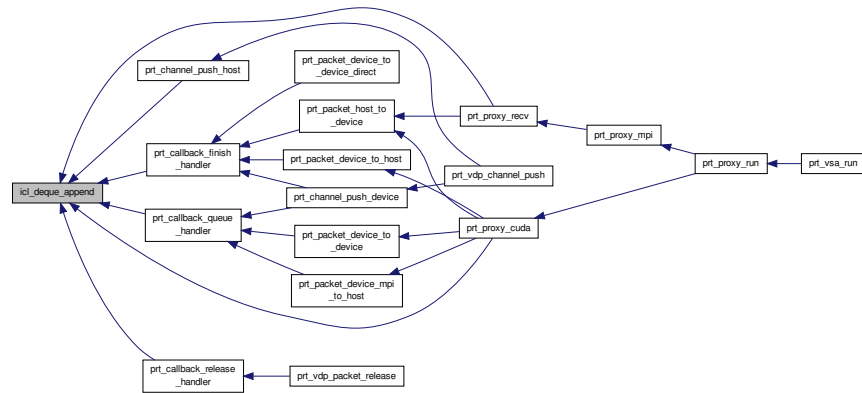
The new node. NULL on error.

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

<i>deque</i>	– The deque to delete from.
<i>node</i>	– The node to delete.
<i>free_func</i>	– The function that frees the node's data.

Return values

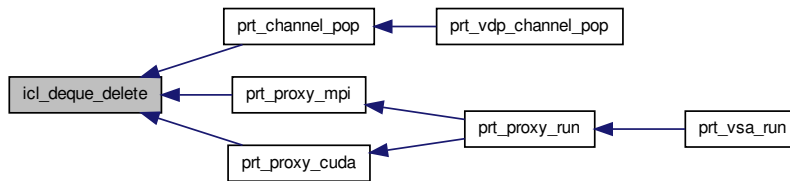
0	on success.
-1	on failure.

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

<i>deque</i>	– The deque to destroy.
<i>free_func</i>	– The function that frees the node's data.

Return values

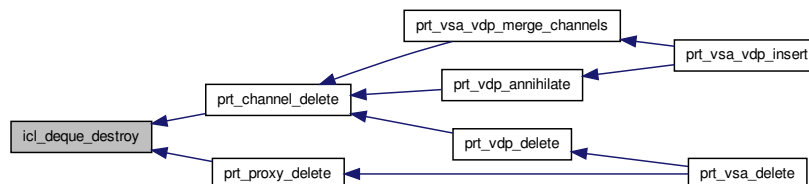
<i>0</i>	on success.
<i>-1</i>	on failure.

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

<i>deque</i>	– The deque to fetch from.
--------------	----------------------------

Returns

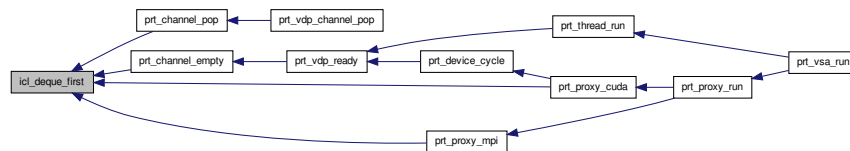
The node at the front of the deque. NULL if empty or error.

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

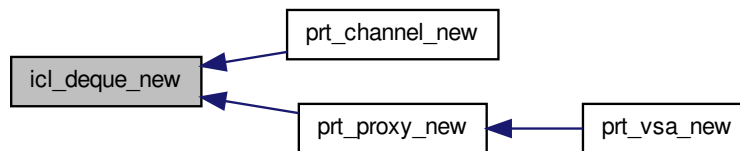
A new deque. NULL on error.

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

<i>deque</i>	– The deque to fetch from.
<i>node</i>	– 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

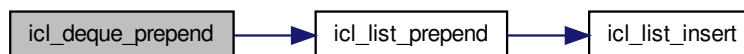
<i>deque</i>	– The deque to prepend to.
<i>data</i>	– The data to prepend.

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

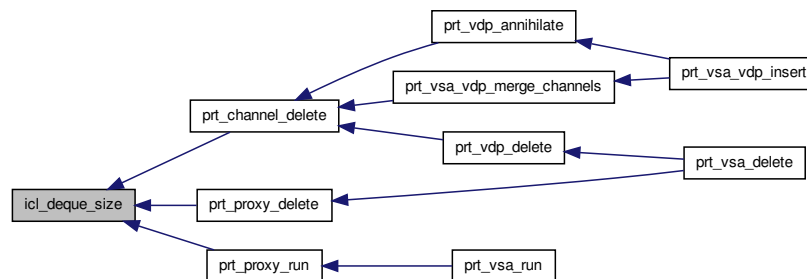
<i>deque</i>	– The deque to get size of.
--------------	-----------------------------

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:

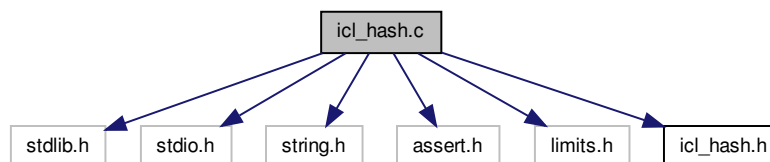


6.7 icl_hash.c File Reference

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

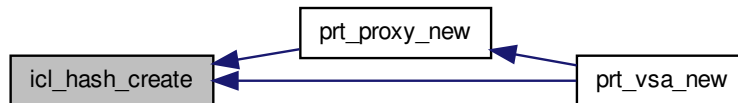
<i>nbuckets</i>	– The number of buckets to create.
<i>hash_function</i>	– The pointer to the hashing function.
<i>hash_key_compare</i>	– The pointer to the hash key comparison function.

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

<i>ht</i>	– The hash table.
<i>key</i>	– The key of the item to be deleted.
<i>free_key</i>	– The pointer to the function that frees the key.
<i>free_data</i>	– The pointer to the function that frees the data.

Return values

<i>0</i>	on success.
<i>-1</i>	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

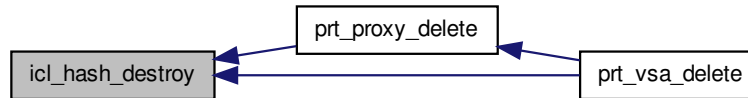
<i>ht</i>	– The hash table to destroy.
<i>free_key</i>	– The pointer to function that frees the keys.
<i>free_data</i>	– The pointer to function that frees the data.

Return values

<i>0</i>	on success.
<i>-1</i>	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

<i>stream</i>	– The file to dump the hash table to.
<i>ht</i>	– 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

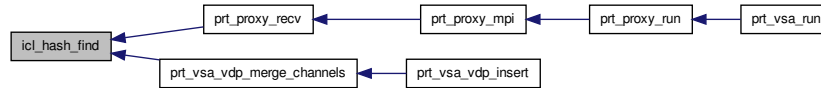
<i>ht</i>	– The hash table to be searched.
<i>key</i>	– The key of the item to search for.

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

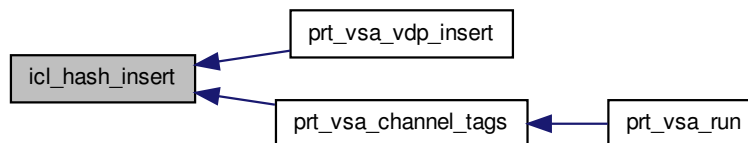
<i>ht</i>	– The hash table.
<i>key</i>	– The key of the new item.
<i>data</i>	– The pointer to the new item's data.

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

<i>ht</i>	– The hash table.
<i>key</i>	– The key of the new item.
<i>data</i>	– The pointer to the new item's data.
<i>olddata</i>	– The pointer to the old item's data (set upon return).

Returns

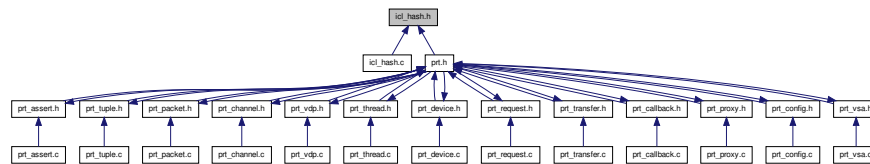
A pointer to the new item. NULL on error.

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

<i>nbuckets</i>	– The number of buckets to create.
<i>hash_function</i>	– The pointer to the hashing function.

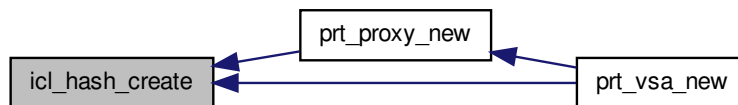
<i>hash_key_compare</i>	– The pointer to the hash key comparison function.
-------------------------	--

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

<i>ht</i>	– The hash table.
<i>key</i>	– The key of the item to be deleted.
<i>free_key</i>	– The pointer to the function that frees the key.
<i>free_data</i>	– The pointer to the function that frees the data.

Return values

<i>0</i>	on success.
<i>-1</i>	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

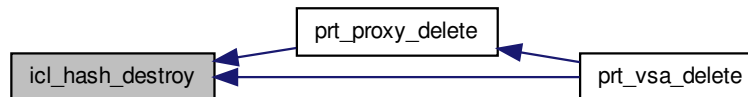
<i>ht</i>	– The hash table to destroy.
<i>free_key</i>	– The pointer to function that frees the keys.
<i>free_data</i>	– The pointer to function that frees the data.

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

<i>stream</i>	– The file to dump the hash table to.
<i>ht</i>	– 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

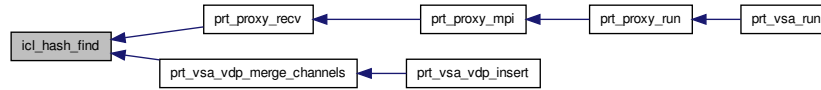
<i>ht</i>	– The hash table to be searched.
<i>key</i>	– The key of the item to search for.

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

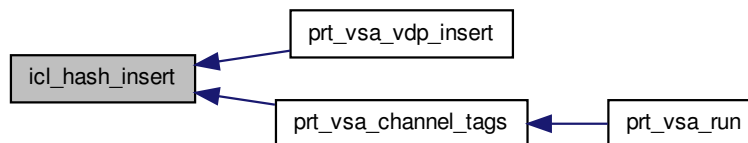
<i>ht</i>	– The hash table.
<i>key</i>	– The key of the new item.
<i>data</i>	– The pointer to the new item's data.

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

<i>ht</i>	– The hash table.
<i>key</i>	– The key of the new item.
<i>data</i>	– The pointer to the new item's data.
<i>olddata</i>	– The pointer to the old item's data (set upon return).

Returns

A pointer to the new item. NULL on error.

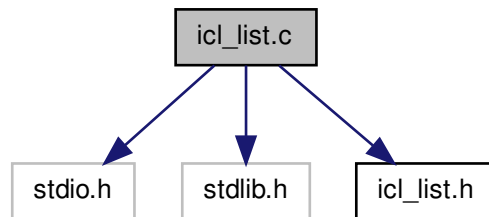
Definition at line 174 of file icl_hash.c.

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

<i>head</i>	– The linked list.
<i>data</i>	– The data to be inserted.

Returns

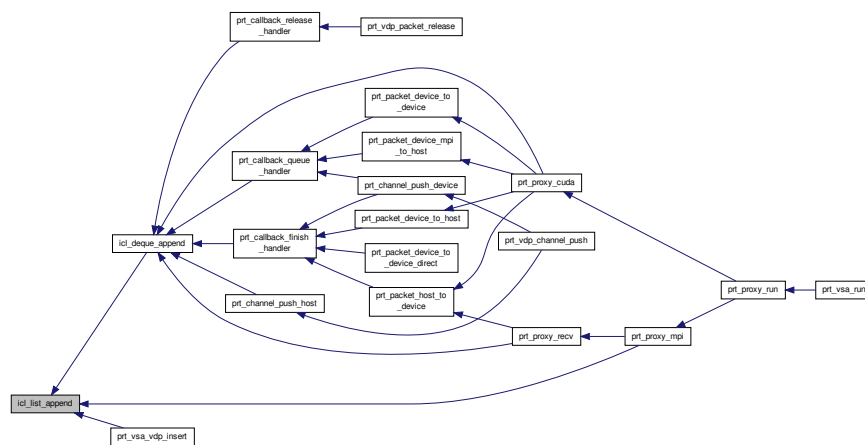
A pointer to the new node. NULL on error.

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

<i>head1</i>	– The first linked list.
<i>head2</i>	– 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

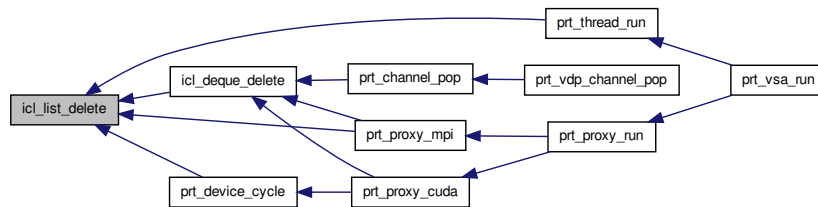
<i>head</i>	– The linked list containing the node to be deleted.
<i>pos</i>	– The node to be deleted.
<i>free_function</i>	– The function that frees the node's data.

Return values

<i>0</i>	on success.
<i>-1</i>	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

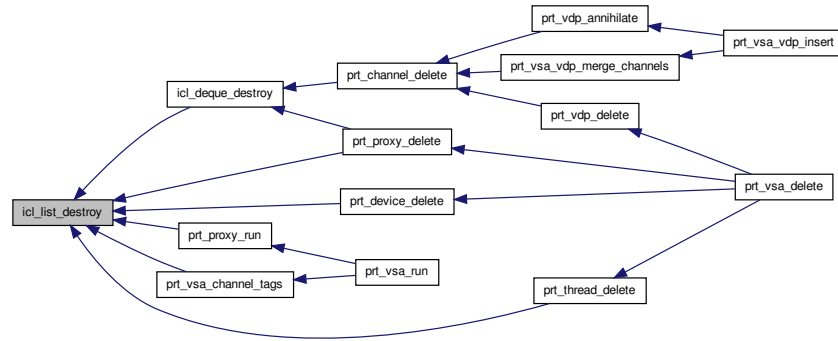
<i>head</i>	– The linked list to be destroyed.
<i>free_function</i>	– The function that frees the node's data.

Return values

<i>0</i>	on success.
<i>-1</i>	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

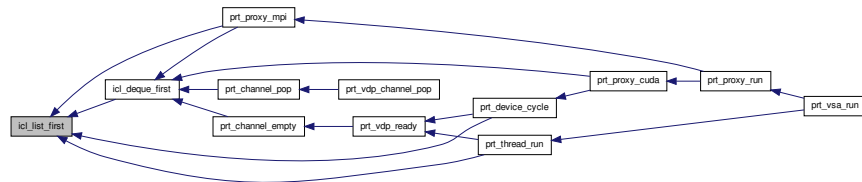
<i>head</i>	– The linked list.
-------------	--------------------

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

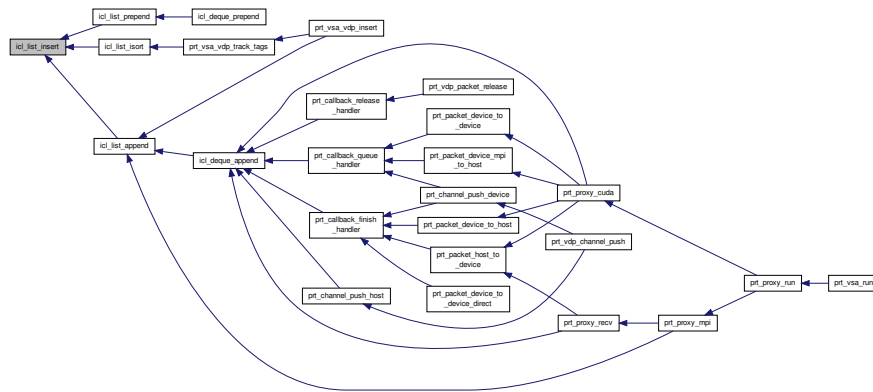
<i>head</i>	– The linked list.
<i>pos</i>	– The insertion position (the node to append to).
<i>data</i>	– The pointer to the data to be inserted.

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

<i>head</i>	– The linked list.
<i>data</i>	– The data to be inserted.
<i>compare</i>	– The function that compares the data items.

Returns

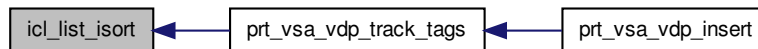
A pointer to the new node. NULL on error.

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

<i>head</i>	– 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 ()

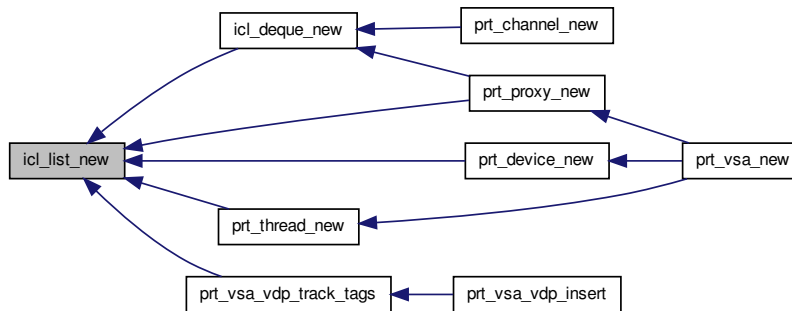
Creates a new linked list.

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

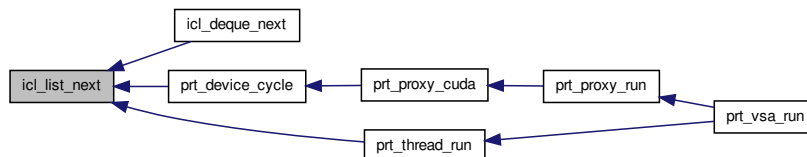
<i>head</i>	– The list containing the specified node.
<i>pos</i>	– The node whose successor should be returned.

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

<i>head</i>	– The linked list.
<i>data</i>	– 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

<i>head</i>	– The list containing the specified node.
<i>pos</i>	– 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

<i>head</i>	– The linked list.
<i>data</i>	– The data to be found.
<i>compare</i>	– 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

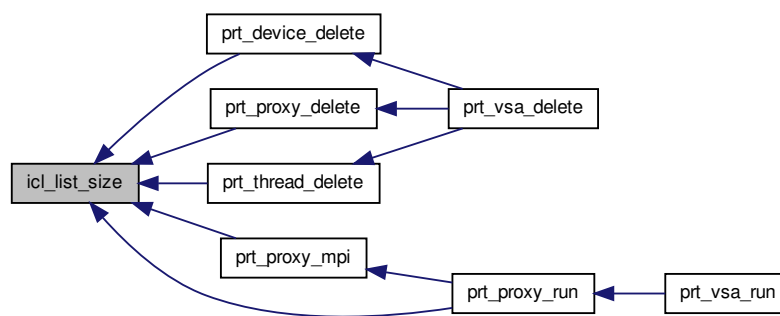
<i>head</i>	– The linked list.
-------------	--------------------

Returns

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

Definition at line 200 of file icl_list.c.

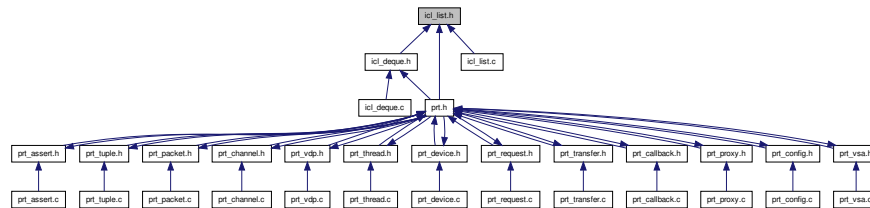
Here is the caller graph for this function:



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

<i>head</i>	– The linked list.
<i>data</i>	– The data to be inserted.

Returns

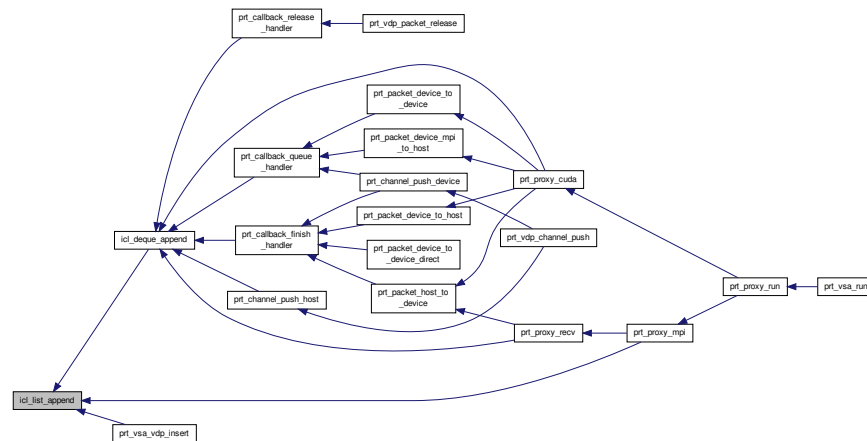
A pointer to the new node. NULL on error.

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

<i>head1</i>	– The first linked list.
<i>head2</i>	– 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

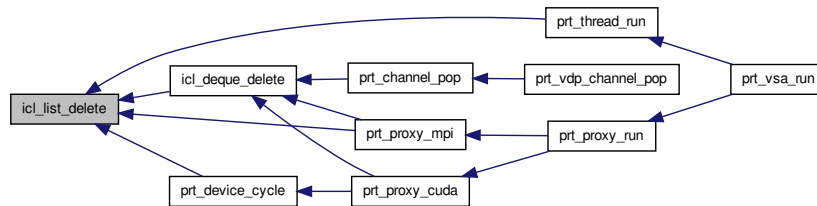
<i>head</i>	– The linked list containing the node to be deleted.
<i>pos</i>	– The node to be deleted.
<i>free_function</i>	– The function that frees the node's data.

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

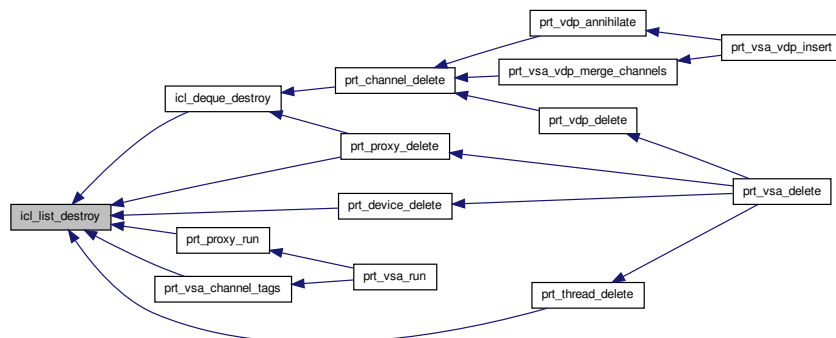
<i>head</i>	– The linked list to be destroyed.
<i>free_function</i>	– The function that frees the node's data.

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

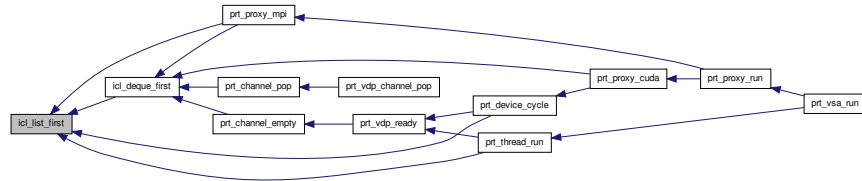
<i>head</i>	– The linked list.
-------------	--------------------

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

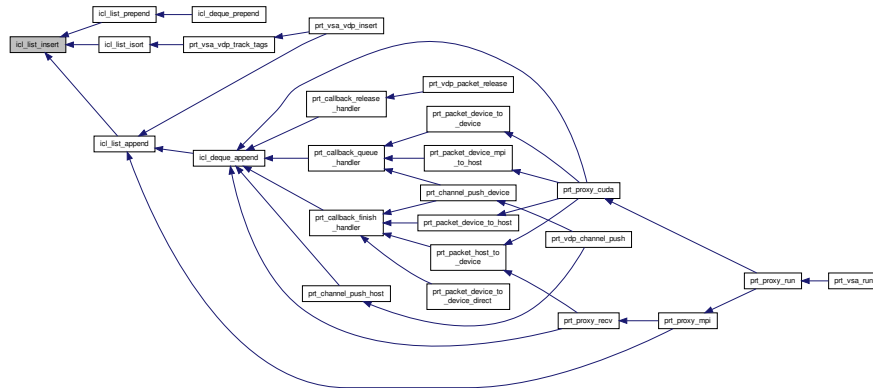
<i>head</i>	– The linked list.
<i>pos</i>	– The insertion position (the node to append to).
<i>data</i>	– The pointer to the data to be inserted.

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

<i>head</i>	– The linked list.
<i>data</i>	– The data to be inserted.
<i>compare</i>	– The function that compares the data items.

Returns

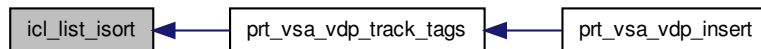
A pointer to the new node. NULL on error.

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

<i>head</i>	– 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 ()`

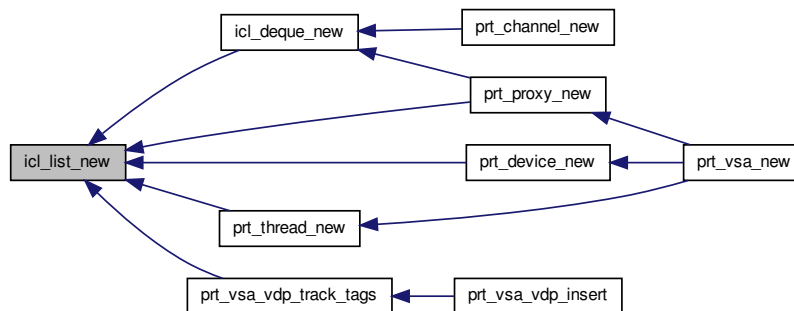
Creates a new linked list.

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

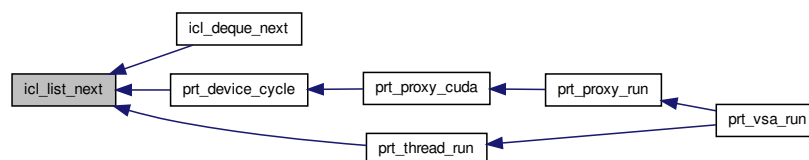
<i>head</i>	– The list containing the specified node.
<i>pos</i>	– The node whose successor should be returned.

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

<i>head</i>	– The linked list.
<i>data</i>	– 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

<i>head</i>	– The list containing the specified node.
<i>pos</i>	– 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

<i>head</i>	– The linked list.
<i>data</i>	– The data to be found.
<i>compare</i>	– 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

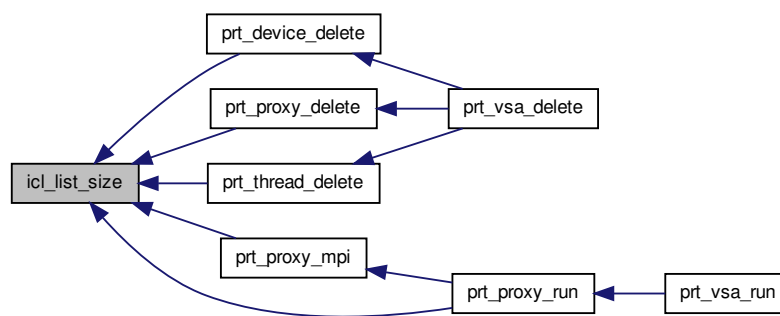
<i>head</i>	– The linked list.
-------------	--------------------

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:

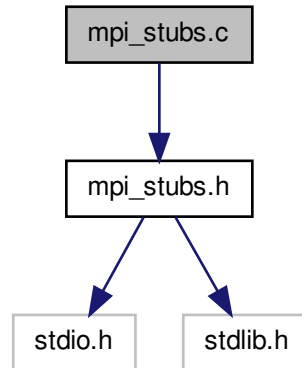


6.11 mpi_stubs.c File Reference

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.

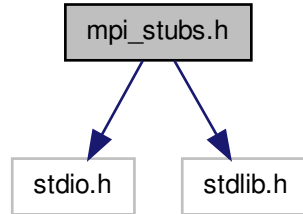
Definition in file [mpi_stubs.c](#).

6.12 mpi_stubs.h File Reference

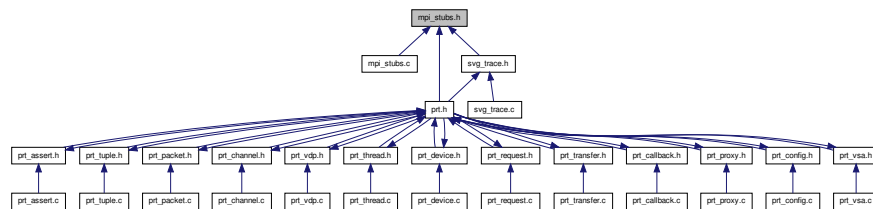
Stubs for a no-MPI build.

```
#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

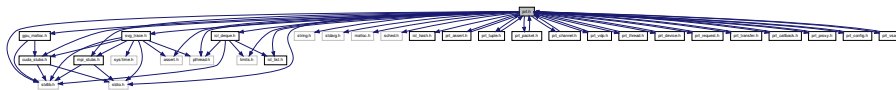
PULSAR Runtime (PRT)

```

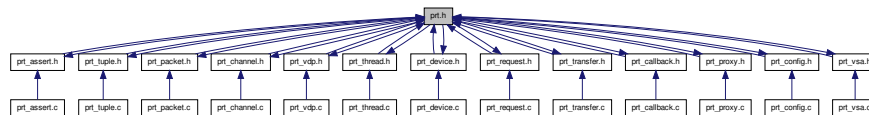
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <string.h>
#include <stdarg.h>
#include <limits.h>
#include <malloc.h>
#include <sched.h>
#include <pthread.h>
#include "mpi_stubs.h"
#include "cuda_stubs.h"
#include "gpu_malloc.h"
#include "icl_list.h"
#include "icl_hash.h"
#include "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

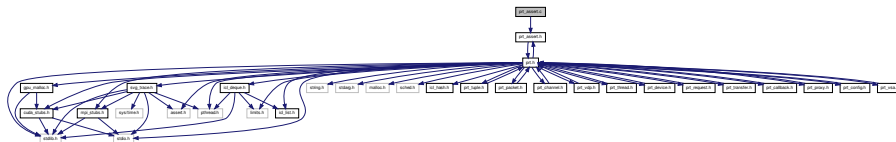
Definition in file [prt.h](#).

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

<i>cond</i>	– The condition.
<i>msg</i>	– The error message.
<i>line</i>	– The line number.
<i>file</i>	– 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

<i>msg</i>	– The error message.
<i>line</i>	– The line number.
<i>file</i>	– 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

<i>msg</i>	– The warning message.
<i>line</i>	– The line number.
<i>file</i>	– The name of the source file.

Definition at line 52 of file prt_assert.c.

6.15 prt_assert.h File Reference

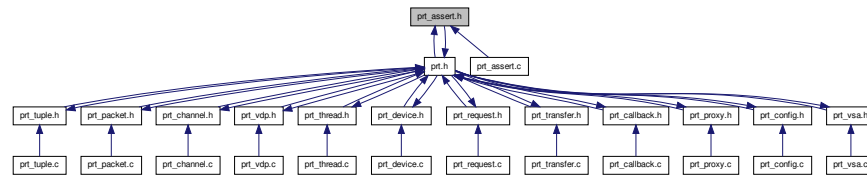
PRT exception handling.

```
#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

<i>cond</i>	– The condition.
<i>msg</i>	– The error message.
<i>line</i>	– The line number.
<i>file</i>	– 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

<i>msg</i>	– The error message.
<i>line</i>	– The line number.
<i>file</i>	– 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

<i>msg</i>	– The warning message.
<i>line</i>	– The line number.
<i>file</i>	– The name of the source file.

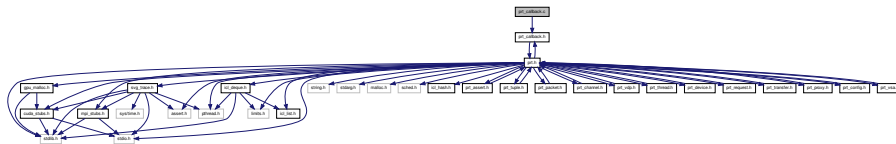
Definition at line 52 of file prt_assert.c.

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 requests 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

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

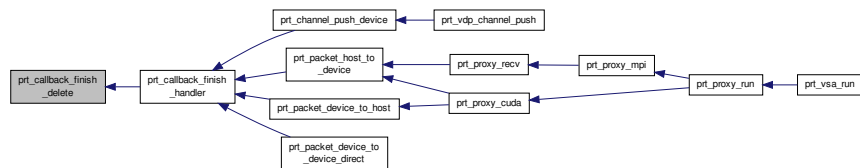
Destroys a callback data structure. This is for the callback that completes a local transfer.

Parameters

<i>callback</i>	– 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 * *cbck*)

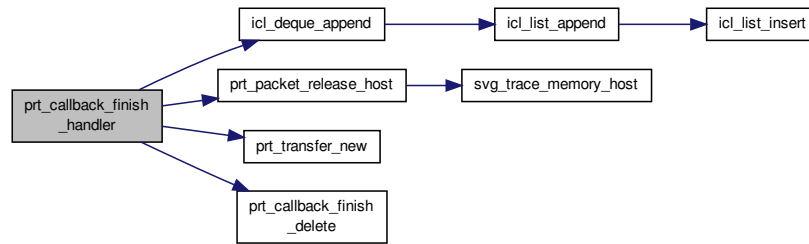
Finishes a local transfer. Puts the packet in the channel after a local transfer finishes. Services host-to-device and device-to-host transfers.

Parameters

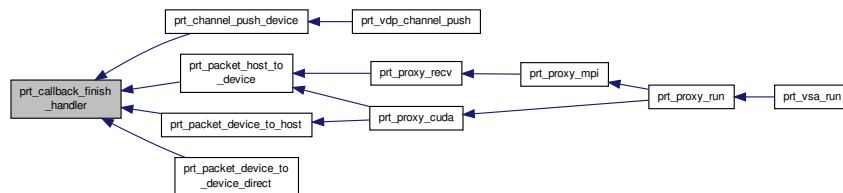
<i>stream</i>	– The callback's stream.
<i>status</i>	– The stream's status.
<i>cbck</i>	– The callback data.

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

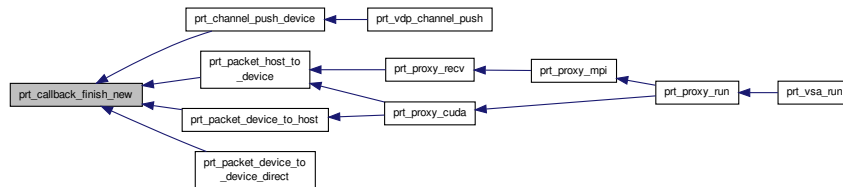
<i>src_packet</i>	– The packet to release when the transfer completes.
<i>dst_packet</i>	– The packet to place in the channel when the transfer completes.
<i>channel</i>	– The channel to insert the packet into.

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)

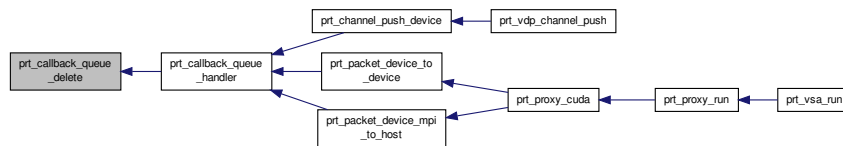
Destroys a callback data structure. This is for the callback that queues a local transfer.

Parameters

<i>callback</i>	– 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)

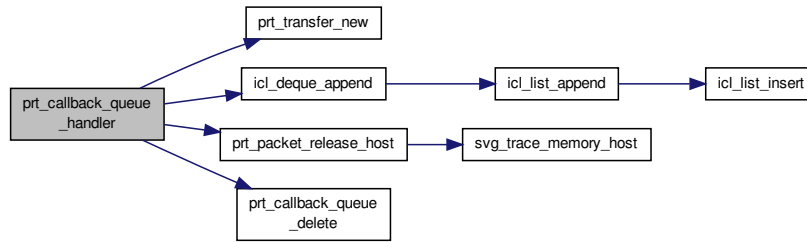
Queues a local transfer request. Services device-to-device requests and MPI requests from a device.

Parameters

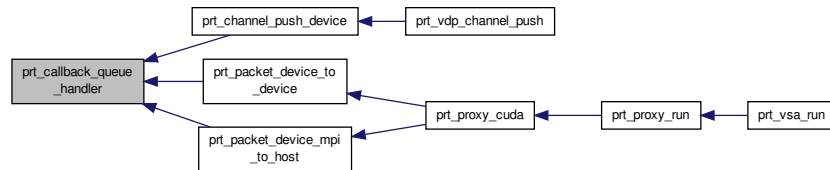
<i>stream</i>	– The callback's stream.
<i>status</i>	– The stream's status.
<i>clbck</i>	– The callback data.

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

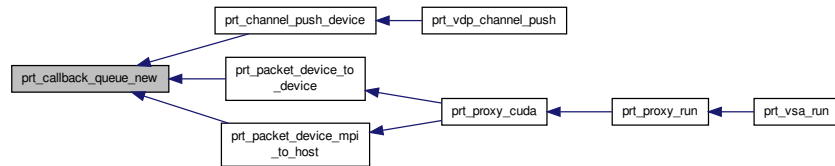
<i>old_packet</i>	– The packet to release when the transfer completes.
<i>src_packet</i>	– The packet to use for the followup transfer request.
<i>channel</i>	– The channel to use for the followup transfer request.
<i>direction</i>	– The direction of the followup transfer request.

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*)

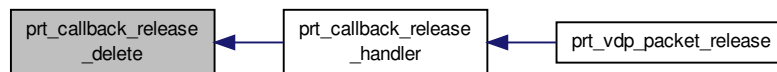
Destroys a callback data structure. This is for the callback that releases a device packet.

Parameters

<i>callback</i>	– 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 * *cbck*)

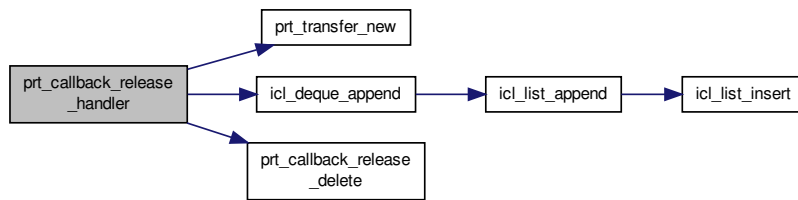
Releases a device packet.

Parameters

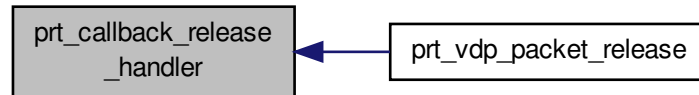
<i>stream</i>	– The callback's stream.
<i>status</i>	– The stream's status.
<i>cbck</i>	– The callback data.

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

<i>vdp</i>	– The VDP releasing the packet.
<i>packet</i>	– 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:

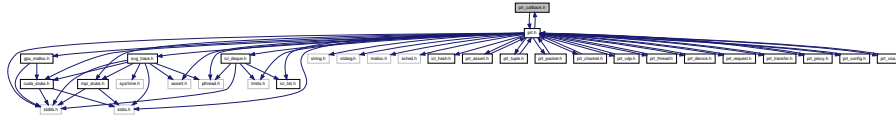


6.17 prt_callback.h File Reference

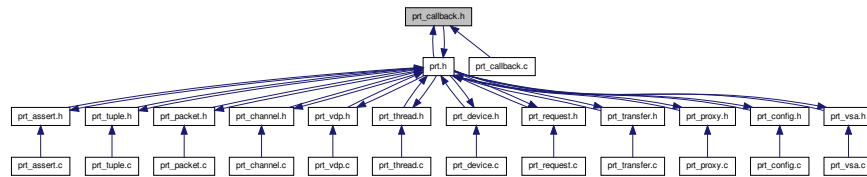
PRT callback.

```
#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 requests 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)

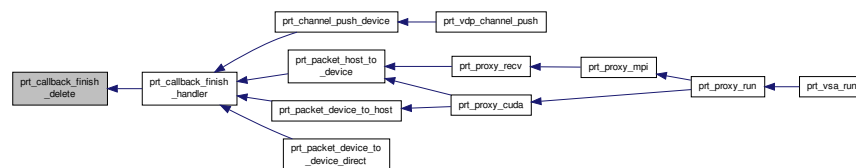
Destroys a callback data structure. This is for the callback that completes a local transfer.

Parameters

<i>callback</i>	– 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 * *cbck*)

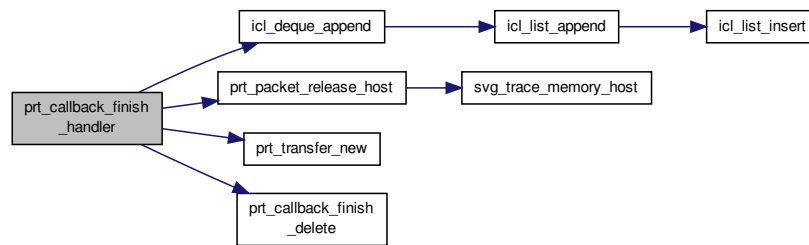
Finishes a local transfer. Puts the packet in the channel after a local transfer finishes. Services host-to-device and device-to-host transfers.

Parameters

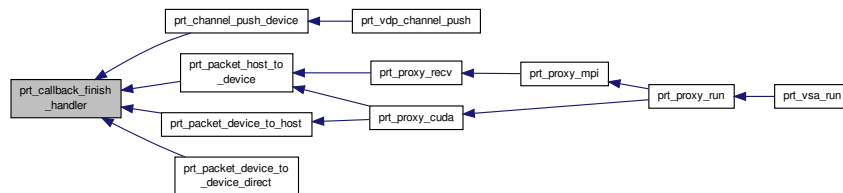
<i>stream</i>	– The callback's stream.
<i>status</i>	– The stream's status.
<i>cbck</i>	– The callback data.

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

<i>src_packet</i>	– The packet to release when the transfer completes.
<i>dst_packet</i>	– The packet to place in the channel when the transfer completes.

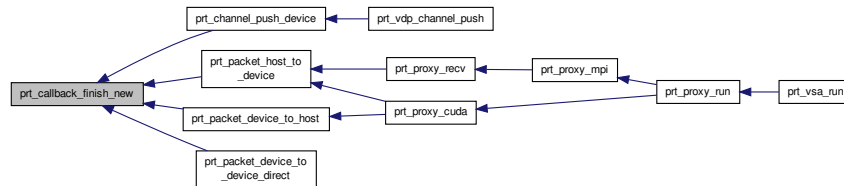
<i>channel</i>	– The channel to insert the packet into.
----------------	--

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)

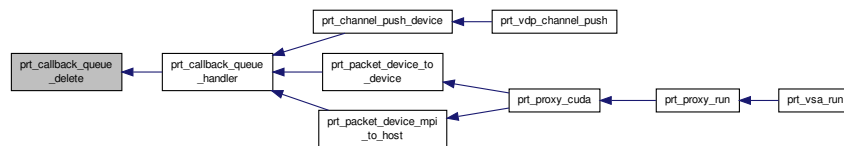
Destroys a callback data structure. This is for the callback that queues a local transfer.

Parameters

<i>callback</i>	– 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 requests and MPI requests from a device.

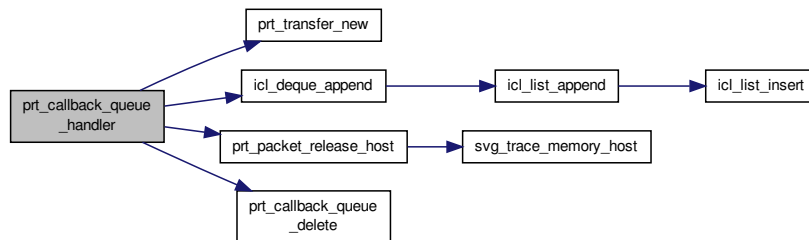
Parameters

<i>stream</i>	– The callback's stream.
---------------	--------------------------

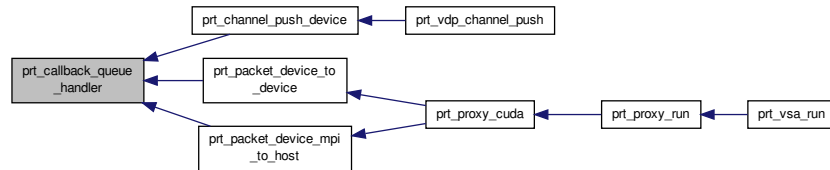
<i>status</i>	– The stream's status.
<i>clbck</i>	– The callback data.

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

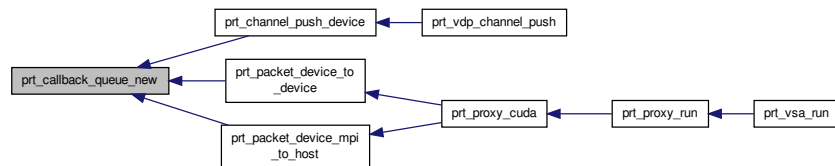
<i>old_packet</i>	– The packet to release when the transfer completes.
<i>src_packet</i>	– The packet to use for the followup transfer request.
<i>channel</i>	– The channel to use for the followup transfer request.
<i>direction</i>	– The direction of the followup transfer request.

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)

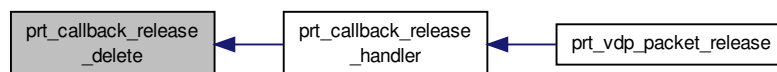
Destroys a callback data structure. This is for the callback that releases a device packet.

Parameters

<i>callback</i>	– 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)

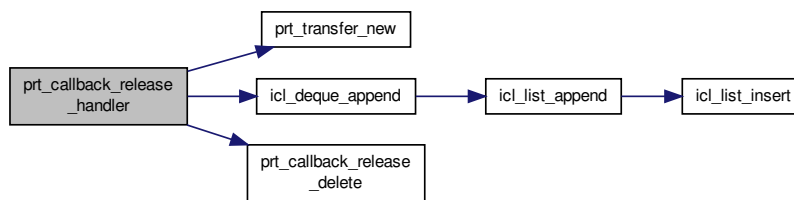
Releases a device packet.

Parameters

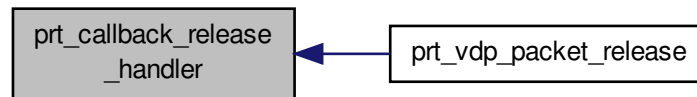
<i>stream</i>	– The callback's stream.
<i>status</i>	– The stream's status.
<i>clbck</i>	– The callback data.

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

<i>vdp</i>	– The VDP releasing the packet.
<i>packet</i>	– 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:

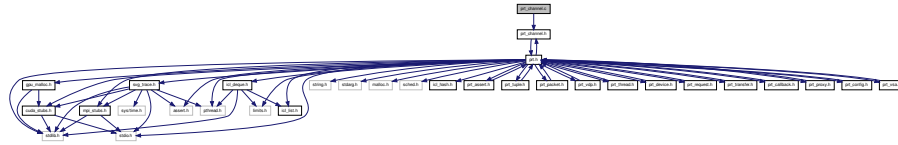


6.18 prt_channel.c File Reference

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 packet 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

<i>channel1</i>	– The first channel.
<i>channel2</i>	– The second channel.

Return values

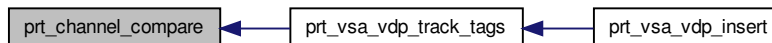
<i>-1</i>	channel1 is less than channel2.
<i>0</i>	channel1 is equal to channel2.
<i>1</i>	channel1 is greater than channel2.

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*)

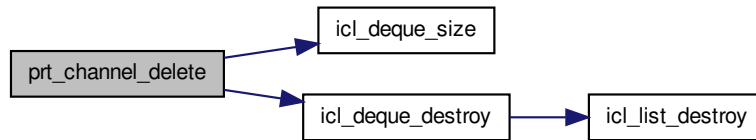
Destroys a channel.

Parameters

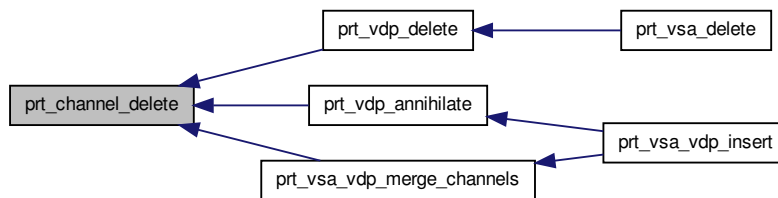
<i>channel</i>	– The channel to destroy.
----------------	---------------------------

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

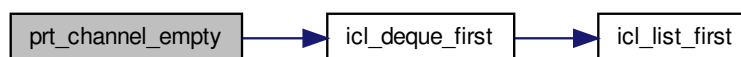
<i>channel</i>	– The channel to check.
----------------	-------------------------

Return values

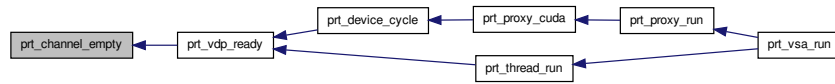
1	if the channel is empty.
0	if the channel is not empty.

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

<i>channel</i>	– 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

<i>channel</i>	- 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 packet from a channel. Does not decrement the number of active references. The packet leaves the channel, but enters the VDP.

Parameters

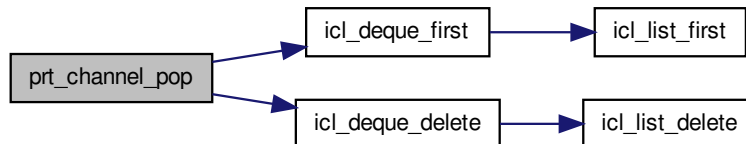
<i>channel</i>	– The channel to fetch the packet from.
----------------	---

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.

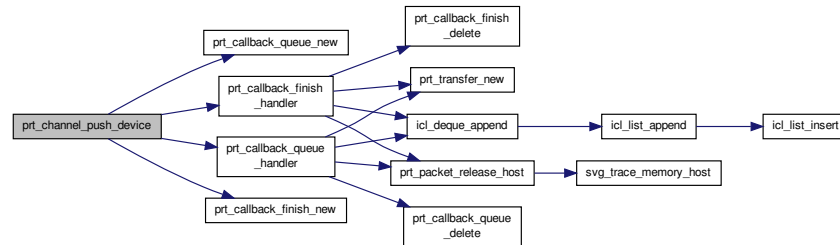
There is no need to set the device here. This function is called by `prt_vdp_channel_push`, which is called from VDP code, where the device is already set.

Parameters

<i>vdp</i>	– The device VDP sending the packet.
<i>channel</i>	– The channel to send the packet to.
<i>packet</i>	– 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*)

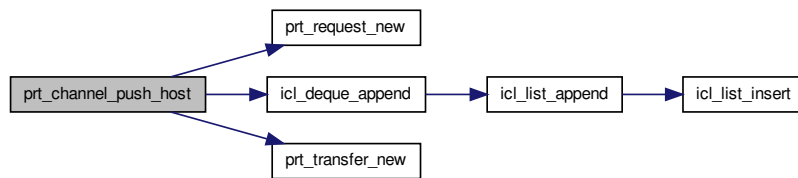
Sends a packet from a host VDP.

Parameters

<i>vdp</i>	– The host VDP sending the packet.
<i>channel</i>	– The channel to send the packet to.
<i>packet</i>	– 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:

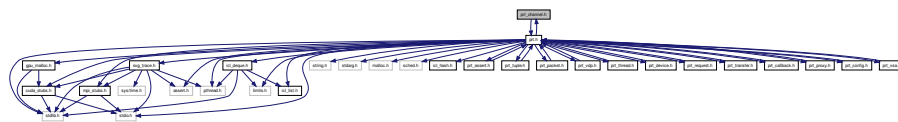


6.19 prt_channel.h File Reference

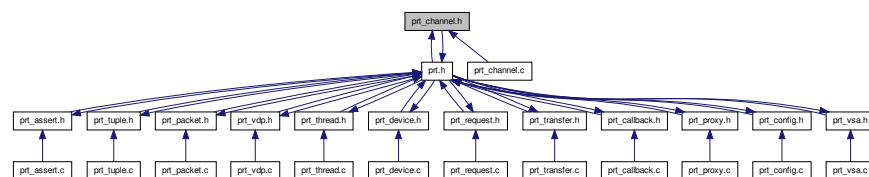
PRT data channel.

```
#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

<i>channel1</i>	– The first channel.
<i>channel2</i>	– The second channel.

Return values

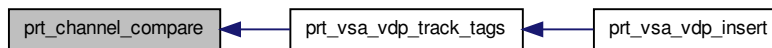
-1	channel1 is less than channel2.
0	channel1 is equal to channel2.
1	channel1 is greater than channel2.

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)

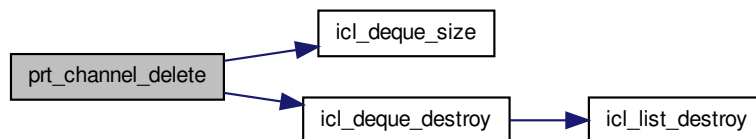
Destroys a channel.

Parameters

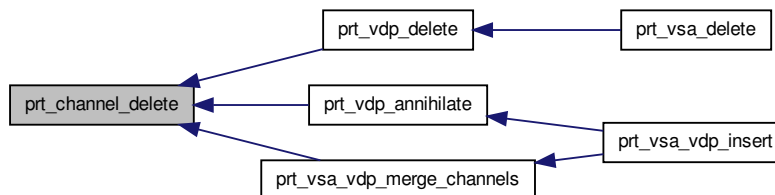
<i>channel</i>	– The channel to destroy.
----------------	---------------------------

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

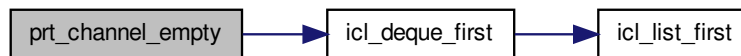
<i>channel</i>	– The channel to check.
----------------	-------------------------

Return values

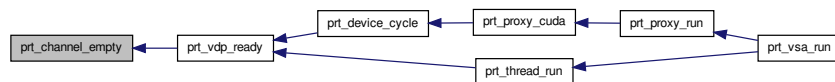
1	if the channel is empty.
0	if the channel is not empty.

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

<i>channel</i>	– 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

<i>channel</i>	- 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 packet from a channel. Does not decrement the number of active references. The packet leaves the channel, but enters the VDP.

Parameters

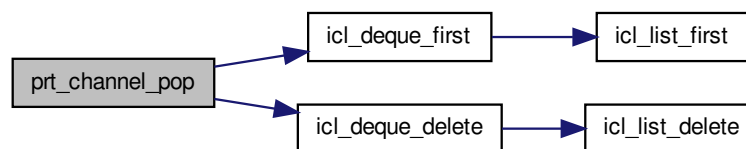
<i>channel</i>	– The channel to fetch the packet from.
----------------	---

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.

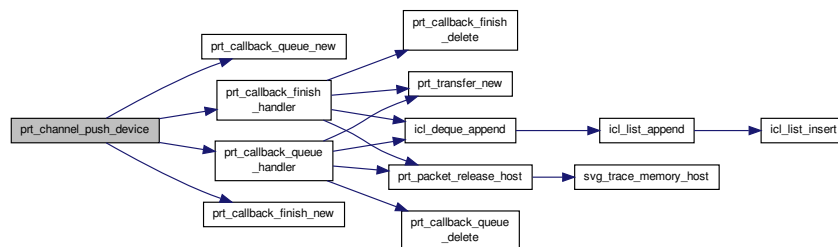
There is no need to set the device here. This function is called by prt_vdp_channel_push, which is called from VDP code, where the device is already set.

Parameters

<i>vdp</i>	– The device VDP sending the packet.
<i>channel</i>	– The channel to send the packet to.
<i>packet</i>	– 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)

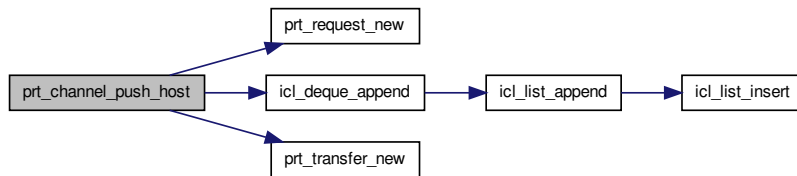
Sends a packet from a host VDP.

Parameters

<i>vdp</i>	– The host VDP sending the packet.
<i>channel</i>	– The channel to send the packet to.
<i>packet</i>	– 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:

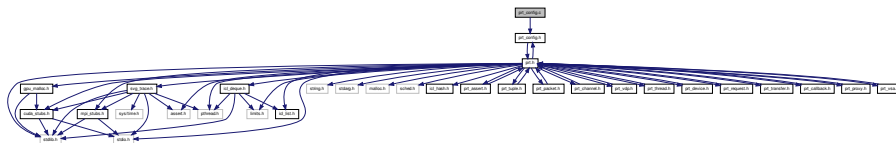


6.20 prt_config.c File Reference

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

<i>config</i>	– 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:

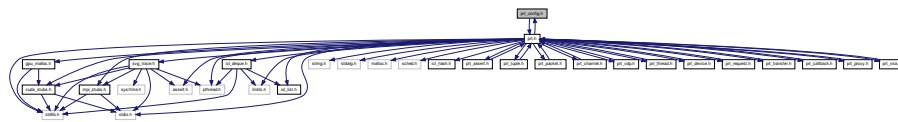


6.21 prt_config.h File Reference

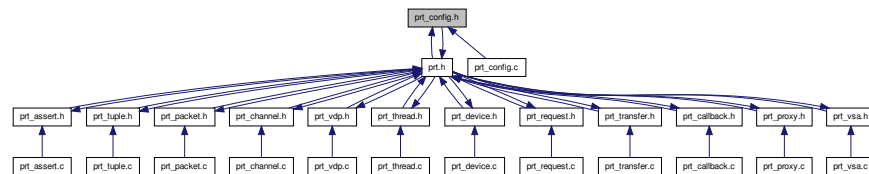
PRT configuration.

```
#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](#).

```
graph LR; prt_vsa_delete --> prt_config_delete
```



```
graph LR; prt_vsa_new --> prt_config_new
```

Generated on Thu Dec 4 2014 14:16:56 for PULSAR by Doxygen

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*)

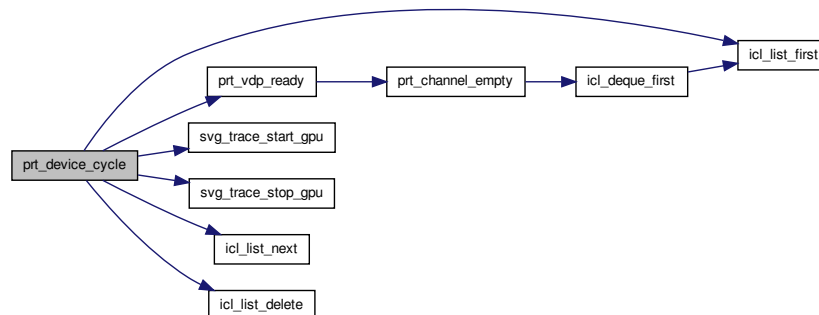
Implements device processing cycle.

Parameters

<i>device</i>	– 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)

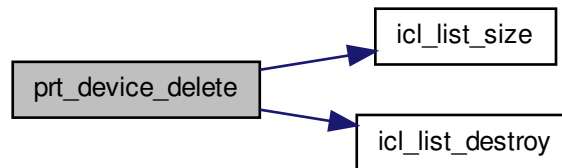
Destroys a device.

Parameters

<i>device</i>	– The device to destroy.
---------------	--------------------------

Definition at line 49 of file prt_device.c.

Here is the call graph for this function:



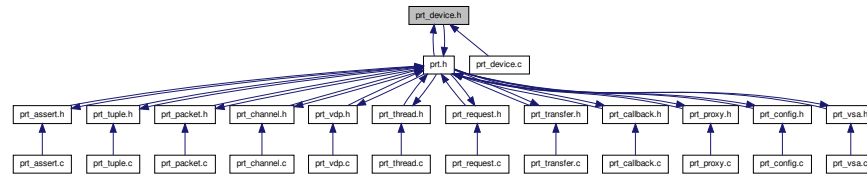
Here is the caller graph for this function:



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

Creates a new device.

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 declared volatile, but no need for atomic access.

6.23.3 Function Documentation

6.23.3.1 void prt_device_cycle (prt_device_t * device)

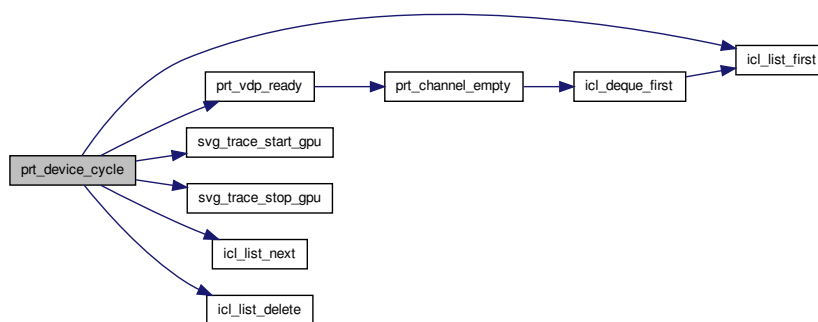
Implements device processing cycle.

Parameters

<i>device</i>	– 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)

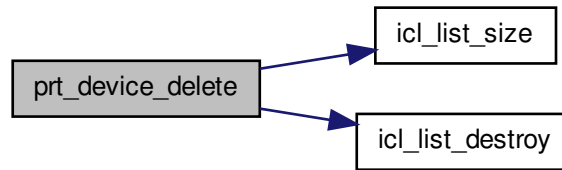
Destroys a device.

Parameters

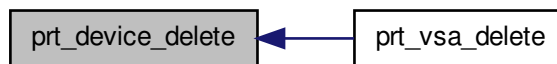
<i>device</i>	– The device to destroy.
---------------	--------------------------

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

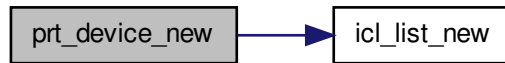
<i>rank</i>	– The local rank of the device.
<i>accelerator</i>	– The global rank of the device.
<i>agent_rank</i>	– The rank of the communication agent.

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:

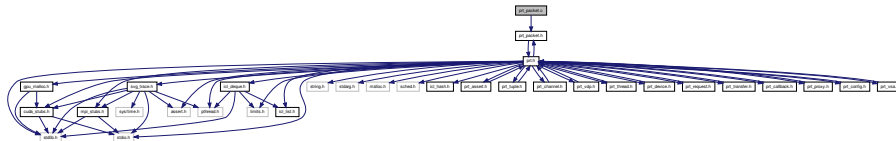


6.24 prt_packet.c File Reference

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, which 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*)

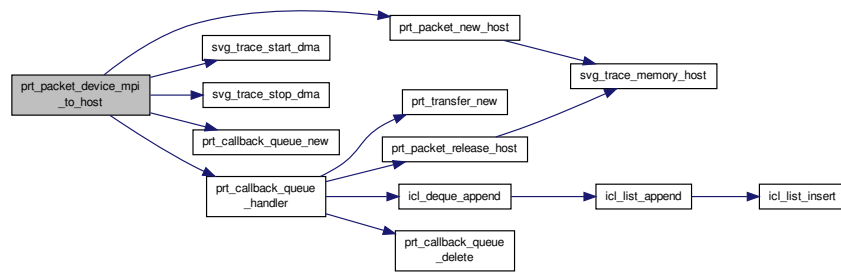
Initiates an MPI transfer from a device. Sends a packet from a device to the host. Then requests an MPI transfer from the host.

Parameters

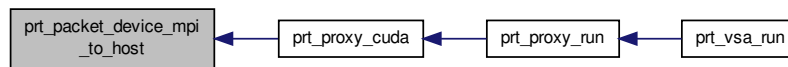
<i>src_packet</i>	– The packet to transfer.
<i>channel</i>	– The destination channel.

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)

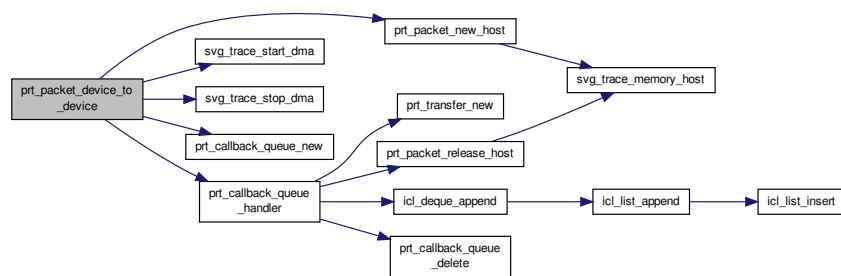
Transfers a packet from a device to another device.

Parameters

<i>src_packet</i>	– The packet to transfer.
<i>channel</i>	– The destination channel.

Definition at line 238 of file prt_packet.c.

Here is the call graph for this function:

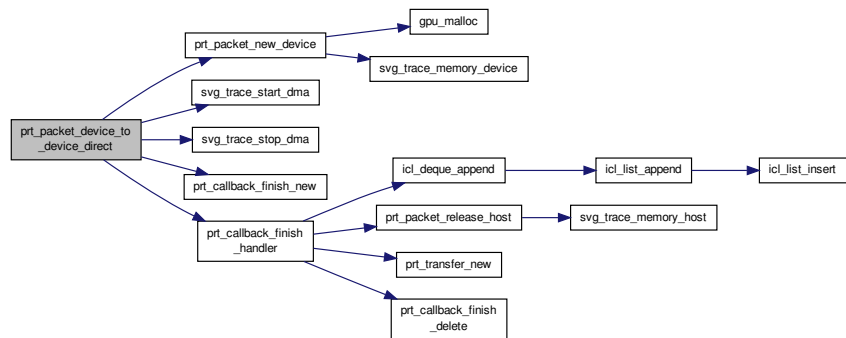



```
graph RL
    prt_vsa_run --> prt_proxy_run
    prt_proxy_run --> prt_proxy_cuda
    prt_proxy_cuda --> prt_packet_device_to_device
```

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

<i>src_packet</i>	– The packet to transfer.
<i>channel</i>	– The destination channel.

Here is the call graph for this function:

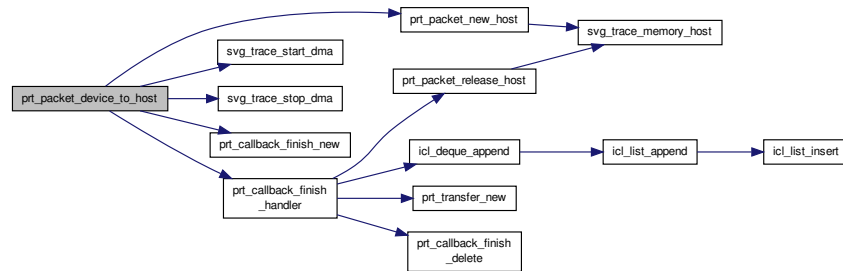


Transfers a packet from the host to a device.

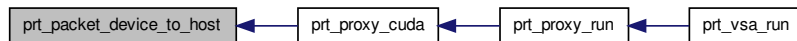
<i>packet</i>	– The packet to transfer.
<i>channel</i>	– The destination channel.
<i>kind</i>	– The direction of the transfer.

Generated on Thu Dec 4 2014 14:16:56 for PULSAR by Doxygen

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)

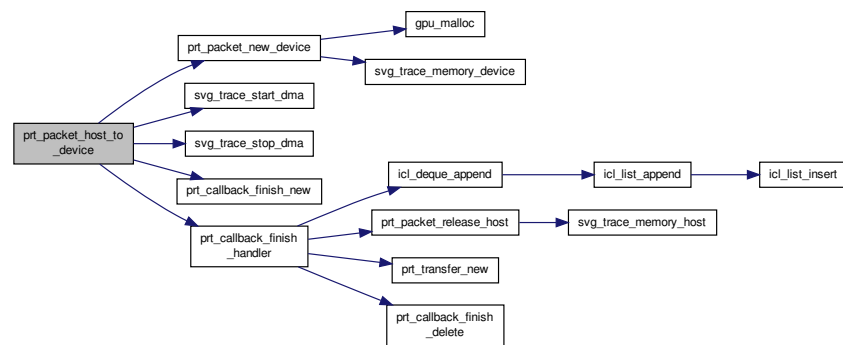
Transfers a packet from the host to a device.

Parameters

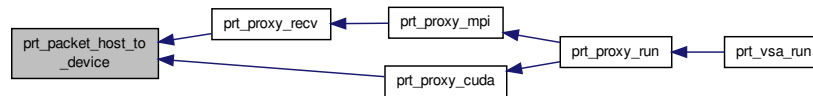
<i>packet</i>	– The packet to transfer.
<i>channel</i>	– The destination channel.
<i>kind</i>	– The direction of the transfer.

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

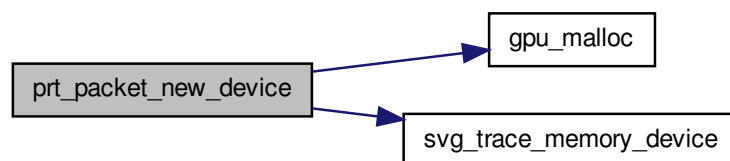
<i>size</i>	– The size of the packet's data.
<i>data</i>	– The pointer to the packet's data.
<i>vdp</i>	– The VDP creating the packet.

Returns

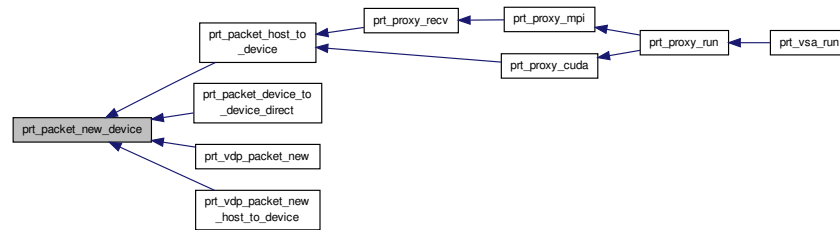
A new packet.

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

<i>size</i>	– The size of packet's data.
<i>data</i>	– The pointer to the packet's data.

Returns

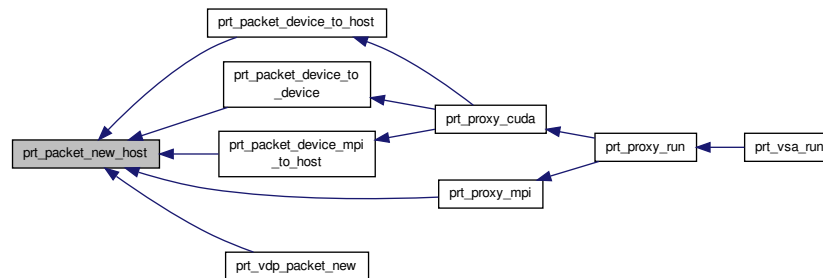
A new packet.

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)

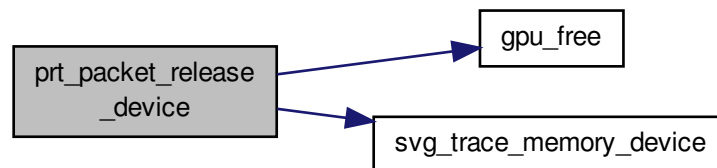
Releases a packet located in device memory. Decrements the number of active references. Destroys the packet when the last reference is removed.

Parameters

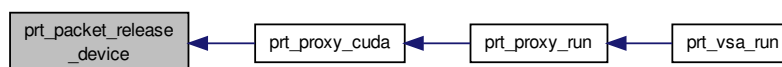
<i>packet</i>	– The device packet to release.
---------------	---------------------------------

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

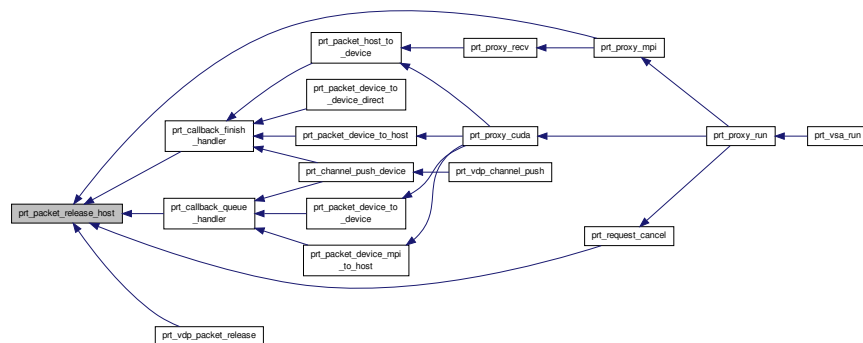
<i>packet</i>	– The host packet to release.
---------------	-------------------------------

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, which initially are allocated with the maximum packet size.

Parameters

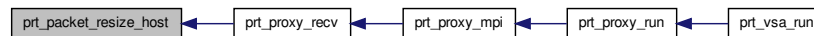
<i>packet</i>	– The packet to resize.
<i>size</i>	– The new size in bytes.

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:

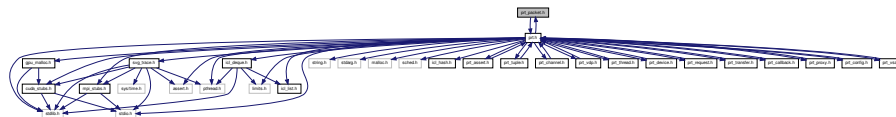


6.25 prt_packet.h File Reference

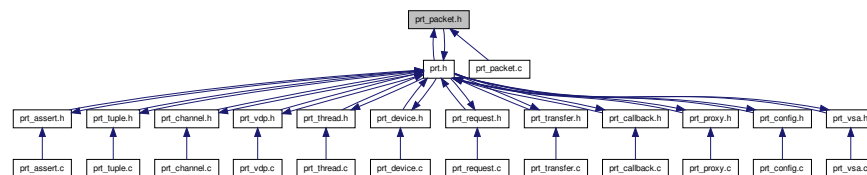
PRT data packet.

```
#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)

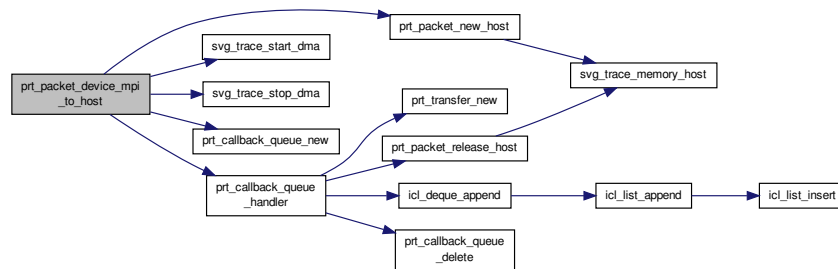
Initiates an MPI transfer from a device. Sends a packet from a device to the host. Then requests an MPI transfer from the host.

Parameters

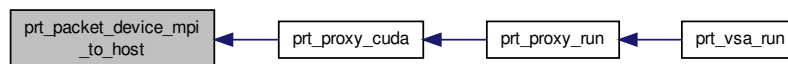
<i>src_packet</i>	– The packet to transfer.
<i>channel</i>	– The destination channel.

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)

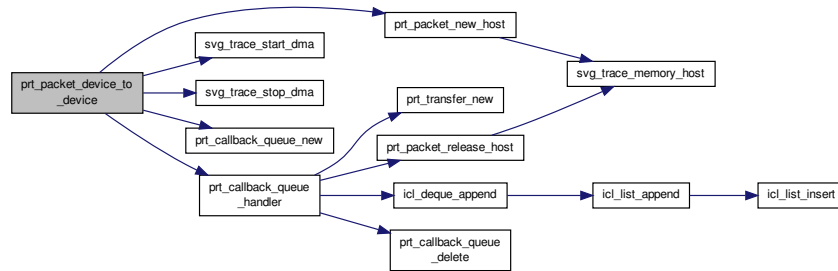
Transfers a packet from a device to another device.

Parameters

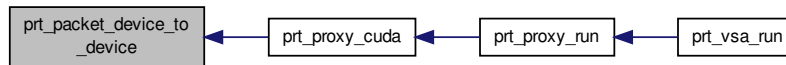
<i>src_packet</i>	– The packet to transfer.
<i>channel</i>	– The destination channel.

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*)

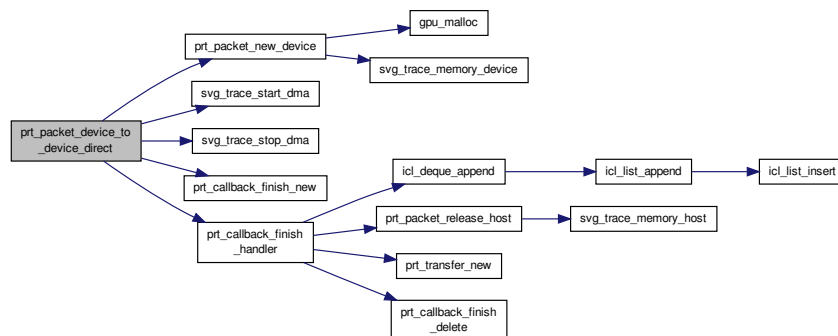
Transfers a packet from a device to another device. Uses a direct copy, without involving the host.

Parameters

<i>src_packet</i>	– The packet to transfer.
<i>channel</i>	– 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)

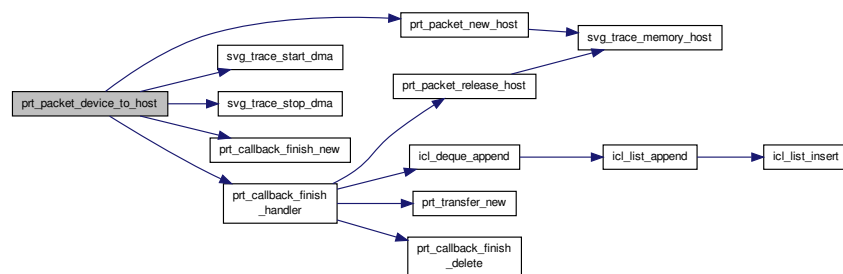
Transfers a packet from the host to a device.

Parameters

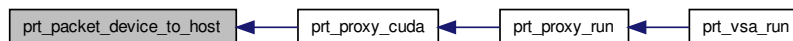
<i>packet</i>	– The packet to transfer.
<i>channel</i>	– The destination channel.
<i>kind</i>	– The direction of the transfer.

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)

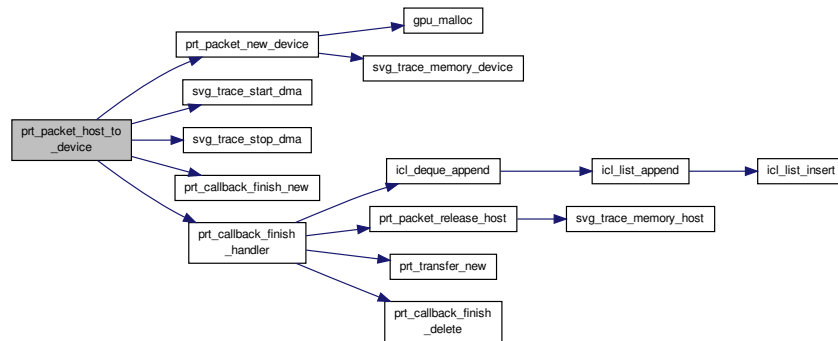
Transfers a packet from the host to a device.

Parameters

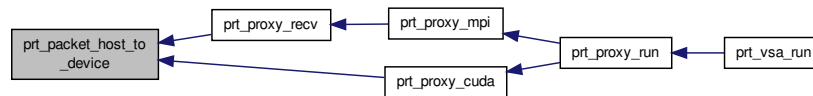
<i>packet</i>	– The packet to transfer.
<i>channel</i>	– The destination channel.
<i>kind</i>	– The direction of the transfer.

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

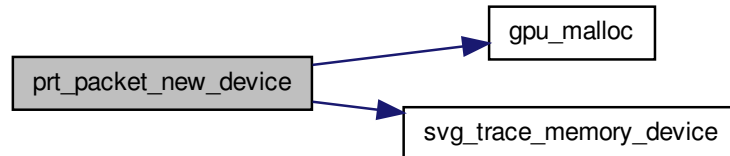
<i>size</i>	– The size of the packet's data.
<i>data</i>	– The pointer to the packet's data.
<i>vdp</i>	– The VDP creating the packet.

Returns

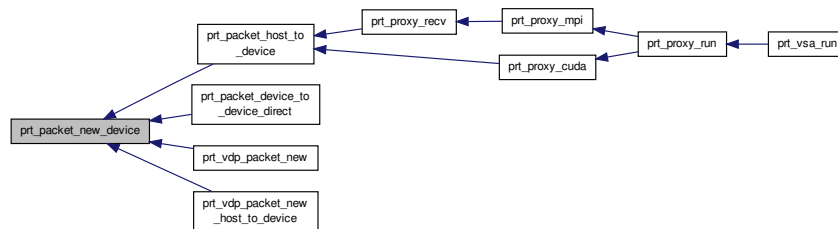
A new packet.

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

<i>size</i>	– The size of packet's data.
<i>data</i>	– The pointer to the packet's data.

Returns

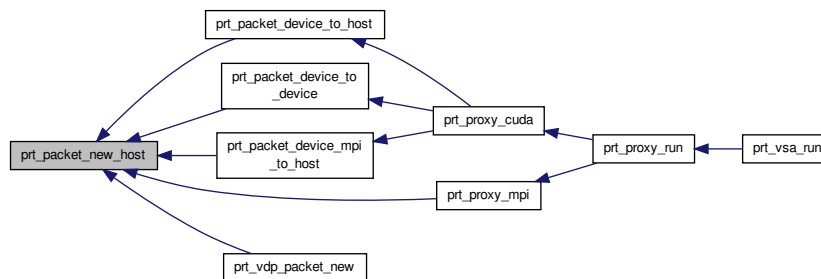
A new packet.

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)

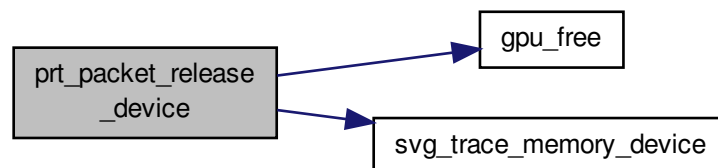
Releases a packet located in device memory. Decrements the number of active references. Destroys the packet when the last reference is removed.

Parameters

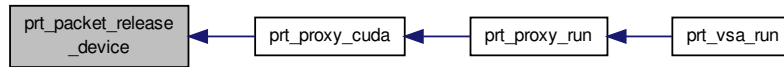
<i>packet</i>	– The device packet to release.
---------------	---------------------------------

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

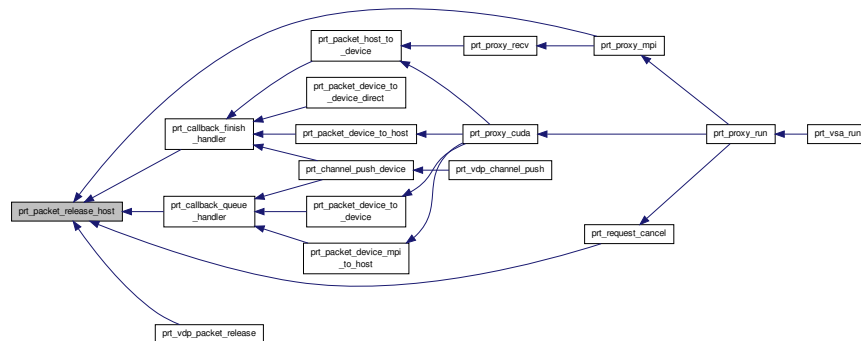
<i>packet</i>	– The host packet to release.
---------------	-------------------------------

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, which initially are allocated with the maximum packet size.

Parameters

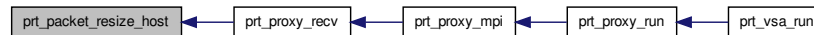
<i>packet</i>	– The packet to resize.
<i>size</i>	– The new size in bytes.

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:

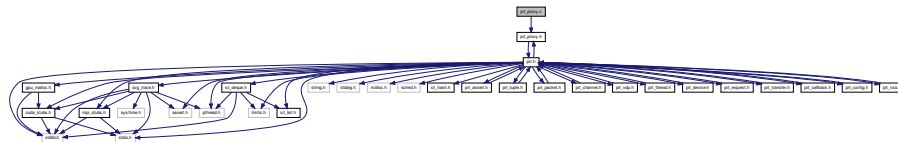


6.26 prt_proxy.c File Reference

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_rcv](#) (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

<code>proxy</code>	– The proxy to cycle CUDA.
--------------------	----------------------------

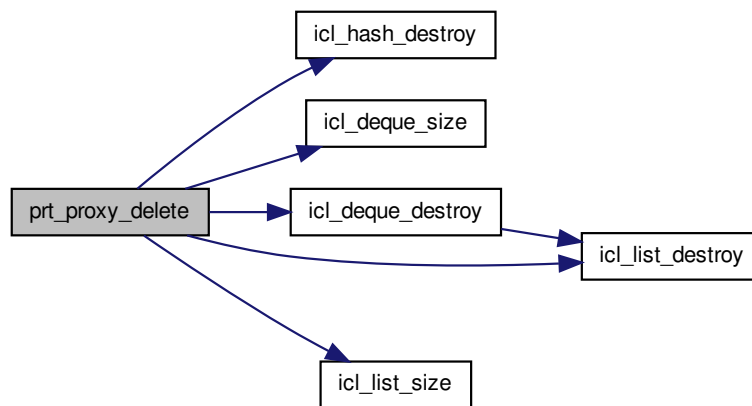
Definition at line 256 of file `prt_proxy.c`.

Parameters

<i>proxy</i>	– 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:



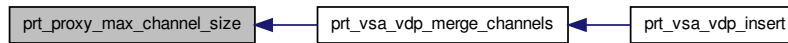
6.26.2.3 void prt_proxy_max_channel_size (prt_proxy_t * proxy, prt_channel_t * channel)

Looks for maximum channel/packet size.

Parameters

<i>proxy</i>	– The proxy registering the size.
<i>channel</i>	– The channel to register the size of.

Definition at line 132 of file prt_proxy.c.



6.26.2.4 void prt_proxy_mpi (prt_proxy_t * proxy)

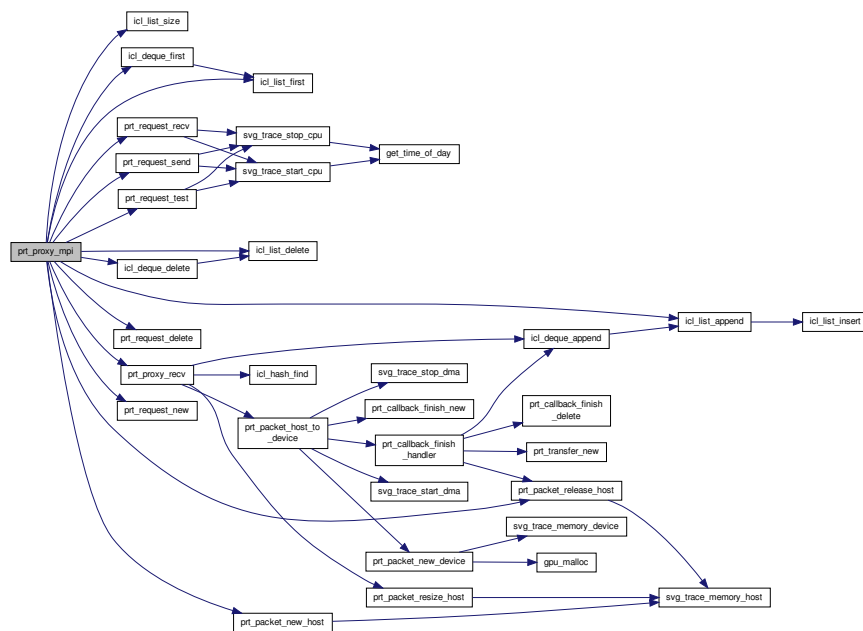
Implements the proxy's MPI cycle. Services all MPI requests.

Parameters

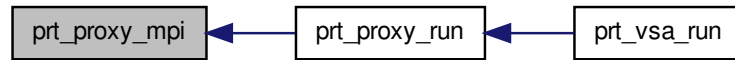
<i>proxy</i>	– The proxy to cycle MPI.
--------------	---------------------------

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

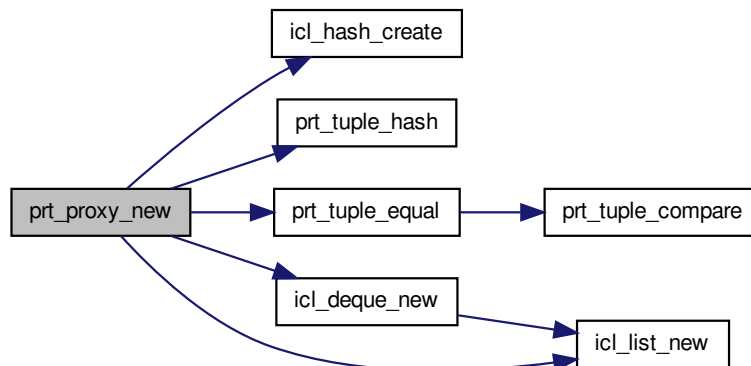
<code>num_agents</code>	– The number of local agents (threads + devices).
-------------------------	---

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_rcv (prt_proxy_t * proxy, prt_request_t * request)

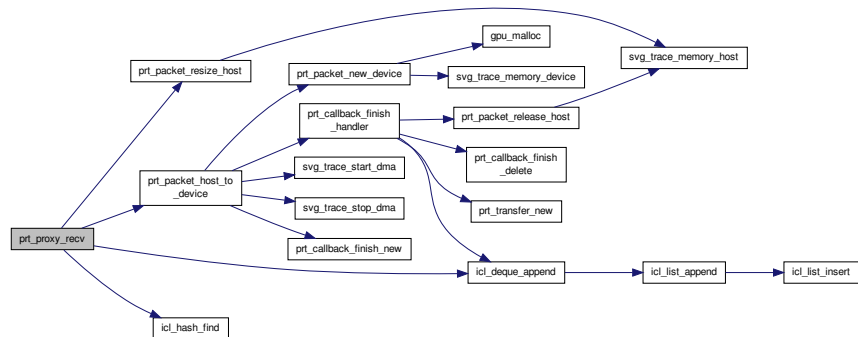
Receives to a channel.

Parameters

<i>proxy</i>	– The proxy to receive the request.
<i>request</i>	– 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:

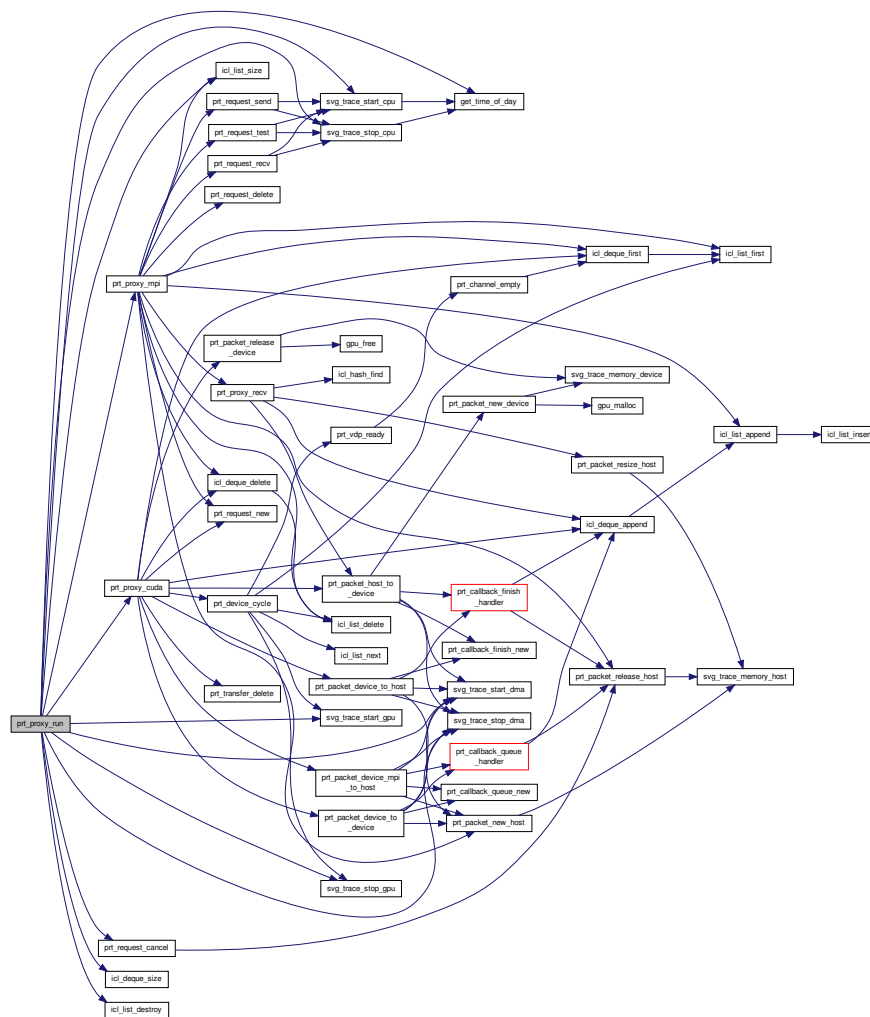


Implements the proxy's production cycle. First, barriers with all MPI processes. Then, barriers with all local worker threads and starts measuring time. When finished, barriers with all local worker threads. Then, barriers with all MPI processes and stops the timer.

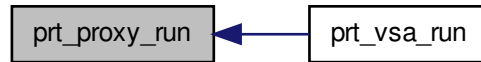
<i>proxy</i>	– The proxy to run.
--------------	---------------------

The execution time.

Here is the call graph for this function:



Here is the caller graph for this function:

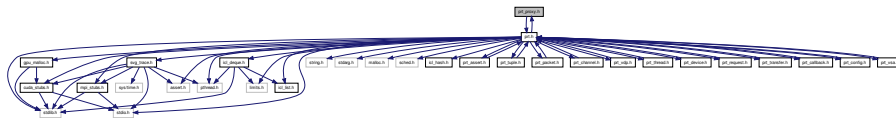


6.27 prt_proxy.h File Reference

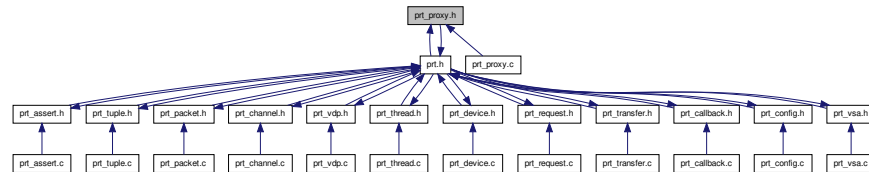
PRT communication proxy.

```
#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)

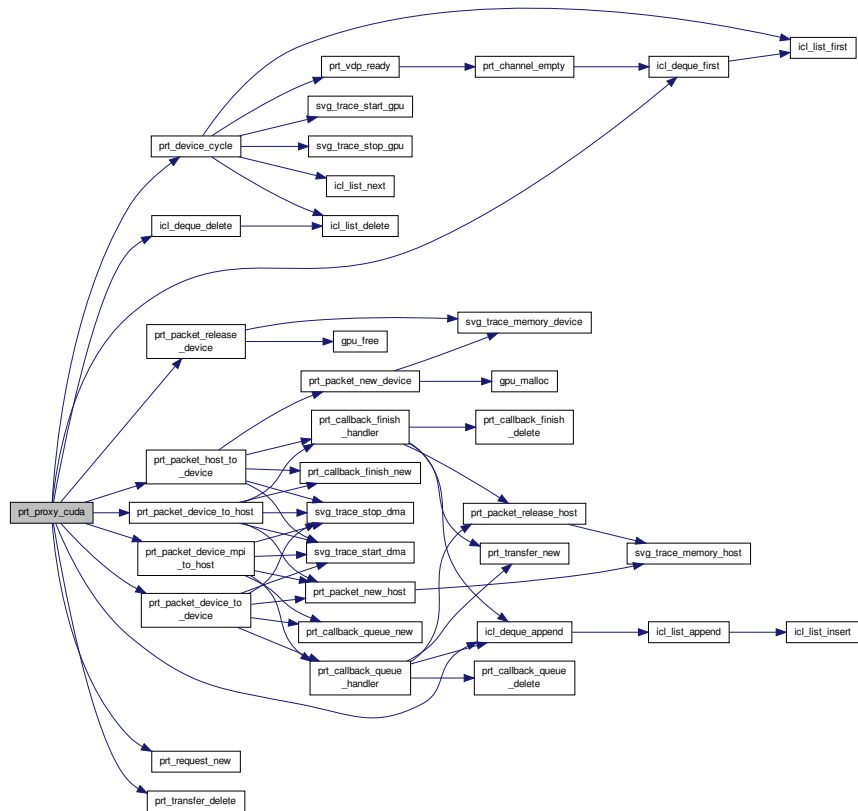
Implements the proxy's CUDA cycle. Services all local transfer requests. Runs all device code.

Parameters

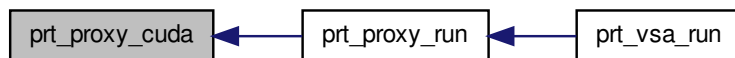
<i>proxy</i>	– The proxy to cycle CUDA.
--------------	----------------------------

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)

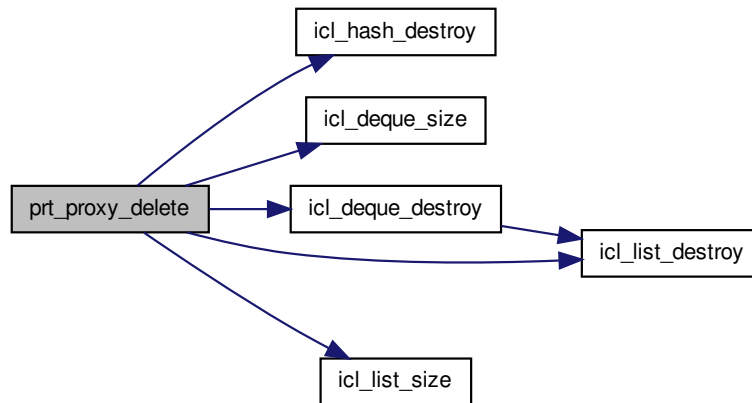
Destroys a proxy. Checks if all the lists are empty at the time of destruction. Not destroying the list of receives (destroyed at the end of the proxy's cycle).

Parameters

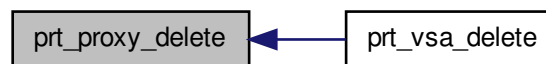
<i>proxy</i>	– The proxy to destroy.
--------------	-------------------------

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:



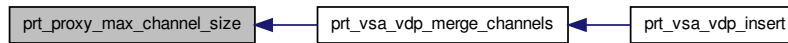
6.27.3.3 void prt_proxy_max_channel_size (prt_proxy_t * proxy, prt_channel_t * channel)

Looks for maximum channel/packet size.

Parameters

<i>proxy</i>	– The proxy registering the size.
<i>channel</i>	– The channel to register the size of.

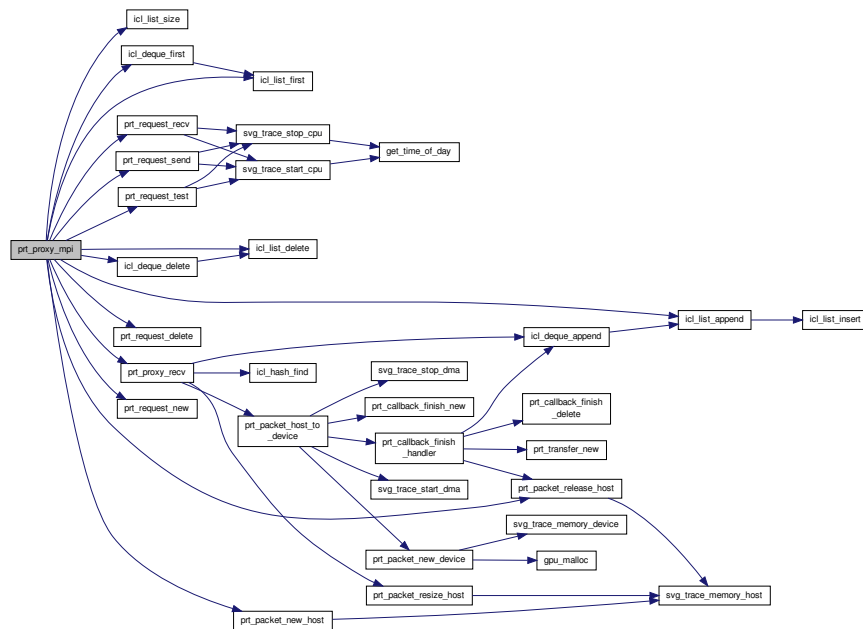
Definition at line 132 of file prt_proxy.c.



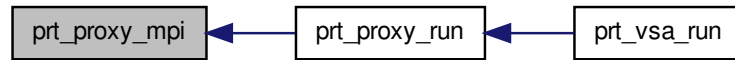
Implements the proxy's MPI cycle. Services all MPI requests.

<i>proxy</i>	– The proxy to cycle MPI.
--------------	---------------------------

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

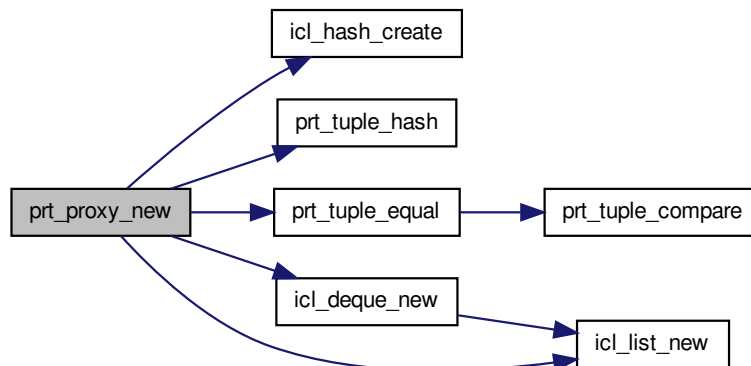
<code>num_agents</code>	– The number of local agents (threads + devices).
-------------------------	---

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_rcv (prt_proxy_t * proxy, prt_request_t * request)

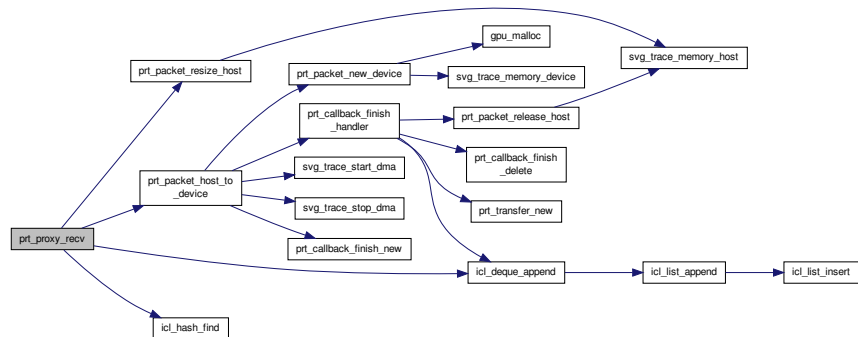
Receives to a channel.

Parameters

<i>proxy</i>	– The proxy to receive the request.
<i>request</i>	– 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

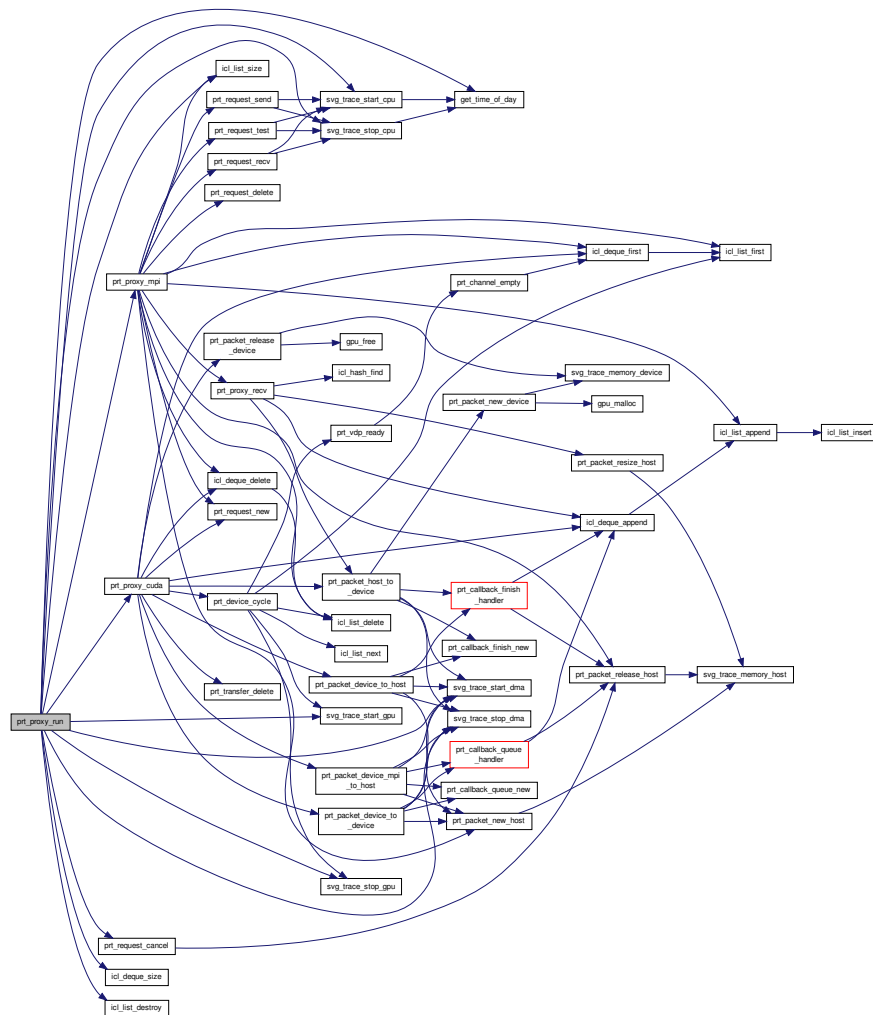
<i>proxy</i>	– The proxy to run.
--------------	---------------------

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:

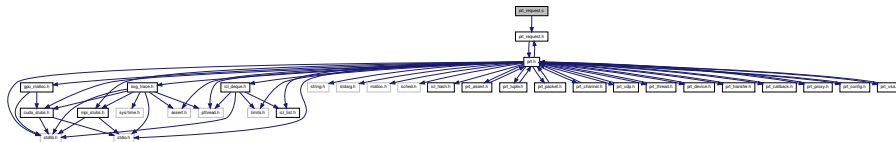


6.28 prt_request.c File Reference

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_rcv (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

PULSAR Runtime <http://icl.utk.edu/pulsar/> Copyright (C) 2012-2015 University of Tennessee.Definition in file [prt_request.c](#).

6.28.2 Function Documentation

6.28.2.1 void prt_request_cancel (prt_request_t * request)

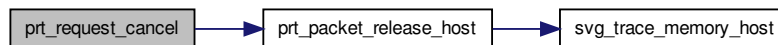
Cancels a request. Cancels them MPI request, releases the packet, frees the request object.

Parameters

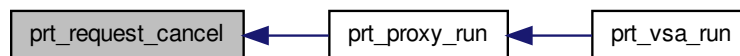
<i>request</i>	– The request to cancel.
----------------	--------------------------

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

<i>request</i>	– 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

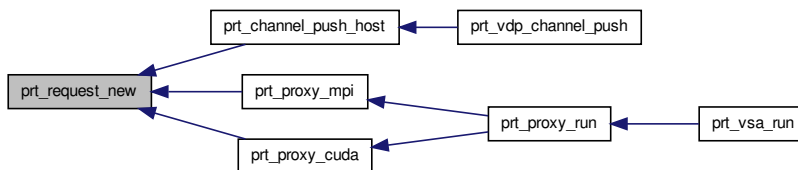
<i>packet</i>	– The packet to create the request for.
<i>count</i>	– The number of data items.
<i>datatype</i>	– The type of data items.
<i>peer</i>	– The peer communicating node.
<i>tag</i>	– The MPI tag of the message.

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_rcv (prt_request_t * request)`

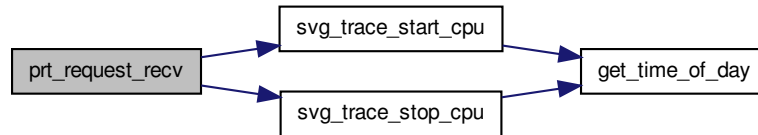
Posts a receive request.

Parameters

<i>request</i>	– 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)

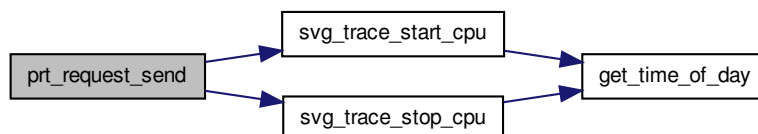
Posts a send request.

Parameters

<i>request</i>	– 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

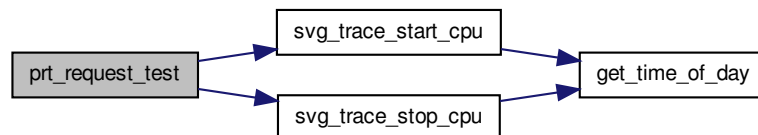
<i>request</i>	– The request to test.
----------------	------------------------

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:



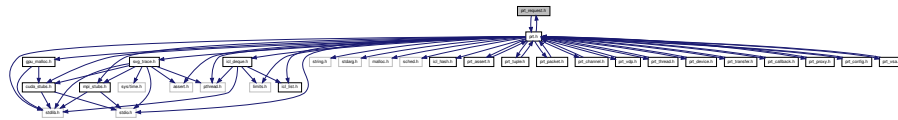
Here is the caller graph for this function:



6.29 prt_request.h File Reference

PRT communication request.

Include dependency graph for prt_request.h:



- struct prt_request_s

Typedefs

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

- `prt_request_t * prt_request_new (struct prt_packet_s *packet, size_t size, int peer, int tag)`

- void prt_request_delete (prt_request_t *request)

- void prt_request_send (prt_request_t *request)

- void prt_request_rcv (prt_request_t *request)

- `int prt_request_test (prt_request_t *request)`

- void prt_request_cancel (prt_request_t *request)

Cancels a request. Cancels the 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)

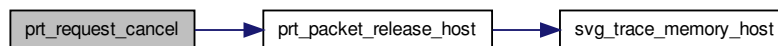
Cancels a request. Cancels them MPI request, releases the packet, frees the request object.

Parameters

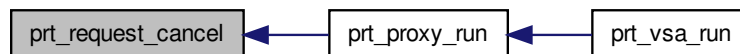
<i>request</i>	– The request to cancel.
----------------	--------------------------

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

<i>request</i>	– 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

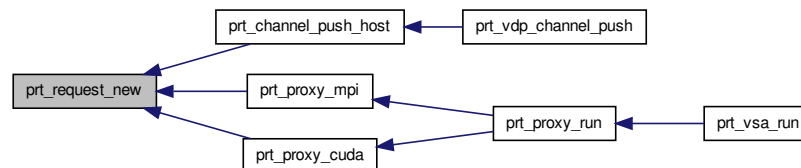
<i>packet</i>	– The packet to create the request for.
<i>count</i>	– The number of data items.
<i>datatype</i>	– The type of data items.
<i>peer</i>	– The peer communicating node.
<i>tag</i>	– The MPI tag of the message.

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)`

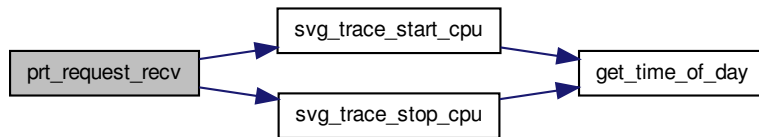
Posts a receive request.

Parameters

<i>request</i>	– 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)

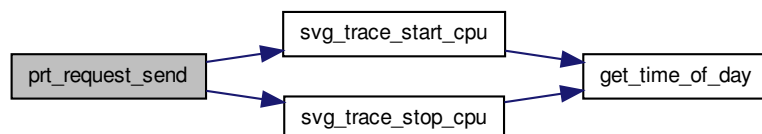
Posts a send request.

Parameters

<i>request</i>	– 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

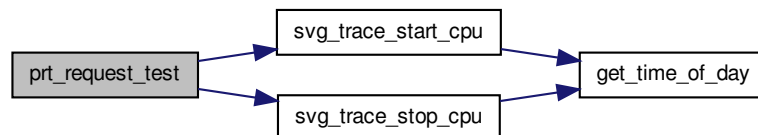
<i>request</i>	– The request to test.
----------------	------------------------

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:



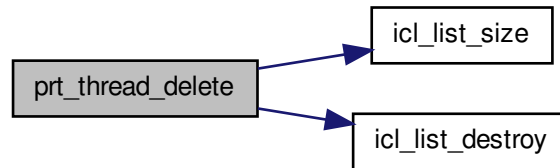
Here is the caller graph for this function:



6.30 prt_thread.c File Reference

PRT thread.

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

<i>rank</i>	– The local rank of the thread.
<i>core</i>	– The global rank of the thread.
<i>agent_rank</i>	- The rank of the communication agent.

Returns

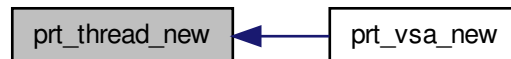
A new thread object.

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

<i>thrd</i>	– The pointer to the thread object.
-------------	-------------------------------------

Definition at line 75 of file prt_thread.c.

```
graph LR; prt_thread_run[prt_thread_run] --> svg_trace_start_cpu[svg_trace_start_cpu]; prt_thread_run --> svg_trace_stop_cpu[svg_trace_stop_cpu]; prt_thread_run --> prt_vdp_ready[prt_vdp_ready]; prt_thread_run --> icl_list_next[icl_list_next]; prt_thread_run --> icl_list_delete[icl_list_delete]; svg_trace_start_cpu --> get_time_of_day[get_time_of_day]; svg_trace_stop_cpu --> get_time_of_day; prt_vdp_ready --> prt_channel_empty[prt_channel_empty]; prt_channel_empty --> icl_deque_first[icl_deque_first]; icl_list_next --> icl_list_first[icl_list_first]; icl_list_delete --> icl_list_first; icl_deque_first --> icl_list_first;
```

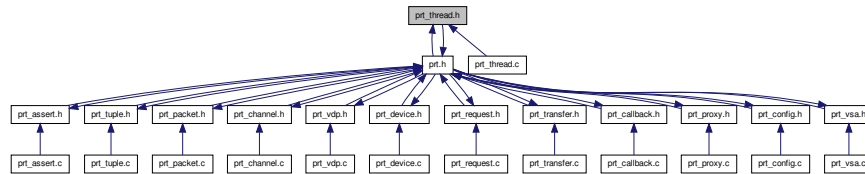
```
graph LR; prt_vsa_run --> prt_thread_run
```

PRT thread.

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 declared volatile, but no need for atomic access.

6.31.3 Function Documentation

6.31.3.1 void prt_thread_delete (prt_thread_t * thread)

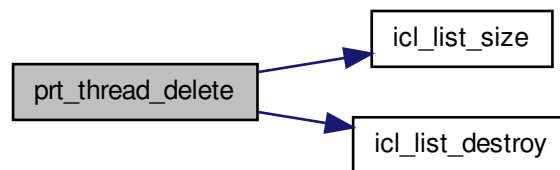
Destroys a thread.

Parameters

<i>thread</i>	– 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

<i>rank</i>	– The local rank of the thread.
<i>core</i>	– The global rank of the thread.
<i>agent_rank</i>	- 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*)

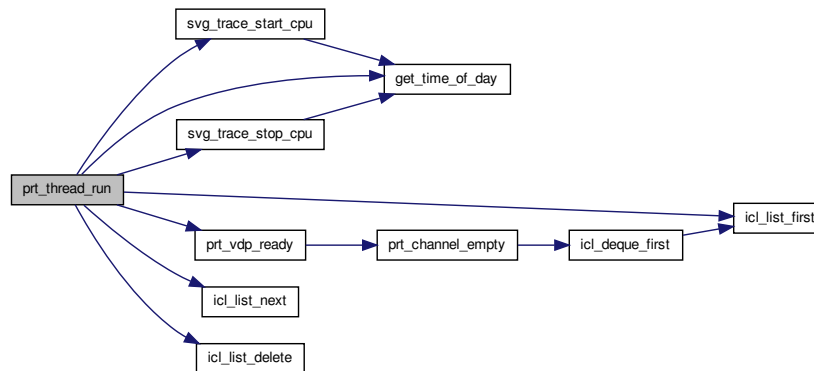
Implements the thread's processing cycle. If set, calls the thread warmup function. Barriers all threads. If the communication proxy is active, it participates in the barrier. Cycles through VDPs. Fires the ones that are ready. Removes the ones which burned out. Quits when the list of VDPs becomes empty. Saves the execution time.

Parameters

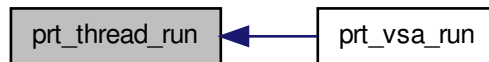
<i>thrd</i>	– The pointer to the thread object.
-------------	-------------------------------------

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:

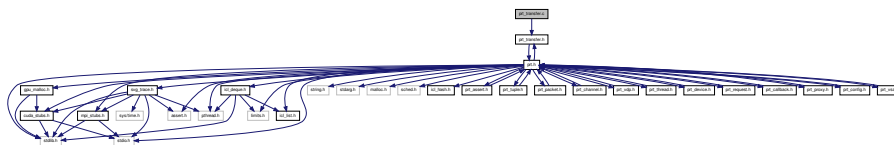


6.32 prt_transfer.c File Reference

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

<i>&ndash;</i>	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

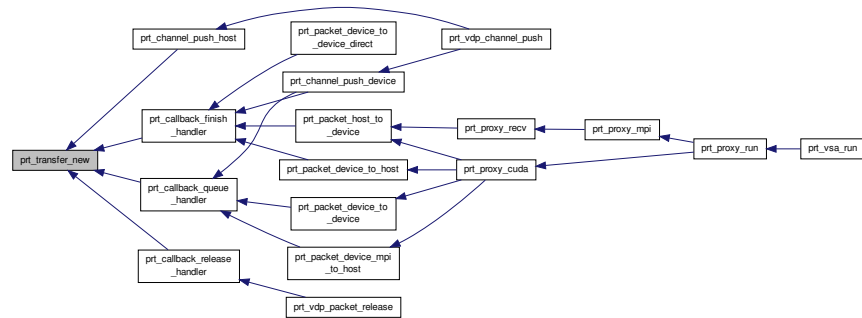
<i>packet</i>	– The packet to transfer.
<i>channel</i>	– The channel to push to.
<i>direction</i>	– The direction of the transfer.
<i>agent</i>	– The number of the communication agent to use.

Returns

A new local transfer object.

Definition at line 24 of file prt_transfer.c.

Here is the caller graph for this function:

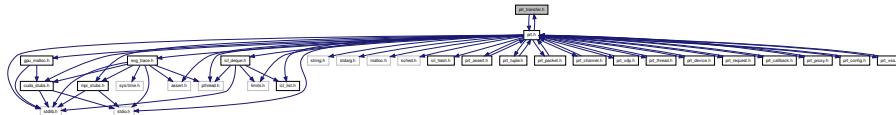


6.33 prt_transfer.h File Reference

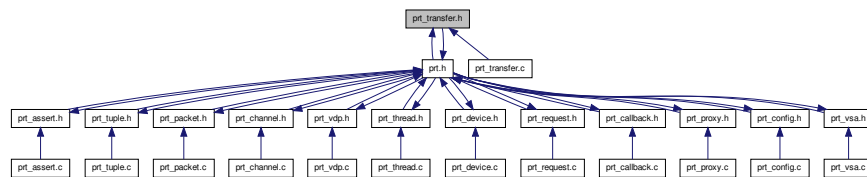
PRT local transfer.

```
#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

<i>&ndash;</i>	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.

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

<i>first_tuple</i>	– 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

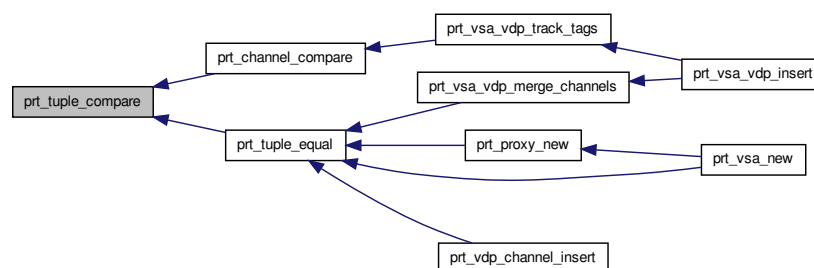
<i>tuple_a</i>	– The first tuple.
<i>tuple_b</i>	– The second tuple.

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

<i>in_tuple</i>	– 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*)

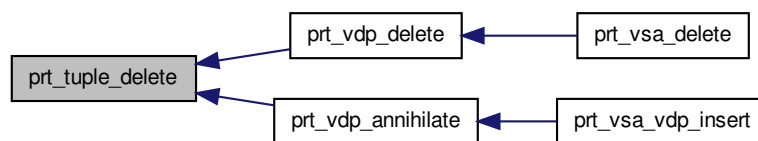
Destroys a tuple.

Parameters

<i>tuple</i>	– 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

<i>tuple_a</i>	– The first tuple.
<i>tuple_b</i>	– The second tuple.

Return values

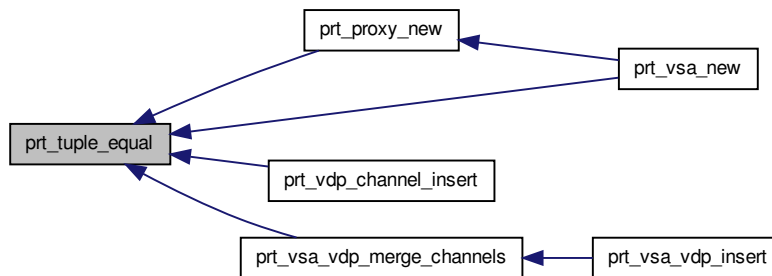
0	if tuples are different.
1	if tuples are identical.

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 length in characters and calls a string hashing function.

Parameters

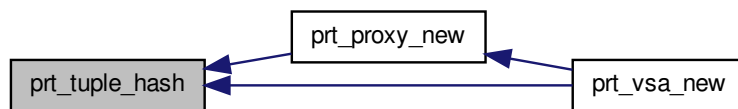
<i>tuple</i>	– The tuple to hash.
--------------	----------------------

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

<i>tuple</i>	– 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

<i>tuple</i>	– The tuple to print.
--------------	-----------------------

Definition at line 178 of file prt_tuple.c.

6.35 prt_tuple.h File Reference

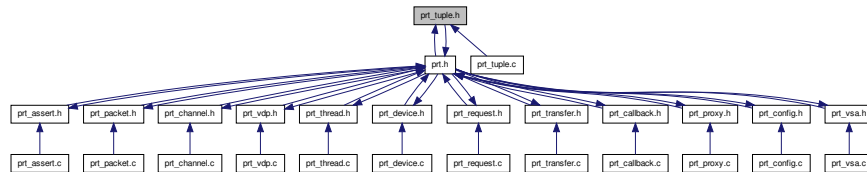
PRT tuple.

```
#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

<i>first_tuple</i>	– The first tuple in the sequence.
<i>...</i>	– 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

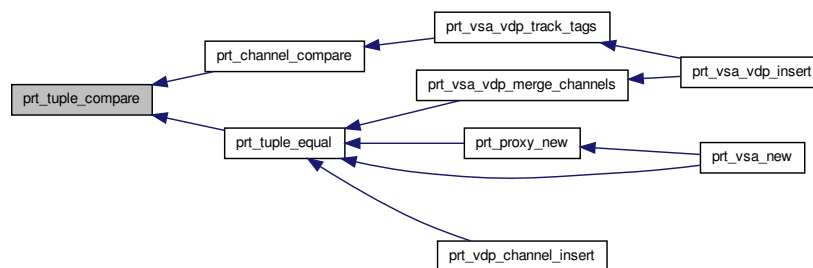
<i>tuple_a</i>	– The first tuple.
<i>tuple_b</i>	– The second tuple.

Return values

<i>-1</i>	if tuple_a is less than tuple_b.
<i>0</i>	if tuple_a is equal to tuple_b.
<i>1</i>	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

<i>in_tuple</i>	– 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)

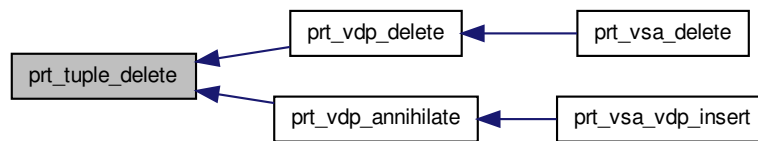
Destroys a tuple.

Parameters

<i>tuple</i>	– 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

<i>tuple_a</i>	– The first tuple.
<i>tuple_b</i>	– 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

<i>tuple</i>	– 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

<i>tuple</i>	– 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

<i>tuple</i>	– The tuple to print.
--------------	-----------------------

Definition at line 178 of file prt_tuple.c.

6.36 prt_vdp.c File Reference

Virtual Data Processor.

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)

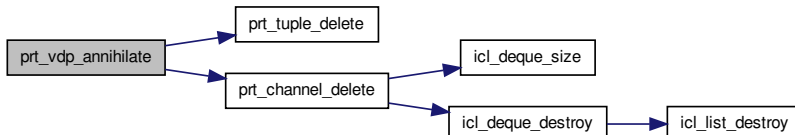
Annihilates a VDP. Used for complete annihilation of VDPs that don't belong in the node. Destroys all input channels. Destroys all output channels.

Parameters

<i>vdp</i>	– 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)

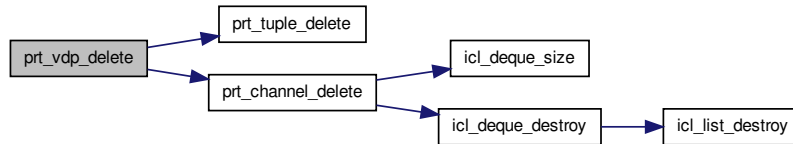
Destroys a VDP. Used for destruction of local VDPs. Destroys all input channels. Destroys all dangling output channels. Local output channels are destroyed as input channels of other local VDPs.

Parameters

<i>vdp</i>	– 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 prevent firing.

Parameters

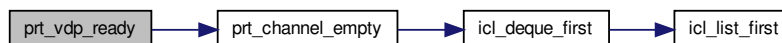
<i>vdp</i>	– The VDP to check.
------------	---------------------

Return values

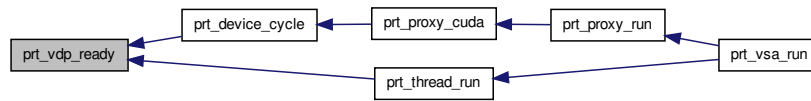
1	if ready.
0	if not ready.

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:

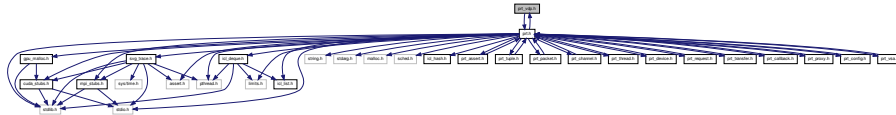


6.37 prt_vdp.h File Reference

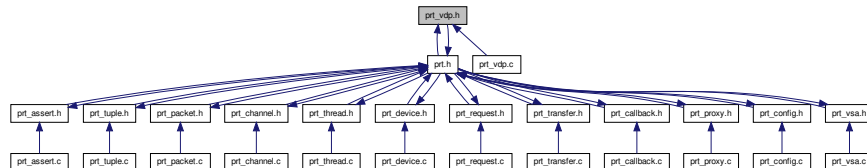
Virtual Data Processor.

```
#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)

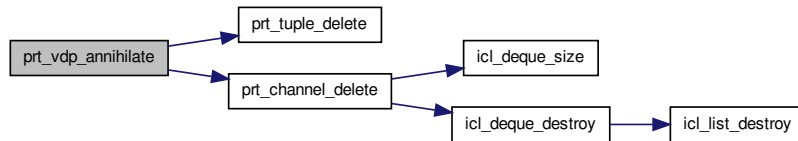
Annihilates a VDP. Used for complete annihilation of VDPs that don't belong in the node. Destroys all input channels. Destroys all output channels.

Parameters

<i>vdp</i>	– 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)

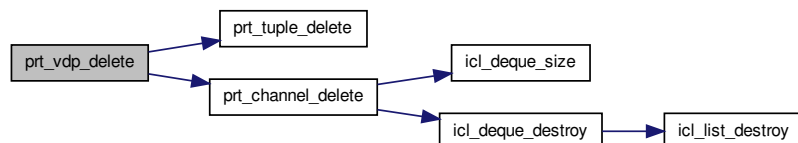
Destroys a VDP. Used for destruction of local VDPs. Destroys all input channels. Destroys all dangling output channels. Local output channels are destroyed as input channels of other local VDPs.

Parameters

<i>vdp</i>	– 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 prevent firing.

Parameters

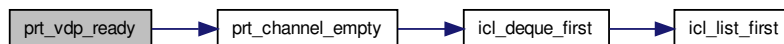
<i>vdp</i>	– The VDP to check.
------------	---------------------

Return values

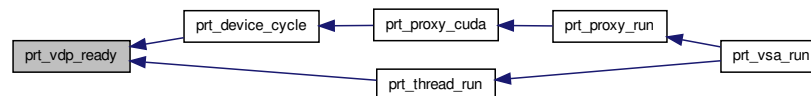
1	if ready.
0	if not ready.

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.

- 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

<i>tuple_a</i>	– The first tuple.
<i>tuple_b</i>	– The second tuple.

Return values

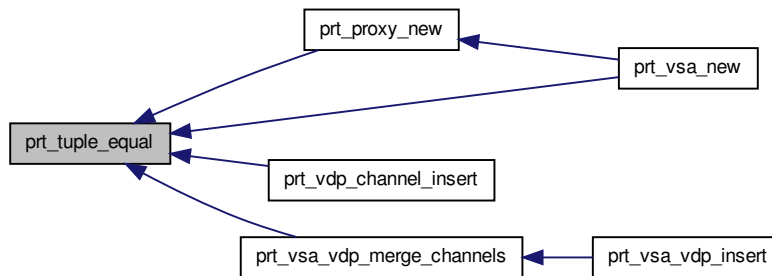
0	if tuples are different.
1	if tuples are identical.

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 length in characters and calls a string hashing function.

Parameters

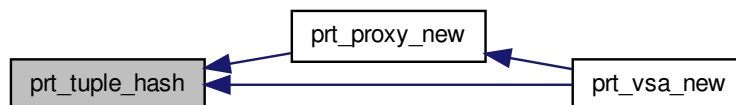
<i>tuple</i>	– The tuple to hash.
--------------	----------------------

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)

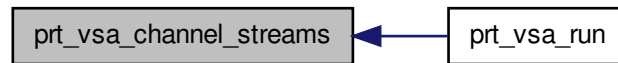
Creates channel streams.

Parameters

<code>vsa</code>	– 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)

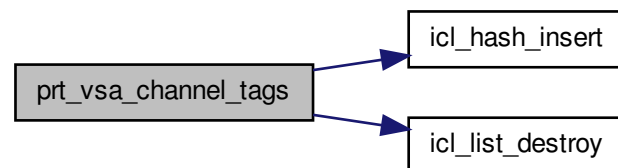
Assigns channel tags. Builds the node-tag lookup. Destroys channel lists.

Parameters

<code>vsa</code>	– 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)

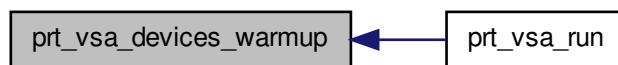
Calls the warmup function for all devices and synchronizes.

Parameters

<i>vsa</i>	– 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)

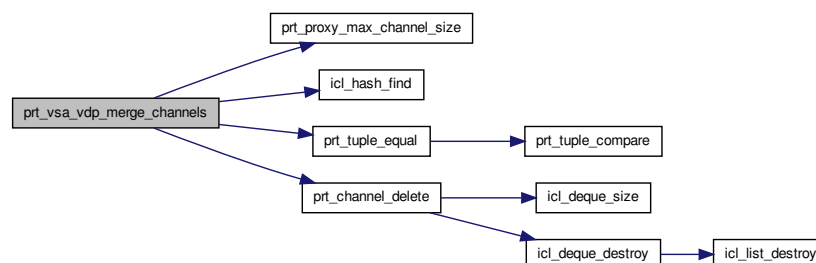
Connects corresponding input and output channels of intra-node VDPs. An input channel always overrides an output channel. This way the on/off switch of the input channel is preserved.

Parameters

<i>vsa</i>	– The VSA to merge channels within.
<i>vdp</i>	– 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)

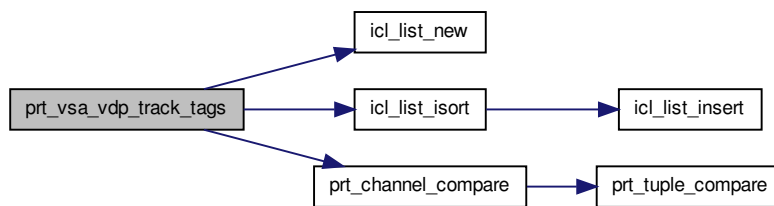
Builds the list of channel connections to other nodes.

Parameters

<i>vsa</i>	– The VSA to track the tags within.
<i>vdp</i>	– 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:

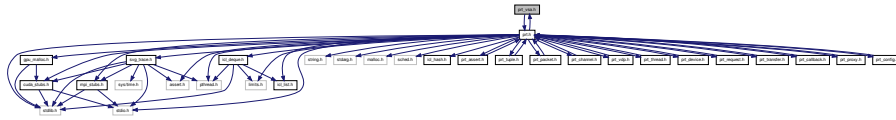


6.39 prt_vsa.h File Reference

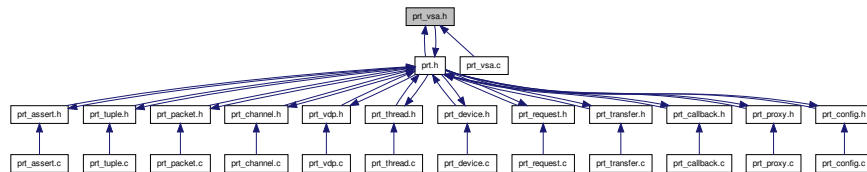
Virtual Systolic Array.

```
#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 anywhere 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)

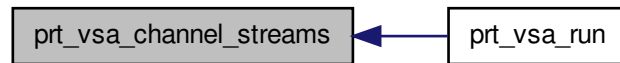
Creates channel streams.

Parameters

<code>vsa</code>	– 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)

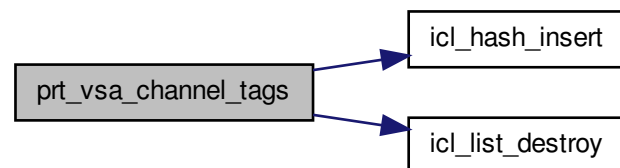
Assigns channel tags. Builds the node-tag lookup. Destroys channel lists.

Parameters

<code>vsa</code>	– 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)

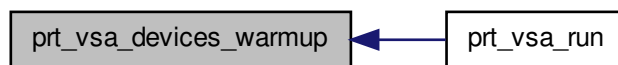
Calls the warmup function for all devices and synchronizes.

Parameters

<i>vsa</i>	– 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)

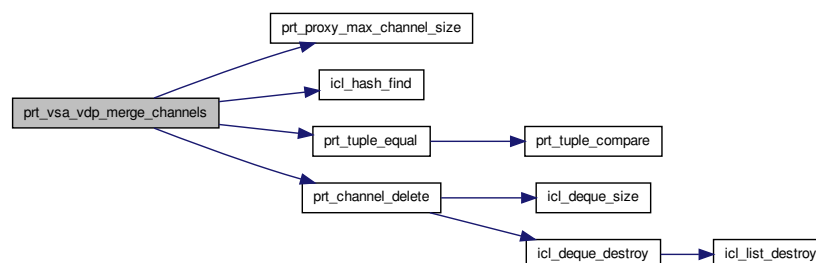
Connects corresponding input and output channels of intra-node VDPs. An input channel always overrides an output channel. This way the on/off switch of the input channel is preserved.

Parameters

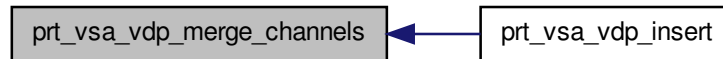
<i>vsa</i>	– The VSA to merge channels within.
<i>vdp</i>	– The VDP to merge channels for.

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)

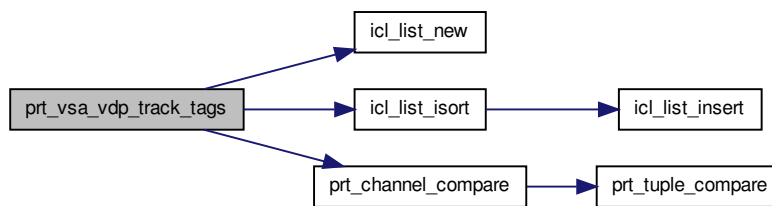
Builds the list of channel connections to other nodes.

Parameters

<i>vsa</i>	– The VSA to track the tags within.
<i>vdp</i>	– 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:

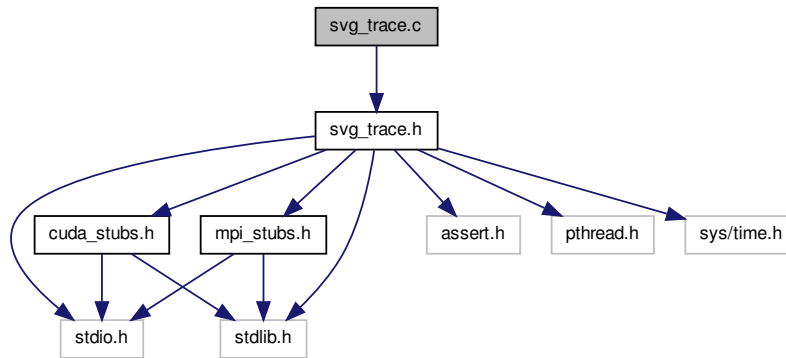


6.40 svg_trace.c File Reference

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 ()

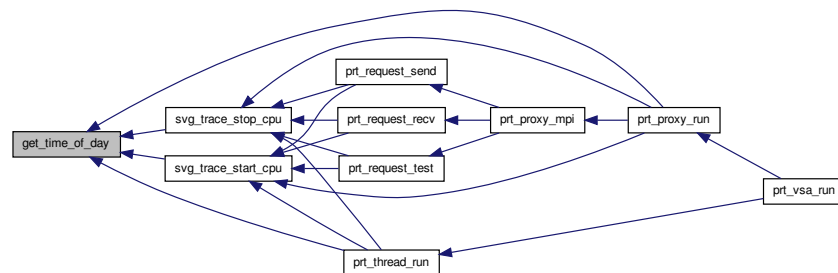
Returns current time.

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)

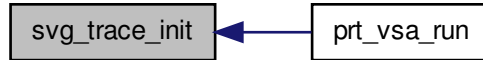
Initializes tracing.

Parameters

<i>num_cores</i>	– The number of cores.
<i>num_devices</i>	– 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*)

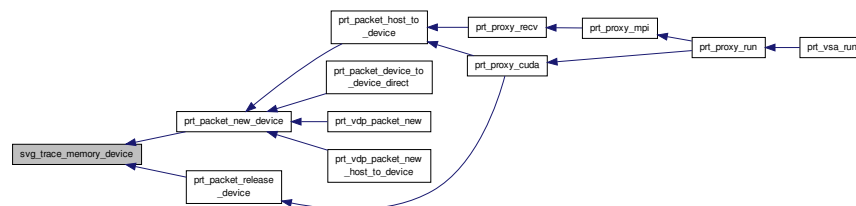
Register device memory usage.

Parameters

<i>delta</i>	– 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*)

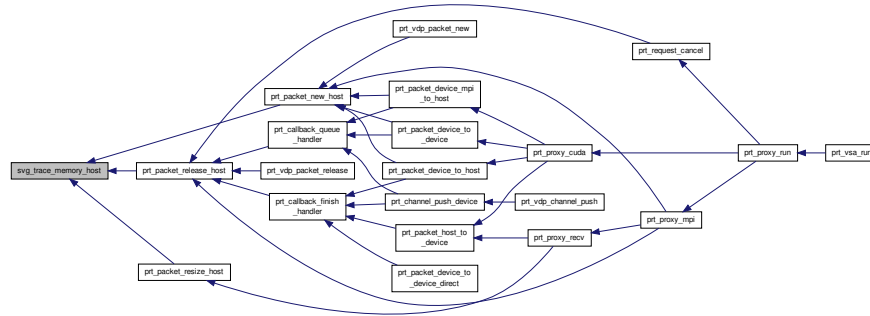
Registers host memory usage. The operation has to be atomic, because it can be invoked by a callback. Because there are two variables to keep track of, the level and the maximum, doing it with atomics is not worth it. Using a spinlock instead.

Parameters

<i>delta</i>	– 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

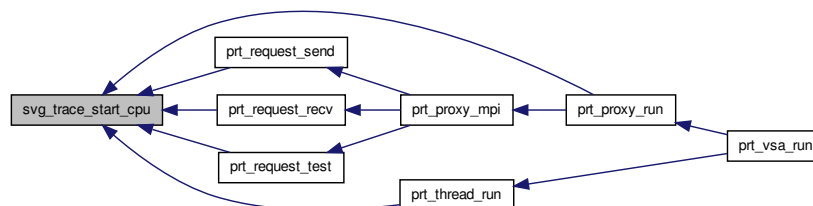
<i>thread_rank</i>	– The rank of the thread.
--------------------	---------------------------

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`)

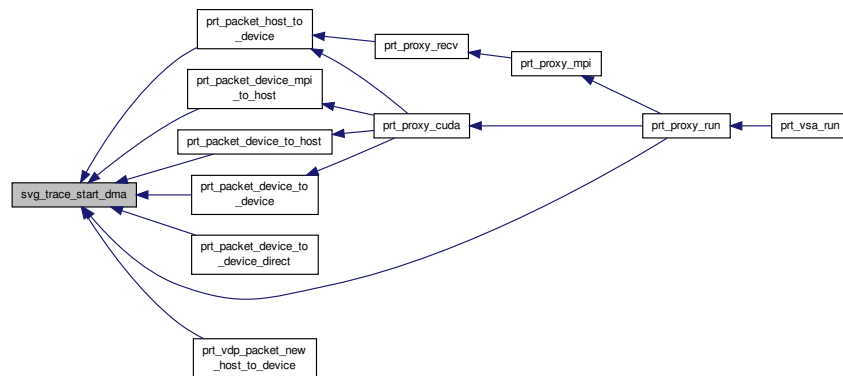
Starts tracing a DMA event.

Parameters

<i>stream</i>	– 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

<i>stream</i>	– 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

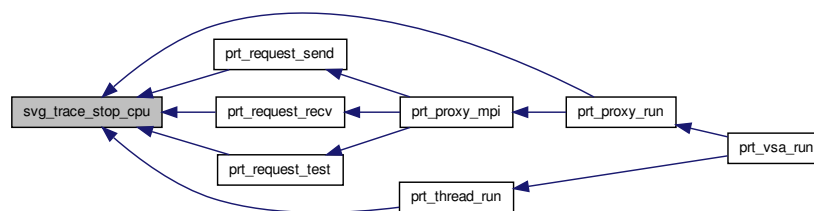
<i>thread_rank</i>	– The rank of the thread.
<i>color</i>	– The RGB color of the SVG box.

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`)

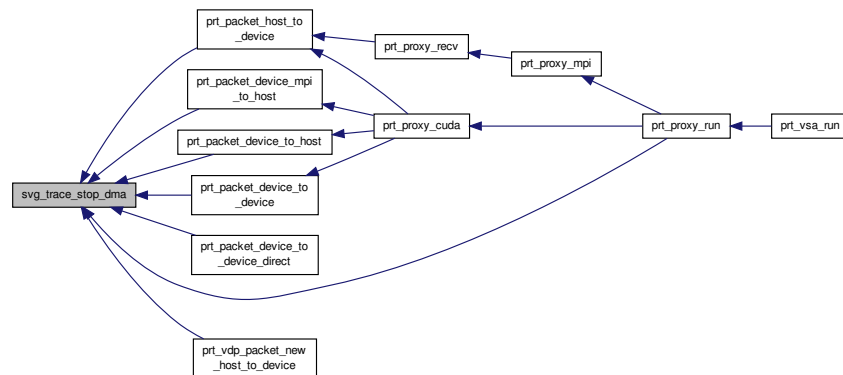
Stops tracing a DMA event.

Parameters

<i>stream</i>	– The stream of the event.
<i>color</i>	– 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)

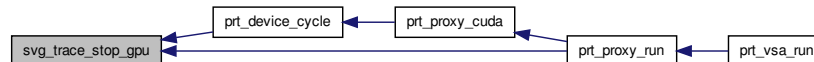
Stops tracing a GPU event.

Parameters

<i>stream</i>	– The stream of the event.
<i>color</i>	– 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

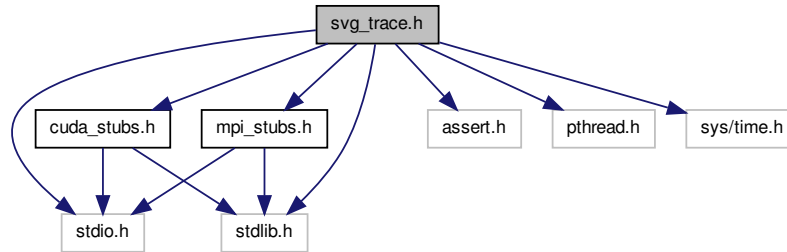
SVG tracing.

```

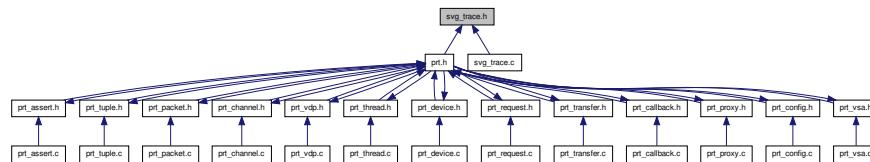
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <pthread.h>
#include <sys/time.h>
#include "mpi_stubs.h"
#include "cuda_stubs.h"

```

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** = 0xFFFFACD,
LightGoldenrodYellow = 0xFAFAD2, **PapayaWhip** = 0xFFEFD5, **Moccasin** = 0xFFE4B5, **PeachPuff** = 0xFFD-

AB9,
PaleGoldenrod = 0xEE8AA, **Khaki** = 0xF0E68C, **DarkKhaki** = 0xBDB76B, **Cornsilk** = 0xFFFF8DC,
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** = 0xFFFFFFFF, **Snow** = 0xFFFFFA, **Honeydew** = 0xF0FFF0,
MintCream = 0xF5FFFA, **Azure** = 0xF0FFFF, **AliceBlue** = 0xF0F8FF, **GhostWhite** = 0xF8F8FF,
WhiteSmoke = 0xF5F5F5, **Seashell** = 0xFFFFEE, **Beige** = 0xF5F5DC, **OldLace** = 0xFDF5E6,
FloralWhite = 0xFFFFF0, **Ivory** = 0FFFFFF0, **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)

Parameters

<i>num_cores</i>	– The number of cores.
<i>num_devices</i>	– 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*)

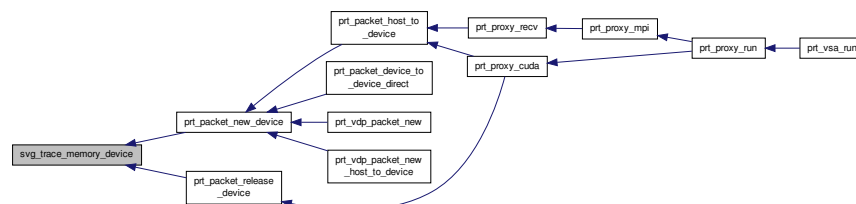
Register device memory usage.

Parameters

<i>delta</i>	– 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*)

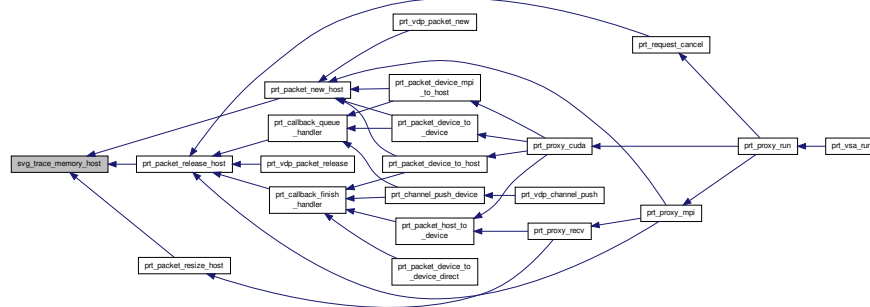
Registers host memory usage. The operation has to be atomic, because it can be invoked by a callback. Because there are two variables to keep track of, the level and the maximum, doing it with atomics is not worth it. Using a spinlock instead.

Parameters

<i>delta</i>	– 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

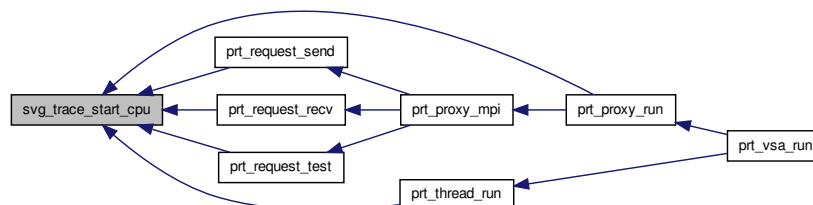
<i>thread_rank</i>	– The rank of the thread.
--------------------	---------------------------

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)

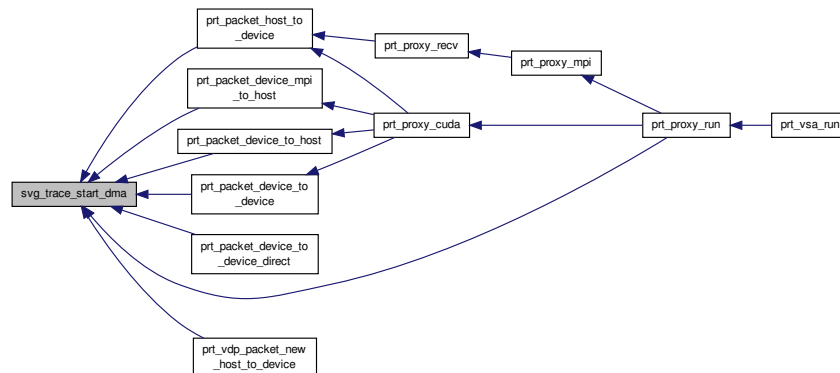
Starts tracing a DMA event.

Parameters

<i>stream</i>	– 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

<i>stream</i>	– 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

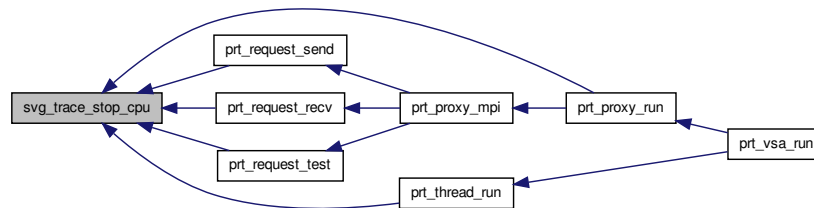
<i>thread_rank</i>	– The rank of the thread.
<i>color</i>	– The RGB color of the SVG box.

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`)

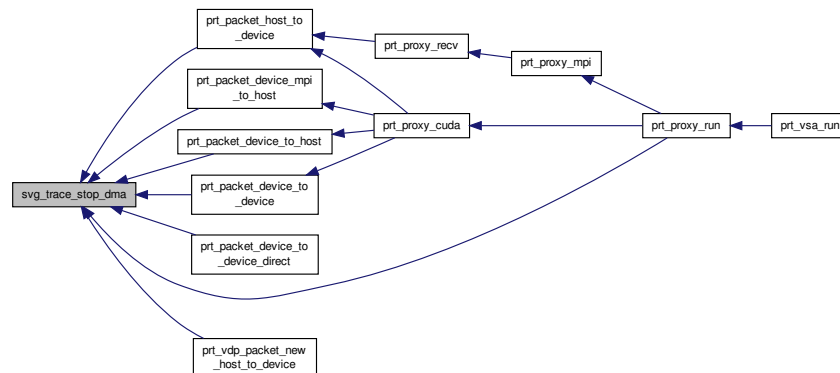
Stops tracing a DMA event.

Parameters

<i>stream</i>	– The stream of the event.
<i>color</i>	– 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

<i>stream</i>	– The stream of the event.
<i>color</i>	– The RGB color of the SVG box.

Definition at line 170 of file `svg_trace.c`.

Here is the caller graph for this function:

