

DiVinE
0.8.0

Generated by Doxygen 1.5.6

Fri Mar 6 09:14:19 2009

Contents

1	Reference Manual	1
2	Error Handling Unit	5
2.1	Overview	6
2.2	Parametrization	6
2.3	Usage	7
2.4	Advanced Usage	8
3	Explicit System	9
3.1	Results of methods for creating successors	10
3.2	Enabled transitions and system transitions	10
3.3	Currently present implementations	11
4	Identifiers used in a symbol table, system and its descendants	13
5	DVE Symbol table	15
6	DVE Symbols	17
6.1	Variables	18
6.2	Process states	18
6.3	Channels	18
6.4	Processes	19
7	System	21
8	Todo List	23
9	Class Index	25
9.1	Class Hierarchy	25

10 Class Index	29
10.1 Class List	29
11 File Index	33
11.1 File List	33
12 Class Documentation	35
12.1 array_of_abstract_t Class Template Reference	35
12.1.1 Detailed Description	37
12.1.2 Member Typedef Documentation	38
12.1.2.1 const_iterator	38
12.1.2.2 iterator	38
12.1.3 Constructor & Destructor Documentation	38
12.1.3.1 array_of_abstract_t	38
12.1.4 Member Function Documentation	38
12.1.4.1 assign_from	38
12.1.4.2 begin	38
12.1.4.3 begin	38
12.1.4.4 clear	39
12.1.4.5 end	39
12.1.4.6 end	39
12.1.4.7 extend	39
12.1.4.8 extend_to	39
12.1.4.9 get_alloc_step	40
12.1.4.10 get_allocated	40
12.1.4.11 last	40
12.1.4.12 last	40
12.1.4.13 pop_back	40
12.1.4.14 push_back	40
12.1.4.15 resize	40
12.1.4.16 set_alloc_step	41
12.1.4.17 shrink_to	41
12.1.4.18 swap	41
12.2 array_t Class Template Reference	42
12.2.1 Detailed Description	44

12.2.2	Member Typedef Documentation	44
12.2.2.1	const_iterator	44
12.2.2.2	iterator	45
12.2.3	Constructor & Destructor Documentation	45
12.2.3.1	array_t	45
12.2.4	Member Function Documentation	45
12.2.4.1	assign_from	45
12.2.4.2	begin	45
12.2.4.3	begin	45
12.2.4.4	clear	45
12.2.4.5	end	46
12.2.4.6	end	46
12.2.4.7	extend	46
12.2.4.8	extend_to	46
12.2.4.9	get_alloc_step	46
12.2.4.10	get_allocated	47
12.2.4.11	last	47
12.2.4.12	last	47
12.2.4.13	pop_back	47
12.2.4.14	push_back	47
12.2.4.15	resize	47
12.2.4.16	set_alloc_step	48
12.2.4.17	shrink_to	48
12.2.4.18	swap	48
12.3	bit_string_t Class Reference	49
12.3.1	Detailed Description	50
12.3.2	Constructor & Destructor Documentation	50
12.3.2.1	bit_string_t	50
12.3.3	Member Function Documentation	50
12.3.3.1	alloc_mem	50
12.3.3.2	clear	51
12.3.3.3	DBG_print	51
12.3.3.4	get_allocated_4bytes_count	51
12.4	bymoc_enabled_trans_t Class Reference	52

12.4.1 Detailed Description	52
12.5 bymoc_explicit_system_t Class Reference	53
12.5.1 Detailed Description	55
12.5.2 Constructor & Destructor Documentation	55
12.5.2.1 bymoc_explicit_system_t	55
12.5.2.2 ~bymoc_explicit_system_t	55
12.5.3 Member Function Documentation	55
12.5.3.1 get_preallocation_count	55
12.5.3.2 is_erroneous	55
12.5.3.3 print_state	56
12.5.3.4 violated_assertion_count	56
12.5.3.5 violated_assertion_string	56
12.5.3.6 violates_assertion	56
12.6 bymoc_expression_t Class Reference	57
12.6.1 Detailed Description	57
12.6.2 Member Function Documentation	58
12.6.2.1 swap	58
12.7 bymoc_process_t Class Reference	59
12.7.1 Detailed Description	60
12.7.2 Member Function Documentation	60
12.7.2.1 add_transition	60
12.8 bymoc_system_t Class Reference	61
12.8.1 Detailed Description	63
12.8.2 Constructor & Destructor Documentation	63
12.8.2.1 bymoc_system_t	63
12.9 bymoc_system_trans_t Class Reference	64
12.9.1 Detailed Description	64
12.10 bymoc_transition_t Class Reference	66
12.10.1 Detailed Description	66
12.11 comm_matrix_t Class Reference	67
12.11.1 Detailed Description	68
12.11.2 Constructor & Destructor Documentation	68
12.11.2.1 comm_matrix_t	68
12.11.3 Member Function Documentation	68

12.11.3.1	getcolcount	68
12.11.3.2	getrowcount	68
12.11.3.3	operator()	69
12.11.4	Friends And Related Function Documentation	69
12.11.4.1	operator*	69
12.11.4.2	operator+	69
12.11.4.3	operator-	70
12.11.4.4	operator-	70
12.12	compacted_t Struct Reference	71
12.12.1	Detailed Description	72
12.12.2	Member Function Documentation	72
12.12.2.1	create_gid	72
12.12.2.2	create_val	72
12.12.2.3	first	72
12.12.2.4	get_arity	72
12.12.2.5	get_gid	73
12.12.2.6	get_operator	73
12.12.2.7	get_value	73
12.12.2.8	join	73
12.12.2.9	last	73
12.12.2.10	left	73
12.12.2.11	right	74
12.12.2.12	to_string	74
12.13	compacted_viewer_t Struct Reference	75
12.13.1	Detailed Description	75
12.14	compressor_t Class Reference	76
12.14.1	Detailed Description	76
12.14.2	Member Function Documentation	76
12.14.2.1	clear	76
12.14.2.2	compress	76
12.14.2.3	decompress	77
12.14.2.4	init	77
12.15	data_t Class Reference	78
12.15.1	Detailed Description	78

12.16	distr_reporter_t Class Reference	79
12.16.1	Detailed Description	80
12.16.2	Member Function Documentation	80
12.16.2.1	collect_and_print	80
12.16.2.2	set_info	80
12.17	distributed_t Class Reference	81
12.17.1	Detailed Description	83
12.17.2	Constructor & Destructor Documentation	83
12.17.2.1	distributed_t	83
12.17.3	Member Function Documentation	83
12.17.3.1	get_all_received_sync_msgs_cnt	83
12.17.3.2	get_all_sent_sync_msgs_cnt	83
12.17.3.3	get_all_sync_barriers_cnt	83
12.17.3.4	get_comm_matrix_rsm	84
12.17.3.5	get_comm_matrix_ssm	84
12.17.3.6	get_proc_msgs_buf_excluseve_mem	84
12.17.3.7	is_manager	84
12.17.3.8	network_initialize	85
12.17.3.9	partition_function	85
12.17.3.10	process_messages	85
12.17.3.11	set_busy	86
12.17.3.12	set_idle	86
12.17.3.13	set_proc_msgs_buf_exclusive_mem	86
12.17.3.14	synchronized	86
12.17.3.15	synchronized	87
12.17.4	Member Data Documentation	87
12.17.4.1	process_user_message	87
12.18	dve_enabled_trans_t Class Reference	89
12.18.1	Detailed Description	89
12.18.2	Constructor & Destructor Documentation	89
12.18.2.1	dve_enabled_trans_t	89
12.18.3	Member Function Documentation	89
12.18.3.1	operator=	89
12.19	dve_explicit_system_t Class Reference	91

12.19.1 Detailed Description	97
12.19.2 Constructor & Destructor Documentation	97
12.19.2.1 dve_explicit_system_t	97
12.19.3 Member Function Documentation	98
12.19.3.1 append_new_enabled	98
12.19.3.2 append_new_enabled_prop_sync	98
12.19.3.3 channel_content_count	98
12.19.3.4 compute_enabled_of_property	99
12.19.3.5 compute_enabled_stage1	99
12.19.3.6 compute_enabled_stage2	99
12.19.3.7 compute_successors_without_sync	99
12.19.3.8 create_error_state	100
12.19.3.9 DBG_print_state	100
12.19.3.10DBG_print_state_CR	101
12.19.3.11eval_expr	101
12.19.3.12get_async_enabled_trans_succ	101
12.19.3.13get_async_enabled_trans_succ_without_property	101
12.19.3.14get_enabled_ith_trans	102
12.19.3.15get_enabled_trans_count	102
12.19.3.16get_ith_succ	102
12.19.3.17get_property_trans	103
12.19.3.18get_receiving_trans	103
12.19.3.19get_sending_or_normal_trans	103
12.19.3.20get_state_creator_value_of_var_type	104
12.19.3.21get_state_of_process	104
12.19.3.22get_succs	104
12.19.3.23get_succs	105
12.19.3.24get_sync_enabled_trans_succ	105
12.19.3.25go_to_error	105
12.19.3.26is_accepting	105
12.19.3.27is_accepting	105
12.19.3.28not_in_glob_conflict	106
12.19.3.29passed_through	106
12.19.3.30read	106

12.19.3.3	read	106
12.19.3.32	set_state_creator_value	107
12.19.3.33	set_state_creator_value_extended	107
12.19.3.34	set_state_creator_value_of_var_type	107
12.19.3.35	set_var_value	108
12.20	dve_explicit_system_t::state_creator_t Struct Reference	109
12.20.1	Detailed Description	109
12.21	dve_expression_t Class Reference	110
12.21.1	Detailed Description	112
12.21.2	Constructor & Destructor Documentation	113
12.21.2.1	~dve_expression_t	113
12.21.2.2	dve_expression_t	113
12.21.2.3	dve_expression_t	113
12.21.2.4	dve_expression_t	113
12.21.3	Member Function Documentation	114
12.21.3.1	arity	114
12.21.3.2	get_ident_gid	114
12.21.3.3	get_operator	114
12.21.3.4	get_value	114
12.21.3.5	set_arity	114
12.21.3.6	set_ident_gid	115
12.21.3.7	set_operator	115
12.21.3.8	set_value	115
12.22	dve_parser_t Class Reference	116
12.22.1	Detailed Description	120
12.22.2	Member Enumeration Documentation	120
12.22.2.1	parser_mode_t	120
12.22.3	Member Function Documentation	121
12.22.3.1	accept_genbuchi_muller_set_complete	121
12.22.3.2	accept_rabin_streett_first_complete	121
12.22.3.3	accept_rabin_streett_pair_complete	121
12.22.3.4	accept_type	121
12.22.3.5	check_restrictions_put_on_property	121
12.22.3.6	expr_array_mem	122

12.22.3.7	expr_assign	122
12.22.3.8	expr_bin	122
12.22.3.9	expr_false	122
12.22.3.10	expr_id	122
12.22.3.11	expr_nat	122
12.22.3.12	expr_parenthesis	123
12.22.3.13	expr_state_of_process	123
12.22.3.14	expr_true	123
12.22.3.15	expr_unary	123
12.22.3.16	expr_var_of_process	123
12.22.3.17	get_expression	123
12.22.3.18	get_mode	123
12.22.3.19	set_fpos	124
12.22.3.20	set_lpos	124
12.22.3.21	state_accept	124
12.22.3.22	state_genbuchi_muller_accept	124
12.22.3.23	state_list_done	124
12.22.3.24	state_rabin_streett_accept	124
12.22.3.25	trans_effect_part	124
12.22.3.26	type_is_const	124
12.22.3.27	type_list_clear	125
12.22.3.28	var_decl_array_size	125
12.22.3.29	var_decl_begin	125
12.22.3.30	var_decl_cancel	125
12.22.3.31	var_decl_create	125
12.22.3.32	var_decl_done	125
12.22.3.33	var_init_field_part	125
12.22.3.34	var_init_is_field	125
12.23	dve_position_t Struct Reference	127
12.23.1	Detailed Description	127
12.24	dve_prob_explicit_system_t Class Reference	128
12.24.1	Detailed Description	128
12.24.2	Constructor & Destructor Documentation	129
12.24.2.1	dve_prob_explicit_system_t	129

12.24.3 Member Function Documentation	129
12.24.3.1 get_succs	129
12.24.3.2 get_succs	129
12.24.3.3 read	130
12.25dve_prob_process_t Class Reference	131
12.25.1 Detailed Description	132
12.26dve_prob_transition_t Class Reference	133
12.26.1 Detailed Description	134
12.27dve_process_decomposition_t Class Reference	135
12.27.1 Detailed Description	135
12.27.2 Member Function Documentation	135
12.27.2.1 get_process_scc_id	135
12.27.2.2 get_process_scc_type	135
12.27.2.3 get_scc_count	136
12.27.2.4 get_scc_type	136
12.27.2.5 is_weak	136
12.27.2.6 parse_process	136
12.28dve_process_t Class Reference	137
12.28.1 Detailed Description	140
12.28.2 Constructor & Destructor Documentation	140
12.28.2.1 dve_process_t	140
12.28.3 Member Function Documentation	140
12.28.3.1 add_assertion	140
12.28.3.2 add_state	140
12.28.3.3 add_transition	140
12.28.3.4 add_variable	141
12.28.3.5 get_acceptance	141
12.28.3.6 get_accepting_group_count	141
12.28.3.7 get_assertion	141
12.28.3.8 get_assertion	141
12.28.3.9 get_trans_count	142
12.28.3.10get_transition	142
12.28.3.11get_transition	142
12.28.3.12get_transition	142

12.28.3.13	get_transition	142
12.28.3.14	set_acceptance	143
12.28.3.15	set_initial_state	143
12.29	dve_source_position_t Class Reference	144
12.29.1	Detailed Description	145
12.29.2	Member Function Documentation	145
12.29.2.1	get_source_pos	145
12.29.2.2	set_source_pos	145
12.30	dve_symbol_t Class Reference	147
12.30.1	Detailed Description	150
12.30.2	Constructor & Destructor Documentation	151
12.30.2.1	dve_symbol_t	151
12.30.3	Member Function Documentation	151
12.30.3.1	create_init_expr_field	151
12.30.3.2	get_channel_item_count	151
12.30.3.3	get_channel_type_list_item	151
12.30.3.4	get_channel_type_list_size	151
12.30.3.5	get_init_expr	152
12.30.3.6	get_init_expr	152
12.30.3.7	get_lid	152
12.30.3.8	get_process_gid	152
12.30.3.9	is_byte	152
12.30.3.10	is_int	153
12.30.3.11	is_no_init_expr_field	153
12.30.3.12	set_channel_item_count	153
12.30.3.13	set_channel_type_list_item	153
12.30.3.14	set_channel_type_list_size	153
12.30.4	Member Data Documentation	153
12.30.4.1	CHANNEL_UNUSED	153
12.31	dve_symbol_table_t Class Reference	154
12.31.1	Detailed Description	156
12.31.2	Constructor & Destructor Documentation	156
12.31.2.1	dve_symbol_table_t	156
12.31.3	Member Function Documentation	156

12.31.3.1	add_process	156
12.31.3.2	find_global_symbol	156
12.31.3.3	find_symbol	156
12.31.3.4	find_visible_symbol	157
12.31.3.5	found_global_symbol	157
12.31.3.6	found_symbol	158
12.31.3.7	get_channel_count	158
12.31.3.8	get_process_count	158
12.31.3.9	get_state_count	158
12.31.3.10	get_symbol	158
12.31.3.11	lget_symbol_count	159
12.31.3.12	get_variable_count	159
12.31.3.13	save_token	159
12.32	dve_system_t Class Reference	160
12.32.1	Detailed Description	164
12.32.2	Constructor & Destructor Documentation	165
12.32.2.1	dve_system_t	165
12.32.3	Member Function Documentation	165
12.32.3.1	add_process	165
12.32.3.2	DBG_print_all_initialized_variables	165
12.32.3.3	eval_expr	165
12.32.3.4	fast_eval	166
12.32.3.5	get_channel_freq_ask	166
12.32.3.6	get_channel_freq_exclaim	166
12.32.3.7	get_global_variable_gid	167
12.32.3.8	get_property_scc_count	167
12.32.3.9	get_property_scc_id	167
12.32.3.10	get_property_scc_type	167
12.32.3.11	lget_property_scc_type	167
12.32.3.12	get_symbol_table	167
12.32.3.13	get_system_synchro	168
12.32.3.14	get_trans_count	168
12.32.3.15	is_property_weak	168
12.32.3.16	set_property_with_synchro	168

12.32.4 Member Data Documentation	168
12.32.4.1 eval_dot	168
12.32.4.2 eval_id	169
12.32.4.3 eval_square_bracket	169
12.32.4.4 initial_values_counts	169
12.32.4.5 parameters	169
12.33dve_system_trans_t Class Reference	170
12.33.1 Detailed Description	171
12.34dve_token_vector_t Class Reference	172
12.34.1 Detailed Description	172
12.35dve_transition_t Class Reference	173
12.35.1 Detailed Description	176
12.35.2 Constructor & Destructor Documentation	177
12.35.2.1 dve_transition_t	177
12.35.3 Member Function Documentation	177
12.35.3.1 alloc_glob_mask	177
12.35.3.2 get_channel_gid	177
12.35.3.3 get_effect	177
12.35.3.4 get_glob_mask	178
12.35.3.5 get_guard	178
12.35.3.6 get_guard	178
12.35.3.7 get_guard_string	178
12.35.3.8 get_partial_id	178
12.35.3.9 get_symbol_table	178
12.35.3.10get_sync_channel_name	179
12.35.3.11get_sync_expr_list_item	179
12.35.3.12get_sync_expr_list_item	179
12.35.3.13get_sync_expr_string	179
12.35.3.14get_valid	180
12.35.3.15set_partial_id	180
12.35.3.16set_process_gid	180
12.36enabled_trans_container_t Class Reference	181
12.36.1 Detailed Description	182
12.36.2 Member Function Documentation	182

12.36.2.1	get_begin	182
12.37	enabled_trans_t Class Reference	183
12.37.1	Detailed Description	183
12.38	ERR_throw_t Struct Reference	184
12.38.1	Detailed Description	184
12.39	ERR_triplet_t Struct Reference	185
12.39.1	Detailed Description	185
12.40	error_string_t Class Reference	186
12.40.1	Detailed Description	186
12.41	error_vector_t Class Reference	187
12.41.1	Detailed Description	189
12.41.2	Constructor & Destructor Documentation	189
12.41.2.1	error_vector_t	189
12.41.2.2	~error_vector_t	189
12.41.3	Member Function Documentation	190
12.41.3.1	flush	190
12.41.3.2	id	190
12.41.3.3	id_front	190
12.41.3.4	operator<<	190
12.41.3.5	operator<<	190
12.41.3.6	operator<<	190
12.41.3.7	operator<<	190
12.41.3.8	operator<<	190
12.41.3.9	operator<<	191
12.41.3.10	perror	191
12.41.3.11	lperror	191
12.41.3.12	perror_back	191
12.41.3.13	perror_back	191
12.41.3.14	perror_front	191
12.41.3.15	perror_front	191
12.41.3.16	pop	191
12.41.3.17	pop	192
12.41.3.18	pop_back	192
12.41.3.19	pop_front	192

12.41.3.20	push	192
12.41.3.21	string	192
12.41.3.22	string_front	192
12.41.3.23	that	193
12.41.3.24	that	193
12.42	ES_parameters_t Struct Reference	194
12.42.1	Detailed Description	194
12.43	explicit_storage_t Class Reference	195
12.43.1	Detailed Description	196
12.43.2	Constructor & Destructor Documentation	196
12.43.2.1	explicit_storage_t	196
12.43.3	Member Function Documentation	196
12.43.3.1	app_by_ref	196
12.43.3.2	delete_all_states	196
12.43.3.3	delete_by_ref	196
12.43.3.4	get_app_by_ref	197
12.43.3.5	get_coltables	197
12.43.3.6	get_ht_occupancy	197
12.43.3.7	get_max_coltable	197
12.43.3.8	get_mem_max_used	197
12.43.3.9	get_mem_used	197
12.43.3.10	get_states_max_stored	197
12.43.3.11	get_states_stored	197
12.43.3.12	init	198
12.43.3.13	insert	198
12.43.3.14	insert	198
12.43.3.15	is_stored	198
12.43.3.16	is_stored	198
12.43.3.17	is_stored_if_not_insert	198
12.43.3.18	reconstruct	199
12.43.3.19	set_app_by_ref	199
12.43.3.20	set_appendix	199
12.43.3.21	set_appendix_size	199
12.43.3.22	set_col_init_size	199

12.43.3.23	set_col_resize	199
12.43.3.24	set_compression_method	199
12.43.3.25	set_hash_function	199
12.43.3.26	set_ht_size	200
12.43.3.27	set_mem_limit	200
12.44	explicit_system_t Class Reference	201
12.44.1	Detailed Description	203
12.44.2	Constructor & Destructor Documentation	203
12.44.2.1	explicit_system_t	203
12.44.3	Member Function Documentation	203
12.44.3.1	can_evaluate_expressions	203
12.44.3.2	can_system_transitions	203
12.44.3.3	eval_expr	203
12.44.3.4	get_enabled_ith_trans	204
12.44.3.5	get_enabled_trans	204
12.44.3.6	get_enabled_trans_count	205
12.44.3.7	get_enabled_trans_succ	205
12.44.3.8	get_enabled_trans_succs	205
12.44.3.9	get_initial_state	206
12.44.3.10	get_preallocation_count	206
12.44.3.11	get_succs	206
12.44.3.12	get_succs	207
12.44.3.13	is_accepting	207
12.44.3.14	is_erroneous	207
12.44.3.15	new_enabled_trans	208
12.44.3.16	violated_assertion_count	208
12.44.3.17	violated_assertion_string	208
12.44.3.18	violates_assertion	209
12.45	expression_t Class Reference	210
12.45.1	Detailed Description	211
12.45.2	Constructor & Destructor Documentation	211
12.45.2.1	expression_t	211
12.45.3	Member Function Documentation	211
12.45.3.1	from_string	211

12.45.3.2	operator=	212
12.45.3.3	read	212
12.45.3.4	to_string	212
12.46	hash_function_t Class Reference	213
12.46.1	Detailed Description	213
12.47	logger_t Class Reference	214
12.47.1	Detailed Description	214
12.47.2	Constructor & Destructor Documentation	214
12.47.2.1	logger_t	214
12.47.2.2	~logger_t	214
12.47.3	Member Function Documentation	215
12.47.3.1	init	215
12.47.3.2	init	215
12.47.3.3	log_now	215
12.47.3.4	register_double	215
12.47.3.5	register_double	216
12.47.3.6	register_int	216
12.47.3.7	register_slong_int	216
12.47.3.8	register_ulong_int	216
12.47.3.9	register_unsigned	216
12.47.3.10	set_storage	217
12.47.3.11	stop_SIGALRM	217
12.47.3.12	use_SIGALRM	217
12.48	message_t Class Reference	218
12.48.1	Detailed Description	220
12.48.2	Constructor & Destructor Documentation	220
12.48.2.1	message_t	220
12.48.2.2	~message_t	221
12.48.3	Member Function Documentation	221
12.48.3.1	append_bool	221
12.48.3.2	append_data	221
12.48.3.3	append_state	221
12.48.3.4	get_written_size	221
12.48.3.5	load_data	222

12.48.3.6 read_data	222
12.48.3.7 read_state	222
12.48.3.8 rewind	222
12.48.3.9 rewind_append	222
12.48.3.10rewind_read	223
12.48.3.11set_data	223
12.48.3.12set_data	223
12.48.3.13set_written_size	223
12.49network_t Class Reference	225
12.49.1 Detailed Description	229
12.49.2 Constructor & Destructor Documentation	229
12.49.2.1 network_t	229
12.49.3 Member Function Documentation	229
12.49.3.1 abort	229
12.49.3.2 all_gather	229
12.49.3.3 barrier	229
12.49.3.4 flush_all_buffers	230
12.49.3.5 flush_all_buffers_timed_out_only	230
12.49.3.6 flush_buffer	231
12.49.3.7 flush_some_buffers	231
12.49.3.8 gather	231
12.49.3.9 get_all_barriers_cnt	232
12.49.3.10get_all_buffers_flushes_cnt	232
12.49.3.11get_all_received_msgs_cnt	233
12.49.3.12get_all_sent_msgs_cnt	233
12.49.3.13get_all_total_pending_size	233
12.49.3.14get_buf_msgs_cnt_limit	234
12.49.3.15get_buf_size_limit	234
12.49.3.16get_buf_time_limit	234
12.49.3.17get_buffer_flushes_cnt	235
12.49.3.18get_cluster_size	235
12.49.3.19get_comm_matrix_rnm	236
12.49.3.20get_comm_matrix_rum	236
12.49.3.21get_comm_matrix_snm	236

12.49.3.22	get_comm_matrix_sum	236
12.49.3.23	get_id	237
12.49.3.24	get_processor_name	237
12.49.3.25	get_rcv_msgs_cnt_rcv_from	237
12.49.3.26	get_rcv_msgs_cnt_rcv_from	238
12.49.3.27	get_sent_msgs_cnt_sent_to	238
12.49.3.28	get_sent_msgs_cnt_sent_to	238
12.49.3.29	get_total_pending_size	239
12.49.3.30	get_user_received_msgs_cnt	239
12.49.3.31	get_user_sent_msgs_cnt	240
12.49.3.32	initialize_buffers	240
12.49.3.33	initialize_network	240
12.49.3.34	is_new_message	241
12.49.3.35	is_new_message_from_source	241
12.49.3.36	is_new_urgent_message	241
12.49.3.37	is_new_urgent_message_from_source	242
12.49.3.38	receive_message	242
12.49.3.39	receive_message	242
12.49.3.40	receive_message_from_source	243
12.49.3.41	receive_message_non_exc	243
12.49.3.42	receive_urgent_message	243
12.49.3.43	receive_urgent_message_from_source	243
12.49.3.44	receive_urgent_message_non_exc	244
12.49.3.45	send_message	244
12.49.3.46	send_message	244
12.49.3.47	send_urgent_message	245
12.49.3.48	send_urgent_message	245
12.50	por_t Class Reference	246
12.50.1	Detailed Description	247
12.50.2	Member Function Documentation	247
12.50.2.1	ample_set	247
12.50.2.2	ample_set_succs	248
12.50.2.3	generate_ample_sets	248
12.50.2.4	generate_composed_ample_sets	249

12.50.2.5 init	250
12.51prob_and_property_trans_t Struct Reference	251
12.51.1 Detailed Description	251
12.51.2 Member Data Documentation	251
12.51.2.1 property_trans_gid	251
12.52prob_explicit_system_t Class Reference	252
12.52.1 Detailed Description	252
12.52.2 Constructor & Destructor Documentation	253
12.52.2.1 prob_explicit_system_t	253
12.52.3 Member Function Documentation	253
12.52.3.1 get_succs	253
12.52.3.2 get_succs	253
12.53prob_process_t Class Reference	255
12.53.1 Detailed Description	256
12.53.2 Constructor & Destructor Documentation	256
12.53.2.1 prob_process_t	256
12.53.2.2 ~prob_process_t	256
12.53.3 Member Function Documentation	256
12.53.3.1 add_prob_transition	256
12.53.3.2 get_prob_transition	256
12.53.3.3 remove_prob_transition	257
12.54prob_succ_container_t Class Reference	258
12.54.1 Detailed Description	258
12.54.2 Constructor & Destructor Documentation	258
12.54.2.1 prob_succ_container_t	258
12.54.2.2 prob_succ_container_t	258
12.55prob_succ_element_t Struct Reference	259
12.55.1 Detailed Description	259
12.56prob_system_t Class Reference	260
12.56.1 Detailed Description	261
12.56.2 Constructor & Destructor Documentation	261
12.56.2.1 prob_system_t	261
12.56.3 Member Function Documentation	261
12.56.3.1 get_index_of_trans_in_prob_trans	261

12.56.3.2	get_prob_trans_of_trans	261
12.57	prob_transition_t Class Reference	263
12.57.1	Detailed Description	264
12.57.2	Member Function Documentation	264
12.57.2.1	get_trans_count	264
12.57.2.2	get_transition	264
12.57.2.3	get_weight_sum	264
12.57.2.4	set_trans_count	265
12.57.2.5	set_transition_and_weight	265
12.58	process_decomposition_t Class Reference	266
12.58.1	Detailed Description	266
12.58.2	Member Function Documentation	266
12.58.2.1	get_process_scc_id	266
12.58.2.2	get_process_scc_type	266
12.58.2.3	get_scc_count	266
12.58.2.4	get_scc_type	267
12.58.2.5	is_weak	267
12.58.2.6	parse_process	267
12.59	process_t Class Reference	268
12.59.1	Detailed Description	270
12.59.2	Constructor & Destructor Documentation	270
12.59.2.1	process_t	270
12.59.2.2	~process_t	270
12.59.3	Member Function Documentation	270
12.59.3.1	add_transition	270
12.59.3.2	from_string	271
12.59.3.3	get_error_vector	271
12.59.3.4	get_trans_count	271
12.59.3.5	get_transition	271
12.59.3.6	get_transition	272
12.59.3.7	read	272
12.59.3.8	remove_transition	272
12.59.3.9	set_error_vector	272
12.59.4	Member Data Documentation	272

12.59.4.1 pproc_terr	272
12.60psh Struct Reference	274
12.60.1 Detailed Description	274
12.61reporter_t Class Reference	275
12.61.1 Detailed Description	276
12.61.2 Member Function Documentation	276
12.61.2.1 get_time	276
12.61.2.2 print	276
12.61.2.3 print	276
12.61.2.4 set_alg_name	276
12.61.2.5 set_file_name	276
12.61.2.6 set_info	276
12.61.2.7 set_obligatory_keys	276
12.61.2.8 set_problem	277
12.61.2.9 set_states_stored	277
12.61.2.10set_succs_calls	277
12.61.2.11start_timer	277
12.61.2.12stop_timer	277
12.62state_ref_t Class Reference	278
12.62.1 Detailed Description	278
12.62.2 Member Function Documentation	278
12.62.2.1 invalidate	278
12.62.2.2 is_valid	278
12.63state_t Struct Reference	279
12.63.1 Detailed Description	279
12.64static_info_t Class Template Reference	280
12.64.1 Detailed Description	280
12.65succ_container_t Class Reference	281
12.65.1 Detailed Description	281
12.65.2 Constructor & Destructor Documentation	281
12.65.2.1 succ_container_t	281
12.65.2.2 succ_container_t	281
12.66SYS_initial_values_t Union Reference	282
12.66.1 Detailed Description	282

12.66.2 Member Data Documentation	282
12.66.2.1 size_t_value	282
12.67SYS_parameters_t Struct Reference	283
12.67.1 Detailed Description	283
12.67.2 Member Data Documentation	283
12.67.2.1 initial_values_counts	283
12.67.2.2 state_lids	283
12.68system_abilities_t Struct Reference	284
12.68.1 Detailed Description	285
12.68.2 Constructor & Destructor Documentation	285
12.68.2.1 system_abilities_t	285
12.69system_t Class Reference	286
12.69.1 Detailed Description	289
12.69.2 Constructor & Destructor Documentation	289
12.69.2.1 system_t	289
12.69.3 Member Function Documentation	289
12.69.3.1 add_process	289
12.69.3.2 can_be_modified	289
12.69.3.3 from_string	289
12.69.3.4 get_abilities	290
12.69.3.5 get_abilities	290
12.69.3.6 get_process	290
12.69.3.7 get_process	290
12.69.3.8 get_process_count	291
12.69.3.9 get_property_gid	291
12.69.3.10get_property_process	291
12.69.3.11get_property_process	291
12.69.3.12get_property_type	292
12.69.3.13get_trans_count	292
12.69.3.14get_transition	292
12.69.3.15get_transition	292
12.69.3.16read	292
12.69.3.17read	293
12.69.3.18remove_process	293

12.69.3.19	set_property_gid	293
12.69.3.20	to_string	293
12.69.3.21	write	294
12.69.3.22	write	294
12.70	system_trans_t Class Reference	295
12.70.1	Detailed Description	296
12.70.2	Member Function Documentation	296
12.70.2.1	get_count	296
12.70.2.2	operator=	296
12.70.2.3	write	296
12.71	thr Struct Reference	297
12.71.1	Detailed Description	297
12.72	timeinfo_t Class Reference	298
12.72.1	Detailed Description	298
12.73	transition_t Class Reference	299
12.73.1	Detailed Description	301
12.73.2	Constructor & Destructor Documentation	301
12.73.2.1	transition_t	301
12.73.3	Member Function Documentation	301
12.73.3.1	can_be_modified	301
12.73.3.2	can_read	301
12.73.3.3	from_string	302
12.73.3.4	get_error_vector	302
12.73.3.5	get_gid	302
12.73.3.6	get_lid	302
12.73.3.7	read	302
12.73.3.8	set_error_vector	303
12.73.3.9	set_gid	303
12.73.3.10	set_lid	303
12.73.3.11	to_string	303
12.73.3.12	write	304
12.73.4	Member Data Documentation	304
12.73.4.1	ptrans_err	304
12.74	updateable_info_t Class Template Reference	305

12.74.1 Detailed Description	305
12.74.2 Member Data Documentation	305
12.74.2.1 const_data	305
12.75updateable_info_t< updateable_data_t, no_const_data_type_t > Class Template Reference	306
12.75.1 Detailed Description	306
12.76vminfo_t Class Reference	307
12.76.1 Detailed Description	307
13 File Documentation	309
13.1 array.hh File Reference	309
13.1.1 Detailed Description	309
13.2 bit_string.hh File Reference	310
13.2.1 Detailed Description	310
13.2.2 Variable Documentation	310
13.2.2.1 bit_values	310
13.3 bymoc_explicit_system.hh File Reference	311
13.3.1 Detailed Description	311
13.4 bymoc_expression.hh File Reference	312
13.4.1 Detailed Description	312
13.5 bymoc_process.hh File Reference	313
13.5.1 Detailed Description	313
13.6 bymoc_process_decomposition.hh File Reference	314
13.6.1 Detailed Description	314
13.7 bymoc_system.hh File Reference	315
13.7.1 Detailed Description	315
13.8 bymoc_system_trans.hh File Reference	316
13.8.1 Detailed Description	316
13.9 bymoc_transition.hh File Reference	317
13.9.1 Detailed Description	317
13.10compressor.hh File Reference	318
13.10.1 Detailed Description	318
13.11data.hh File Reference	319
13.11.1 Detailed Description	319

13.12distr_reporter.hh File Reference	320
13.12.1 Detailed Description	320
13.13distributed.hh File Reference	321
13.13.1 Detailed Description	321
13.14dve_commonparse.hh File Reference	322
13.14.1 Detailed Description	322
13.14.2 Function Documentation	323
13.14.2.1 dve_eerror	323
13.14.2.2 dve_pperror	323
13.14.2.3 dve_tterror	323
13.14.2.4 dve_yyerror	323
13.15dve_explicit_system.hh File Reference	324
13.15.1 Detailed Description	324
13.15.2 Typedef Documentation	324
13.15.2.1 dve_state_int_t	324
13.15.3 Variable Documentation	325
13.15.3.1 ES_FMT_PRINT_ALL_NAMES	325
13.16dve_expression.hh File Reference	326
13.16.1 Detailed Description	326
13.17dve_grammar.hh File Reference	327
13.17.1 Detailed Description	327
13.18dve_parser.hh File Reference	328
13.18.1 Detailed Description	328
13.19dve_prob_explicit_system.hh File Reference	329
13.19.1 Detailed Description	329
13.20dve_prob_process.hh File Reference	330
13.20.1 Detailed Description	330
13.21dve_prob_system.hh File Reference	331
13.21.1 Detailed Description	331
13.22dve_prob_transition.hh File Reference	332
13.22.1 Detailed Description	332
13.23dve_process.hh File Reference	333
13.23.1 Detailed Description	333
13.23.2 Variable Documentation	333

13.23.2.1 DVE_PROCESS_ALLOC_STEP	333
13.24dve_process_decomposition.hh File Reference	334
13.24.1 Detailed Description	334
13.25dve_source_position.hh File Reference	335
13.25.1 Detailed Description	335
13.26dve_symbol_table.hh File Reference	336
13.26.1 Detailed Description	336
13.27dve_system.hh File Reference	337
13.27.1 Detailed Description	337
13.27.2 Enumeration Type Documentation	337
13.27.2.1 system_synchronicity_t	337
13.28dve_system_trans.hh File Reference	338
13.28.1 Detailed Description	338
13.29dve_token_vector.hh File Reference	339
13.29.1 Detailed Description	339
13.30dve_transition.hh File Reference	340
13.30.1 Detailed Description	340
13.30.2 Enumeration Type Documentation	340
13.30.2.1 sync_mode_t	340
13.30.3 Variable Documentation	341
13.30.3.1 TR_effects_alloc_step	341
13.30.3.2 TR_effects_default_alloc	341
13.31error.hh File Reference	342
13.31.1 Detailed Description	343
13.31.2 Define Documentation	343
13.31.2.1 UNIMPLEMENTED	343
13.31.3 Typedef Documentation	344
13.31.3.1 ERR_char_string_t	344
13.31.3.2 ERR_psh_callback_t	344
13.31.3.3 ERR_thr_callback_t	344
13.31.4 Function Documentation	344
13.31.4.1 ERR_default_psh_callback	344
13.31.4.2 ERR_default_thr_callback	345
13.31.5 Variable Documentation	345

13.31.5.1 ERR_UNKNOWN_ID	345
13.31.5.2 ERR_UNKNOWN_TYPE	345
13.31.5.3 gerr	345
13.32explicit_storage.hh File Reference	347
13.32.1 Detailed Description	347
13.33explicit_system.hh File Reference	348
13.33.1 Detailed Description	348
13.33.2 Function Documentation	349
13.33.2.1 succs_deadlock	349
13.33.2.2 succs_error	349
13.33.2.3 succs_normal	349
13.33.3 Variable Documentation	350
13.33.3.1 SUCC_DEADLOCK	350
13.33.3.2 SUCC_ERROR	350
13.33.3.3 SUCC_NORMAL	350
13.34expression.hh File Reference	351
13.34.1 Detailed Description	351
13.35hash_function.hh File Reference	352
13.35.1 Detailed Description	352
13.36huffman.hh File Reference	353
13.36.1 Detailed Description	353
13.37inttostr.hh File Reference	354
13.37.1 Detailed Description	354
13.38logger.hh File Reference	355
13.38.1 Detailed Description	355
13.39mcr12_explicit_system.hh File Reference	356
13.39.1 Detailed Description	356
13.40mcr12_system.hh File Reference	357
13.40.1 Detailed Description	357
13.41network.hh File Reference	358
13.41.1 Detailed Description	359
13.41.2 Typedef Documentation	360
13.41.2.1 pcomm_matrix_t	360
13.41.3 Variable Documentation	360

13.41.3.1 NET_TAG_NORMAL	360
13.41.3.2 NET_TAG_URGENT	360
13.42path.hh File Reference	361
13.42.1 Detailed Description	361
13.42.2 Define Documentation	361
13.42.2.1 PATH_CYCLE_SEPARATOR	361
13.43por.hh File Reference	362
13.43.1 Detailed Description	362
13.44prob_explicit_system.hh File Reference	363
13.44.1 Detailed Description	363
13.45prob_process.hh File Reference	364
13.45.1 Detailed Description	364
13.46prob_system.hh File Reference	365
13.46.1 Detailed Description	365
13.47process.hh File Reference	366
13.47.1 Detailed Description	366
13.48process_decomposition.hh File Reference	367
13.48.1 Detailed Description	367
13.49reporter.hh File Reference	368
13.49.1 Detailed Description	368
13.50state.hh File Reference	369
13.50.1 Detailed Description	370
13.50.2 Function Documentation	371
13.50.2.1 new_state	371
13.50.2.2 set_to_state_pos	371
13.50.2.3 state_pos_to	371
13.51sysopen.hh File Reference	372
13.51.1 Detailed Description	372
13.52system.hh File Reference	373
13.52.1 Detailed Description	373
13.53system_abilities.hh File Reference	374
13.53.1 Detailed Description	374
13.54system_trans.hh File Reference	375
13.54.1 Detailed Description	375

Chapter 1

Reference Manual

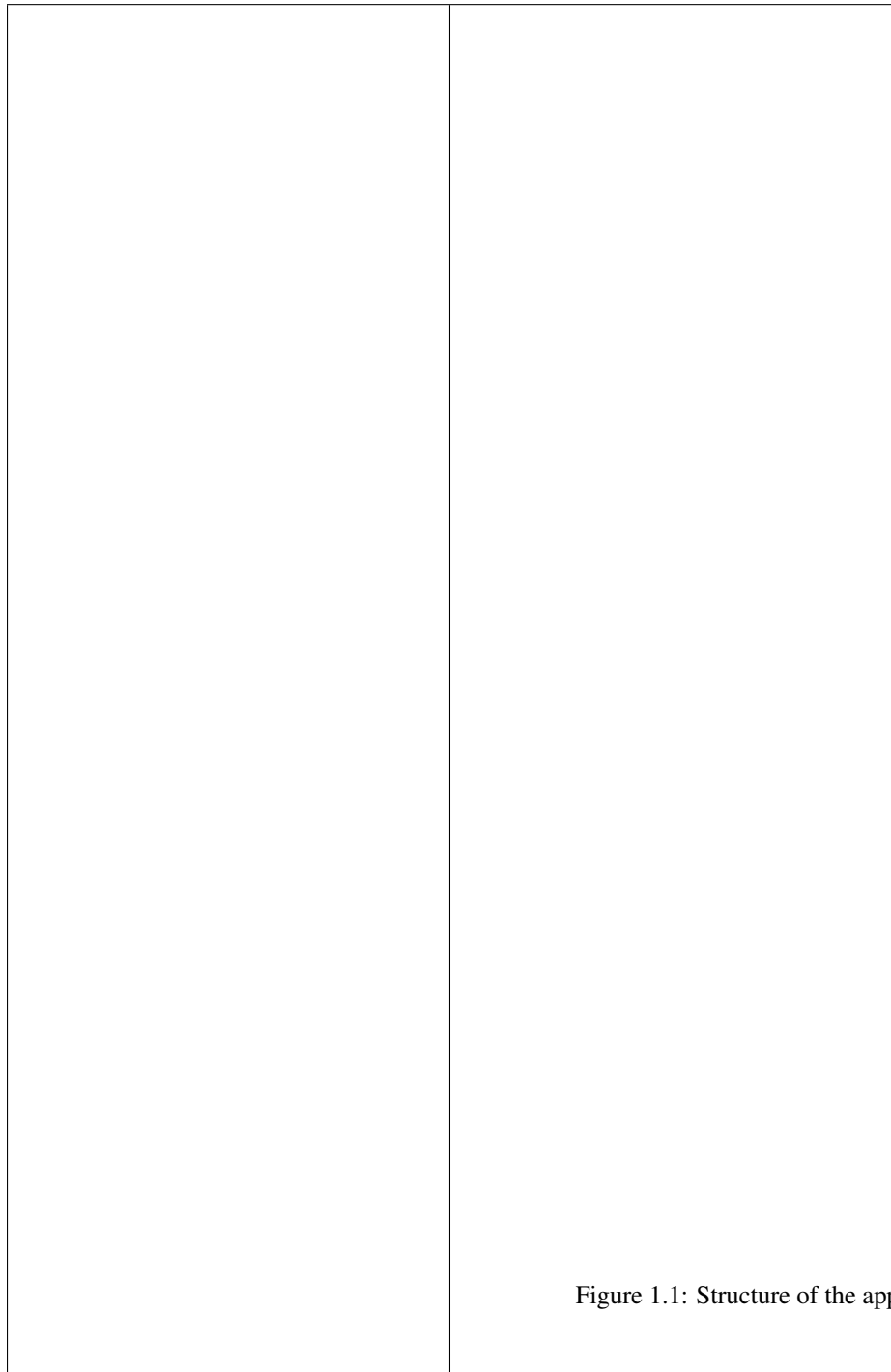
This is a reference manual of DiVinE = Distributed Verification Environment.

DiVinE is an open-source library aimed at an easy implementation of distributed verification algorithms. The big contribution of DiVinE project is its own modularity. Well defined interfaces of separate units allows to the developer to use any of "boxes," that he needs for his application. Furthermore the abstract interface of [system](#) allows to use

DiVinE is **not** a model checker! But there is a support for a LTL model checking algorithms:

- property process in a syntax of DVE source files
- transformation of a LTL formula to the property process

Nevertheless DiVinE itself can't do LTL modelchecking at all. You have to write your own program that uses DiVinE library or use any program, that is based on DiVinE library, but is not the part of DiVinE Library. Currently there are 8 model checking algorithms based of the DiVinE Library. This collection of tools forms so called DiVinE ToolSet.



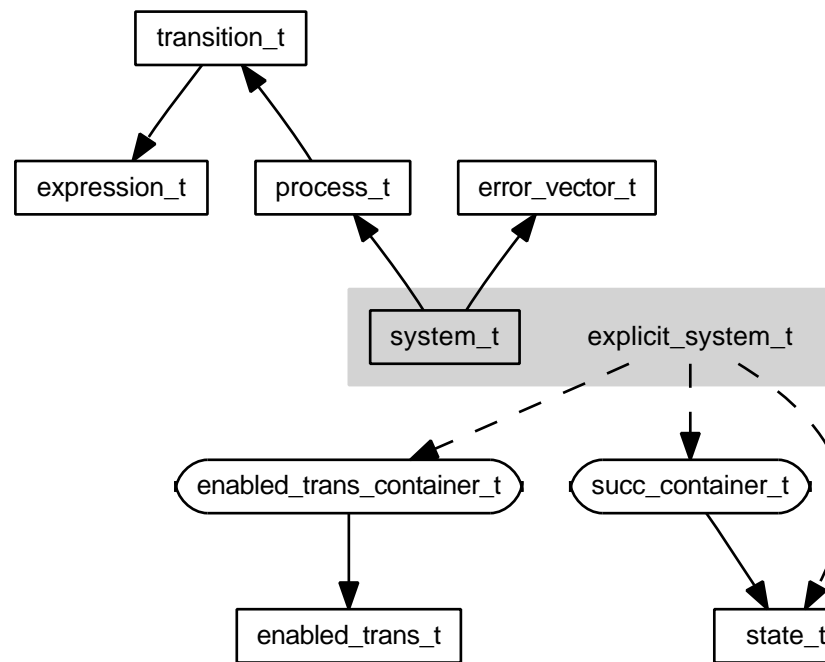


Figure 1.2: Structure of the explicit system and relations with other classes used in the library

The main parts of this project:

- [system](#) - represents the model of the verified system. The system is understood as a system of parallelly running processes (class [system_t](#))
- [explicit system](#) - descendant of [system](#), adds a support for an enumerative model checking, i. e. initial state and its successors generation (class [explicit_system_t](#))

- it is a child class of [system_t](#)
- storage - implementaion of a set of states (class [explicit_storage_t](#))
- distributed layer - communication primitives, barrier synchronization and termination detection (classes [distributed_t](#) and [network_t](#))
- reporter - standardized output of applications (classes [distr_reporter_t](#) and [reporter_t](#))
- POR - sample implementation of partial order reduction - currently only for DVE system (class [por_t](#))
- HW informations unit - monitors resources taken by the application (classes [vminfo_t](#) and [timeinfo_t](#))

For details see [Related Pages](#) and the [Compound List](#).

Another parts of this library:

- [error handling unit](#) - unified unit used to handle errors in whole project

Chapter 2

Error Handling Unit

2.1 Overview

Error Handling Unit is a unit for a facile creating and handling error messages. First it was motivated by the fact that creating a string containing names and current values of variables was not a single-line statement.

Second there was a motivation originating from a parser creation. We sometimes only want to *store* the information about the (possible) problem, but we do not want to stop the syntax analysis. Therefore we need some mechanism where to (temporarily) store a list of found errors (we would rather call such errors as 'warnings').

Third there was a need to be able to manage the messages in a different (but in enclosed parts of program in a unified) way. For example we sometimes need to flush all the error messages immediately to the standard error output. But in a graphical environment we would like to have a list of errors in a separate window (or more windows - each for some part of the system).

The main class that implements these features is `error_vector_t`. Its instances are called **error vectors**.

2.2 Parametrization

2 Classes of Messages

`Error vector` recognizes 2 classes of error messages: warnings and errors. They are maintained in a different ways. Warnings represent such errors, which do not seriously threaten the computation. Errors represent such errors, which are really fatal for the computation and a program have to jump out of such dangerous place in a code to another place where the exceptions are caught.

Error Identifiers and Types

Error messages can have identifiers. These identifiers are numbers of type `ERR_id_t` and they should be unique in your program (therefore you should divide carefully an identifier space between another units). There exists also constant `ERR_UNKNOWN_ID`, that is equal to zero (you can count on it) and represents unspecified `error ID`.

Error messages can also have types. Error types are numbers of type `ERR_throw_t` and they are the numbers passed to the `catch` block by a `throw` statement. This error handling unit sometimes throws such exceptions and you can use error types for example to divide behaviour according to the seriousness of the situation (specified when an exception was called).

Callbacks

In a `error vector` there exist callback functions called 'warning handling callback' and 'error handling callback' (see `ERR_psh_callback_t` and `ERR_thr_callback_t`). Their default behaviour is described in a reference manual to `ERR_default_psh_callback()` and `ERR_default_thr_callback()` functions.

When use use `error_vector_t::set_push_callback()` and `error_vector_t::set_throw_callback()` functions to change the default behaviour of 'warning/error handling callbacks', you preserve some conventions like: 'warning handling callback' will never call `throw`, 'error handling callback' will call `throw` and moreover it will print some messages on the output (terminal, window, ...), etc.

2.3 Usage

The basic usage is the following:

```
gerr << "I'm a mess" << "age" << " nbr. " << 1 << thr(3,13565);
```

This causes creating and storing of message "I'm a message nbr. 1" with **error ID** 13565 and type 3. Then the default 'error handling callback' flushes all messages stored in a memory to the standard error output, clears the memory of messages and calls `throw 3;`.

By the same token:

```
gerr << "I'm a mess" << "age" << " nbr. " << 1 << psh(3,13565);
```

This code causes the end of creation of an error message and message is stored into the list of errors. Then the default 'error handling callback' prints this newly created message to the standard error output and erases a printed message from a memory. Default 'error handling callback' do not use an error type.

Note:

In both - `psh` and `thr`: Parameters error type and **error ID** are optional. If you want to set **error ID** (e.g. 1777) and you do not want to set the error type, feel free to write 0 instead of error type (e.g. `thr(0,1777)`). You can also use predefined constant `ERR_UNKNOWN_TYPE`, that is also equal to zero.

Note 2:

Default 'warning/error handling callbacks do not use a list of errors They always store only 1 message and speedily they remove it from a list immediately after printing to the standard error output.

Catching Exceptions:

If you have set 'error handling callback' to throw exceptions (default behaviour) you should also catch these exceptions. To catch an exception produced by `terr << thr()`, you should use construction

```
try { ... } catch (ERR_throw_t & err_type) { ... }
```

The above code should be a wrapper for all commands of DiVinE, which you call in your program. The simplest way is:

```
int main(int argc, char * argv[])
{
    try
    {

    }
    catch (ERR_throw_t & err_type)
    { return err_type; }
    return 0;
}
```

But you can also create more complex hierarchy of catching exceptions and manipulate them in a various ways. You can also want to move the most of functionality to the 'error handling callback' (that you can set by [error_vector_t::set_throw_callback\(\)](#) function). It really depends on a type of application, where you use this unit.

2.4 Advanced Usage

There are many functions to access the messages in an [error vector](#). For detailed information about complete interface of [error vector](#) see [error_vector_t](#) class reference. Here we will discuss only some classes of these advanced methods:

- 'that' methods - serve to throw a prepared error message (with given [error ID](#) and type)
- 'push' method - inserts a message without calling 'error/warning handling callback'
- 'pop*', 'clear' and 'flush' methods
 - serve to erase selected or all messages from a memory
- 'empty' method - returns whether there exists any stored message
- 'perror*' methods - serve to print a stored message to the standard error output
- 'string*' methods - serve to get a text of a stored message
- 'id*' methods - serve to get an [error ID](#) of a stored message

Chapter 3

Explicit System

Explicit system is the [system](#) extended by a capability of creating successors.

Its abstract interface is the class [explicit_system_t](#) (which is a descendant of [system_t](#)).

It can also compute the [enabled transitions](#) in any state of the system using [explicit_system_t::get_enabled_trans\(\)](#) method.

The basic functionality of this class is the generation of states of the [system](#) using functions [explicit_system_t::get_initial_state\(\)](#) and [explicit_system_t::get_succs\(\)](#). The rest of basic functions are contained in the group of methods called "Obligatory part of abstract interface" in the description of [explicit_system_t](#).

3.1 Results of methods for creating successors

All methods for creating successors of a state of the system (we denote them as `get_succs()` methods) return a bitwise sum of constants `SUCC_NORMAL`, `SUCC_ERROR` and `SUCC_DEADLOCK`. They have the following meaning:

- `SUCC_NORMAL` ... is equal to 0. If `get_succs()` method returns `SUCC_NORMAL`, it means that some successors were generated, if property is not in a deadlock. It also means that there were no registered errors during the computation of successors.
- `SUCC_ERROR` ... if `SUCC_ERROR` is comprised in a return value of `get_succs()` method it means that there was registered at least one error during the computation of successors
- `SUCC_DEADLOCK` ... if `SUCC_DEADLOCK` is comprised in a return value of `get_succs()` method. In the system without property process it means that no successors were generated. In the system with property it means, that the system excluding property is in a deadlock, but property process is not in a deadlock.

Warning:

In the case of a system with a property process successors are generated even when `get_succs()` returns `SUCC_DEADLOCK`

To simply test the return values of `get_succs()` methods you can use inlined functions [succs_normal\(\)](#), [succs_error\(\)](#) and [succs_deadlock\(\)](#).

3.2 Enabled transitions and system transitions

This section is valid only for systems, which are able to work with system transitions and enabled transitions (e. g. [DVE system](#))

System transition is a tuple of process transitions. It can be understood as a product of transitions of processes.

Enabled transition is a system transition + erroneousess of evaluation of guards. It can be understood as a product of transition of processes with satisfied guards in a given

state. Erroneousness is added because of possible errors during an evaluation of guards (e. g. division by 0).

It depends on the system, which system/enabled transitions are valid.

Example 1: In an asynchronous DVE systems synchronized transitions are represented by system/enabled transitions containing 2 transitions. In the case of the system with property process there is one additional transition in each system/enabled transition.

Example 2: In a case of synchronous DVE systems each system/enabled transition contains exactly *process_count* transitions.

For generation of enabled transitions in a given state use a method `get_enabled_trans()`.

To create the successors of all enabled transitions stored in a container of enabled transitions (see [enabled_trans_container_t](#)) you can use the method `get_enabled_trans_succs()`.

For further informations see [explicit_system_t](#).

3.3 Currently present implementations

Currently there are 2 implementations of explicit system present in DiVinE:

- DVE system - full featured implementation
- BYMOC system - very limited implementation - used for future support of Promela modeling language

Chapter 4

Identifiers used in a symbol table, system and its descendants

The identifiers used in DiVinE are always indexes to the array. Therefore if there exist COUNT objects of some type, you can be sure that there exist objects of such type with identifiers 0, 1, 2, ... , (COUNT-1).

- *SID* = symbol identifier = unique ID of symbols (symbol is a channel, a variable, a process state or a process)

Usually you can use it when you work with names of symbols (e. g. when you are searching in a [symbol table](#)).

- *GID* = global identifier = identifier of a particular type of object. It is unique in whole [system](#).

These types of objects have their own GIDs: transitions, processes, process states, variables and channels.

GIDs are unique only for a particular type of object. E. g. there may exist a transition with the same GID as some variable, but there don't exist any two different transitions with the same GID.

- *LID* = local identifier = identifier of a particular type of object. It is unique in a given process.

These types of objects have their own LIDs: transitions, process states and variables.

LIDs are unique only for a particular type of object. E. g. there may exist a transition with the same GID as some variable declared in a same process, but there don't exist any two different transitions with the same LID in the same process. On the other hand in the different processes there may exist two transitions with the same LID.

- *Partial ID* - Special identifier for transitions which identifies uniquely a transition with a particular synchronisation in a given process.

You can obtain this identifier by `transition_t::get_partial_id()` or you can use this identifier in functions `process_t::get_transition(const sync_mode_t sync_mode, const std::size_t trans_nbr)` and `process_t::get_trans_count(const sync_mode_t sync_mode)`.

Maybe you do not need such an identifier, but the generator of successors do.

Chapter 5

DVE Symbol table

Symbol table is an object implemented by class [dve_symbol_table_t](#). Therefore see [dve_symbol_table_t](#) for details.

Symbol table contains all declarations of symbols (see [dve_symbol_t](#) and [DVE Symbols](#)). Symbol is a channel, process, [variable](#) or process state.

Symbol table is contained in a DVE [system](#) ([dve_system_t](#)) and it is created at the same time as [system](#).

Chapter 6

DVE Symbols

There exist 4 types of DVE symbols:

6.1 Variables

Variables can be constant or not. Constant variables do not influence the complexity of computation and they are almost the same as numeral constants. To determine whether the variable is constant you can use method `is_const()`.

There are 2 types of variables

- byte variables - see `VAR_BYTE`
- integer variables - see `VAR_INT`

You can determine the type of variable by method `get_var_type()` or by methods `is_byte()` and `is_int()`.

Variables can be also divided to scalar and vector. You can find out whether a variable is vector by the method `is_vector()`.

Scalar variables

There is a set of methods determined specially for scalar variables. This time there is only the method `get_init_expr()`, which returns an expression initializing the variable.

Vector variables

There is a set of methods determined specially for vector variables. You can get the size of vector using a method `get_vector_size()`. Furthermore there is a set of methods determined to get the initializing expressions of the vector: `get_init_expr_count()`, `get_init_expr(const std::size_t i)`.

6.2 Process states

Process states are the states of a single process. They can be declared only locally in each process. Each process has a special process state `__error__`, which is used for error detection (variable overflow, division by zero etc.).

6.3 Channels

Channels can be declared only globally. There are 2 types of channels:

- channels which pass the value
- channels which do not pass the value See `channel_type_t` for details (in fact there is one more type of channel - unused channel). Use can get the type of channel using the method `get_channel_type()`.

6.4 Processes

Processes can be declared only globally.

Chapter 7

System

In DiVinE word "system" means a simulated model. The system is read in and stored using the class `system_t`, which forms the abstract interface to the system together with classes `process_t`, `transition_t` and `expression_t`.

Class `system_t` itself can do only a little. It contains a list of processes (see `process_t`). To obtain any process you can use the function `system_t::get_process()`. To obtain a count of all processes in a system (including invalid) you can use the function `system_t::get_process_count()`.

Because the class `system_t` is only the abstract interface, there may be many implementations of this interface. The abstract interface is designed in the way, that provides the full access to the structure of the system, if the implementation of the system allows this access. And at the same time it makes only small requirements to the basic well working implementation of the system. This is done using so called "abilities". The developer of an application based on the DiVinE Library should use `system_t::get_abilities()` or `can_*` methods to derive, which features of the system can be used in the application. Even if all `can_*` methods return false, i. e. the system has no advanced abilities, it is still possible to write the model checking algorithm based on such a system. But in that case for example no access to the structure of the system is provided (thus no partial order reduction is possible) etc.

Example 1: DVE system - the full-featured implementation of `system_t` interface

Example 2: BYMOC system - the very limited implementation of `system_t` interface allowing only to read in the system and generate its states

Chapter 8

Todo List

Class `reporter_t` Currently it prints a memory consumption only in the end of the run of the program. It should print a maximum during a run of the program

Chapter 9

Class Index

9.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

array_of_abstract_t	35
array_of_abstract_t< enabled_trans_t, system_new_enabled_trans >	35
enabled_trans_container_t	181
array_t	42
array_t< enabled_trans_t * >	42
array_t< prob_succ_element_t >	42
prob_succ_container_t	258
array_t< state_t >	42
succ_container_t	281
bit_string_t	49
comm_matrix_t	67
compacted_t	71
compacted_viewer_t	75
compressor_t	76
data_t	78
distributed_t	81
dve_explicit_system_t::state_creator_t	109
dve_parser_t	116
dve_position_t	127
dve_source_position_t	144
dve_expression_t	110
dve_transition_t	173
dve_symbol_t	147
dve_symbol_table_t	154
dve_token_vector_t	172
ERR_throw_t	184
ERR_triplet_t	185
error_string_t	186

error_vector_t	187
ES_parameters_t	194
explicit_storage_t	195
expression_t	210
bymoc_expression_t	57
dve_expression_t	110
hash_function_t	213
logger_t	214
message_t	218
network_t	225
por_t	246
prob_and_property_trans_t	251
prob_succ_element_t	259
process_decomposition_t	266
dve_process_decomposition_t	135
process_t	268
bymoc_process_t	59
dve_process_t	137
dve_prob_process_t	131
prob_process_t	255
dve_prob_process_t	131
psh	274
reporter_t	275
distr_reporter_t	79
state_ref_t	278
state_t	279
static_info_t	280
SYS_initial_values_t	282
SYS_parameters_t	283
system_abilities_t	284
system_t	286
bymoc_system_t	61
bymoc_explicit_system_t	53
dve_system_t	160
dve_explicit_system_t	91
dve_prob_explicit_system_t	128
explicit_system_t	201
bymoc_explicit_system_t	53
dve_explicit_system_t	91
prob_explicit_system_t	252
dve_prob_explicit_system_t	128
prob_system_t	260
prob_explicit_system_t	252
system_trans_t	295
bymoc_system_trans_t	64
bymoc_enabled_trans_t	52
dve_system_trans_t	170

dve_enabled_trans_t	89
enabled_trans_t	183
bymoc_enabled_trans_t	52
dve_enabled_trans_t	89
thr	297
timeinfo_t	298
transition_t	299
bymoc_transition_t	66
dve_transition_t	173
prob_transition_t	263
dve_prob_transition_t	133
updateable_info_t	305
updateable_info_t< updateable_data_t, no_const_data_type_t >	306
vminfo_t	307

Chapter 10

Class Index

10.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

array_of_abstract_t (Simple resizable container representing 1-dimensional array)	35
array_t (Simple resizable container representing 1-dimensional array)	42
bit_string_t (Class for impementation of field of bits)	49
bymoc_enabled_trans_t (Class implementing enabled trasion in BYMOC system)	52
bymoc_explicit_system_t	53
bymoc_expression_t (Class representing an expression in BYMOC system)	57
bymoc_process_t (Class representing a process in BYMOC system)	59
bymoc_system_t (Class for Promela/Bytecode system representation)	61
bymoc_system_trans_t (Class implementing system trasion in BYMOC system)	64
bymoc_transition_t (Class representing a transition in BYMOC system)	66
comm_matrix_t (Communication matrix)	67
compacted_t (Class to access compacted expressions)	71
compacted_viewer_t (Structure of a single compacted expression. In the memory block, this initial structure is followed by the left subexpression (if present), and then by right subexpression (if present). r_offset is an offset to the right subexpression)	75
compressor_t (Compression implementation class)	76
data_t (Class representing a general data type)	78
distr_reporter_t	79
distributed_t (Main distributed support class)	81
dve_enabled_trans_t (Class implementing enabled trasion in DVE system)	89
dve_explicit_system_t (Class in DVE system interpretation)	91
dve_explicit_system_t::state_creator_t	109
dve_expression_t (Class representing an expression in DVE system)	110
dve_parser_t (Class that provides an interface to the parsing of DVE sources)	116
dve_position_t (Structure for storing of position in a source code)	127

dve_prob_explicit_system_t (Class in DVE system interpretation)	128
dve_prob_process_t (Class representing a DVE process in probabilistic system)	131
dve_prob_transition_t (Class representing a DVE transition in probabilistic system)	133
dve_process_decomposition_t	135
dve_process_t (Class representing a process)	137
dve_source_position_t (Class for storage of position in a source code)	144
dve_symbol_t	147
dve_symbol_table_t (Class which stores the declaration of symbols (see dve_symbol_t))	154
dve_system_t (Class for a DVE system representation)	160
dve_system_trans_t (Class implementing system transition in DVE system) . .	170
dve_token_vector_t (Class used by dve_symbol_table_t to store the names of symbols)	172
dve_transition_t (Class representing a transition)	173
enabled_trans_container_t (Container determined for storing enabled processes in one state)	181
enabled_trans_t (Class storing informations about one enabled transition) . .	183
ERR_throw_t	184
ERR_triplet_t	185
error_string_t (Class determined for storing error messages)	186
error_vector_t (The main class in error.hh determined for storing)	187
ES_parameters_t (Structure determined for passing parameters to ES_*_eval() functions)	194
explicit_storage_t (Explicit storage class)	195
explicit_system_t	201
expression_t (Abstract interface of a class representing an expression)	210
hash_function_t (Class that unifies hash functions in the library)	213
logger_t	214
message_t (Class representing a data to send or receive using network_t) . .	218
network_t (Network communication support class)	225
por_t (Class for utilization of partial order reduction)	246
prob_and_property_trans_t	251
prob_explicit_system_t	252
prob_process_t	255
prob_succ_container_t (Class determined to store probabilistic successors of some state)	258
prob_succ_element_t (Single element to store in prob_succ_container_t) . .	259
prob_system_t (Abstract interface of a class representing a model of a system)	260
prob_transition_t (Abstract interface of a class representing a probabilistic transition)	263
process_decomposition_t	266
process_t (Abstract interface of a class representing a process of a system) . .	268
psh (Structure determined for causing storing/printing error messages)	274
reporter_t (Class for measuring and reporting)	275
state_ref_t (State reference class)	278
state_t (Structure representing the state of the system)	279
static_info_t (Class used to collect data during synchronization)	280
succ_container_t (Class determined to store successors of some state)	281

SYS_initial_values_t (Internal type for storing initial values of various identifiers)	282
SYS_parameters_t	283
system_abilities_t (Structure storing abilities of system and its subordinate components)	284
system_t (Abstract interface of a class representing a model of a system) . .	286
system_trans_t (Class storing informations about one system transition) . . .	295
thr (Structure determined for causing storing/printing error messages) . . .	297
timeinfo_t (Class for a time measuring)	298
transition_t (Abstract interface of a class representing a transition)	299
updateable_info_t (Class used to collect data during synchronization)	305
updateable_info_t< updateable_data_t, no_const_data_type_t > (Class used to collect data during synchronization)	306
vminfo_t (Class for a memory consumption measuring)	307

Chapter 11

File Index

11.1 File List

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

array.hh	309
array_of_abstract.hh	??
bit_string.hh	310
bymoc_explicit_system.hh	311
bymoc_expression.hh	312
bymoc_process.hh	313
bymoc_process_decomposition.hh	314
bymoc_system.hh	315
bymoc_system_trans.hh	316
bymoc_transition.hh	317
compressor.hh	318
data.hh	319
deb.hh	??
distr_reporter.hh	320
distributed.hh	321
dve_commonparse.hh	322
dve_explicit_system.hh	324
dve_expression.hh	326
dve_flex_lexer.hh	??
dve_grammar.hh	327
dve_gramsympb.hh	??
dve_parser.hh	328
dve_prob_explicit_system.hh	329
dve_prob_process.hh	330
dve_prob_system.hh	331
dve_prob_transition.hh	332
dve_process.hh	333
dve_process_decomposition.hh	334
dve_source_position.hh	335

dve_symbol.hh	??
dve_symbol_table.hh	336
dve_system.hh	337
dve_system_trans.hh	338
dve_token_vector.hh	339
dve_transition.hh	340
error.hh	342
explicit_storage.hh	347
explicit_system.hh	348
expression.hh	351
hash_function.hh	352
huffman.hh	353
inttostr.hh	354
logger.hh	355
mcrl2_explicit_system.hh	356
mcrl2_system.hh	357
mcrl2_system_trans.hh	??
message.hh	??
network.hh	358
path.hh	361
por.hh	362
prob_explicit_system.hh	363
prob_process.hh	364
prob_system.hh	365
prob_transition.hh	??
process.hh	366
process_decomposition.hh	367
reporter.hh	368
spor.hh	??
state.hh	369
sysinfo.hh	??
sysopen.hh	372
system.hh	373
system_abilities.hh	374
system_trans.hh	375
transition.hh	??
types.hh	??
undeb.hh	??

Chapter 12

Class Documentation

12.1 `array_of_abstract_t` Class Template Reference

Simple resizable container representing 1-dimensional array.

```
#include <array_of_abstract.hh>
```

Public Types

- `typedef array_of_abstract_iterator_t< T, const T ** > const_iterator`
Constant iterator.
- `typedef array_of_abstract_iterator_t< T, T ** > iterator`
Iterator.

Public Member Functions

- `array_of_abstract_t (const array_of_abstract_t< T, SingleAlloc > &to_copy)`
A copy constructor.
- `array_of_abstract_t (const void *params, size_int_t allocate=2, size_int_t step=2)`
- `void assign_from (const array_of_abstract_t< T, SingleAlloc > &to_copy)`
Copies 'to_copy' to this container.
- `const T & back () const`
Returns a constant reference to the last item in the container.
- `T & back ()`
Returns a reference to the last item in the container.

- `const_iterator begin ()` const
Returns a constant iterator pointing to the first item of the container.
- `iterator begin ()`
Returns an iterator pointing to the first item of the container.
- `void clear ()`
Lowers the size of array to zero, but does not release allocated memory.
- `const_iterator end ()` const
- `iterator end ()`
- `void extend (const size_int_t count)`
*Extends the container **by** count members.*
- `void extend_to (const size_int_t count)`
Extends the container to the size of 'count' elements.
- `const T & front ()` const
Returns a constant reference to the first item in the container.
- `T & front ()`
Returns a reference to the first item in the container.
- `size_int_t get_alloc_step ()` const
Returns an allocation step.
- `size_int_t get_allocated ()` const
Returns a count of items allocated in a memory of this container.
- `const_iterator last ()` const
Returns a constant iterator pointing to the last item of the container.
- `iterator last ()`
Returns an iterator pointing to the last item of the container.
- `const T & operator[] (const size_int_t i)` const
Returns a constant reference to 'i'-th item in the container.
- `T & operator[] (const size_int_t i)`
Returns a reference to 'i'-th item in the container.
- `T pop_back ()`
Removes the last item from the container.
- `void push_back (T what)`

Appends 'what' to the end of the container.

- void [resize](#) (const size_int_t count)

Resizes the container to the size of 'count' elements.

- void [set_alloc_step](#) (const size_int_t new_alloc_step)

Sets an alloc_step step.

- void [shrink_to](#) (const size_int_t count)

Shrinks the container to the size of 'count' elements.

- size_int_t [size](#) () const

Returns a count of items stored in a container.

- void [swap](#) (array_of_abstract_t< T, SingleAlloc > &second)

Swaps the containment of instance 'second' and this.

- [~array_of_abstract_t](#) ()

A destructor.

Protected Attributes

- T * [field](#)

The entire field of objects stored in [array_t](#).

12.1.1 Detailed Description

template<class T, T (*)(const void *params) SingleAlloc> class array_of_abstract_t< T, SingleAlloc >

Simple resizable container representing 1-dimensional array.

This container reimplements [array_t](#) template for cases of abstract classes. They cannot be normally instantiated, therefore this container stores only pointers to them. In fact it stores pointers to the childs of the abstract class.

This container presumes, that all items have the same (but unknown) type derived from T (where T is a parameter of template). The unknown type is given by the function SingleAlloc given as a second parameter of a template. SingleAlloc creates the instance of the unknown type and returns it in the form of pointer to the abstract class.

This has been especially useful in an implementation of [enabled_trans_container_t](#), derived from [array_of_abstract_t](#). It enables [enabled_trans_container_t](#) to be relatively fast (no redundant allocation and deallocation) and universal for all possible systems.

12.1.2 Member Typedef Documentation

12.1.2.1 `typedef array_of_abstract_iterator_t<T, const T**> const_iterator`

Constant iterator.

Constant iterator - you cannot change the value to which the iterator points. Dereferencing, increasing and decreasing takes time $O(1)$ and it is really fast.

12.1.2.2 `typedef array_of_abstract_iterator_t<T, T**> iterator`

Iterator.

Iterator. Dereferencing, increasing and decreasing takes time $O(1)$ and it is really fast.

12.1.3 Constructor & Destructor Documentation

12.1.3.1 `array_of_abstract_t (const void * params, size_int_t allocate = 2, size_int_t step = 2) [inline]`

A constructor

Parameters:

params = parameters of allocation

12.1.4 Member Function Documentation

12.1.4.1 `void assign_from (const array_of_abstract_t< T, SingleAlloc > & to_copy) [inline]`

Copies 'to_copy' to this container.

Copies the entire contents of one instance of container to another one.

This operation takes both memory and time $O(n)$, where n is a number of items in *to_copy* instance of the container.

12.1.4.2 `const_iterator begin () const [inline]`

Returns a constant iterator pointing to the first item of the container.

It is really fast operation running in time $O(1)$

12.1.4.3 `iterator begin () [inline]`

Returns an iterator pointing to the first item of the container.

It is really fast operation running in time $O(1)$

Referenced by `dve_explicit_system_t::append_new_enabled()`, and `dve_explicit_system_t::get_async_enabled_trans()`.

12.1.4.4 `void clear ()` [inline]

Lowers the size of array to zero, but does not release allocated memory.

Lowers the size of array to zero, but does not release allocated memory. It is the same as `shrink_to(0)`.

Reimplemented in [enabled_trans_container_t](#).

Referenced by `enabled_trans_container_t::clear()`.

12.1.4.5 `const_iterator end () const` [inline]

Returns a constant iterator pointing immediatelly behind the last item of the container

It is really fast operation running in time $O(1)$

12.1.4.6 `iterator end ()` [inline]

Returns an iterator pointing immediatelly behind the last item of the container

It is really fast operation running in time $O(1)$

12.1.4.7 `void extend (const size_int_t count)` [inline]

Extends the container **by** count members.

Parameters:

count = the count of items we want to add to the container

Its running time is $O(n)$, where n is a number of items stored in the container.

Referenced by `dve_explicit_system_t::append_new_enabled()`, `dve_explicit_system_t::append_new_enabled_prop_sync()`, `dve_explicit_system_t::compute_successors_without_sync()`, `por_t::generate_composed_ample_sets()`, `dve_explicit_system_t::get_async_enabled_trans()`, and `dve_explicit_system_t::get_sync_succs_internal()`.

12.1.4.8 `void extend_to (const size_int_t count)` [inline]

Extends the container to the size of 'count' elements.

Warning:

Important: This method presumes, that `count` \geq [size\(\)](#).

Its running time is $O(n)$, where n is a number of items stored in the container.

12.1.4.9 `size_int_t get_alloc_step () const` [inline]

Returns an allocation step.

Returns an allocation step. Allocation step is the least step of allocation of new items (we always allocate the number of items, which is divisible by [get_alloc_step\(\)](#))

12.1.4.10 `size_int_t get_allocated () const` [inline]

Returns a count of items allocated in a memory of this container.

Returns a count of items allocated in a memory of this container. It's return value is always more or equal to return value of [size\(\)](#)

12.1.4.11 `const_iterator last () const` [inline]

Returns a constant iterator pointing to the last item of the container.

It is really fast operation running in time $O(1)$

12.1.4.12 `iterator last ()` [inline]

Returns an iterator pointing to the last item of the container.

It is really fast operation running in time $O(1)$

12.1.4.13 `T pop_back ()` [inline]

Removes the last item from the container.

Removes the last item from the container and returns its value. It doesn't release the memory allocated for the last item (this memory will be reused in the next [push_back\(\)](#)) - therefore it runs in a time $O(1)$ and it is really fast operation.

12.1.4.14 `void push_back (T what)` [inline]

Appends 'what' to the end of the container.

Appends *what* to the end of the container. If necessary it extends the allocated memory. Therefore in that case it runs in a time $O(n)$, where n is a number of items stored in the container.

12.1.4.15 `void resize (const size_int_t count)` [inline]

Resizes the container to the size of 'count' elements.

It is implemented using [shrink_to\(\)](#) and [extend_to\(\)](#) methods. Therefore if *count* \leq [size\(\)](#) it runs in a time $O(1)$ (it uses [shrink_to\(\)](#) method), otherwise it runs in a time

$O(n)$ (it uses [extend_to\(\)](#) method), where n is a number of items stored in the container.

12.1.4.16 void set_alloc_step (const size_int_t new_alloc_step) [inline]

Sets an alloc_step step.

Sets an allocation step. Allocation step is the least step of allocation of new items (we always allocate the number of items, which is divisible by [get_alloc_step\(\)](#))

12.1.4.17 void shrink_to (const size_int_t count) [inline]

Shrinks the container to the size of 'count' elements.

Warning:

Important: This method presumes, that $count \leq size()$.

Its running time is $O(1)$ and it is really fast operation

12.1.4.18 void swap (array_of_abstract_t< T, SingleAlloc > & second) [inline]

Swaps the containment of instance 'second' and this.

Swaps the containment of instance 'second' and this. On one hand (unlike [assign_from\(\)](#)) it changes its parameter, but on the other hand it runs only in $O(1)$ time, what is much faster than the running time of [assign_from\(\)](#)

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

- array_of_abstract.hh

12.2 array_t Class Template Reference

Simple resizable container representing 1-dimensional array.

```
#include <array.hh>
```

Public Types

- typedef const T * [const_iterator](#)
Constant iterator.
- typedef T * [iterator](#)
Iterator.

Public Member Functions

- [array_t](#) (const [array_t](#)< T, Alloc > &to_copy)
A copy constructor.
- [array_t](#) (size_int_t allocate=2, size_int_t step=2)
A constructor.
- void [assign_from](#) (const [array_t](#)< T, Alloc > &to_copy)
Copies 'to_copy' to this container.
- const T & [back](#) () const
Returns a constant reference to the last item in the container.
- T & [back](#) ()
Returns a reference to the last item in the container.
- [const_iterator](#) [begin](#) () const
Returns a constant iterator pointing to the first item of the container.
- [iterator](#) [begin](#) ()
Returns an iterator pointing to the first item of the container.
- void [clear](#) ()
Lowers the size of array to zero, but does not release allocated memory.
- [const_iterator](#) [end](#) () const
- [iterator](#) [end](#) ()
- void [extend](#) (const size_int_t count)
Extends the container by count members.

- void [extend_to](#) (const size_int_t count)
Extends the container to the size of 'count' elements.
- const T & [front](#) () const
Returns a constant reference to the first item in the container.
- T & [front](#) ()
Returns a reference to the first item in the container.
- size_int_t [get_alloc_step](#) () const
Returns an allocation step.
- size_int_t [get_allocated](#) () const
Returns a count of items allocated in a memory of this container.
- const_iterator [last](#) () const
Returns a constant iterator pointing to the last item of the container.
- iterator [last](#) ()
Returns an iterator pointing to the last item of the container.
- const T & [operator\[\]](#) (const size_int_t i) const
Returns a constant reference to 'i'-th item in the container.
- T & [operator\[\]](#) (const size_int_t i)
Returns a reference to 'i'-th item in the container.
- T [pop_back](#) ()
Removes the last item from the container.
- void [push_back](#) (T what)
Appends 'what' to the end of the container.
- void [resize](#) (const size_int_t count)
Resizes the container to the size of 'count' elements.
- void [set_alloc_step](#) (const size_int_t new_alloc_step)
Sets an alloc_step step.
- void [shrink_to](#) (const size_int_t count)
Shrinks the container to the size of 'count' elements.
- size_int_t [size](#) () const
Returns a count of items stored in a container.
- void [swap](#) (array_t< T, Alloc > &second)

Swaps the containment of instance 'second' and this.

- [~array_t\(\)](#)

A destructor.

Protected Attributes

- [T * field](#)

The entire field of objects stored in [array_t](#).

12.2.1 Detailed Description

```
template<class T, T (*)(const size_int_t count) Alloc = default_new_field_of_ -
objects<T>> class array_t< T, Alloc >
```

Simple resizable container representing 1-dimensional array.

This container implement a resizable 1-dimensional array of a choosen type. The access to the single item of an array is implemented simply using operator [].

Constraints imposed on type that can be paramater of this template: Type must not have contructor or destructor. It should be a scalar type like integer, pointer, etc.

You can do reallocation using methods [resize\(\)](#), [shrink_to\(\)](#), [extend_to\(\)](#), [extend\(\)](#) or [push_back\(\)](#).

The purpose of this container is to implement container with really fast random access times to it. The penalty for the really fast read/write operations is a possibly slow reallocation. Reallocation is implemented such way that if the container has not allocated sufficiently large memory, reallocation methods [resize\(\)](#), [extend\(\)](#), [extend_to\(\)](#) or [push_back\(\)](#) allocate a larger piece of memory. It means that [resize\(\)](#), [extend\(\)](#), [extend_to\(\)](#) and [push_back\(\)](#) may have time complexity $O(n)$, where n is a number items in an array. You can influence how often the array will be reallocated using [set_alloc_step\(\)](#) method.

There are defined functions [swap\(\)](#) and [assign_from\(\)](#) to copy the contents of one instace of the container to another instance.

12.2.2 Member Typedef Documentation

12.2.2.1 typedef const T* const_iterator

Constant iterator.

Constant iterator - you cannot change the value to which the iterator points. Dereferencing, increasing and descresing takes time $O(1)$ and it is really fast.

12.2.2.2 typedef T* iterator

Iterator.

Iterator. Dereferencing, increasing and decreasing takes time $O(1)$ and it is really fast.

12.2.3 Constructor & Destructor Documentation

12.2.3.1 array_t(size_int_t allocate = 2, size_int_t step = 2) [inline]

A constructor.

Parameters:

allocate = the number of items to pre-allocate

step = the step of allocation in case of extending the container

12.2.4 Member Function Documentation

12.2.4.1 void assign_from(const array_t< T, Alloc > &to_copy) [inline]

Copies 'to_copy' to this container.

Copies the entire contents of one instance of container to another one.

This operation takes both memory and time $O(n)$, where n is a number of items in *to_copy* instance of the container.

12.2.4.2 const_iterator begin() const [inline]

Returns a constant iterator pointing to the first item of the container.

It is really fast operation running in time $O(1)$

12.2.4.3 iterator begin() [inline]

Returns an iterator pointing to the first item of the container.

It is really fast operation running in time $O(1)$

Referenced by `dve_explicit_system_t::get_sync_succs_internal()`, and `dve_transition_t::~~dve_transition_t()`.

12.2.4.4 void clear() [inline]

Lowers the size of array to zero, but does not release allocated memory.

Lowers the size of array to zero, but does not release allocated memory. It is the same as `shrink_to(0)`.

Referenced by `dve_explicit_system_t::get_async_enabled_trans_succs()`, `dve_prob_explicit_system_t::get_succs()`, `bymoc_explicit_system_t::get_succs()`, `dve_explicit_system_t::get_sync_enabled_trans()`, `dve_explicit_system_t::get_sync_succs_internal()`, `dve_transition_t::read()`, and `dve_prob_transition_t::read()`.

12.2.4.5 `const_iterator end () const` [inline]

Returns a constant iterator pointing immediately behind the last item of the container

It is really fast operation running in time $O(1)$

12.2.4.6 `iterator end ()` [inline]

Returns an iterator pointing immediately behind the last item of the container

It is really fast operation running in time $O(1)$

Referenced by `dve_explicit_system_t::get_sync_succs_internal()`, and `dve_transition_t::~dve_transition_t()`.

12.2.4.7 `void extend (const size_int_t count)` [inline]

Extends the container **by** count members.

Parameters:

count = the count of items we want to add to the container

Its running time is $O(n)$, where n is a number of items stored in the container.

Referenced by `dve_process_t::add_assertion()`, `dve_process_t::add_state()`, and `dve_expression_t::dve_expression_t()`.

12.2.4.8 `void extend_to (const size_int_t count)` [inline]

Extends the container to the size of 'count' elements.

Warning:

Important: This method presumes, that *count* \geq `size()`.

Its running time is $O(n)$, where n is a number of items stored in the container.

12.2.4.9 `size_int_t get_alloc_step () const` [inline]

Returns an allocation step.

Returns an allocation step. Allocation step is the least step of allocation of new items (we always allocate the number of items, which is divisible by `get_alloc_step()`)

Referenced by `array_t< dve_symbol_t * >::assign_from()`.

12.2.4.10 size_int_t get_allocated () const [inline]

Returns a count of items allocated in a memory of this container.

Returns a count of items allocated in a memory of this container. It's return value is always more or equal to return value of [size\(\)](#)

Referenced by `array_t< dve_symbol_t * >::assign_from()`.

12.2.4.11 const_iterator last () const [inline]

Returns a constant iterator pointing to the last item of the container.

It is really fast operation running in time $O(1)$

12.2.4.12 iterator last () [inline]

Returns an iterator pointing to the last item of the container.

It is really fast operation running in time $O(1)$

12.2.4.13 T pop_back () [inline]

Removes the last item from the container.

Removes the last item from the container and returns its value. It doesn't release the memory allocated for the last item (this memory will be reused in the next [push_back\(\)](#)) - therefore it runs in a time $O(1)$ and it is really fast operation.

12.2.4.14 void push_back (T what) [inline]

Appends 'what' to the end of the container.

Appends *what* to the end of the container. If necessary it extends the allocated memory. Therefore in that case it runs in a time $O(n)$, where n is a number of items stored in the container.

Referenced by `dve_symbol_table_t::add_channel()`, `dve_prob_process_t::add_prob_transition()`, `dve_system_t::add_process()`, `dve_symbol_table_t::add_process()`, `dve_symbol_table_t::add_state()`, `dve_process_t::add_state()`, `dve_process_t::add_transition()`, `dve_symbol_table_t::add_variable()`, `por_t::ample_set_succs()`, `dve_explicit_system_t::compute_successors_without_sync()`, `prob_system_t::consolidate()`, `dve_explicit_system_t::get_async_enabled_trans_succs()`, `dve_prob_explicit_system_t::get_succs()`, `dve_explicit_system_t::get_sync_enabled_trans_succs()`, and `dve_explicit_system_t::get_sync_succs_internal()`.

12.2.4.15 void resize (const size_int_t count) [inline]

Resizes the container to the size of 'count' elements.

It is implemented using [shrink_to\(\)](#) and [extend_to\(\)](#) methods. Therefore if $count \leq size()$ it runs in a time $O(1)$ (it uses [shrink_to\(\)](#) method), otherwise it runs in a time $O(n)$ (it uses [extend_to\(\)](#) method), where n is a number of items stored in the container.

Referenced by `dve_expression_t::assign()`, and `prob_transition_t::set_trans_count()`.

12.2.4.16 `void set_alloc_step (const size_int_t new_alloc_step)` [inline]

Sets an `alloc_step` step.

Sets an allocation step. Allocation step is the least step of allocation of new items (we always allocate the number of items, which is divisible by [get_alloc_step\(\)](#))

12.2.4.17 `void shrink_to (const size_int_t count)` [inline]

Shrinks the container to the size of 'count' elements.

Warning:

Important: This method presumes, that $count \leq size()$.

Its running time is $O(1)$ and it is really fast operation

12.2.4.18 `void swap (array_t< T, Alloc > &second)` [inline]

Swaps the containment of instance 'second' and this.

Swaps the containment of instance 'second' and this. On one hand (unlike [assign_from\(\)](#)) it changes its parameter, but on the other hand it runs only in $O(1)$ time, what is much faster than the running time of [assign_from\(\)](#)

Referenced by `dve_expression_t::swap()`.

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

- [array.hh](#)

12.3 bit_string_t Class Reference

Class for impementation of field of bits.

```
#include <bit_string.hh>
```

Public Member Functions

- void [add](#) (const [bit_string_t](#) &from)
*Bitwisely adds 'from' to (*this).*
- void [alloc_mem](#) (const size_int_t bit_count)
Allocation/reallocation method.
- [bit_string_t](#) (const size_int_t bit_count)
A constructor allocating space for 'bit_count' bits.
- [bit_string_t](#) ()
A default constructor.
- [bit_string_t](#) (const [bit_string_t](#) &bit_str2)
A copy constructor.
- void [clear](#) ()
Sets all bits to 0.
- void [DBG_print](#) (std::ostream &outs=cerr) const
- void [disable_bit](#) (const size_int_t i)
Sets 'i'-th bit to 0.
- void [enable_bit](#) (const size_int_t i)
Sets 'i'-th bit to 1.
- size_int_t [get_allocated_4bytes_count](#) () const
Return a count of 4-byte items allocated.
- bool [get_bit](#) (const size_int_t i) const
Returns a value of 'i'-th bit.
- size_int_t [get_bit_count](#) () const
Returns a count of allocated bits.
- byte_t * [get_mem](#) ()
Returns a memory containing array of bits.
- size_int_t [get_mem_size](#) ()

Returns a size of allocated memory in bytes.

- void `invert_bit` (const `size_int_t` i)
Inverts 'i'-th bit.
- `bit_string_t` & `operator=` (const `bit_string_t` &bit_str2)
Copies a content of right side to the `bit_string_t` instance on the left side.
- void `set_bit` (const `size_int_t` i, const bool value)
Sets 'i'-th bit to 'value'.
- `~bit_string_t` ()
A destructor.

Friends

- bool `operator &` (const `bit_string_t` &bs1, const `bit_string_t` &bs2)
Returns true iff ('bs1' & 'bs2') != 0.
- bool `operator ^` (const `bit_string_t` &bs1, const `bit_string_t` &bs2)
Returns true iff ('bs1' xor 'bs2') != 0.
- bool `operator |` (const `bit_string_t` &bs1, const `bit_string_t` &bs2)
Returns true iff ('bs1' | 'bs2') != 0.

12.3.1 Detailed Description

Class for impementation of field of bits.

12.3.2 Constructor & Destructor Documentation

12.3.2.1 `bit_string_t` (const `size_int_t` bit_count) [inline]

A constructor allocating space for 'bit_count' bits.

This is a simple contructor, that automatically allocates a memory for number of bits given in *bit_count*

All bits are initially 0.

12.3.3 Member Function Documentation

12.3.3.1 void `alloc_mem` (const `size_int_t` bit_count) [inline]

Allocation/reallocation method.

This method can be used for initial allocation of memory space (given in count of bits to be allocated - parameter *bitcount*).

All bits are initially 0.

Referenced by `por_t::init()`.

12.3.3.2 void clear () [inline]

Sets all bits to 0.

Warning:

Do not use this function, if no bits are allocated before!

Referenced by `por_t::init()`.

12.3.3.3 void DBG_print (std::ostream & outs = cerr) const [inline]

Prints a sequence of zeros and ones representing a content to output stream 'outs'.

Referenced by `por_t::init()`.

12.3.3.4 size_int_t get_allocated_4bytes_count () const [inline]

Return a count of 4-byte items allocated.

Storage in this class is implemented using allocation of field of 4-byte variables in a memory. This method returns a count of these variables.

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

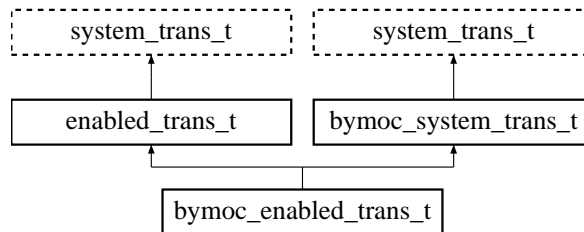
- [bit_string.hh](#)

12.4 bymoc_enabled_trans_t Class Reference

Class implementing enabled trasition in BYMOC [system](#).

```
#include <bymoc_system_trans.hh>
```

Inheritance diagram for bymoc_enabled_trans_t::



Public Member Functions

- virtual [enabled_trans_t](#) & operator= (const [enabled_trans_t](#) &second)

An assignment operator.

12.4.1 Detailed Description

Class implementing enabled trasition in BYMOC [system](#).

BYMOC [system](#) does not support system transitions and enabled transitions. This class is here only for abstract interface compatibility reasons.

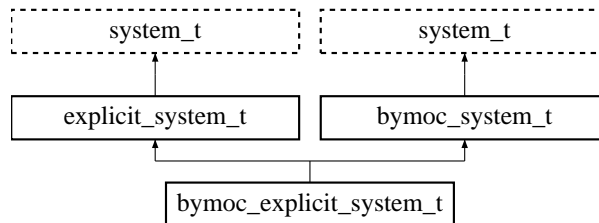
The documentation for this class was generated from the following files:

- [bymoc_system_trans.hh](#)
- [bymoc_system_trans.cc](#)

12.5 bymoc_explicit_system_t Class Reference

```
#include <bymoc_explicit_system.hh>
```

Inheritance diagram for bymoc_explicit_system_t:



Public Member Functions

- [bymoc_explicit_system_t](#) ([error_vector_t](#) &evect)
A constructor.
- [process_decomposition_t](#) * [get_property_decomposition](#) ()
Returns property decomposition, or 0, if subsystem is not available.
- virtual [~bymoc_explicit_system_t](#) ()
A destructor.

Methods for expression evaluation

These methods are not implemented and [can_evaluate_expressions\(\)](#) returns false

- virtual bool [eval_expr](#) (const [expression_t](#) *const expr, const [state_t](#) state, [data_t](#) &data) const
Not implemented in BYMOC [system](#) - throws error message.

Methods working with system transitions and enabled transitions

These methods are not implemented and [can_system_transitions\(\)](#) returns false

- virtual int [get_enabled_ith_trans](#) (const [state_t](#) state, const size_int_t i, [enabled_trans_t](#) &enb_trans)
Not implemented in BYMOC [system](#) - throws error message.
- virtual int [get_enabled_trans](#) (const [state_t](#) state, [enabled_trans_container_t](#) &enb_trans)
Not implemented in BYMOC [system](#) - throws error message.
- virtual int [get_enabled_trans_count](#) (const [state_t](#) state, size_int_t &count)
Not implemented in BYMOC [system](#) - throws error message.

- virtual bool `get_enabled_trans_succ` (const `state_t` state, const `enabled_trans_t` &enabled, `state_t` &new_state)
Not implemented in BYMOC system - throws error message.
- virtual bool `get_enabled_trans_succs` (const `state_t` state, `succ_container_t` &succs, const `enabled_trans_container_t` &enabled_trans)
Not implemented in BYMOC system - throws error message.
- virtual int `get_succs` (`state_t` state, `succ_container_t` &succs, `enabled_trans_container_t` &etc)
Not implemented in BYMOC system - throws error message.
- virtual `enabled_trans_t * new_enabled_trans` () const
Not implemented in BYMOC system - throws error message.

Obligatory part of abstract interface

These methods have to implemented in each implementation of *explicit_system_t*

- virtual `state_t get_initial_state` ()
Implements `explicit_system_t::get_initial_state()` in BYMOC system.
- virtual int `get_ith_succ` (`state_t` state, const int i, `state_t` &succ)
Implements `explicit_system_t::get_ith_succ()` in BYMOC system.
- virtual `size_int_t get_preallocation_count` () const
- virtual `property_type_t get_property_type` ()
Implements `explicit_system_t::get_preproperty_type()`.
- virtual int `get_succs` (`state_t` state, `succ_container_t` &succs)
Implements `explicit_system_t::get_succs()` in BYMOC system.
- virtual bool `is_accepting` (`state_t` state, `size_int_t` acc_group=0, `size_int_t` pair_member=1)
Implements `explicit_system_t::is_accepting()` in BYMOC system.
- virtual bool `is_erroneous` (`state_t` state)
- virtual void `print_state` (`state_t` state, `std::ostream` &outs=std::cout)
- virtual `size_int_t violated_assertion_count` (const `state_t` state) const
Implements `explicit_system_t::violated_assertion_count()` in BYMOC.
- virtual `std::string violated_assertion_string` (const `state_t` state, const `size_int_t` index) const
Implements `explicit_system_t::violated_assertion_string()` in BYMOC.
- virtual bool `violates_assertion` (const `state_t` state) const
Implements `explicit_system_t::violates_assertion()` in BYMOC.

12.5.1 Detailed Description

Class serving for evaluation of possible transitions (of a system given by bytecode source) by the way of explicit state creating.

[bymoc_explicit_system_t](#) is the immediate descendant of a class [system_t](#).

It takes over the system of expression evaluation from [system_t](#). Only for evaluating variables, fields and state identifiers there are defined special functions, which return their value according to a state of system (given by a piece of a memory).

12.5.2 Constructor & Destructor Documentation

12.5.2.1 bymoc_explicit_system_t (error_vector_t & evec)

A constructor.

Parameters:

evec = [error vector](#) used for reporting of error messages

References [system_t::get_abilities\(\)](#), and [system_abilities_t::system_can_decompose_property](#).

12.5.2.2 ~bymoc_explicit_system_t () [virtual]

A destructor.

A destructor.

12.5.3 Member Function Documentation

12.5.3.1 size_int_t get_preallocation_count () const [virtual]

Implements [explicit_system_t::print_state\(\)](#) in BYMOC [system](#), but see also implementation specific notes below

This method always returns 10000. No better estimation is implemented.

Implements [explicit_system_t](#).

12.5.3.2 bool is_erroneous (state_t state) [virtual]

Implements [explicit_system_t::is_erroneous\(\)](#) in BYMOC [system](#), but see also implementation specific notes below

It constantly returns true - till now virtual machine does not support any control of error states (created e. g. by division by zero)

Implements [explicit_system_t](#).

12.5.3.3 `void print_state (state_t state, std::ostream & outs = std::cout)` [virtual]

Implements [explicit_system_t::print_state\(\)](#) in BYMOC [system](#), but see also implementation specific notes below

Implements [explicit_system_t](#).

References `state_t::ptr`.

12.5.3.4 `virtual size_int_t violated_assertion_count (const state_t state) const` [inline, virtual]

Implements [explicit_system_t::violated_assertion_count\(\)](#) in BYMOC.

Currently it only returns 0, because assertions are not supported by BYMOC

Implements [explicit_system_t](#).

References `state_t::ptr`.

12.5.3.5 `virtual std::string violated_assertion_string (const state_t state, const size_int_t index) const` [inline, virtual]

Implements [explicit_system_t::violated_assertion_string\(\)](#) in BYMOC.

Currently it only returns empty string, because assertions are not supported by BYMOC

Implements [explicit_system_t](#).

References `state_t::ptr`.

12.5.3.6 `virtual bool violates_assertion (const state_t state) const` [inline, virtual]

Implements [explicit_system_t::violates_assertion\(\)](#) in BYMOC.

Currently it only returns false, because assertions are not supported by BYMOC

Implements [explicit_system_t](#).

References `state_t::ptr`.

The documentation for this class was generated from the following files:

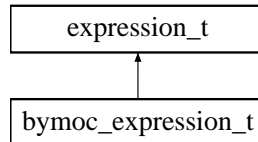
- [bymoc_explicit_system.hh](#)
- [bymoc_explicit_system.cc](#)

12.6 bymoc_expression_t Class Reference

Class representing an expression in BYMOC [system](#).

```
#include <bymoc_expression.hh>
```

Inheritance diagram for bymoc_expression_t::



Public Member Functions

- virtual void [assign](#) (const [expression_t](#) &expr)
Not implemented in BYMOC [system](#) - throws error message.
- **bymoc_expression_t** ([system_t](#) *const system=0)
- virtual int [from_string](#) (std::string &expr_str, const size_int_t process_gid=NO_ID)
Not implemented in BYMOC [system](#) - throws error message.
- virtual int [read](#) (std::istream &istr, size_int_t process_gid=NO_ID)
Not implemented in BYMOC [system](#) - throws error message.
- virtual void [swap](#) ([expression_t](#) &expr)
- virtual std::string [to_string](#) () const
Not implemented in BYMOC [system](#) - throws error message.
- virtual void [write](#) (std::ostream &ostr) const
Not implemented in BYMOC [system](#) - throws error message.
- virtual [~bymoc_expression_t](#) ()
Not implemented in BYMOC [system](#) - throws error message.

12.6.1 Detailed Description

Class representing an expression in BYMOC [system](#).

BYMOC [system](#) does not support expressions. This class is here only for abstract interface compatibility reasons.

12.6.2 Member Function Documentation

12.6.2.1 `void swap (expression_t & expr)` [virtual]

IMPLEMENTATION OF VIRTUAL INTERFACE OF `expression_t` ///// Not implemented in BYMOC `system` - throws error message

Reimplemented from `expression_t`.

References gerr.

The documentation for this class was generated from the following files:

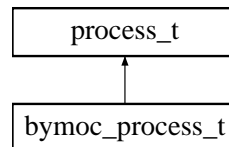
- `bymoc_expression.hh`
- `bymoc_expression.cc`

12.7 bymoc_process_t Class Reference

Class representing a process in BYMOC [system](#).

```
#include <bymoc_process.hh>
```

Inheritance diagram for bymoc_process_t::



Public Member Functions

- virtual void [add_transition](#) ([transition_t](#) *const transition)
Adds new transition to the process.
- **bymoc_process_t** ([system_t](#) *const system)
- virtual int [from_string](#) (std::string &proc_str)
Not implemented in BYMOC [system](#) - throws error message.
- virtual size_int_t [get_trans_count](#) () const
Not implemented in BYMOC [system](#) - throws error message.
- virtual const [transition_t](#) * [get_transition](#) (const size_int_t id) const
Not implemented in BYMOC [system](#) - throws error message.
- virtual [transition_t](#) * [get_transition](#) (const size_int_t id)
Not implemented in BYMOC [system](#) - throws error message.
- virtual int [read](#) (std::istream &istr)
Not implemented in BYMOC [system](#) - throws error message.
- virtual void [remove_transition](#) (const size_int_t transition_gid)
Not implemented in BYMOC [system](#) - throws error message.
- virtual std::string [to_string](#) () const
Not implemented in BYMOC [system](#) - throws error message.
- virtual void [write](#) (std::ostream &ostr) const
Not implemented in BYMOC [system](#) - throws error message.
- virtual [~bymoc_process_t](#) ()
Not implemented in BYMOC [system](#) - throws error message.

12.7.1 Detailed Description

Class representing a process in BYMOC [system](#).

BYMOC [system](#) does not support processes. This class is here only for abstract interface compatibility reasons.

12.7.2 Member Function Documentation

12.7.2.1 `void add_transition(transition_t *const transition)` [virtual]

Adds new transition to the process.

Parameters:

transition = pointer to the transition to add

This method modifies the added transition because it has to set [transition LID](#) and [Partial ID](#).

Implements [process_t](#).

References [gerr](#).

The documentation for this class was generated from the following files:

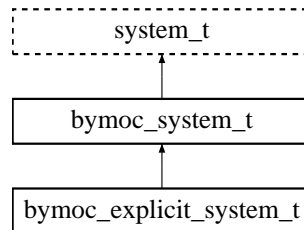
- [bymoc_process.hh](#)
- [bymoc_process.cc](#)

12.8 bymoc_system_t Class Reference

Class for Promela/Bytecode system representation.

```
#include <bymoc_system.hh>
```

Inheritance diagram for bymoc_system_t::



Public Member Functions

- [bymoc_system_t](#) ([error_vector_t](#) &evect=[gerr](#))

A constructor.

- virtual [~bymoc_system_t](#) ()

A destructor.

Methods modifying a system

These methods are not implemented and [can_be_modified\(\)](#) returns false.

- virtual void [add_process](#) ([process_t](#) *const process)
Not imlemented in BYMOC [system](#) - throws error message.
- virtual void [remove_process](#) (const [size_int_t](#) process_id)
Not imlemented in BYMOC [system](#) - throws error message.

Obligatory part of abstact interface

These methods have to implemented in each implementation of [system_t](#)

- virtual [slong_int_t](#) [from_string](#) (const [std::string](#) str)
Warning - this method is still not implemented - TODO.
- virtual [slong_int_t](#) [read](#) (const [char](#) *const filename)
*Implements [system_t::read\(const char * const filename\)](#) in BYMOC [system](#).*
- virtual [slong_int_t](#) [read](#) ([std::istream](#) &ins=[std::cin](#))
Warning - this method is still not implemented - TODO.

- virtual std::string [to_string](#) ()
Warning - this method is still not implemented - TODO.
- virtual void [write](#) (std::ostream &outs=std::cout)
Warning - this method is still not implemented - TODO.
- virtual bool [write](#) (const char *const filename)
Warning - this method is still not implemented - TODO.

Methods working with processes

These methods are not implemented and [can_processes\(\)](#) returns false.

- virtual const [process_t](#) * [get_process](#) (const size_int_t id) const
Not imlemented in BYMOC [system](#) - throws error message.
- virtual [process_t](#) * [get_process](#) (const size_int_t gid)
Not imlemented in BYMOC [system](#) - throws error message.
- virtual size_int_t [get_process_count](#) () const
Not imlemented in BYMOC [system](#) - throws error message.
- virtual property_type_t [get_property_type](#) ()
Not implemented in BYMOC [system](#) - throws error message.

Methods working with property process

These methods are not implemented and [can_property_process\(\)](#) returns false

- virtual size_int_t [get_property_gid](#) () const
Not imlemented in BYMOC [system](#) - throws error message.
- virtual const [process_t](#) * [get_property_process](#) () const
Not imlemented in BYMOC [system](#) - throws error message.
- virtual [process_t](#) * [get_property_process](#) ()
Not imlemented in BYMOC [system](#) - throws error message.
- virtual void [set_property_gid](#) (const size_int_t gid)
Not imlemented in BYMOC [system](#) - throws error message.

Methods working with transitions

These methods are not implemented and [can_transitions\(\)](#) returns false.

- virtual size_int_t [get_trans_count](#) () const
Not imlemented in BYMOC [system](#) - throws error message.
- virtual const [transition_t](#) * [get_transition](#) (size_int_t gid) const
Not imlemented in BYMOC [system](#) - throws error message.
- virtual [transition_t](#) * [get_transition](#) (size_int_t gid)
Not imlemented in BYMOC [system](#) - throws error message.

Protected Attributes

- nipsvm_t nipsvm

Friends

- class bymoc_process_decomposition_t

12.8.1 Detailed Description

Class for Promela/Bytecode system representation.

This class implements the abstract interface [system_t](#)

This implementation is based on external virtual machine for special bytecode. Therefore this [system](#) is called BYMOC [system](#).

It supports only very basic functionality of [system_t](#) interface (processes, transition and expressions are not supported). The calls of non-implemented methods cause error messages.

12.8.2 Constructor & Destructor Documentation

12.8.2.1 bymoc_system_t (error_vector_t & *evect* = gerr)

A constructor.

Parameters:

estack = the **error vector**, that will be used by created instance of [system_t](#)

The documentation for this class was generated from the following files:

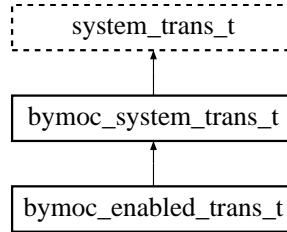
- [bymoc_system.hh](#)
- [bymoc_system.cc](#)

12.9 bymoc_system_trans_t Class Reference

Class implementing system transition in BYMOC [system](#).

```
#include <bymoc_system_trans.hh>
```

Inheritance diagram for bymoc_system_trans_t::



Public Member Functions

- virtual `size_int_t get_count () const`
Not implemented in BYMOC [system](#) - throws error message.
- virtual `system_trans_t & operator= (const system_trans_t &second)`
Not implemented in BYMOC [system](#) - throws error message.
- virtual `transition_t *const & operator[] (const int i) const`
Not implemented in BYMOC [system](#) - throws error message.
- virtual `transition_t *& operator[] (const int i)`
Not implemented in BYMOC [system](#) - throws error message.
- virtual void `set_count (const size_int_t new_count)`
Not implemented in BYMOC [system](#) - throws error message.
- virtual `std::string to_string () const`
Not implemented in BYMOC [system](#) - throws error message.
- virtual void `write (std::ostream &ostr) const`
Not implemented in BYMOC [system](#) - throws error message.

12.9.1 Detailed Description

Class implementing system transition in BYMOC [system](#).

BYMOC [system](#) does not support system transitions and enabled transitions. This class is here only for abstract interface compatibility reasons.

The documentation for this class was generated from the following files:

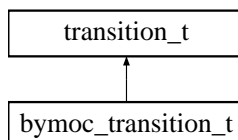
- [bymoc_system_trans.hh](#)
- bymoc_system_trans.cc

12.10 bymoc_transition_t Class Reference

Class representing a transition in BYMOC [system](#).

```
#include <bymoc_transition.hh>
```

Inheritance diagram for bymoc_transition_t:



Public Member Functions

- **bymoc_transition_t** ([system_t](#) *const system)
- virtual int [from_string](#) (std::string &trans_str, const size_int_t process_gid=NO_ID)
Not implemented in BYMOC [system](#) - throws error message.
- virtual int [read](#) (std::istream &istr, size_int_t process_gid=NO_ID)
Not implemented in BYMOC [system](#) - throws error message.
- virtual std::string [to_string](#) () const
Not implemented in BYMOC [system](#) - throws error message.
- virtual void [write](#) (std::ostream &ostr) const
Not implemented in BYMOC [system](#) - throws error message.
- virtual [~bymoc_transition_t](#) ()
Not implemented in BYMOC [system](#) - throws error message.

12.10.1 Detailed Description

Class representing a transition in BYMOC [system](#).

BYMOC [system](#) does not support transitions. This class is here only for abstract interface compatibility reasons.

The documentation for this class was generated from the following files:

- [bymoc_transition.hh](#)
- [bymoc_transition.cc](#)

12.11 comm_matrix_t Class Reference

Communication matrix.

```
#include <network.hh>
```

Public Member Functions

- [comm_matrix_t](#) (int rows, int col, [error_vector_t](#) &arg0=[gerr](#))
A constructor, creates matrix with rows rows and col columns.
- int [getcolcount](#) (void)
Function that returns number of columns in the matrix.
- int [getrowcount](#) (void)
Function that returns number of rows in the matrix.
- int & [operator\(\)](#) (int row, int col)
Operator that retrieves elements of the matrix.
- [~comm_matrix_t](#) ()
A destructor, frees space allocated for the matrix.

Protected Attributes

- [error_vector_t](#) & [errvec](#)

Friends

- [pcomm_matrix_t operator*](#) (const [pcomm_matrix_t](#) &m, int a)
Operator that multiplies matrix m by integer a, analogous to [operator\(\)](#).*
- [pcomm_matrix_t operator*](#) (int a, const [pcomm_matrix_t](#) &m)
Operator that multiplies matrix m by integer a.
- [pcomm_matrix_t operator+](#) (const [pcomm_matrix_t](#) &m1, const [pcomm_matrix_t](#) &m2)
Operator that adds matrix m1 to matrix m2.
- [pcomm_matrix_t operator-](#) (const [pcomm_matrix_t](#) &m1, const [pcomm_matrix_t](#) &m2)
Operator that subtracts matrix m2 from matrix m1 (element-wise).
- [pcomm_matrix_t operator-](#) (const [pcomm_matrix_t](#) &m)

Operator that inverts all elements in the matrix ("puts minus" in front of every element).

12.11.1 Detailed Description

Communication matrix.

Matrix returned by methods of [network_t](#) is usually of size $cl_size * cl_size$, where cl_size is the number of computers in the cluster, which can be retrieved by [network_t::get_cluster_size\(\)](#) method. In i -th row and j -th column of the matrix is an integer that stores information about data sent by workstation with id i to workstation with id j .

12.11.2 Constructor & Destructor Documentation

12.11.2.1 `comm_matrix_t (int rows, int col, error_vector_t & arg0 = gerr)`

A constructor, creates matrix with *rows* rows and *col* columns.

Parameters:

- rows* - number of rows of the matrix to be created
- col* - number of columns of the matrix to be created
- arg0* - object used for error reporting

Use this constructor to create matrix with *rows* rows and *col* columns. If you don't want errors to be handled in a default way, pass your own error handling object of type [error_vector_t](#).

12.11.3 Member Function Documentation

12.11.3.1 `int getcolcount (void) [inline]`

Function that returns number of columns in the matrix.

Returns:

number of columns in the matrix

Use this function to get the number of columns in the matrix.

12.11.3.2 `int getrowcount (void) [inline]`

Function that returns number of rows in the matrix.

Returns:

number of rows in the matrix

Use this function to get the number of rows in the matrix.

12.11.3.3 int & operator() (int row, int col)

Operator that retrieves elements of the matrix.

Parameters:

- row* - specifies the row of the element to be retrieved
- col* - specifies the column of the element to be retrieved

Returns:

value of the retrieved element

Use this operator to retrieve element at position (i, j) in the matrix by passing i as the first parameter and j as the second parameter.

12.11.4 Friends And Related Function Documentation

12.11.4.1 pcomm_matrix_t operator* (int a, const pcomm_matrix_t & m) [friend]

Operator that multiplies matrix m by integer a .

Parameters:

- a - integer that is to be multiplied with all elements of the matrix passed as second parameter
- m - pointer to matrix whose elements are to be multiplied by the integer passed as first parameter

Returns:

pointer to matrix whose elements are elements of m multiplied by a

Use this operator to multiply all elements of matrix by an integer.

12.11.4.2 pcomm_matrix_t operator+ (const pcomm_matrix_t & m1, const pcomm_matrix_t & m2) [friend]

Operator that adds matrix $m1$ to matrix $m2$.

Parameters:

- $m1$ - pointer to matrix, first addend
- $m2$ - pointer to matrix, second addend

Returns:

pointer to sum matrix of $m1$ and $m2$

Use this operator to add two matrices.

12.11.4.3 pcomm_matrix_t operator- (const pcomm_matrix_t & *m1*, const pcomm_matrix_t & *m2*) [`friend`]

Operator that subtracts matrix *m2* from matrix *m1* (element-wise).

Parameters:

m1 - pointer to matrix, minuend

m2 - pointer to matrix, subtrahend

Returns:

pointer to difference matrix between *m1* and *m2*

Use this operator to subtract two matrices.

12.11.4.4 pcomm_matrix_t operator- (const pcomm_matrix_t & *m*) [`friend`]

Operator that inverts all elements in the matrix ("puts minus" in front of every element).

Parameters:

m - pointer to matrix whose elements are to be inverted

Returns:

pointer to matrix with inverted elements

Use this operator to get matrix with all elements inverted.

The documentation for this class was generated from the following files:

- [network.hh](#)
- network.cc

12.12 compacted_t Struct Reference

Class to access compacted expressions.

```
#include <dve_expression.hh>
```

Public Member Functions

- void [create_gid](#) (int _op, size_int_t _gid) const
Constructor – Creates s compacted representation of a unary leaf in the syntax tree.
- void [create_val](#) (int _op, all_values_t _value) const
Creates a compacted representation of a unary leaf in the syntax tree.
- [compacted_viewer_t * first](#) () const
Returns pointer to the first subexpression.
- int [get_arity](#) () const
Returns arity.
- size_int_t [get_gid](#) () const
Returns gid of stored in T_ID, T_SQUARE_BRACKET, or T_DOT leaf.
- int [get_operator](#) () const
Returns operator.
- all_values_t [get_value](#) () const
Returns value of a T_NAT leaf.
- void [join](#) (int _op, [compacted_viewer_t](#) *_left, [compacted_viewer_t](#) *_right) const
Constructor – Joins to compacted expression into one with given operator.
- [compacted_viewer_t * last](#) () const
Returns pointer to the last subexpression.
- [compacted_viewer_t * left](#) () const
Returns pointers to left subexpression.
- [compacted_viewer_t * right](#) () const
Returns pointers to right subexpression.
- std::string [to_string](#) ()
To string.

Public Attributes

- [compacted_viewer_t](#) * ptr

12.12.1 Detailed Description

Class to access compacted expressions.

This class is to view compacted expressions. It uses [compacted_viewer_t](#) to view the memory block pointed by the member pointer.

12.12.2 Member Function Documentation

12.12.2.1 void create_gid (int _op, size_int_t _gid) const

Constructor – Creates s compacted representation of a unary leaf in the syntax tree.

Creates memory block that keeps, in a compacted way, a leaf of type different from T_NAT of the tree of subexpressions.

References compacted_viewer_t::arity, compacted_viewer_t::op, compacted_viewer_t::r_offset, and compacted_viewer_t::size.

Referenced by dve_expression_t::compaction().

12.12.2.2 void create_val (int _op, all_values_t _value) const

Creates a compacted representation of a unary leaf in the syntax tree.

Creates memory block that keeps, in a compacted way, a leaf of type T_NAT of the tree of subexpressions.

References compacted_viewer_t::arity, compacted_viewer_t::op, compacted_viewer_t::r_offset, and compacted_viewer_t::size.

Referenced by dve_expression_t::compaction().

12.12.2.3 compacted_viewer_t* first () const [inline]

Returns pointer to the first subexpression.

Returns pointer to the first subexpression in a given compacted subexpression.

12.12.2.4 int get_arity () const [inline]

Returns arity.

Returns arity of the compacted expression.

12.12.2.5 size_int_t get_gid () const [inline]

Returns gid of stored in T_ID, T_SQUARE_BRACKET, or T_DOT leaf.

Returns gid of stored in T_ID, T_SQUARE_BRACKET, or T_DOT leaf in compacted expression.

Referenced by dve_system_t::fast_eval(), and to_string().

12.12.2.6 int get_operator () const [inline]

Returns operator.

Returns operator of the compacted expression.

Referenced by dve_system_t::fast_eval().

12.12.2.7 all_values_t get_value () const [inline]

Returns value of a T_NAT leaf.

Returns value of a T_NAT leaf in compacted expression.

Referenced by dve_system_t::fast_eval(), and to_string().

12.12.2.8 void join (int _op, compacted_viewer_t * _left, compacted_viewer_t * _right) const

Constructor – Joins to compacted expression into one with given operator.

Creates memory block that keeps, concatenation of given subexpression that is preceded with the connecting connective in a compacted way. of the tree of subexpressions.

References compacted_viewer_t::arity, compacted_viewer_t::op, compacted_viewer_t::r_offset, and compacted_viewer_t::size.

Referenced by dve_expression_t::compaction().

12.12.2.9 compacted_viewer_t* last () const [inline]

Returns pointer to the last subexpression.

Returns pointer to the last subexpression in a given compacted subexpression.

12.12.2.10 compacted_viewer_t* left () const [inline]

Returns pointers to left subexpression.

Returns pointer to the left subexpression in a given compacted subexpression.

Referenced by dve_system_t::fast_eval(), and to_string().

12.12.2.11 compacted_viewer_t* right () const [inline]

Returns pointers to right subexpression.

Returns pointer to the right subexpression in a given compacted subexpression.

Referenced by `dve_system_t::fast_eval()`, and `to_string()`.

12.12.2.12 std::string to_string ()

To string.

Prints compacted expression to string.

References `get_gid()`, `get_value()`, `left()`, `compacted_viewer_t::op`, `ptr`, `right()`, and `to_string()`.

Referenced by `dve_system_t::fast_eval()`, and `to_string()`.

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

- [dve_expression.hh](#)
- `dve_expression.cc`

12.13 compacted_viewer_t Struct Reference

Structure of a single compacted expression. In the memory block, this initial structure is followed by the left subexpression (if present), and then by right subexpression (if present). `r_offset` is an offset to the right subexpression.

```
#include <dve_expression.hh>
```

Public Attributes

- `int arity`
- `int op`
- `int r_offset`
- `int size`

12.13.1 Detailed Description

Structure of a single compacted expression. In the memory block, this initial structure is followed by the left subexpression (if present), and then by right subexpression (if present). `r_offset` is an offset to the right subexpression.

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

- [dve_expression.hh](#)

12.14 compressor_t Class Reference

Compression implementation class.

```
#include <compressor.hh>
```

Public Member Functions

- void **clear** ()
Clears data structures initialized inside compressor.
- bool **compress** (state_t, char *&pointer, int &size)
Compress a state according current compression method.
- bool **decompress** (state_t &, char *pointer, int size)
Decompress the state using current compression method.
- bool **init** (int method, int appendix_size)
Initializes compressor instance.

12.14.1 Detailed Description

Compression implementation class.

12.14.2 Member Function Documentation

12.14.2.1 void clear ()

Clears data structures initialized inside compressor.

Deallocates data that are created during initialization of compressor. This is needed especially for clearing the arena of compressor.

Referenced by explicit_storage_t::init().

12.14.2.2 bool compress (state_t, char *& pointer, int & size)

Compress a state according current compression method.

Returns where the state is compressed and how long the compressed representation is.

Referenced by explicit_storage_t::insert(), explicit_storage_t::is_stored(), and explicit_storage_t::is_stored_if_not_insert().

12.14.2.3 bool decompress (state_t &, char * *pointer*, int *size*)

Decompress the state using current compression method.

Decompress the state from the given pointer and size using current compression method.

Referenced by explicit_storage_t::reconstruct().

12.14.2.4 bool init (int *method*, int *appendix_size*)

Initializes compressor instance.

Requires identification of compression method, extra space that should be allocated at each compressed state (appendix) and pointer to [explicit_system_t](#).

Referenced by explicit_storage_t::init().

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

- [compressor.hh](#)

12.15 data_t Class Reference

Class representing a general data type.

```
#include <data.hh>
```

Public Member Functions

- `template<class T>`
void **assign** (const T &value)

12.15.1 Detailed Description

Class representing a general data type.

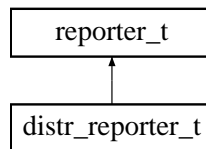
The documentation for this class was generated from the following file:

- [data.hh](#)

12.16 distr_reporter_t Class Reference

```
#include <distr_reporter.hh>
```

Inheritance diagram for distr_reporter_t::



Public Member Functions

- void **collect** ()
- void **collect_and_print** (std::size_t type_of_output=[REPORTER_OUTPUT_NORMAL](#))
Same as [collect_and_print\(\)](#) but prints the report into the standard file.
- void **collect_and_print** (std::size_t type_of_output, std::ostream &out)
Collects informations from workstations.
- [distr_reporter_t](#) ([distributed_t](#) *d)
A constructor.
- void **print** (std::size_t type_of_output, std::ostream &out)
- void **set_global_info** (std::string s, std::string a, const std::string &long_name)
- void [set_global_info](#) (std::string s, std::string a)
Sets global information to report.
- void **set_global_info** (std::string s, double a, const std::string &long_name)
- void [set_global_info](#) (std::string s, double a)
Sets global information to report.
- void **set_info** (std::string s, double a, const std::string &long_name, size_t flag=[REPORTER_MASTER](#))
- void [set_info](#) (std::string s, double a, size_t flag=[REPORTER_MASTER](#))
Sets specific information to report.

Protected Member Functions

- void **_set_pr** (std::ostream &out, double a)
- void **print_specific_value** (std::ostream &out, const std::string label, const divine::size_int_t i)

Protected Attributes

- [distributed_t](#) * [distributed](#)

Pointer to the instance of [distributed_t](#).

- `std::map< std::string, global_info_value_t >` **global_info**
- `std::map< std::string, std::string >` **global_long_name**
- `std::vector< std::vector< double > >` **results**
- `std::map< std::string, std::size_t >` [specific_info_flag](#)

Specific information of the report - adjustable by [set_info\(\)](#).

Static Protected Attributes

- static const `size_int_t` **BASIC_ITEMS** = 6

12.16.1 Detailed Description

Class [distr_reporter_t](#) extends the class [reporter_t](#) with support for distributed computation.

12.16.2 Member Function Documentation

12.16.2.1 `void collect_and_print (std::size_t type_of_output, std::ostream & out)`

Collects informations from workstations.

Master prints it into the given ostream *out*. The *type_of_output* specifies the verbosity of the output.

12.16.2.2 `void set_info (std::string s, double a, size_t flag = REPORTER_MASTER) [inline]`

Sets specific information to report.

The last argumet specifies what should be reported in the case of `REPORTER_OUTPUT_SHORT` (only one value is reported in this case).

The documentation for this class was generated from the following files:

- [distr_reporter.hh](#)
- [distr_reporter.cc](#)

12.17 distributed_t Class Reference

Main distributed support class.

```
#include <distributed.hh>
```

Public Member Functions

- [distributed_t](#) (divine::error_vector_t &arg0=[divine::gerr](#))
- void [finalize](#) ()
Finalizes network. Remember that you can finalize only once.
- void [force_poll](#) (void)
- int [get_all_received_sync_msgs_cnt](#) (void)
Get all received synchronization messages count.
- int [get_all_sent_sync_msgs_cnt](#) (void)
Get all sent synchronization messages count.
- int [get_all_sync_barriers_cnt](#) (void)
Get all synchronization barriers count.
- void [get_comm_matrix_rsm](#) (pcomm_matrix_t &ret, int target)
Get communication matrix of received sync messages.
- void [get_comm_matrix_ssm](#) (pcomm_matrix_t &ret, int target)
Get communication matrix of sent sync messages.
- const message_processor_t * [get_message_processor](#) () const
Returns a message processor.
- void [get_proc_msgs_buf_excluseve_mem](#) (bool &value)
- int [get_state_net_id](#) (divine::state_t &state)
The same as [partition_function\(\)](#), left here for historical reasons.
- void [initialize](#) ()
Completes initialization. Remember that you can initialize only once.
- bool [is_manager](#) () const
Returns, whether this workstation is a manager of the distributed computation.
- void [network_initialize](#) (int &argc, char **&argv)
Network initialization.
- int [partition_function](#) (divine::state_t &state)
Function to determine owner of a state.

- void [process_messages](#) (void)
Function that checks for arrived messages and participates in synchronization.
- void [set_busy](#) (void)
Avoids synchronization.
- void [set_hash_function](#) (hash_functions_t)
Sets hash function to be used for partitioning.
- void [set_idle](#) (void)
Allows synchronization.
- void [set_message_processor](#) (message_processor_t *proc_message)
Sets a message processor.
- void [set_proc_msgs_buf_exclusive_mem](#) (bool value)
- bool [synchronized](#) (abstract_info_t &info)
- bool [synchronized](#) (void)
Function that together with process_messages does synchronization.

Public Attributes

- int [cluster_size](#)
Total number of computers.
- [network_t](#) [network](#)
instance of [network_t](#), you can use it to send/receive messages, etc.
- int [network_id](#)
Unique computer identifier.
- void(* [process_user_message](#))(char *buf, int size, int src, int tag, bool urgent)
Pointer to user-defined function that processes user messages.
- string [processor_name](#)
Name of the computer.

Protected Types

- enum [distr_mode_t](#) { [PRE_INIT](#), [PRE_INIT_NET_OK](#), [NORMAL](#) }

Protected Attributes

- `divine::error_vector_t` & **errvec**
- [hash_function_t](#) **hasher**
- `distr_mode_t` **mode**
- `sync_data_t` **sync_collector**
- `sync_data_t` **sync_one_result**

12.17.1 Detailed Description

Main distributed support class.

The class provides support for common operations necessary for distributed computing such as partition function and termination detection (synchronization).

12.17.2 Constructor & Destructor Documentation

12.17.2.1 `distributed_t` (`divine::error_vector_t` & *arg0* = `divine::gerr`)

A constructor, does't initialize anything!,
non-default error handling vector can be specified

12.17.3 Member Function Documentation

12.17.3.1 `int get_all_received_sync_msgs_cnt` (`void`)

Get all received synchronization messages count.

Returns:

Number of synchronization messages received by the calling workstation.

12.17.3.2 `int get_all_sent_sync_msgs_cnt` (`void`)

Get all sent synchronization messages count.

Returns:

Number of synchronization messages sent by the calling workstation.

12.17.3.3 `int get_all_sync_barriers_cnt` (`void`)

Get all synchronization barriers count.

Returns:

Number of synchronization barriers on the calling workstation.

12.17.3.4 void get_comm_matrix_rsm (pcomm_matrix_t & ret, int target)

Get communication matrix of received sync messages.

Similar to [get_comm_matrix_ssm\(\)](#), but the position (i, j) of the matrix contains the count of synchronization messages received on workstation with id i from workstation with id j .

12.17.3.5 void get_comm_matrix_ssm (pcomm_matrix_t & ret, int target)

Get communication matrix of sent sync messages.

Parameters:

ret - output parameter, pointer to communication matrix (of type [comm_matrix_t](#)) that contains at position (i, j) the count of synchronization messages sent by workstation with id i to workstation with id j

target - only one workstation gets valid pointer ret, this workstation is specified by *target* parameter, which must be the same on all workstations

Returns:

true if the function succeeds, **false** otherwise

Synchronization process requires communication between workstations. Use this function to get the counts of synchronization messages sent by each workstation to each workstation. The function must be called in a way explained for the [network_t::barrier\(\)](#) function.

12.17.3.6 void get_proc_msgs_buf_excluseve_mem (bool & value)

Gets the value of a variable that determines whether the parameter *buf* in [process_user_message\(\)](#) points directly to internal buffers.

true means *buf* points to newly allocated (exclusive) memory, **false** means it points to internal buffers.

false is default, it saves both time and memory.

12.17.3.7 bool is_manager () const [inline]

Returns, whether this workstation is a manager of the distributed computation.

The same as

```
distributed.network_id == NETWORK_ID_MANAGER
```

References NETWORK_ID_MANAGER.

12.17.3.8 void network_initialize (int &argc, char **&argv)

Network initialization.

Parameters:

argc = Number of command line parameters.

Passing the first argument of main is common.

argv = Array of command line parameters.

Passing the second argument of main is common.

Initializes network, since then cluster_size, network_id and processor_name are valid. You can also set send buffer's limits using functions of network. But you still cannot send/receive messages, etc. To complete initialization call the [initialize\(\)](#) function.

References cluster_size, network_t::get_cluster_size(), network_t::get_id(), network_t::get_processor_name(), network_t::initialize_network(), network, network_id, and processor_name.

12.17.3.9 int partition_function (divine::state_t &state)

Function to determine owner of a state.

Parameters:

state = a state, which association with a computer is to be retrieved

Returns:

computer unique identifier of the state owner

Partition function, returns the number of the computer which the state passed as the argument belongs to.

12.17.3.10 void process_messages (void)

Function that checks for arrived messages and participates in synchronization.

This function checks for new messages. If some message arrived, it receives it and calls [process_user_message\(\)](#) function, that's function user must write and give to [distributed_t](#) by assigning it's pointer to [process_user_message\(\)](#).

When called on manager workstation (the one with id NETWORK_ID_MANAGER), it checks whether the workstation is idle (that is no user message was received in this [process_messages\(\)](#) call and user did not call [set_busy\(\)](#)), if yes and user also called [synchronized\(\)](#) function previously, then synchronization process is initiated.

References DIVINE_TAG_SYNC_COMPLETION, DIVINE_TAG_SYNC_ONE, DIVINE_TAG_SYNC_READY, DIVINE_TAG_SYNC_TWO, DIVINE_TAG_USER, network_t::flush_all_buffers_timed_out_only(), network_t::get_all_received_msgs_cnt(), network_t::get_all_sent_msgs_cnt(), network_t::is_new_message(),

`network_t::is_new_urgent_message()`, `network`, `network_id`, `NETWORK_ID_MANAGER`, `process_user_message`, `network_t::receive_message()`, `network_t::receive_message_non_exc()`, `network_t::receive_urgent_message()`, `network_t::receive_urgent_message_non_exc()`, `network_t::send_urgent_message()`, `network_t::stats_num()`, `network_t::stats_Recv_bytes`, and `network_t::stats_Sent_bytes_local`.

12.17.3.11 void set_busy (void)

Avoids synchronization.

This method must be called before `process_messages()` to be effective.

12.17.3.12 void set_idle (void)

Allows synchronization.

This method must be called before `process_messages()` to be effective.

References `network_t::flush_all_buffers()`, `network_t::flush_some_buffers()`, and `network`.

12.17.3.13 void set_proc_msgs_buf_exclusive_mem (bool value)

Determines whether the parameter `buf` in the `process_user_message()` function points directly to internal buffers

Parameters:

value - if **true**, then the *buf* parameter in `process_user_message()` points to exclusive memory and the user is responsible for freeing it. if **false**, then no special memory is allocated and *buf* points directly to internal buffer and user must not free it.

12.17.3.14 bool synchronized (abstract_info_t & info)

Function that together with `process_messages` does synchronization and allows to collect some information during the synchronization process

Similar to `synchronized()` but it allows to collect some information. Let's take updateable info as an example (see `updateable_info_t`). You must create a structure (of type `struct`), attributes of the structure will be used for the collected data. The structure also contains `update()` function, which manipulates with the attributes. Create an instance of `updateable_info_t` and pass the structure as its template parameter (the actual structure used for collected data is accessible via the `updateable_info_t::data` parameter). Pass the instance as the parameter of this function. After all workstations are synchronized, manager workstation (0) sends the contents of the structure to workstation 1, workstation 1 sends it to workstation 2, etc. The last workstation then completes the round. On every workstation the `update()` function is called, which can manipulate with the attributes.

12.17.3.15 bool synchronized (void)

Function that together with process_messages does synchronization.

Returns:

true if workstations are synchronized.

It works in the following way. By first call to this function (on all workstations in the cluster) you are telling that you want to perform some distributed computation and at the end you want to synchronize. During the computation you send messages using [network_t::send_message\(\)](#) or [network_t::send_urgent_message\(\)](#). To receive message use the [process_messages\(\)](#) function which calls [process_user_message\(\)](#), where you can do something with the received message. By subsequent calls to [synchronized\(\)](#) you determine if all workstations finished their work. In distributed computation, work mostly implies sending messages, but if you want to do some bigger amount of work without sending messages and you don't want to synchronize, use the [set_busy\(\)](#) function. Don't forget to call [set_idle\(\)](#) function when your work is finished and remember to call them before [process_messages\(\)](#). [synchronized\(\)](#) and [process_messages\(\)](#) should be called quite often to make the computation efficient.

Warning:

Synchronization is performed in [process_messages\(\)](#) method.

References [network_t::barrier\(\)](#), [DIVINE_TAG_SYNC_READY](#), and [network](#).

12.17.4 Member Data Documentation

12.17.4.1 void(* process_user_message)(char *buf, int size, int src, int tag, bool urgent)

Pointer to user-defined function that processes user messages.

Parameters:

- buf* - pointer to message data, it points either to newly allocated memory or to internal buffers (see [set_proc_msgs_buf_exclusive_mem\(\)](#))
- size* - size of the received message
- src* - id of the sender workstation
- tag* - tag of the message (Remember that all user messages must have tag greater or equal to [DIVINE_TAG_USER](#)).
- urgent* - specifies whether the received message is urgent or not

To be able to use synchronization together with sending your own messages, you must write this function and assign it's pointer to [process_user_message\(\)](#).

Referenced by [process_messages\(\)](#).

The documentation for this class was generated from the following files:

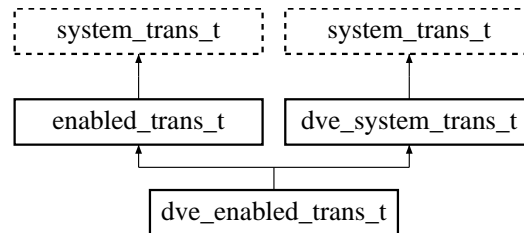
- [distributed.hh](#)
- distributed.cc

12.18 dve_enabled_trans_t Class Reference

Class implementing enabled transition in DVE [system](#).

```
#include <dve_system_trans.hh>
```

Inheritance diagram for dve_enabled_trans_t::



Public Member Functions

- [dve_enabled_trans_t](#) (const [dve_enabled_trans_t](#) &second)
A copy constructor.
- [dve_enabled_trans_t](#) ()
- virtual [enabled_trans_t](#) & operator= (const [enabled_trans_t](#) &second)

12.18.1 Detailed Description

Class implementing enabled transition in DVE [system](#).

12.18.2 Constructor & Destructor Documentation

12.18.2.1 dve_enabled_trans_t () [inline]

CONSTRUCTORS /// A constructor

12.18.3 Member Function Documentation

12.18.3.1 enabled_trans_t & operator= (const enabled_trans_t & second) [virtual]

VIRTUAL INTERFACE OF ENABLED_TRANS_T /// Implements [enabled_trans_t](#) operator=() in DVE [system](#)

Implements [enabled_trans_t](#).

The documentation for this class was generated from the following files:

- [dve_system_trans.hh](#)

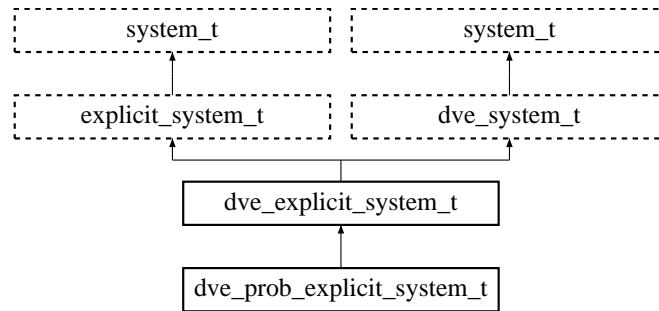
- `dve_system_trans.cc`

12.19 dve_explicit_system_t Class Reference

Class in DVE system interpretation.

```
#include <dve_explicit_system.hh>
```

Inheritance diagram for dve_explicit_system_t::



Public Member Functions

- [dve_explicit_system_t](#) ([error_vector_t](#) &evect=[gerr](#))
A constructor.
- [size_int_t](#) [get_channel_pos](#) (const [size_int_t](#) gid) const
Returns a position of channel in a state of the system with *process GID* 'gid'.
- [size_int_t](#) [get_process_pos](#) (const [size_int_t](#) gid) const
Returns a position of process in a state of the system with *process GID* 'gid'.
- [size_int_t](#) [get_process_size](#) (const [size_int_t](#) gid) const
Returns a size of process in a state of the system with *process GID* 'gid'.
- [size_int_t](#) [get_var_pos](#) (const [size_int_t](#) gid) const
Returns a position of variable in a state of the system with *process GID* 'gid'.
- virtual [~dve_explicit_system_t](#) ()
A destructor.

DVE system specific methods

These methods are implemented only in DVE system and they cannot be run using an abstract interface of [system_t](#).

- void [DBG_print_state](#) ([state_t](#) state, [std::ostream](#) &outs=[std::cerr](#), const [ulong_int_t](#) format=[ES_FMT_PRINT_ALL_NAMES](#))
Prints a state to output stream 'outs' formatted according to 'format'.

- void `DBG_print_state_CR` (`state_t` state, `std::ostream` &outs=`std::cerr`, `const` `ulong_int_t` format=`ES_FMT_PRINT_ALL_NAMES`)
- `all_values_t eval_expr` (`const dve_expression_t` *const expr, `const state_t` state, `bool` &eval_err) `const`
Evaluates the expression in the context of the state of [explicit system](#).
- `dve_transition_t` * `get_property_trans` (`const system_trans_t` &sys_trans) `const`
Returns a transition of property process contained in enabled transition.
- `dve_transition_t` * `get_receiving_trans` (`const system_trans_t` &sys_trans) `const`
Returns a receiving transition contained in enabled transition.
- `dve_transition_t` * `get_sending_or_normal_trans` (`const system_trans_t` &sys_trans) `const`
Returns a normal or a sending transition contained in enabled transition.
- `size_int_t get_space_sum` () `const`
Returns a number of bytes necessary to store a state of the system.
- `size_int_t get_state_of_process` (`const state_t` state, `size_int_t` process_id) `const`
- `all_values_t get_var_value` (`const state_t` state, `size_int_t` var_gid, `const` `size_int_t` index=0)
Returns the value of the [variable](#) in a given state of the system.
- `bool is_accepting` (`state_t` state, `size_int_t` process_id)
Returns, whether process 'process_id' is in a accepting process state.
- `virtual slong_int_t read` (`const` `char` *const filename, `bool` do_comp)
Read in a DVE source given by 'filename' and avoids expression compaction if 'do_comp' is false.
- `virtual slong_int_t read` (`const` `char` *const filename)
Reads in a DVE source given by 'filename'.
- `bool set_var_value` (`state_t` state, `const` `size_int_t` var_gid, `const` `all_values_t` v, `const` `size_int_t` index=0)
Changes the value of the [variable](#) in a given state of the system.

Methods for expression evaluation

These methods are implemented and `can_evaluate_expressions()` returns true

- `virtual bool eval_expr` (`const expression_t` *const expr, `const state_t` state, `data_t` &data) `const`
Implements `explicit_system_t::eval_expr()` in DVE system.

Methods working with system transitions and enabled transitions

These methods are implemented and *can_system_transitions()* returns true

- virtual int *get_enabled_ith_trans* (const *state_t* state, const size_int_t i, *enabled_trans_t* &enb_trans)
- virtual int *get_enabled_trans* (const *state_t* state, *enabled_trans_container_t* &enb_trans)
Implements explicit_system_t::get_enabled_trans() in DVE system.
- virtual int *get_enabled_trans_count* (const *state_t* state, size_int_t &count)
Implements explicit_system_t::get_enabled_trans_count() in DVE system.
- virtual bool *get_enabled_trans_succ* (const *state_t* state, const *enabled_trans_t* &enabled, *state_t* &new_state)
Implements explicit_system_t::get_enabled_trans_succ() in DVE system.
- virtual bool *get_enabled_trans_succs* (const *state_t* state, *succ_container_t* &succs, const *enabled_trans_container_t* &enabled_trans)
Implements explicit_system_t::get_enabled_trans_succs() in DVE system.
- virtual int *get_succs* (*state_t* state, *succ_container_t* &succs, *enabled_trans_container_t* &etc)
- virtual *enabled_trans_t* * *new_enabled_trans* () const
Implements explicit_system_t::new_enabled_trans() in DVE system.

Obligatory part of abstract interface

These methods have to implemented in each implementation of *explicit_system_t*

- virtual *state_t* *get_initial_state* ()
Implements explicit_system_t::get_initial_state() in DVE system.
- virtual int *get_ith_succ* (*state_t* state, const int i, *state_t* &succ)
- size_int_t *get_preallocation_count* () const
Implements explicit_system_t::get_preallocation_count() in DVE system.
- virtual int *get_succs* (*state_t* state, *succ_container_t* &succs)
- virtual bool *is_accepting* (*state_t* state, size_int_t acc_group=0, size_int_t pair_member=1)
- virtual bool *is_erroneous* (*state_t* state)
Implements explicit_system_t::is_erroneous() in DVE system.
- virtual void *print_state* (*state_t* state, std::ostream &outs=std::cout)
Implements explicit_system_t::print_state() in DVE system.
- virtual size_int_t *violated_assertion_count* (const *state_t* state) const
Implements explicit_system_t::violated_assertion_count() in DVE system.
- virtual std::string *violated_assertion_string* (const *state_t* state, const size_int_t index) const

Implements *explicit_system_t::violated_assertion_string()* in DVE system.

- virtual bool `violates_assertion` (const `state_t` state) const
Implements *explicit_system_t::violates_assertion()* in DVE system.

Methods to check the SCCs of property process graph

- `process_decomposition_t * get_property_decomposition` ()
Returns property decomposition.

Protected Member Functions

- void `append_new_enabled` (`dve_transition_t` *const t_answ, `dve_transition_t` *const t_ask, const bool trans_err)
- void `append_new_enabled_prop_sync` (`dve_transition_t` *const t_answ, `dve_transition_t` *const t_prop, const bool trans_err)
Appends a new enabled transition given by sending and receiving property.
- bool `apply_effect` (const `state_t` state, const `dve_expression_t` *const effect)
Creates a successor of 'state' by applying effect 'effect'.
- bool `apply_transition_effects` (const `state_t` state, const `dve_transition_t` *const trans)
Creates a successor of 'state' by applying effects of the transition 'trans'.
- size_int_t `channel_content_count` (const `state_t` &state, const size_int_t gid)
- bool `channel_is_empty` (const `state_t` &state, const size_int_t gid)
Returns true iff channel with *GID* 'gid' is empty in state 'state'.
- bool `channel_is_full` (const `state_t` &state, const size_int_t gid)
Returns true iff channel with *GID* 'gid' is full in state 'state'.
- bool `compute_enabled_of_property` (const `state_t` state)
- bool `compute_enabled_stage1` (const size_int_t process_number, channels_t *channels, const `state_t` state, const bool only_committed)
- bool `compute_enabled_stage2` (const size_int_t process_number, channels_t *channels, const `state_t` state, const bool only_committed)
- bool `compute_successors_without_sync` (const size_int_t process_number, succ_container_t &succs, const `state_t` state)
- `state_t create_error_state` ()
Creates "erroneous state".
- int `get_async_enabled_trans` (const `state_t` state, `enabled_trans_container_t` &enb_trans)
Implementation of *get_enabled_trans()* in asynchronous DVE systems.

- bool `get_async_enabled_trans_succ` (const `state_t` state, const `enabled_trans_t` &enabled, `state_t` &new_state, const `state_t` property_state)
- bool `get_async_enabled_trans_succ` (const `state_t` state, const `enabled_trans_t` &enabled, `state_t` &new_state)

Implementation of `get_enabled_trans_succ()` in asynchronous DVE systems.

- bool `get_async_enabled_trans_succ_without_property` (const `state_t` state, const `enabled_trans_t` &enabled, `state_t` &new_state)
- bool `get_async_enabled_trans_succs` (const `state_t` state, `succ_container_t` &succs, const `enabled_trans_container_t` &enabled_trans)

Implementation of `get_enabled_trans_succs()` in asynchronous DVE systems.

- int `get_async_succs` (const `state_t` state, `succ_container_t` &succs, `enabled_trans_container_t` &enb_trans)

Implementation of `get_succs()` in asynchronous DVE systems.

- int `get_async_succs` (const `state_t` state, `succ_container_t` &succs)

Implementation of `get_succs()` in asynchronous DVE systems.

- int `get_async_succs_internal` (const `state_t` state, `succ_container_t` &succs)

Internal method for implementation of both variants of `get_async_succs()`.

- `all_values_t` `get_state_creator_value_of_var_type` (const `state_t` state, `size_int_t` var_gid, const `dve_var_type_t` var_type, const `size_int_t` index=0)

*Returns a value of a variable with *GID* 'var_gid' in a state 'state'.*

- int `get_sync_enabled_trans` (const `state_t` state, `enabled_trans_container_t` &enb_trans)

!Implementation of `get_enabled_trans()` in synchronous DVE systems

- bool `get_sync_enabled_trans_succ` (const `state_t` state, const `enabled_trans_t` &enabled, `state_t` &new_state)

Implementation of `get_enabled_trans_succ()` in synchronous DVE systems.

- bool `get_sync_enabled_trans_succs` (const `state_t` state, `succ_container_t` &succs, const `enabled_trans_container_t` &enabled_trans)

Implementation of `get_enabled_trans_succs()` in synchronous DVE systems.

- int `get_sync_succs` (`state_t` state, `succ_container_t` &succs, `enabled_trans_container_t` &etc)

Implementation of `get_succs()` in synchronous DVE systems.

- int `get_sync_succs` (`state_t` state, `succ_container_t` &succs)

Implementation of `get_succs()` in synchronous DVE systems.

- int `get_sync_succs_internal` (`state_t` state, `succ_container_t` &succs, `enabled_trans_container_t` *const etc)

Internal method for implementation of both variants of `get_sync_succs()`.

- void `go_to_error` (`state_t` state)
- bool `is_committed` (`state_t` state)

Returns, whether at least one process is in a committed state.

- bool `not_in_glob_conflict` (const `dve_transition_t` *const t1, const `dve_transition_t` *const t2)
- bool `passed_through` (const `state_t` state, const `dve_transition_t` *const t, const `dve_state_int_t` state1, bool &eval_err)
- void `pop_front_channel` (`state_t` &state, const `size_int_t` gid)
- void `prepare_channels_info` ()

Clears list of transitions created by `compute_enabled_stage1()`.

- void `push_back_channel` (`state_t` &state, const `size_int_t` gid)
- `all_values_t` `read_from_channel` (const `state_t` &state, const `size_int_t` gid, const `size_int_t` item_index, const `size_int_t` elem_index=0)
- `all_values_t` `retype` (`dve_var_type_t` type, `all_values_t` value)
- bool `set_state_creator_value` (`state_t` state, const `size_int_t` var_gid, const `all_values_t` v, const `size_int_t` index=0)
- void `set_state_creator_value_extended` (const `state_t` &state, const `state_t` &new_state, const `dve_expression_t` &to_assign, const `all_values_t` &val, bool &error)

Sets a value 'val' to the variable given by 'to_assign'.

- bool `set_state_creator_value_of_var_type` (`state_t` state, const `size_int_t` var_gid, const `dve_var_type_t` var_type, const `all_values_t` v, const `size_int_t` index)

*Sets a value of a variable with *GID* 'var_gid' in a state 'state'.*

- void `write_to_channel` (`state_t` &state, const `size_int_t` gid, const `size_int_t` item_index, const `all_values_t` value)

Protected Attributes

- `size_int_t` * `array_sizes`
- `enabled_trans_container_t` * `aux_enabled_trans`
- `enabled_trans_container_t` * `aux_enabled_trans2`
- `succ_container_t` * `aux_succ_container`
- `size_int_t` * `channel_buffer_size`
- `size_int_t` * `channel_element_size`
- `size_int_t` ** `channel_item_pos`
- `dve_var_type_t` ** `channel_item_type`
- `channels_t` * `channels`
- `size_int_t` `first_in_succs`

- size_int_t **glob_count**
- [array_t](#)< byte_t * > **glob_filters**
- process_position_t **global_position**
- size_int_t **max_succ_count**
- [enabled_trans_container_t](#) * **p_enabled_trans**
- size_int_t **process_count**
- process_position_t * **process_positions**
- size_int_t **prop_begin**
- size_int_t **prop_size**
- size_int_t **property_position**
- [array_t](#)< prop_trans_t > **property_trans**
- size_int_t **space_sum**
- std::vector< [state_creator_t](#) > **state_creators**
- size_int_t **state_creators_count**
- size_int_t * **state_positions_channel**
- size_int_t * **state_positions_proc**
- size_int_t * **state_positions_state**
- size_int_t * **state_positions_var**
- size_int_t * **state_sizes_var**
- size_int_t **trans_count**

Classes

- struct [state_creator_t](#)

12.19.1 Detailed Description

Class in DVE system interpretation.

This class implements the abstract interface [explicit_system_t](#).

It is a child of [dve_system_t](#) - thus it also contains the representation of DVE [system](#).

DVE system interpretation in this case comprises state generation, enabled transitions generation and expression evaluation.

This implementation of the abstract interface implements full set of its methods. Furthermore It takes over the system of expression evaluation from [system_t](#). Only for evaluating variables, fields and state identifiers there are defined special functions, which return their value according to a state of system (given by a piece of a memory).

Furthermore it provides the set of methods, which are purely DVE system specific.

12.19.2 Constructor & Destructor Documentation

12.19.2.1 dve_explicit_system_t (error_vector_t & *evect* = gerr)

A constructor.

Parameters:

evect = [error vector](#) used for reporting of error messages

References `system_t::get_abilities()`, and `system_abilities_t::system_can_decompose_property`.

12.19.3 Member Function Documentation
**12.19.3.1 void append_new_enabled (dve_transition_t *const *t_answ*,
dve_transition_t *const *t_ask*, const bool *trans_err*)** `[protected]`

Appends an new enabled transition given by sending and receiving transition and erroneousousness.

It also combines sending and receiving transitions with property transitions stored in private variable `property_trans`, if there are any. Thus it stores 1 enabled transition for a system without property process and n enabled transition for a system with property process with n executable transitions stored in `property_trans`.

References `array_of_abstract_t::back()`, `array_of_abstract_t::begin()`, `array_of_abstract_t::extend()`, `system_t::get_with_property()`, `system_trans_t::set_count()`, `enabled_trans_t::set_erroneous()`, and `array_of_abstract_t::size()`.

Referenced by `compute_enabled_stage2()`.

**12.19.3.2 void append_new_enabled_prop_sync (dve_transition_t *const
t_answ, dve_transition_t *const *t_prop*, const bool *trans_err*)**
`[protected]`

Appends a new enabled transition given by sending and receiving property.

Parameters:

t_answ = sending transition

t_prop = receiving transition of the property process

trans_err = says, whether an evaluation of some of guards of transitions was erroneous

References `array_of_abstract_t::extend()`, `system_trans_t::set_count()`, and `enabled_trans_t::set_erroneous()`.

Referenced by `compute_enabled_stage2()`.

**12.19.3.3 size_int_t channel_content_count (const state_t & *state*, const
size_int_t *gid*)** `[protected]`

Returns a count of elements stored in a channel with [GID](#) '*gid*' in state '*state*'

Referenced by `channel_is_empty()`, `channel_is_full()`, and `DBG_print_state()`.

12.19.3.4 **bool compute_enabled_of_property (const state_t state)** [protected]

Computes transitions of property process with satisfied guards and stores them to the list `property_trans`

References `compute_successors_without_sync()`, `process_t::get_gid()`, `dve_process_t::get_trans_count()`, `dve_process_t::get_transition()`, `passed_through()`, `dve_system_t::pproperty`, and `SYNC_NO_SYNC`.

Referenced by `get_async_enabled_trans()`.

12.19.3.5 **bool compute_enabled_stage1 (const size_int_t process_number, channels_t * channels, const state_t state, const bool only_committed)** [protected]

Searching for receiving transitions with satisfied guards (stores them to ‘channels’ - transition sorted by the used channel)

References `dve_process_t::get_committed()`, `process_t::get_gid()`, `dve_process_t::get_trans_count()`, `dve_process_t::get_transition()`, `passed_through()`, `dve_system_t::processes`, and `SYNC_ASK`.

Referenced by `compute_successors_without_sync()`, and `get_async_enabled_trans()`.

12.19.3.6 **bool compute_enabled_stage2 (const size_int_t process_number, channels_t * channels, const state_t state, const bool only_committed)** [protected]

Searching for sending transitions and transitions without synchronization with satisfied guards.

It combines them with transitions from ‘channels’ created by [compute_enabled_stage1\(\)](#). It creates a list of enabled transitions in `p_enabled_trans`.

References `append_new_enabled()`, `append_new_enabled_prop_sync()`, `channel_is_empty()`, `channel_is_full()`, `dve_transition_t::get_channel_gid()`, `dve_process_t::get_committed()`, `process_t::get_gid()`, `dve_system_t::get_property_gid()`, `dve_process_t::get_trans_count()`, `dve_process_t::get_transition()`, `system_t::get_with_property()`, `passed_through()`, `dve_system_t::processes`, `SYNC_ASK`, `SYNC_ASK_BUFFER`, `SYNC_EXCLAIM`, `SYNC_EXCLAIM_BUFFER`, and `SYNC_NO_SYNC`.

Referenced by `get_async_enabled_trans()`.

12.19.3.7 **bool compute_successors_without_sync (const size_int_t process_number, succ_container_t & succs, const state_t state)** [protected]

Computes successors of ‘state’ using transitions of process with [GID](#) ‘process_number’ without any regard to synchronization

References `apply_transition_effects()`, `array_of_abstract_t::back()`, `compute_enabled_stage1()`, `array_of_abstract_t::extend()`, `process_t::get_gid()`, `dve_process_t::get_trans_count()`, `dve_process_t::get_transition()`, `go_to_error()`, `passed_through()`, `dve_system_t::processes`, `array_t::push_back()`, `system_trans_t::set_count()`, and `enabled_trans_t::set_erroneous()`.

Referenced by `compute_enabled_of_property()`, and `get_sync_succs_internal()`.

12.19.3.8 `state_t create_error_state ()` [protected]

Creates "erroneous state".

Creates "erroneous state", which is the unique in whole system. This state should not be interpreted (you would get unexpected values of [variables](#), which you want to see).

You can detect error states by function [is_erroneous\(\)](#).

References `get_space_sum()`, and `dve_system_t::processes`.

12.19.3.9 `void DBG_print_state (state_t state, std::ostream & outs = std::cerr, const ulong_int_t format = ES_FMT_PRINT_ALL_NAMES)`

Prints a state to output stream 'outs' formatted according to 'format'.

This is a function meant especially for debugging purposes.

Parameters:

state = state of the system to print

outs = output stream where you want to print the state

format = format of output - you can set how the output should be structured and which names of identifiers should appear in the printout of the state (bitwise sum of the constants [ES_FMT_PRINT_STATE_NAMES](#), [ES_FMT_PRINT_VAR_NAMES](#), [ES_FMT_PRINT_PROCESS_NAMES](#), [ES_FMT_DIVIDE_PROCESSES_BY_CR](#)).

References `dve_explicit_system_t::state_creator_t::array_size`, `channel_content_count()`, `ES_FMT_DIVIDE_PROCESSES_BY_CR`, `ES_FMT_PRINT_PROCESS_NAMES`, `ES_FMT_PRINT_STATE_NAMES`, `ES_FMT_PRINT_VAR_NAMES`, `dve_symbol_table_t::get_channel()`, `dve_symbol_t::get_channel_type_list_size()`, `dve_symbol_t::get_name()`, `dve_symbol_table_t::get_process()`, `dve_symbol_table_t::get_state()`, `dve_symbol_table_t::get_variable()`, `dve_explicit_system_t::state_creator_t::gid`, `is_erroneous()`, `dve_system_t::processes`, `dve_system_t::psymbol_table`, `dve_explicit_system_t::state_creator_t::type`, and `dve_explicit_system_t::state_creator_t::var_type`.

12.19.3.10 `void DBG_print_state_CR (state_t state, std::ostream
& outs = std::cerr, const ulong_int_t format =
ES_FMT_PRINT_ALL_NAMES) [inline]`

It is the same as [DBG_print_state\(\)](#), but finally calls ‘outs << std::endl’ (thus prints ‘new-line’ character and flushes the buffer).

Referenced by [get_initial_state\(\)](#).

12.19.3.11 `all_values_t eval_expr (const dve_expression_t *const expr, const
state_t state, bool & eval_err) const [inline]`

Evaluates the expression in the context of the state of [explicit system](#).

Parameters:

expr = pointer to the expression represented by [dve_expression_t](#)

state = state of the system

eval_err = if there was an error during an evaluation of an expression, *eval_err* is set to true. Otherwise it is **unchanged**.

Returns:

The value of the expression *expr* in the context of *state*

References [dve_system_t::eval_expr\(\)](#), [dve_system_t::fast_eval\(\)](#), [dve_expression_t::get_p_compact\(\)](#), [dve_expression_t::is_compacted\(\)](#), and [ES_parameters_t::state](#).

12.19.3.12 `bool get_async_enabled_trans_succ (const state_t state, const
enabled_trans_t & enabled, state_t & new_state, const state_t
property_state) [protected]`

Creates a successor of ‘state’ using the enabled transition ‘enabled’ and a successor ‘property_state’ gained by the transition of property

This function optimizes a generation of successors of one state, because it does not compute transitions of property process any more. It uses precomputed state of property process.

References [get_async_enabled_trans_succ_without_property\(\)](#), [get_property_trans\(\)](#), and [state_t::ptr](#).

12.19.3.13 `bool get_async_enabled_trans_succ_without_property (const state_t
state, const enabled_trans_t & enabled, state_t & new_state)
[protected]`

Creates a successor of ‘state’ using the enabled transition ‘enabled’ and doesn’t use the property component of the enabled transition.

References `apply_transition_effects()`, `eval_expr()`, `dve_symbol_table_t::get_channel()`, `dve_transition_t::get_channel_gid()`, `dve_symbol_t::get_channel_type_list_item()`, `dve_symbol_t::get_channel_typed()`, `enabled_trans_t::get_erroneous()`, `get_receiving_trans()`, `get_sending_or_normal_trans()`, `dve_transition_t::get_sync_expr_list_item()`, `dve_transition_t::get_sync_expr_list_size()`, `go_to_error()`, `dve_system_t::psymbol_table`, `set_state_creator_value_extended()`, `SYNC_ASK_BUFFER`, and `SYNC_EXCLAIM_BUFFER`.

Referenced by `get_async_enabled_trans_succ()`.

12.19.3.14 `int get_enabled_ith_trans (const state_t state, const size_int_t i, enabled_trans_t & enb_trans)` [virtual]

Implements `explicit_system_t::get_enabled_ith_trans()` in DVE system, but see also implementation specific notes below

Warning:

Computing of this method is the same as the running time of `get_enabled_trans()` (it is more efficient to use `get_enabled_trans()` and store generated enabled transitions, if it is possible).

Implements `explicit_system_t`.

References `get_enabled_trans()`.

Referenced by `get_ith_succ()`.

12.19.3.15 `int get_enabled_trans_count (const state_t state, size_int_t & count)` [virtual]

Implements `explicit_system_t::get_enabled_trans_count()` in DVE system.

The running time of this method is the same as the running time of `get_enabled_trans()`.

Implements `explicit_system_t`.

References `get_enabled_trans()`, and `array_of_abstract_t::size()`.

12.19.3.16 `int get_ith_succ (state_t state, const int i, state_t & succ)` [virtual]

Implements `explicit_system_t::get_ith_succ()` in DVE system, but see also implementation specific notes below

Warning:

The running time of this function is relatively high. The running time is the running time of `get_succs()`. It is usually more efficient to use `get_succs()` and to store all generated states, if it is possible.

Implements [explicit_system_t](#).

References [get_async_enabled_trans_succ\(\)](#), [get_enabled_ith_trans\(\)](#), and [SUCC_ERROR](#).

Referenced by [por_t::ample_set_succs\(\)](#).

12.19.3.17 `dve_transition_t * get_property_trans (const system_trans_t & sys_trans) const`

Returns a transition of property process contained in enabled transition.

If the system contains a property process, this method returns the pointer to the property process's transition contained in an enabled transition.

Otherwise it returns 0.

Warning:

This method presumes, that *sys_trans* has been created by this class.

References [system_trans_t::get_count\(\)](#), and [system_t::get_with_property\(\)](#).

Referenced by [get_async_enabled_trans_succ\(\)](#), and [dve_prob_explicit_system_t::get_succs\(\)](#).

12.19.3.18 `dve_transition_t * get_receiving_trans (const system_trans_t & sys_trans) const`

Returns a receiving transition contained in enabled transition.

If [get_sending_or_normal_trans\(sys_trans\)](#) returns sending transition, this method returns its receiving counterpart.

Otherwise it returns 0.

Warning:

This method presumes, that *sys_trans* has been created by this class.

References [system_trans_t::get_count\(\)](#), and [system_t::get_with_property\(\)](#).

Referenced by [por_t::generate_ample_sets\(\)](#), [por_t::generate_composed_ample_sets\(\)](#), and [get_async_enabled_trans_succ_without_property\(\)](#).

12.19.3.19 `dve_transition_t * get_sending_or_normal_trans (const system_trans_t & sys_trans) const`

Returns a normal or a sending transition contained in enabled transition.

Transition of DVE system can consist of at most 3 transitions of processes. It always contains either non-synchronized transition or a sending transition (and its receiving counterpart)

This method guaranties to always return the transition, if *sys_trans* contains any (which should be truth, if enabled transition has been created by this class).

Warning:

This method presumes, that *sys_trans* has been created by this class.

Referenced by `por_t::generate_ample_sets()`, `por_t::generate_composed_ample_sets()`, `get_async_enabled_trans_succ_without_property()`, and `dve_prob_explicit_system_t::get_succs()`.

12.19.3.20 `all_values_t get_state_creator_value_of_var_type (const state_t state, size_int_t var_gid, const dve_var_type_t var_type, const size_int_t index = 0)` [protected]

Returns a value of a variable with [GID](#) 'var_gid' in a state 'state'.

Parameters:

state = state of system

var_gid = [GID](#) of variable

var_type = type of variable

index = index for a case of vector of variables

Returns:

a value of variable with [GID](#) *var_gid* in a state *state*. It uses *var_type* to determine the type of conversion and *index* (voluntary) for determining an pointer to the variable in a vector

12.19.3.21 `size_int_t get_state_of_process (const state_t state, size_int_t process_id) const`

Returns a [state LID](#) of state of a given process in a given state of the system.

References `dve_system_t::processes`.

Referenced by `por_t::generate_ample_sets()`, `por_t::generate_composed_ample_sets()`, `dve_process_decomposition_t::get_process_scc_id()`, and `dve_process_decomposition_t::get_process_scc_type()`.

12.19.3.22 `virtual int get_succs (state_t state, succ_container_t & succs, enabled_trans_container_t & etc)` [inline, virtual]

Implements `explicit_system_t::get_succs(state_t state, succ_container_t & succs, enabled_trans_container_t & etc)` in DVE [system](#)

Implements [explicit_system_t](#).

12.19.3.23 `virtual int get_succs (state_t state, succ_container_t & succs)`
`[inline, virtual]`

Implements [explicit_system_t::get_succs\(state_t state, succ_container_t & succs\)](#) in DVE [system](#)

Implements [explicit_system_t](#).

Referenced by `dve_prob_explicit_system_t::get_succs()`.

12.19.3.24 `bool get_sync_enabled_trans_succ (const state_t state, const enabled_trans_t & enabled, state_t & new_state)` `[protected]`

Implementation of [get_enabled_trans_succ\(\)](#) in synchronous DVE systems.

!!!Transmitting of values through channels is ignored!!!!

References `apply_transition_effects()`, `system_trans_t::get_count()`, and `enabled_trans_t::get_erroneous()`.

Referenced by `get_sync_enabled_trans_succs()`.

12.19.3.25 `void go_to_error (state_t state)` `[protected]`

METHODS: Changes the state of process (repr. by 'process') to 'error'

References `dve_system_t::processes`.

Referenced by `compute_successors_without_sync()`, `get_async_enabled_trans_succ_without_property()`, and `get_sync_succs_internal()`.

12.19.3.26 `bool is_accepting (state_t state, size_int_t process_id)` `[inline]`

Returns, whether process 'process_id' is in a accepting process state.

Parameters:

state = state of the system

process_id = [GID](#) of process, which we are asking for an acceptance

Returns:

whether a process with [GID](#) *process_ID* is in an accepting state or not

12.19.3.27 `virtual bool is_accepting (state_t state, size_int_t acc_group = 0, size_int_t pair_member = 1)` `[inline, virtual]`

Implements [explicit_system_t::is_accepting\(\)](#) in DVE [system](#). Returns true if the property automaton state belongs to the by-parameters-specified set of accepting condition. The set of accepting condition is specified using accepting group id and possibly pair

member. Accepting !groups correspond to individual sets of states, or pairs of sets of states given by the accepting condition of the property automaton. Groups are identified by the numbers including zero (0,...,n-1) while pair members using numbers 1 and 2.

For example let Streett's accepting condition be (L1,U1), (L2,U2), (L3,U3). To check whether state *q* is present in U2, `is_accepting(q,1,2)` should be used.

Implements [explicit_system_t](#).

12.19.3.28 `bool not_in_glob_conflict (const dve_transition_t *const t1, const dve_transition_t *const t2) [inline, protected]`

Returns true iff transitions 't1' and 't2' assign to the same global [variables](#).

References `dve_transition_t::get_glob_mask()`.

12.19.3.29 `bool passed_through (const state_t state, const dve_transition_t *const t, const dve_state_int_t state1, bool & eval_err) [inline, protected]`

Returns true iff transition 't' is a transition from a process state 'state1' with satisfied guard in a system state 'state'

Returns also errors during an evaluation to the variable *eval_err*

References `dve_transition_t::get_guard()`, and `dve_transition_t::get_state1_lid()`.

Referenced by `compute_enabled_of_property()`, `compute_enabled_stage1()`, `compute_enabled_stage2()`, and `compute_successors_without_sync()`.

12.19.3.30 `long_int_t read (const char *const filename, bool do_comp) [virtual]`

Read in a DVE source given by 'filename' and avoids expression compaction if 'do_comp' is false.

Is uses [system_t::read\(const char * const filename\)](#), therefore see that function for more information (about return value etc.) Furthermore it makes some analysis and extraction of information from the [system](#) and [symbol table](#). Moreover if 'do_comp' is false avoids expression compaction

References `dve_system_t::get_property_gid()`, `system_t::get_with_property()`, `process_decomposition_t::parse_process()`, and `dve_system_t::read()`.

12.19.3.31 `virtual long_int_t read (const char *const filename) [inline, virtual]`

Reads in a DVE source given by 'filename'.

Is uses [system_t::read\(const char * const filename\)](#), therefore see that function for more information (about return value etc.) Furthermore, it makes some analysis and extraction of information from the [system](#) and [symbol table](#)

Reimplemented from [dve_system_t](#).

Reimplemented in [dve_prob_explicit_system_t](#).

Referenced by `dve_prob_explicit_system_t::read()`.

12.19.3.32 `bool set_state_creator_value (state_t state, const size_int_t var_gid, const all_values_t v, const size_int_t index = 0) [inline, protected]`

[set_state_creator_value_of_var_type\(\)](#) with ‘var_type’ set to ‘var_types[var_gid]’

Referenced by `apply_effect()`, and `set_state_creator_value_extended()`.

12.19.3.33 `void set_state_creator_value_extended (const state_t & state, const state_t & new_state, const dve_expression_t & to_assign, const all_values_t & val, bool & error) [protected]`

Sets a value ‘val’ to the variable given by ‘to_assign’.

Parameters:

state = original state

new_state = new state, where the value of variable should be assigned

to_assign = expression representing a variable of indexed element of array (index is evaluated in a context of *state*)

val = value to assign

error = erroneousess of possible evaluation of index of array or potential breaking of bound of an array

References `eval_expr()`, `dve_expression_t::get_ident_gid()`, `dve_expression_t::get_operator()`, `dve_expression_t::left()`, and `set_state_creator_value()`.

Referenced by `get_async_enabled_trans_succ_without_property()`.

12.19.3.34 `bool set_state_creator_value_of_var_type (state_t state, const size_int_t var_gid, const dve_var_type_t var_type, const all_values_t v, const size_int_t index) [protected]`

Sets a value of a variable with [GID](#) ‘var_gid’ in a state ‘state’.

Parameters:

state = state of system

var_gid = [GID](#) of variable

var_type = type of variable

index = index for a case of vector of variables

v = value to assign

Returns:

true iff no error occurred during a call (i. e. value is in bounds of its type)

References `dve_symbol_t::get_name()`, `dve_symbol_table_t::get_variable()`, and `dve_system_t::psymbol_table`.

Referenced by `get_initial_state()`.

12.19.3.35 `bool set_var_value (state_t state, const size_int_t var_gid, const all_values_t v, const size_int_t index = 0) [inline]`

Changes the value of the [variable](#) in a given state of the system.

Parameters:

state ... system state to which the variable value should be set

var_gid ... [GID](#) of variable to set

v ... value to set (be aware of bounds of byte and int types)

index ... optional parameter for the case of arrays - it is the index to an array

Returns:

true iff setting value was not successful - i. e. value breaks the bounds of the type of variable, or variable is the constant

The documentation for this class was generated from the following files:

- [dve_explicit_system.hh](#)
- [dve_explicit_system.cc](#)

12.20 dve_explicit_system_t::state_creator_t Struct Reference

```
#include <dve_explicit_system.hh>
```

Public Types

- enum **state_creator_type_t** { **VARIABLE**, **PROCESS_STATE**, **CHANNEL_BUFFER** }

Public Member Functions

- **state_creator_t** (const state_creator_type_t sc_type, dve_var_type_t vtype, const size_int_t arr_size, const size_int_t gid_arg)

Public Attributes

- size_int_t **array_size**
- size_int_t **gid**
- state_creator_type_t **type**
- dve_var_type_t **var_type**

12.20.1 Detailed Description

Internal structure used to store informations about non-constant [variables](#), process states and channel buffers.

Programmer of algorithms using DiVinE usually should not use this structure.

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

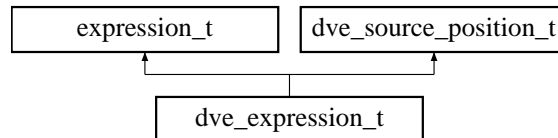
- [dve_explicit_system.hh](#)

12.21 dve_expression_t Class Reference

Class representing an expression in DVE [system](#).

```
#include <dve_expression.hh>
```

Inheritance diagram for dve_expression_t:



Public Member Functions

- virtual void [assign](#) (const [expression_t](#) &expr)
Implements [expression_t::assign\(\)](#) in DVE [system](#).
- [dve_expression_t](#) (const [dve_expression_t](#) &expr)
A copy constructor.
- [dve_expression_t](#) (const std::string &str_expr, [system_t](#) *const system, size_int_t process_gid=NO_ID)
A constructor from a string.
- [dve_expression_t](#) (int expr_op, const [dve_expression_t](#) &expr_left, [system_t](#) *const system=0)
- [dve_expression_t](#) (int expr_op, const [dve_expression_t](#) &expr_left, const [dve_expression_t](#) &expr_right, [system_t](#) *const system=0)
- [dve_expression_t](#) (int expr_op, std::stack< [dve_expression_t](#) * > &subexprs_stack, const short int ar=0, [system_t](#) *const system=0)
A constructor used by a parser.
- [dve_expression_t](#) ([system_t](#) *const system=0)
A constructor of empty expression.
- virtual int [from_string](#) (std::string &expr_str, const size_int_t process_gid=NO_ID)
Implements [expression_t::from_string\(\)](#) in DVE [system](#).
- virtual int [read](#) (std::istream &istr, size_int_t process_gid=NO_ID)
Implements [expression_t::read\(\)](#) in DVE [system](#).
- virtual void [swap](#) ([expression_t](#) &expr)
Implements [expression_t::swap\(\)](#) in DVE [system](#).

- virtual std::string [to_string](#) () const
Implements [expression_t::to_string\(\)](#) in DVE system.
- virtual void [write](#) (std::ostream &ostr) const
Implements [expression_t::write\(\)](#) in DVE system.
- virtual [~dve_expression_t](#) ()

DVE system specific methods

These methods are implemented only in DVE system and they cannot be run using an abstract interface of [expression_t](#)

- size_int_t [arity](#) () const
- void [compaction](#) ()
Computes compact version of the expression (single memory block).
- size_int_t [get_ident_gid](#) () const
- int [get_operator](#) () const
- [compacted_viewer_t](#) * [get_p_compact](#) () const
Returns pointer to compacted representation.
- [dve_symbol_table_t](#) * [get_symbol_table](#) () const
Returns a pointer to the [symbol table](#) corresponding to this expression.
- all_values_t [get_value](#) () const
- bool [is_compacted](#) () const
Says whether the expression is compacted.
- const [dve_expression_t](#) * [left](#) () const
Returns the left-most constant immediate subexpression.
- [dve_expression_t](#) * [left](#) ()
Returns the left-most immediate subexpression.
- const [dve_expression_t](#) * [right](#) () const
Returns the right-most constant immediate subexpression.
- [dve_expression_t](#) * [right](#) ()
Returns the right-most immediate subexpression.
- void [set_arity](#) (const size_int_t new_arity)
- void [set_ident_gid](#) (const size_int_t gid)
- void [set_operator](#) (const int oper)
- void [set_value](#) (const all_values_t value)
- const [dve_expression_t](#) * [subexpr](#) (size_int_t index) const
Returns the 'index'-th constant immediate subexpression.
- [dve_expression_t](#) * [subexpr](#) (size_int_t index)
Returns the 'index'-th immediate subexpression.

12.21.1 Detailed Description

Class representing an expression in DVE [system](#).

This class implements the abstract interface of [expression_t](#) and moreover it provides many methods beside methods of that interface.

Expressions are stored in a structure of a syntax tree. The methods of class [dve_expression_t](#) enables to obtain or change the informations about the top-most node in a syntax tree of the expression. To edit the lower levels of the syntax tree you can use methods [left\(\)](#), [right\(\)](#) and [subexpr\(\)](#).

The syntax tree is little different than you are probably used to. The inner nodes may contain following operators:

- () ... T_PARENTHESIS ... parenthesis,
- [] ... T_SQUARE_BRACKETS ... array indexing,
- < ... T_LT,
- <= ... T_LEQ,
- = ... T_EQ,
- != ... T_NEQ,
- >= ... T_GEQ,
- > ... T_GT,
- + ... T_PLUS,
- - ... T_MINUS,
- * ... T_MULT,
- / ... T_DIV,
- % ... T_MOD,
- & ... T_AND,
- | ... T_OR,
- ^ ... T_XOR,
- << ... T_LSHIFT,
- >> ... T_RSHIFT,
- and ... T_BOOL_AND,
- or ... T_BOOL_OR,
- -> ... T_IMPLY,
- - ... T_UNARY_MINUS ... unary minus

- `~ ... T_TILDE ...` bitwise negation
- `not ... T_BOOL_NOT ...` logical negation
- `= ... T_ASSIGN ...` assignment operation (permitted only in effects)

The leaf nodes may contain following operators:

- `. ... T_DOT ...` question on a state of process
- `<number> ... T_NAT ...` integer constant
- `<variable> ... T_ID ...` variable

Note:

In a `T_DOT` node you can use `get_ident_gid()` method to get a **GID** of process state contained in an expression. The identifier of process which owns that process state can be obtained by the function `symbol_t::get_process_gid()`, e.g. `table->get_state(expr->get_ident_gid())->get_process_gid()`.

12.21.2 Constructor & Destructor Documentation

12.21.2.1 `~dve_expression_t()` [virtual]

IMPLEMENTATION OF VIRTUAL INTERFACE OF `expression_t` ///// A destructor.

12.21.2.2 `dve_expression_t(int expr_op, const dve_expression_t & expr_left, const dve_expression_t & expr_right, system_t *const system = 0)` [inline]

A constructor that creates a new expression from 2 expressions and an operator.

12.21.2.3 `dve_expression_t(int expr_op, const dve_expression_t & expr_left, system_t *const system = 0)` [inline]

A constructor that creates a new expression from 1 expression and an operator.

12.21.2.4 `dve_expression_t(const std::string & str_expr, system_t *const system, size_int_t process_gid = NO_ID)`

A constructor from a string.

Parameters:

str_expr = string assigning the expression to represent

par_table = symbol table containing symbols needed to recognize in the expression (e.g. variable names)

process_gid = [GID](#) of process in which the expression is given (NO_ID means no process, that is a global environment)

References `gerr`, and `read()`.

12.21.3 Member Function Documentation

12.21.3.1 `size_int_t arity () const` [inline]

Returns a number of children of the top-most node in the syntax tree of the expression. Referenced by `compaction()`, and `dve_system_t::eval_expr()`.

12.21.3.2 `size_int_t get_ident_gid () const` [inline]

Returns an identifier contained in the top-most node of the syntax tree of the expression. It can be used only if operator in this node is `T_ID` (scalar variable), `T_SQUARE_BRACKETS` (vector variable) or `T_DOT` (process state).

It returns [GID](#) of the object of a certain type - the type is given by `get_operator()` (values `T_ID`, `T_SQUARE_BRACKETS`, `T_DOT`).

Referenced by `dve_parser_t::check_restrictions_put_on_property()`, `compaction()`, `dve_system_t::eval_expr()`, `dve_parser_t::expr_assign()`, `dve_explicit_system_t::set_state_creator_value_extended()`, and `dve_parser_t::trans_sync()`.

12.21.3.3 `int get_operator () const` [inline]

Returns an operator in the top-most node of the syntax tree of the expression.

Referenced by `compaction()`, `dve_system_t::eval_expr()`, `dve_parser_t::expr_assign()`, `dve_explicit_system_t::set_state_creator_value_extended()`, and `dve_parser_t::trans_sync()`.

12.21.3.4 `all_values_t get_value () const` [inline]

Returns a value contained in the top-most node of the syntax tree of the expression. It can be used only if operator in this node is `T_NAT` (number).

Referenced by `compaction()`, and `dve_system_t::eval_expr()`.

12.21.3.5 `void set_arity (const size_int_t new_arity)` [inline]

Sets a number of children of the top-most node in the syntax tree of the expression.

12.21.3.6 void set_ident_gid (const size_int_t *gid*) [inline]

Sets an identifier contained in the top-most node of the syntax tree of the expression.

It can be used only if operator in this node is T_ID (scalar variable), T_SQUARE_BRACKETS (vector variable) or T_DOT (process state).

It returns [GID](#) of the object of a certain type - the type is given by [get_operator\(\)](#) (values T_ID, T_SQUARE_BRACKETS, T_DOT).

12.21.3.7 void set_operator (const int *oper*) [inline]

Sets an operator in the top-most node of the syntax tree of the expression.

12.21.3.8 void set_value (const all_values_t *value*) [inline]

Sets a value contained in the top-most node of the syntax tree of the expression. It can be used only if operator in this node is T_NAT (number).

The documentation for this class was generated from the following files:

- [dve_expression.hh](#)
- [dve_expression.cc](#)

12.22 dve_parser_t Class Reference

Class that provides an interface to the parsing of DVE sources.

```
#include <dve_parser.hh>
```

Public Types

- enum [parser_mode_t](#) {
[SYSTEM](#), [PROCESS](#), [PROB_TRANSITION](#), [TRANSITION](#),
[EXPRESSION](#) }
Modes of parsing.

Public Member Functions

- void [check_restrictions_put_on_property](#) ()
- [dve_parser_t](#) ([error_vector_t](#) &evect, [dve_expression_t](#) *my_expression)
A constructor for creating of a parser of DVE expression.
- [dve_parser_t](#) ([error_vector_t](#) &evect, [dve_prob_transition_t](#) *my_transition)
A constructor for creating of a parser of DVE transition.
- [dve_parser_t](#) ([error_vector_t](#) &evect, [dve_transition_t](#) *my_transition)
A constructor for creating of a parser of DVE transition.
- [dve_parser_t](#) ([error_vector_t](#) &evect, [dve_prob_process_t](#) *my_process)
A constructor for creating of a parser of probabilistic DVE process.
- [dve_parser_t](#) ([error_vector_t](#) &evect, [dve_process_t](#) *my_process)
A constructor for creating of a parser of DVE process.
- [dve_parser_t](#) ([error_vector_t](#) &evect, [dve_system_t](#) *my_system)
A constructor for creating of a parser of entire DVE source file.
- [size_int_t](#) [get_current_process](#) () const
Returns the context for parsing of expression or transition.
- [dve_expression_t](#) * [get_dve_expression_to_fill](#) ()
Returns a expression, which is filled in by the parser.
- [dve_prob_transition_t](#) * [get_dve_prob_transition_to_fill](#) ()
Returns a probabilistic transition, which is filled in by the parser.
- [dve_process_t](#) * [get_dve_process_to_fill](#) ()

Returns a process, which is filled in by the parser.

- [dve_system_t * get_dve_system_to_fill \(\)](#)

Returns a system, which is filled in by the parser.

- [dve_transition_t * get_dve_transition_to_fill \(\)](#)

Returns a transition, which is filled in by the parser.

- [dve_expression_t * get_expression \(\)](#) const

Returns the last parsed expression.

- [parser_mode_t get_mode \(\)](#)

Returns a mode of parsing.

- [bool get_probabilistic \(\)](#)

Returns, whether parsed system can be probabilistic.

- [void set_current_process \(size_int_t process_gid\)](#)

Sets the context for parsing of expression or transition.

- [void set_dve_expression_to_fill \(dve_expression_t *const expr\)](#)

Sets a expression, which should be filled in a parser.

- [void set_dve_prob_transition_to_fill \(dve_prob_transition_t *const trans\)](#)

Sets a probabilistic transition, which should be filled in a parser.

- [void set_dve_process_to_fill \(dve_process_t *const proc\)](#)

Sets a process, which should be filled in a parser.

- [void set_dve_system_to_fill \(dve_system_t *const sys\)](#)

Sets a system, which should be filled in a parser.

- [void set_dve_transition_to_fill \(dve_transition_t *const trans\)](#)

Sets a transition, which should be filled in a parser.

- [void set_mode \(const parser_mode_t new_mode\)](#)

Sets a mode of parsing.

- [void set_probabilistic \(const bool can_be_prob\)](#)

Sets, whether parsed system can be probabilistic.

- [~dve_parser_t \(\)](#)

A destructor.

Parsing-time methods

These methods are called by a Yacc/Bison generated parser and they are determined to store the informations from the syntax analysis.

These functions should not be interesting for a usual developer.

- void [accept_genbuchi_muller_set_complete](#) ()
- void [accept_rabin_streett_first_complete](#) ()
- void [accept_rabin_streett_pair_complete](#) ()
- void [accept_type](#) (int type)
- void [assertion_create](#) (const char *state_name)
Called by Bison parser after parsing of assertion definition.
- void [channel_decl](#) (const char *name)
Called by Bison parser, when declaration of channel is parsed.
- void [done](#) ()
Method called by Bison parser after finished parsing.
- void [expr_array_mem](#) (const char *name, int expr_op=T_SQUARE_BRACKETS)
- void [expr_assign](#) (const int assignType)
- void [expr_bin](#) (const int binaryType)
- void [expr_false](#) ()
- void [expr_id](#) (const char *name, int expr_op=T_ID)
- void [expr_nat](#) (const int num)
- void [expr_parenthesis](#) ()
- void [expr_state_of_process](#) (const char *proc_name, const char *name)
- void [expr_true](#) ()
- void [expr_unary](#) (const int unaryType)
- void [expr_var_of_process](#) (const char *proc_name, const char *name, const bool array=false)
- void [expression_list_store](#) ()
Called by Bison parser, when the item of the list of expressions is parsed.
- int [get_fcol](#) ()
Returns the first column of the current position of Bison parser.
- int [get_fline](#) ()
Returns the first line of the current position of Bison parser.
- int [get_lcol](#) ()
Returns the column line of the current position of Bison parser.
- int [get_lline](#) ()
Returns the last line of the current position of Bison parser.
- void [prob_trans_create](#) (const char *name)
Called by Bison parser, when probabilistic transition is detected.
- void [prob_transition_part](#) (const char *name, const int weight)

Called by Bison parser, when a part of probabilistic transition is read.

- void [proc_decl_begin](#) (const char *name)
Called by Bison parser, when it observes the begin of process.
- void [proc_decl_done](#) ()
Called by Bison parser, when it observes the end of process.
- void [set_fpos](#) (int line, int col)
- void [set_lpos](#) (int line, int col)
- void [state_accept](#) (const char *name)
- void [state_commit](#) (const char *name)
Called by Bison parser, when declaration of committed state is parsed.
- void [state_decl](#) (const char *name)
Called by Bison parser, when declaration of state is parsed.
- void [state_genbuchi_muller_accept](#) (const char *name)
- void [state_init](#) (const char *name)
Called by Bison parser, when declaration of initial state is parsed.
- void [state_list_done](#) ()
- void [state_rabin_streett_accept](#) (const char *name)
- void [system_property](#) (const char *name)
Called by Bison parser, when a name of the property process is parsed.
- void [system_synchronicity](#) (const int sync, const bool prop=false)
Called by Bison parser, when synchronicity of a system is parsed.
- void [take_expression](#) ()
Called by Bison parser, when parsing of expression is expected.
- void [take_expression_cancel](#) ()
Called by Bison parser in case of error during parsing of expression.
- void [trans_create](#) (const char *name1, const char *name2, const bool has_guard, const int sync, const int effect_count)
Called by Bison parser after parsing of transition.
- void [trans_effect_list_begin](#) ()
Called by Bison parser after keyword "effect" is parsed.
- void [trans_effect_list_cancel](#) ()
Called by Bison parser, when error occurs during parsing of effects.
- void [trans_effect_list_end](#) ()
Called by Bison parser after all effects are parsed.
- void [trans_effect_part](#) ()
Called by Bison parser, when single effect is parsed.

- void [trans_guard_expr](#) ()
Called by Bison parser, when guard in a transition is parsed.
- void [trans_sync](#) (const char *name, const int sync, const bool sync_val)
Called by Bison parser, when synchronization in a transition is parsed.
- void [type_is_const](#) (const bool const_prefix)
- void [type_list_add](#) (const int type_nbr)
Called by Bison parser, when the item of the list of types is parsed.
- void [type_list_clear](#) ()
- void [typed_channel_decl](#) (const char *name, const int buffer_size)
Called by Bison parser, when declaration of typed channel is parsed.
- void [var_decl_array_size](#) (const int bound)
- void [var_decl_begin](#) (const int type_nbr)
Called by Bison parser, when variable declaration begins.
- void [var_decl_cancel](#) ()
- void [var_decl_create](#) (const char *name, const int arrayDimensions, const bool initialized)
- void [var_decl_done](#) ()
Called by Bison parser after finished parsing of declaration of variable.
- void [var_init_field_part](#) ()
- void [var_init_is_field](#) (const bool is_field)
Called by Bison parser, during parsing of initial value of variable.

12.22.1 Detailed Description

Class that provides an interface to the parsing of DVE sources.

It is used by reading methods of [dve_system_t](#), [dve_process_t](#), [dve_transition_t](#) and [dve_expression_t](#)

12.22.2 Member Enumeration Documentation

12.22.2.1 enum parser_mode_t

Modes of parsing.

Used in methods [get_mode\(\)](#), [set_mode\(\)](#). Mode of parsing is automatically set by used constructor of [dve_parser_t](#)

Enumerator:

SYSTEM Parser will parse entire DVE source file.

PROCESS Parser will parse only DVE process.

PROB_TRANSITION Parser will parse only DVE prob. transition.

TRANSITION Parser will parse a DVE transition.

EXPRESSION Parser will parse a DVE expression.

12.22.3 Member Function Documentation

12.22.3.1 void accept_genbuchi_muller_set_complete ()

Called by Bison parser, when the declaration of an accepting set in GenBuchi or Muller accepting condition is complete

12.22.3.2 void accept_rabin_streett_first_complete ()

Called by Bison parser, when the declaration of first part of the accepting pair in Rabin or Street accepting condition is complete

12.22.3.3 void accept_rabin_streett_pair_complete ()

Called by Bison parser, when the declaration of an accepting pair in Rabin or Street accepting condition is complete

12.22.3.4 void accept_type (int type)

Called by Bison parser, when property type is clear (ie complete accepting condition parsed)

References `dve_process_t::set_acceptance()`, and `dve_process_t::set_acceptance_type_and_groups()`.

12.22.3.5 void check_restrictions_put_on_property ()

Takes a property process currently set in a filled system and checks, whether all restrictions are fulfilled. If not, it throws errors.

References `dve_symbol_table_t::get_channel()`, `dve_transition_t::get_channel_gid()`, `dve_transition_t::get_effect()`, `dve_transition_t::get_effect_count()`, `dve_expression_t::get_ident_gid()`, `dve_symbol_t::get_process_gid()`, `dve_system_t::get_property_process()`, `dve_source_position_t::get_source_first_col()`, `dve_source_position_t::get_source_first_line()`, `dve_source_position_t::get_source_last_col()`, `dve_source_position_t::get_source_last_line()`, `dve_transition_t::get_sync_expr_list_size()`, `dve_transition_t::get_sync_mode()`, `process_t::get_trans_count()`, `process_t::get_transition()`, `dve_symbol_table_t::get_variable()`, `dve_expression_t::left()`, `SYNC_EXCLAIM`, and `SYNC_NO_SYNC`.

Referenced by `system_property()`.

12.22.3.6 void `expr_array_mem` (const char * *name*, int *expr_op* = T_SQUARE_BRACKETS)

Called by Bison parser, when token representing an identifier of array is read during parsing of an expression

References `dve_symbol_table_t::find_global_symbol()`, `dve_symbol_table_t::find_visible_symbol()`, `dve_symbol_table_t::get_symbol()`, `dve_symbol_t::is_variable()`, and `dve_symbol_t::is_vector()`.

Referenced by `expr_var_of_process()`.

12.22.3.7 void `expr_assign` (const int *assignType*)

Called by Bison parser, when assignment is read during parsing of an expression

Assignment is usually not allowed in the syntax of expressions, but this method is only called in effect of transition.

References `dve_expression_t::get_ident_gid()`, `dve_symbol_t::get_lid()`, `dve_symbol_t::get_name()`, `dve_expression_t::get_operator()`, `dve_symbol_t::get_process_gid()`, `dve_symbol_table_t::get_symbol()`, `dve_symbol_table_t::get_variable()`, `dve_symbol_t::is_const()`, and `dve_symbol_t::is_variable()`.

12.22.3.8 void `expr_bin` (const int *binaryType*)

Called by Bison parser, when binary operator is read during parsing of an expression

12.22.3.9 void `expr_false` ()

Called by Bison parser, when token "true" is read during parsing of an expression

12.22.3.10 void `expr_id` (const char * *name*, int *expr_op* = T_ID)

Called by Bison parser, when token representing an identifier of variable is read during parsing of an expression

Variable must not be an array - arrays are covered by method [expr_array_mem\(\)](#)

References `dve_symbol_table_t::find_global_symbol()`, `dve_symbol_table_t::find_visible_symbol()`, `dve_symbol_table_t::get_symbol()`, and `dve_symbol_t::is_variable()`.

Referenced by `expr_var_of_process()`.

12.22.3.11 void `expr_nat` (const int *num*)

Called by Bison parser, when token representing a natural number is read during parsing of an expression

12.22.3.12 void expr_parenthesis ()

Called by Bison parser, when parenthesis are read during parsing of an expression

At the time of calling of this method everything inside them is already parsed.

12.22.3.13 void expr_state_of_process (const char * *proc_name*, const char * *name*)

Called by Bison parser, when a test on a state of a process has been read during parsing of an expression

References dve_symbol_table_t::find_global_symbol(), dve_symbol_table_t::find_symbol(), dve_symbol_t::get_gid(), dve_symbol_t::get_name(), dve_symbol_table_t::get_symbol(), dve_symbol_t::get_symbol_type(), dve_symbol_t::get_valid(), and SYSTEM.

12.22.3.14 void expr_true ()

Called by Bison parser, when token "false" is read during parsing of an expression

12.22.3.15 void expr_unary (const int *unaryType*)

Called by Bison parser, when unary operator is read during parsing of an expression

12.22.3.16 void expr_var_of_process (const char * *proc_name*, const char * *name*, const bool *array* = false)

Called by Bison parser, when a value of a foreign variable has been read during parsing

References expr_array_mem(), expr_id(), dve_symbol_table_t::find_global_symbol(), dve_symbol_t::get_gid(), dve_symbol_table_t::get_symbol(), and dve_symbol_t::get_valid().

12.22.3.17 dve_expression_t * get_expression () const

Returns the last parsed expression.

Causes an exception, when there is not exactly one expression stored in a stack of parsed expressions

12.22.3.18 parser_mode_t get_mode () [inline]

Returns a mode of parsing.

Parser is able to parse a system (in DVE format), process, transition or expression according to the pointer given in a constructor

12.22.3.19 void set_fpos (int *line*, int *col*)

Called by Bison parser in case of reporting of the beginning of the current position to this class

12.22.3.20 void set_lpos (int *line*, int *col*)

Called by Bison parser in case of reporting of the end of the current position to this class

12.22.3.21 void state_accept (const char * *name*)

Called by Bison parser, when the declaration of accepting state in Buchi accepting condition is done

References dve_symbol_table_t::find_symbol(), dve_symbol_t::get_lid(), and dve_symbol_table_t::get_symbol().

Referenced by state_genbuchi_muller_accept(), and state_rabin_streett_accept().

12.22.3.22 void state_genbuchi_muller_accept (const char * *name*)

Called by Bison parser, when the declaration of accepting state in GenBuchi or Muller accepting set is done

References state_accept().

12.22.3.23 void state_list_done ()

Called by Bison parser, when declarations of all states of current process are parsed

12.22.3.24 void state_rabin_streett_accept (const char * *name*)

Called by Bison parser, when the declaration of accepting state in Rabin or Streett accepting set is done

References state_accept().

12.22.3.25 void trans_effect_part ()

Called by Bison parser, when single effect is parsed.

Last expression (containing effect's assignment) is then taken as an effect of the current transition

12.22.3.26 void type_is_const (const bool *const_prefix*) [inline]

Called by Bison parser, when it observes, that newly declared variable is constant

12.22.3.27 void type_list_clear ()

Called by Bison parser, when the item of the list of types is already copied somewhere and thus it can be cleared

12.22.3.28 void var_decl_array_size (const int *bound*)

Called by Bison parser, when number representing a bound of array is parsed.

References DVE_MAX_ARRAY_SIZE.

12.22.3.29 void var_decl_begin (const int *type_nbr*)

Called by Bison parser, when variable declaration begins.

Bison parser tells to this class the type of currently declared variable in parameter *type_nbr*

12.22.3.30 void var_decl_cancel ()

Called by Bison parser, when error occurs during parsing of declaration of variable

12.22.3.31 void var_decl_create (const char * *name*, const int *arrayDimensions*, const bool *initialized*)

Called by Bison parser after parsing of identifier and initial values

Creates a symbol in [Symbol table](#)

References take_expression_cancel().

12.22.3.32 void var_decl_done ()

Called by Bison parser after finished parsing of declaration of variable.

Currently empty method - variable is really created in [var_decl_create\(\)](#)

12.22.3.33 void var_init_field_part ()

Called by Bison parser, while it finishes a parsing of expression initialization

Stores last parsed expression to the vector of initial values

12.22.3.34 void var_init_is_field (const bool *is_field*)

Called by Bison parser, during parsing of initial value of variable.

Parser reports, whether initial value is an array of scalar value

The documentation for this class was generated from the following files:

- [dve_parser.hh](#)
- dve_parser.cc

12.23 dve_position_t Struct Reference

Structure for storing of position in a source code.

```
#include <dve_commonparse.hh>
```

Public Member Functions

- **dve_position_t** (const [dve_position_t](#) &loc)
- void **lines** (int num)
- void **reset** ()
- void **step** ()

Public Attributes

- int **first_column**
- int **first_line**
- int **last_column**
- int **last_line**

12.23.1 Detailed Description

Structure for storing of position in a source code.

This structure is used in an implementation of parser of DVE files You should not need to use it during an implementation of program based on this library

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

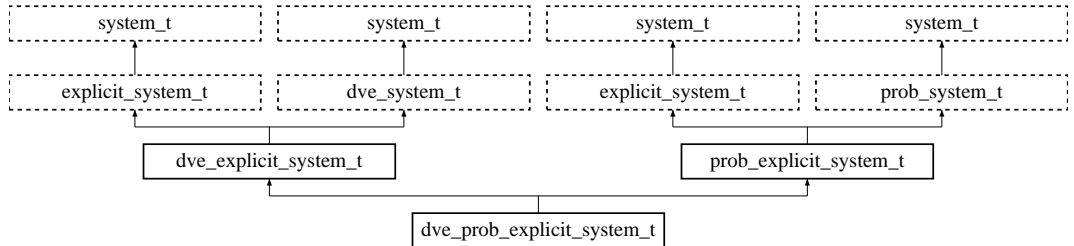
- [dve_commonparse.hh](#)

12.24 dve_prob_explicit_system_t Class Reference

Class in DVE system interpretation.

```
#include <dve_prob_explicit_system.hh>
```

Inheritance diagram for `dve_prob_explicit_system_t`:



Public Member Functions

- `dve_prob_explicit_system_t (error_vector_t &evect=gerr)`
A constructor.
- `virtual int get_succs (state_t state, prob_succ_container_t &succs)`
Creates probabilistic successors of state 'state'.
- `virtual int get_succs (state_t state, prob_succ_container_t &succs, enabled_trans_container_t &etc)`
Creates probabilistic successors of state 'state'.
- `virtual int get_succs_ordered_by_prob_and_property_trans (state_t state, prob_succ_container_t &succs)`
- `virtual slong_int_t read (const char *const filename)`
Reads in a DVE source given by 'filename'.
- `virtual slong_int_t read (std::istream &ins)`
Implements `system_t::read(std::istream & ins = std::cin)` in DVE `system`.
- `virtual ~dve_prob_explicit_system_t ()`
A destructor.

12.24.1 Detailed Description

Class in DVE system interpretation.

This class implements the abstract interface `explicit_system_t`.

It is a child of `dve_system_t` - thus it also contains the representation of DVE `system`.

DVE system interpretation in this case comprises state generation, enabled transitions generation and expression evaluation.

This implementation of the abstract interface implements full set of its methods. Furthermore It takes over the system of expression evaluation from [system_t](#). Only for evaluating variables, fields and state identifiers there are defined special functions, which return their value according to a state of system (given by a piece of a memory).

Furthermore it provides the set of methods, which are purely DVE system specific.

12.24.2 Constructor & Destructor Documentation

12.24.2.1 `dve_prob_explicit_system_t (error_vector_t & evec = gerr)` [inline]

A constructor.

Parameters:

evec = [error vector](#) used for reporting of error messages

12.24.3 Member Function Documentation

12.24.3.1 `virtual int get_succs (state_t state, prob_succ_container_t & succs)` [inline, virtual]

Creates probabilistic successors of state 'state'.

Creates probabilistic successors of state *state* and saves them to successor container *succs* (see [prob_succ_container_t](#)).

Parameters:

state = state of the system

succs = successors container for storage of successors of *state*

Returns:

bitwise OR of SUCC_NORMAL, SUCC_ERROR and SUCC_DEADLOCK (use functions [succs_normal\(\)](#), [succs_error\(\)](#) and [succs_deadlock\(\)](#) for testing)

Implements [prob_explicit_system_t](#).

12.24.3.2 `int get_succs (state_t state, prob_succ_container_t & succs, enabled_trans_container_t & etc)` [virtual]

Creates probabilistic successors of state 'state'.

Creates probabilistic successors of state *state*. In addition to [get_succs\(state_t state, prob_succ_container_t & succs\)](#) this method also creates a piece of information about enabled transitions used for successor generation.

Together with methods [prob_system_t::get_prob_trans_of_trans\(\)](#), [prob_system_t::get_index_of_trans_in_prob_trans\(\)](#) and methods of [prob_transition_t](#) it is possible to extract all additional information (and even more) that is stored in [prob_succ_container_t](#).

Implements [prob_explicit_system_t](#).

References [enabled_trans_container_t::clear\(\)](#), [array_t::clear\(\)](#), [transition_t::get_gid\(\)](#), [prob_system_t::get_index_of_trans_in_prob_trans\(\)](#), [prob_system_t::get_prob_trans_of_trans\(\)](#), [prob_system_t::get_prob_transition\(\)](#), [dve_explicit_system_t::get_property_trans\(\)](#), [dve_explicit_system_t::get_sending_or_normal_trans\(\)](#), [dve_explicit_system_t::get_succs\(\)](#), [prob_transition_t::get_weight\(\)](#), [prob_transition_t::get_weight_sum\(\)](#), [system_t::get_with_property\(\)](#), [array_t::push_back\(\)](#), and [array_t::size\(\)](#).

12.24.3.3 **virtual slong_int_t read (const char *const filename)** [inline, virtual]

Reads in a DVE source given by 'filename'.

Is uses [system_t::read\(const char * const filename\)](#), therefore see that function for more information (about return value etc.) Furthermore, it makes some analysis and extraction of information from the [system](#) and [symbol table](#)

Reimplemented from [dve_explicit_system_t](#).

References [dve_explicit_system_t::read\(\)](#).

The documentation for this class was generated from the following files:

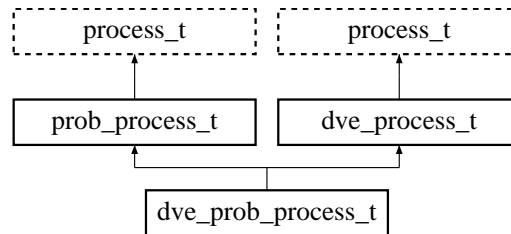
- [dve_prob_explicit_system.hh](#)
- [dve_prob_explicit_system.cc](#)

12.25 dve_prob_process_t Class Reference

Class representing a DVE process in probabilistic system.

```
#include <dve_prob_process.hh>
```

Inheritance diagram for dve_prob_process_t::



Public Member Functions

- virtual void [add_prob_transition](#) ([prob_transition_t](#) *const prob_trans)
Implements [prob_process_t::add_prob_transition\(\)](#) for DVE probabilistics process.
- [dve_prob_process_t](#) ([prob_system_t](#) *const system)
A constructor.
- [dve_prob_process_t](#) ()
A constructor.
- virtual int [from_string](#) (std::string &proc_str)
Implements [process_t::from_string\(\)](#) for DVE probabilistics process.
- virtual size_int_t [get_prob_trans_count](#) () const
Implements [prob_process_t::get_prob_trans_count\(\)](#) for DVE probabilistics process.
- virtual const [prob_transition_t](#) * [get_prob_transition](#) (const size_int_t prob_trans_lid) const
Implements [prob_process_t::get_prob_transition\(\)](#) for DVE probabilistics process.
- virtual [prob_transition_t](#) * [get_prob_transition](#) (const size_int_t prob_trans_lid)
Implements [prob_process_t::get_prob_transition\(\)](#) for DVE probabilistics process.
- virtual int [read](#) (std::istream &istr)
Implements [process_t::read\(\)](#) for DVE probabilistics process.
- virtual void [remove_prob_transition](#) (const size_int_t transition_index)
Implements [prob_process_t::remove_prob_transition\(\)](#) for DVE probabilistics process.

- virtual std::string [to_string](#) () const
Implements [process_t::to_string\(\)](#) for DVE probabilistics process.
- virtual void [write](#) (std::ostream &ostr) const
Implements [process_t::write\(\)](#) for DVE probabilistics process.
- virtual [~dve_prob_process_t](#) ()
A destructor.

12.25.1 Detailed Description

Class representing a DVE process in probabilistic system.

This class implements the abstract interface [prob_process_t](#).

The documentation for this class was generated from the following files:

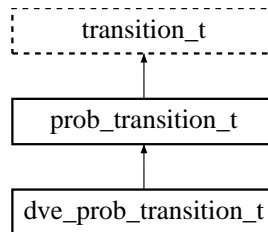
- [dve_prob_process.hh](#)
- [dve_prob_process.cc](#)

12.26 dve_prob_transition_t Class Reference

Class representing a DVE transition in probabilistic system.

```
#include <dve_prob_transition.hh>
```

Inheritance diagram for dve_prob_transition_t::



Public Member Functions

- [dve_prob_transition_t](#) ([prob_system_t](#) *const system)
A constructor.
- [dve_prob_transition_t](#) ()
A constructor.
- virtual int [from_string](#) (std::string &trans_str, const size_int_t process_gid=NO_ID)
Implements [transition_t::from_string\(\)](#) in probabilistic DVE system.
- bool [get_valid](#) () const
- virtual int [read](#) (std::istream &istr, size_int_t process_gid=NO_ID)
Implements [transition_t::read\(\)](#) in probabilistic DVE system.
- void [set_valid](#) (const bool is_valid)
Sets validity of the transition ('true' means valid).
- virtual std::string [to_string](#) () const
Implements [transition_t::to_string\(\)](#) in probabilistic DVE system.
- virtual void [write](#) (std::ostream &ostr) const
Implements [transition_t::write\(\)](#) in probabilistic DVE system.

Protected Attributes

- bool **valid**

12.26.1 Detailed Description

Class representing a DVE transition in probabilistic system.

This class implements the abstract interface [prob_transition_t](#).

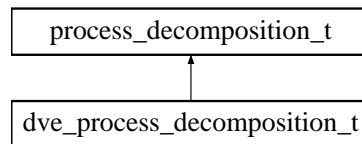
The documentation for this class was generated from the following files:

- [dve_prob_transition.hh](#)
- [dve_prob_transition.cc](#)

12.27 dve_process_decomposition_t Class Reference

```
#include <dve_process_decomposition.hh>
```

Inheritance diagram for dve_process_decomposition_t:



Public Member Functions

- **dve_process_decomposition_t** ([dve_explicit_system_t](#) &system)
- int [get_process_scc_id](#) ([state_t](#) &)
- int [get_process_scc_type](#) ([state_t](#) &)
- int [get_scc_count](#) ()
- int [get_scc_type](#) (int)
- bool [is_weak](#) ()
- void [parse_process](#) (std::size_t)

12.27.1 Detailed Description

This class is used to decompose a graph of a single process specified in a dve file into SCCs. The decomposition is not accessible directly but with member functions returning for a given state SCC id and type.

12.27.2 Member Function Documentation

12.27.2.1 int [get_process_scc_id](#) ([state_t](#) &) [virtual]

Returns id of an SCC that the given local state of the process belongs to.

Implements [process_decomposition_t](#).

References [dve_explicit_system_t::get_state_of_process\(\)](#).

12.27.2.2 int [get_process_scc_type](#) ([state_t](#) &) [virtual]

Returns type of an SCC that the given local state belongs to. Returned values: 0 means nonaccepting component, 1 means partially accepting component, and 2 means fully accepting component.

Implements [process_decomposition_t](#).

References [dve_explicit_system_t::get_state_of_process\(\)](#).

12.27.2.3 `int get_scc_count ()` [virtual]

Returns the number of SCCs in the decomposition.

Implements [process_decomposition_t](#).

12.27.2.4 `int get_scc_type (int)` [virtual]

Returns type of the given SCC, where 0 means nonaccepting component, 1 means partially accepting component, and 2 means fully accepting component.

Implements [process_decomposition_t](#).

12.27.2.5 `bool is_weak ()` [virtual]

Returns whether the process has a weak graph.

Implements [process_decomposition_t](#).

12.27.2.6 `void parse_process (std::size_t)` [virtual]

Performs the decomposition of a process with a given (global) id.

Implements [process_decomposition_t](#).

References `dve_process_t::get_acceptance()`, `dve_process_t::get_initial_state()`, `dve_system_t::get_process()`, `dve_transition_t::get_state1_lid()`, `dve_transition_t::get_state2_lid()`, `dve_process_t::get_state_count()`, `dve_process_t::get_trans_count()`, and `dve_process_t::get_transition()`.

The documentation for this class was generated from the following files:

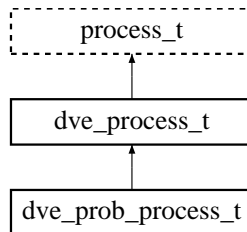
- [dve_process_decomposition.hh](#)
- [dve_process_decomposition.cc](#)

12.28 dve_process_t Class Reference

Class representing a process.

```
#include <dve_process.hh>
```

Inheritance diagram for dve_process_t::



Public Member Functions

- [dve_process_t](#) ([system_t](#) *const system)
A constructor (initializes the [process_t](#) part and DVE specific parts).
- [dve_process_t](#) ()
- virtual [~dve_process_t](#) ()
A destructor.

DVE system specific methods

These methods are implemented only in DVE system and they cannot be run using an abstract interface of [process_t](#).

- void [add_assertion](#) (size_int_t state_lid, [dve_expression_t](#) *const assert_expr)
Associates state with [LID](#) 'state_lid' with assertion 'assert_expr'.
- void [add_state](#) (const size_int_t state_gid)
- void [add_variable](#) (const size_int_t var_gid)
- bool [get_acceptance](#) (size_int_t lid, size_int_t group=0, size_int_t pair_member=1) const
- size_int_t [get_accepting_group_count](#) () const
- property_type_t [get_accepting_type](#) () const
Returns property accepting type.
- const [dve_expression_t](#) * [get_assertion](#) (const size_int_t state_lid, const size_int_t index) const
Returns a constant pointer to expression representing assertion.
- [dve_expression_t](#) * [get_assertion](#) (const size_int_t state_lid, const size_int_t index)

Returns a pointer to expression representing assertion.

- `size_int_t get_assertion_count (const size_int_t state_lid) const`
*Returns a count of assertions associated with state with **LID** 'state_lid'.*
- `bool get_committed (size_int_t lid) const`
*Returns, whether state with **LID** = 'lid' is committed or not.*
- `size_int_t get_initial_state () const`
*Returns a **LID** of initial state of this process.*
- `size_int_t get_state_count () const`
Returns a count of process states in this process.
- `size_int_t get_state_gid (size_int_t lid) const`
*Returns a **GID** of process state with **LID** 'lid'.*
- `dve_symbol_table_t * get_symbol_table () const`
*Returns a **symbol table** of the **system**, where this process is contained.*
- `size_int_t get_trans_count (const sync_mode_t sync_mode) const`
Returns a count of transitions of given synchronization mode.
- `const dve_transition_t * get_transition (const sync_mode_t sync_mode, const size_int_t trans_nbr) const`
- `dve_transition_t * get_transition (const sync_mode_t sync_mode, const size_int_t trans_nbr)`
- `bool get_valid () const`
Returns true iff process is valid.
- `size_int_t get_variable_count () const`
Returns a count of local variables in this process.
- `size_int_t get_variable_gid (size_int_t lid) const`
*Returns a **GID** of variable with **LID** 'lid'.*
- `void set_acceptance (size_int_t lid, bool is_accepting, size_int_t group=0, size_int_t pair_member=1)`
- `void set_acceptance_type_and_groups (property_type_t prop_type, size_int_t groups_count)`
Sets accepting type and number of accepting groups.
- `void set_committed (size_int_t lid, bool is_committed)`
*Sets, whether state with **LID** = 'lid' is committed or not.*
- `void set_initial_state (const size_int_t state_lid)`
Sets, which process state is initial.
- `void set_valid (const bool is_valid)`
Sets validity of the process ('true' means valid).

Methods modifying a process

These methods are implemented only if `can_transitions()` returns true.

- virtual void `add_transition` (`transition_t` *const transition)
- virtual void `remove_transition` (const size_int_t transition_lid)

Implements `process_t::remove_transition()` in DVE system.

Methods for reading a process from a string representation

These methods are implemented and `can_read()` returns true.

- virtual int `from_string` (std::string &proc_str)

Implements `process_t::from_string()` in DVE system.

- virtual int `read` (std::istream &istr)

Implements `process_t::read()` in DVE system.

Methods working with transitions of a process

These methods are implemented and `can_transitions()` returns true.

- virtual size_int_t `get_trans_count` () const

Implements `process_t::get_trans_count()` in DVE system.

- virtual const `transition_t` * `get_transition` (const size_int_t lid) const
- virtual `transition_t` * `get_transition` (const size_int_t lid)

Obligatory part of abstract interface

- virtual std::string `to_string` () const

Implements `process_t::to_string()` in DVE system.

- virtual void `write` (std::ostream &ostr) const

Implements `process_t::write()` in DVE system.

Protected Member Functions

- int `read_using_given_parser` (std::istream &istr, `dve_parser_t` &parser)
- void `write_declarations` (std::ostream &ostr) const

12.28.1 Detailed Description

Class representing a process.

This class implements the abstract interface [process_t](#)

Notice, there are methods, which are not virtual and they are already present in the class [process_t](#) (e. g. [process_t::get_gid\(\)](#)) and there are also virtual methods with default virtual implementation. They are not changed in [dve_transition_t](#) (e. g. [process_t::set_gid\(\)](#)).

It supports the full set of methods of an abstract interface and adds many DVE specific methods to this class, corresponding to the DVE point of view to transitions of processes.

12.28.2 Constructor & Destructor Documentation

12.28.2.1 [dve_process_t\(\)](#)

IMPLEMENTATION OF VIRTUAL INTERFACE ///// A constructor (initializes the [process_t](#) part and DVE specific parts)

12.28.3 Member Function Documentation

12.28.3.1 [void add_assertion \(size_int_t state_lid, dve_expression_t *const assert_expr\)](#)

Associates state with [LID](#) 'state_lid' with assertion 'assert_expr'.

Several assertions can be associated with the same state

References [array_t::back\(\)](#), and [array_t::extend\(\)](#).

Referenced by [dve_parser_t::assertion_create\(\)](#).

12.28.3.2 [void add_state \(const size_int_t state_gid\)](#)

This function should not be used if the process is the part of the [system](#) Otherwise is may cause inconsistencies in a [system](#).

References [array_t::back\(\)](#), [array_t::extend\(\)](#), [dve_symbol_table_t::get_state\(\)](#), [get_symbol_table\(\)](#), [array_t::push_back\(\)](#), and [dve_symbol_t::set_lid\(\)](#).

12.28.3.3 [void add_transition \(transition_t *const transition\)](#) [virtual]

Implements [process_t::add_transition\(\)](#) in DVE [system](#), but see also implementation specific notes below

This method modifies the added transition because it has to set [transition LID](#) and [Partial ID](#).

Implements [process_t](#).

References [dve_transition_t::get_sync_mode\(\)](#), [array_t::push_back\(\)](#), [transition_t::set_lid\(\)](#), [dve_transition_t::set_partial_id\(\)](#), and [array_t::size\(\)](#).

12.28.3.4 void add_variable (const size_int_t var_gid)

This function should not be used if the process is the part of the [system](#) Otherwise is may cause inconsistencies in a [system](#).

12.28.3.5 bool get_acceptance (size_int_t lid, size_int_t group = 0, size_int_t pair_member = 1) const [inline]

Returns, whether state with [LID](#) = 'lid' in accepting group 'group' and pair member 'pair_member' is s accepting or not.

Referenced by [dve_process_decomposition_t::parse_process\(\)](#).

12.28.3.6 size_int_t get_accepting_group_count () const [inline]

Returns number of accepting groups. Accepting groups correspond to the number of sets in accepting condition of generalized Buchi automaton or number of pairs in accepting conditions of Rabin and Streett automata.

12.28.3.7 const dve_expression_t* get_assertion (const size_int_t state_lid, const size_int_t index) const [inline]

Returns a constant pointer to expression representing assertion.

Parameters:

state_lid = [LID](#) of state, which the assertion is associated with

index = number from 0 to [get_assertion_count\(state_lid\)](#)-1

12.28.3.8 dve_expression_t* get_assertion (const size_int_t state_lid, const size_int_t index) [inline]

Returns a pointer to expression representing assertion.

Parameters:

state_lid = [LID](#) of state, which the assertion is associated with

index = number from 0 to [get_assertion_count\(state_lid\)](#)-1

12.28.3.9 `size_int_t get_trans_count (const sync_mode_t sync_mode) const` [inline]

Returns a count of transitions of given synchronization mode.

Returns a count of transitions of synchronization mode given by parameter *sync_mode*

[Partial ID](#) then can be from 0 to (get_trans_count(sync_mode) - 1)

12.28.3.10 `const dve_transition_t* get_transition (const sync_mode_t sync_mode, const size_int_t trans_nbr) const` [inline]

Returns a pointer to the selected constant transition. For further informations see `get_transition(const sync_mode_t sync_mode, const size_int_t trans_nbr)` above.

12.28.3.11 `dve_transition_t* get_transition (const sync_mode_t sync_mode, const size_int_t trans_nbr)` [inline]

Returns a pointer to the transition given by the synchronization and [Partial ID](#).

Returns a pointer to the transition given by parameters:

Parameters:

sync_mode = synchronization mode of the transition (SYNC_NO_SYNC, SYNC_ASK or SYNC_EXCLAIM)

trans_nbr = ID of the transition in this process

12.28.3.12 `virtual const transition_t* get_transition (const size_int_t lid) const` [inline, virtual]

Implements [process_t::get_transition\(const size_int_t lid\) const](#) in DVE system

Implements [process_t](#).

12.28.3.13 `virtual transition_t* get_transition (const size_int_t lid)` [inline, virtual]

Implements [process_t::get_transition\(const size_int_t lid\)](#) in DVE system

Implements [process_t](#).

Referenced by `dve_explicit_system_t::compute_enabled_of_property()`, `dve_explicit_system_t::compute_enabled_stage1()`, `dve_explicit_system_t::compute_enabled_stage2()`, `dve_explicit_system_t::compute_successors_without_sync()`, `dve_process_decomposition_t::parse_process()`, `dve_parser_t::prob_trans_create()`, `dve_system_t::set_property_gid()`, and `dve_prob_process_t::write()`.

12.28.3.14 `void set_acceptance (size_int_t lid, bool is_accepting, size_int_t group = 0, size_int_t pair_member = 1) [inline]`

Sets, whether state with [LID](#) = 'lid' in accepting group 'group' and pair member 'pair_member' is accepting or not.

Referenced by `dve_parser_t::accept_type()`.

12.28.3.15 `void set_initial_state (const size_int_t state_lid) [inline]`

Sets, which process state is initial.

Sets, which process state is initial.

Parameters:

state_lid = [LID](#) of initial state.

Referenced by `dve_parser_t::state_init()`.

The documentation for this class was generated from the following files:

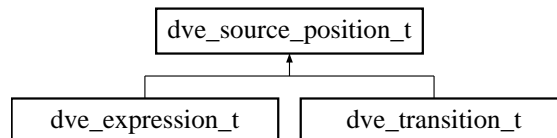
- [dve_process.hh](#)
- `dve_process.cc`

12.29 dve_source_position_t Class Reference

Class for storage of position in a source code.

```
#include <dve_source_position.hh>
```

Inheritance diagram for dve_source_position_t::



Public Member Functions

- [dve_source_position_t](#) (const size_int_t fline, const size_int_t fcol, const size_int_t lline, const size_int_t lcol)
A constructor.
- [dve_source_position_t](#) ()
A constructor (initializes all position parameters to zero).
- size_int_t [get_source_first_col](#) () const
Returns the first column of the position in a source code.
- size_int_t [get_source_first_line](#) () const
Returns the first line of the position in a source code.
- size_int_t [get_source_last_col](#) () const
Returns the last column of the position in a source code.
- size_int_t [get_source_last_line](#) () const
Returns the last line of the position in a source code.
- void [get_source_pos](#) (dve_source_position_t &second)
Sets a position in a source code (copies parameters from the given object).
- void [get_source_pos](#) (size_int_t &fline, size_int_t &fcol, size_int_t &lline, size_int_t &lcol) const
Copies the position in a source code to four given parameters.
- void [set_source_pos](#) (const dve_source_position_t &second)
Copies the position in a source code to the given object.
- void [set_source_pos](#) (const size_int_t fline, const size_int_t fcol, const size_int_t lline, const size_int_t lcol)

Sets a position in a source code given by four given parameters.

12.29.1 Detailed Description

Class for storage of position in a source code.

This class represents a position in a source code. Other classes representing various objects in the source code derive from this class to be possible to locate them in the code.

Position in the source code consists of four parameters:

- first line
- first column
- last line
- last column

12.29.2 Member Function Documentation

12.29.2.1 void get_source_pos (size_int_t & *fline*, size_int_t & *fcol*, size_int_t & *lline*, size_int_t & *lcol*) const

Copies the position in a source code to four given parameters.

Parameters:

fline = first line of position
fcol = first column of position
lline = last line of position
lcol = last column of position

12.29.2.2 void set_source_pos (const size_int_t *fline*, const size_int_t *fcol*, const size_int_t *lline*, const size_int_t *lcol*)

Sets a position in a source code given by four given parameters.

Parameters:

fline = first line of position
fcol = first column of position
lline = last line of position
lcol = last column of position

Referenced by `dve_expression_t::assign()`, and `dve_expression_t::swap()`.

The documentation for this class was generated from the following files:

- [dve_source_position.hh](#)
- `dve_source_position.cc`

12.30 dve_symbol_t Class Reference

```
#include <dve_symbol.hh>
```

Public Member Functions

- [dve_symbol_t](#) (const dve_sym_type_t t, const char *const n, const size_int_t l)
A constructor.
- size_int_t [get_gid](#) () const
Returns a [GID](#) of symbol.
- size_int_t [get_lid](#) () const
Returns a [LID](#) of symbol.
- const char * [get_name](#) () const
Returns a name of symbol.
- size_int_t [get_name_length](#) () const
Returns a length of name of symbol.
- size_int_t [get_process_gid](#) () const
Returns a [GID](#) of process in which the symbol is declared.
- size_int_t [get_sid](#) () const
Returns a [SID](#) of symbol.
- dve_sym_type_t [get_symbol_type](#) () const
Returns a type of symbol - see [dve_sym_type_t](#).
- bool [get_valid](#) () const
Returns whether symbol is considered valid (invalid==ignored).
- bool [is_channel](#) () const
Returns true iff symbol is a channel.
- bool [is_process](#) () const
Returns true iff symbol is a process.
- bool [is_state](#) () const
Returns true iff symbol is a process state.
- bool [is_variable](#) () const
Returns true iff symbol is a variable.
- void [set_gid](#) (const size_int_t gid_arg)

Sets a [GID](#) of symbol.

- void [set_lid](#) (const size_int_t lid_arg)
Sets a [LID](#) of symbol.
- void [set_name](#) (const char *const str, const size_int_t length)
Sets a name of a symbol and its length.
- void [set_name](#) (const char *new_name)
Sets a name of symbol - be aware of correct length of 'new_name'.
- void [set_process_gid](#) (const size_int_t nbr)
Sets a [GID](#) of process in which the symbol is declared.
- void [set_sid](#) (const size_int_t symb_id)
Sets a [SID](#) of symbol.
- void [set_symbol_type](#) (const dve_sym_type_t symb_type)
Sets a type of symbol - see [dve_sym_type_t](#).
- void [set_valid](#) (const bool is_valid)
Sets whether symbol is considered valid (invalid==ignored).
- [~dve_symbol_t](#) ()
A destructor.

Following functions can be called only for vector variables

- void [create_init_expr_field](#) ()
- const [dve_expression_t](#) * [get_init_expr](#) (const size_int_t i) const
Returns a pointer to the i-th initial value.
- size_int_t [get_init_expr_count](#) () const
Returns a count of values initializing the vector.
- std::vector< [dve_expression_t](#) * > * [get_init_expr_field](#) ()
Returns a pointer to the vector of initial values.
- size_int_t [get_vector_size](#) () const
Returns a size of the vector.
- void [no_init_expr_field](#) ()
Tells that array has no initial values.
- void [set_init_expr](#) (const size_int_t i, [dve_expression_t](#) *const trans)
Sets 'i'-th initial value from initial expression field.

- void [set_vector_size](#) (const size_int_t ar_size)

Sets a size of the vector.

Following functions can be called only for channel symbols

- slong_int_t [get_channel_buffer_size](#) () const
Returns a size of a buffer of typed channel.
- size_int_t [get_channel_item_count](#) () const
Returns a number of items transmittable through a channel simultaneously.
- dve_var_type_t [get_channel_type_list_item](#) (const size_int_t index) const
- size_int_t [get_channel_type_list_size](#) () const
- bool [get_channel_typed](#) () const
Returns true iff the channel is a typed channel.
- void [set_channel_buffer_size](#) (const slong_int_t buffer_size)
Sets a size of a buffer of typed channel.
- void [set_channel_item_count](#) (const size_int_t size)
Sets a number of items transmittable through a channel simultaneously.
- void [set_channel_type_list_item](#) (const size_int_t index, const dve_var_type_t type_item)
- void [set_channel_type_list_size](#) (const size_int_t size)
- void [set_channel_typed](#) (const bool typed)
Sets, whether channel is typed ('typed'==true) or not ('typed'==false).

Following functions can be called only for scalar variables

- const dve_expression_t * [get_init_expr](#) () const
- dve_expression_t * [get_init_expr](#) ()
Returns an initial value of scalar variable.
- void [set_init_expr](#) (dve_expression_t *const p_expr)
Sets an initial value of scalar variable.

Following functions can be called only for variable symbols

- dve_var_type_t [get_var_type](#) () const
Returns a type of variable - see dve_var_type_t.
- bool [is_byte](#) () const
Returns true iff a variable symbol is of byte type.
- bool [is_const](#) () const
Returns true iff a variable symbol is constant.

- bool [is_int](#) () const
Returns true iff a variable symbol is of integer type.
- bool [is_vector](#) () const
Returns true iff a variable symbol is a vector.
- void [set_const](#) (const bool const_var)
Sets, whether a variable symbol is constant.
- void [set_var_type](#) (const dve_var_type_t var_type)
Sets a type of variable - see [dve_var_type_t](#).
- void [set_vector](#) (const bool is_vector)
Sets, whether a variable symbol is a vector.

Static Public Attributes

- static const size_int_t [CHANNEL_UNUSED](#) = MAX_SIZE_INT
Return value [get_channel_item_count\(\)](#) iff channel has not been used.

12.30.1 Detailed Description

Class that represents the declaration of symbol (= process, channel, variable or process state).

[dve_symbol_t](#) represents so called symbol. Symbol is a named entity in a [system](#).

The common properties of symbols Each symbol has a unique identifier [SID](#). You can obtain it using [get_sid\(\)](#).

Each symbol has also an identifier [GID](#). [GID](#) is unique too, but only among the symbols of the same type. You can get it using [get_gid\(\)](#).

Locally declared symbols have also an identifier [LID](#). [LID](#) is unique among all symbols inside the same process. Only variables and process states can be declared locally. You can get [LID](#) of symbol using [get_lid\(\)](#). Method [get_lid\(\)](#) has a **special semantics in case of global symbols**.

Each locally declared symbol has a unique parent process (a process where the symbol was declared). You can obtain a [GID](#) of this process by method [get_process_gid\(\)](#). In case of global symbols method [get_process_gid\(\)](#) returns [NO_ID](#).

Each symbol has its name of course. You can get it using a method [get_name\(\)](#). The length of this name is available using a method [get_name_length\(\)](#).

It is requested to distinguish among separate types of symbols. You can do it using a method [get_symbol_type\(\)](#) or using a set of functions [is_variable\(\)](#), [is_process\(\)](#), [is_channel\(\)](#) and [is_state\(\)](#).

There are altogether 4 types of symbols - see [DVE Symbols](#) for details.

12.30.2 Constructor & Destructor Documentation

12.30.2.1 dve_symbol_t (const dve_sym_type_t t, const char *const n, const size_int_t l) [inline]

A constructor.

Parameters:

t = type of a symbol

n = name of a symbol

l = length of a name of a symbol

12.30.3 Member Function Documentation

12.30.3.1 void create_init_expr_field () [inline]

Creates a new field of initial values (be aware: old one should be deleted first)

12.30.3.2 size_int_t get_channel_item_count () const [inline]

Returns a number of items transmittable through a channel simultaneously.

For typed channels it is the same as [get_channel_type_list_size\(\)](#)

Returns:

initially CHANNEL_USUSED, until the number of items is set

Referenced by dve_parser_t::trans_sync().

12.30.3.3 dve_var_type_t get_channel_type_list_item (const size_int_t index) const [inline]

Returns a type of 'index'-th element of data sent simultaneously through a typed channel

Referenced by dve_explicit_system_t::get_async_enabled_trans_succ_without_property(), and dve_system_t::write().

12.30.3.4 size_int_t get_channel_type_list_size () const [inline]

Returns a count of types of values transmitted simultaneously through a typed channel

Referenced by dve_explicit_system_t::DBG_print_state(), dve_parser_t::trans_sync(), and dve_system_t::write().

12.30.3.5 `const dve_expression_t* get_init_expr () const` [inline]

The same as [get_init_expr\(\)](#) above, but for constant instances of this class.

12.30.3.6 `dve_expression_t* get_init_expr ()` [inline]

Returns an initial value of scalar variable.

Returns an initial value of scalar variable. Initial value is represented by a pointer to the expression (see [dve_expression_t](#)) and it can be 0, when there was no initial value.

Referenced by `dve_system_t::write()`.

12.30.3.7 `size_int_t get_lid () const` [inline]

Returns a [LID](#) of symbol.

Returns a [LID](#) of symbol. If the symbol is global, it returns:

- [GID](#) in case of process, channel and state
- index to the list of global variables (you can use it as a parameter of `system_t::get_global_variable_gid()`)

Referenced by `dve_parser_t::assertion_create()`, `dve_parser_t::expr_assign()`, `dve_transition_t::set_state1_gid()`, `dve_transition_t::set_state2_gid()`, `dve_parser_t::state_accept()`, `dve_parser_t::state_commit()`, `dve_parser_t::state_init()`, and `dve_parser_t::trans_sync()`.

12.30.3.8 `size_int_t get_process_gid () const` [inline]

Returns a [GID](#) of process in which the symbol is declared.

Returns a [GID](#) of process in which the symbol is declared. If the symbol is global, returns `NO_ID`.

Referenced by `dve_symbol_table_t::add_state()`, `dve_symbol_table_t::add_variable()`, `dve_parser_t::check_restrictions_put_on_property()`, `dve_parser_t::expr_assign()`, `dve_parser_t::trans_sync()`, and `dve_expression_t::write()`.

12.30.3.9 `bool is_byte () const` [inline]

Returns true iff a variable symbol is of byte type.

Returns true iff a variable symbol is of type byte - that is `dve_symbol_t::get_var_type()` returns `VAR_BYTE`.

Referenced by `dve_system_t::write()`.

12.30.3.10 bool is_int () const [inline]

Returns true iff a variable symbol is of integer type.

Returns true iff a variable symbol is of integer type - that is [dve_symbol_t::get_var_type\(\)](#) returns VAR_INT.

12.30.3.11 void no_init_expr_field () [inline]

Tells that array has no initial values.

Sets a internal pointer to is field of initial values to zero. Thus tells, that there are no initial values for the field

12.30.3.12 void set_channel_item_count (const size_int_t size) [inline]

Sets a number of items transmittable through a channel simultaneously.

For typed channels it is the same as [set_channel_type_list_size\(\)](#)

Referenced by [dve_parser_t::trans_sync\(\)](#).

12.30.3.13 void set_channel_type_list_item (const size_int_t index, const dve_var_type_t type_item) [inline]

Sets a type of 'index'-th element of data sent simultaneously through a typed channel

12.30.3.14 void set_channel_type_list_size (const size_int_t size) [inline]

Sets a count of types of values transmitted simultaneously through a typed channel

12.30.4 Member Data Documentation**12.30.4.1 const size_int_t CHANNEL_UNUSED = MAX_SIZE_INT**
[static]

Return value [get_channel_item_count\(\)](#) iff channel has not been used.

Constant returned by [get_channel_item_count\(\)](#) if the number of items transmittable simultaneously through a channel has not been initialized (by the first usage of channel)

Referenced by [dve_parser_t::trans_sync\(\)](#).

The documentation for this class was generated from the following files:

- [dve_symbol.hh](#)
- [dve_symbol.cc](#)

12.31 dve_symbol_table_t Class Reference

Class which stores the declaration of symbols (see [dve_symbol_t](#)).

```
#include <dve_symbol_table.hh>
```

Public Member Functions

- void [add_channel](#) ([dve_symbol_t](#) *const symbol)
Appends a symbol of channel to the list of symbols of channels.
- void [add_process](#) ([dve_symbol_t](#) *const symbol)
Appends a symbol of process to the list of symbols of processes.
- void [add_state](#) ([dve_symbol_t](#) *const symbol)
Appends a symbol of state to the list of symbols of process states.
- void [add_variable](#) ([dve_symbol_t](#) *const symbol)
Appends a symbol of variable to the list of symbols of variables.
- [dve_symbol_table_t](#) ([error_vector_t](#) &evect)
- std::size_t [find_global_symbol](#) (const char *name) const
Searches for a symbol in the list of global symbols.
- std::size_t [find_symbol](#) (const char *name, const std::size_t proc_gid) const
Searches for a symbol in a given process.
- std::size_t [find_visible_symbol](#) (const char *name, const std::size_t proc_gid) const
- bool [found_global_symbol](#) (const char *name) const
Returns, whether there exists a global symbol of name 'name'.
- bool [found_symbol](#) (const char *name, const std::size_t proc_gid) const
Returns, whether symbol exists.
- const [dve_symbol_t](#) * [get_channel](#) (const std::size_t gid) const
Returns a pointer to the constant symbol of channel with [GID](#) 'gid'.
- [dve_symbol_t](#) * [get_channel](#) (const std::size_t gid)
Returns a pointer to the symbol of channel with [GID](#) 'gid'.
- std::size_t [get_channel_count](#) () const
Returns a count of all channels.
- const [dve_symbol_t](#) * [get_process](#) (const std::size_t gid) const
Returns a pointer to the constant symbol of process with [GID](#) 'gid'.

- [dve_symbol_t * get_process](#) (const std::size_t gid)
*Returns a pointer to the symbol of process with **GID** 'gid'.*
- std::size_t [get_process_count](#) () const
Returns a count of all variables.
- const [dve_symbol_t * get_state](#) (const std::size_t gid) const
*Returns a pointer to the constant symbol of process state with **GID** 'gid'.*
- [dve_symbol_t * get_state](#) (const std::size_t gid)
*Returns a pointer to the symbol of process state with **GID** 'gid'.*
- std::size_t [get_state_count](#) () const
Returns a count of all variables.
- const [dve_symbol_t * get_symbol](#) (const std::size_t sid) const
- [dve_symbol_t * get_symbol](#) (const std::size_t sid)
*Returns the pointer on a symbol that has a given **SID**.*
- std::size_t [get_symbol_count](#) () const
Returns the count of all symbols.
- const [dve_symbol_t * get_variable](#) (const std::size_t gid) const
*Returns a pointer to the constant symbol of variable with **GID** 'gid'.*
- [dve_symbol_t * get_variable](#) (const std::size_t gid)
*Returns a pointer to the symbol of variable with **GID** 'gid'.*
- std::size_t [get_variable_count](#) () const
Returns a count of all variables.
- const char * [save_token](#) (const char *const token)

Protected Attributes

- array_t< [dve_symbol_t](#) * > **channel_syms**
- array_t< [dve_symbol_t](#) * > **global_symbols**
- array_t< [dve_symbol_t](#) * > **process_syms**
- array_t< std::size_t > **processes_of_symbols**
- array_t< [dve_symbol_t](#) * > **state_syms**
- array_t< [dve_symbol_t](#) * > **symbols**
- array_t< array_t< [dve_symbol_t](#) * > * > **symbols_of_processes**
- **error_vector_t** & **terr**
- [dve_token_vector_t](#) **token_vector**
- array_t< [dve_symbol_t](#) * > **variable_syms**

12.31.1 Detailed Description

Class which stores the declaration of symbols (see [dve_symbol_t](#)).

12.31.2 Constructor & Destructor Documentation

12.31.2.1 `dve_symbol_table_t (error_vector_t & evect)` `[inline]`

METHODS: A constructor.

12.31.3 Member Function Documentation

12.31.3.1 `void add_process (dve_symbol_t *const symbol)`

Appends a symbol of process to the list of symbols of processes.

Warning:

Important: If the [symbol table](#) is the part of [system](#), **DO NOT USE THIS METHOD**. It would cause the inconsistencies. To avoid it, please use [dve_system_t::add_process\(\)](#).

References [dve_symbol_t::get_name\(\)](#), [array_t::push_back\(\)](#), [save_token\(\)](#), [dve_symbol_t::set_gid\(\)](#), [dve_symbol_t::set_name\(\)](#), [dve_symbol_t::set_sid\(\)](#), and [array_t::size\(\)](#).

12.31.3.2 `std::size_t find_global_symbol (const char * name) const`

Searches for a symbol in the list of global symbols.

Parameters:

name = name of symbol

Returns:

[SID](#) of symbol if the symbol exists in the list of global symbols, otherwise it returns [NO_ID](#).

References [find_visible_symbol\(\)](#), and [array_t::size\(\)](#).

Referenced by [dve_parser_t::expr_array_mem\(\)](#), [dve_parser_t::expr_id\(\)](#), [dve_parser_t::expr_state_of_process\(\)](#), [dve_parser_t::expr_var_of_process\(\)](#), [find_visible_symbol\(\)](#), [dve_parser_t::system_property\(\)](#), and [dve_parser_t::trans_sync\(\)](#).

12.31.3.3 `std::size_t find_symbol (const char * name, const std::size_t proc_gid) const`

Searches for a symbol in a given process.

Parameters:

name = name of symbol

proc_gid = GID of process in which we look for a symbol

Returns:

SID of symbol if the symbol exists in a process with GID *proc_gid*, otherwise it returns NO_ID.

References array_t::size().

Referenced by dve_parser_t::assertion_create(), dve_parser_t::expr_state_of_process(), find_visible_symbol(), dve_parser_t::state_accept(), dve_parser_t::state_commit(), dve_parser_t::state_init(), and dve_parser_t::trans_create().

12.31.3.4 std::size_t find_visible_symbol (const char * *name*, const std::size_t *proc_gid*) const

Searches for a symbol in a given process and in the list of global symbols.

This method searches for a symbol visible from a process with GID *proc_gid*. First it searches in the list of symbols of a given process and only then it searches in the list of global symbols.

Parameters:

name = name of symbol

proc_gid = GID of process in which we look for a symbol

Returns:

SID of symbol if the symbol exists in a process or in the list of global symbols, otherwise it returns NO_ID.

References find_global_symbol(), and find_symbol().

Referenced by dve_parser_t::expr_array_mem(), dve_parser_t::expr_id(), and find_global_symbol().

12.31.3.5 bool found_global_symbol (const char * *name*) const

Returns, whether there exists a global symbol of name 'name'.

Searches for a symbol in the list of global symbols.

Parameters:

name = name of symbol

Returns:

true iff symbol of name *name* exists and it is a global symbol

References array_t::size().

12.31.3.6 `bool found_symbol (const char * name, const std::size_t proc_gid) const`

Returns, whether symbol exists.

Searches for a symbol in a selected process

Parameters:

name = name of symbol

proc_gid = [GID](#) of process

Returns:

true iff symbol was found

References `array_t::size()`.

12.31.3.7 `std::size_t get_channel_count () const` `[inline]`

Returns a count of all channels.

The set of channels in a [symbol table](#) should be the same as the set of channels in the entire [system](#) that contains this [symbol table](#) (if there is such a [system](#))

12.31.3.8 `std::size_t get_process_count () const` `[inline]`

Returns a count of all variables.

The set of processes in a [symbol table](#) should be the same as the set of processes in the entire [system](#) that contains this [symbol table](#) (if there is such a [system](#))

12.31.3.9 `std::size_t get_state_count () const` `[inline]`

Returns a count of all variables.

The set of process states in a [symbol table](#) should be the same as the set of process states in the entire [system](#) that contains this [symbol table](#) (if there is such a [system](#))

Referenced by `por_t::init()`.

12.31.3.10 `const dve_symbol_t* get_symbol (const std::size_t sid) const` `[inline]`

The same as [get_symbol\(\)](#) above, but this is used in a constant instances of [dve_symbol_table_t](#)

12.31.3.11 std::size_t get_symbol_count () const [inline]

Returns the count of all symbols.

Returns the count of all symbols stored in a [symbol table](#). The set of symbols in a [symbol table](#) should be the same as the union of sets of all variables, channels, processes and process states in the entire [system](#) that contains this [symbol table](#) (if there is such a [system](#))

12.31.3.12 std::size_t get_variable_count () const [inline]

Returns a count of all variables.

The set of variables in a [symbol table](#) should be the same as the set of variables in the entire [system](#) that contains this [symbol table](#) (if there is such a [system](#))

12.31.3.13 const char* save_token (const char *const token) [inline]

Stores given string into the internal structure and returns the pointer to this (newly created) string.

Referenced by `add_channel()`, `add_process()`, `add_state()`, and `add_variable()`.

The documentation for this class was generated from the following files:

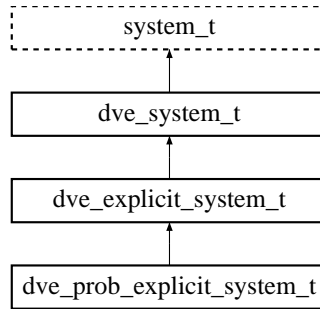
- [dve_symbol_table.hh](#)
- [dve_symbol_table.cc](#)

12.32 dve_system_t Class Reference

Class for a DVE system representation.

```
#include <dve_system.hh>
```

Inheritance diagram for dve_system_t::



Public Member Functions

- [dve_system_t](#) ([error_vector_t](#) &evect=[gerr](#))

A constructor.

- virtual [~dve_system_t](#) ()

A destructor.

Methods modifying a system

These methods are implemented and [can_be_modified\(\)](#) returns true.

- virtual void [add_process](#) ([process_t](#) *const process)
- virtual void [remove_process](#) (const [size_int_t](#) process_id)

Implements [system_t::remove_process\(\)](#).

DVE system specific methods

These methods are implemented only in DVE system and they cannot be run using an abstract interface of [system_t](#).

- void [DBG_print_all_initialized_variables](#) ()

For debugging purposes.

- [size_int_t](#) [get_channel_count](#) () const

Returns a count of channels in the system.

- [size_int_t](#) [get_channel_freq](#) ([size_int_t](#) ch_gid) const

Returns count of channel usage in processes ('ch_gid' = GID of channel).

- size_int_t [get_channel_freq_ask](#) (size_int_t ch_gid) const
- size_int_t [get_channel_freq_exclaim](#) (size_int_t ch_gid) const
- size_int_t [get_global_variable_count](#) () const
Returns a count of global [variables](#).
- size_int_t [get_global_variable_gid](#) (const size_int_t i) const
Returns a [GID](#) of global [variables](#).
- bool [get_property_with_synchro](#) () const
Returns true, iff whether the property process contains any synchronizaiton.
- dve_symbol_table_t * [get_symbol_table](#) () const
Returns pointer to the [symbol table](#).
- system_synchronicity_t [get_system_synchro](#) () const
Returns a synchronicity of the [system](#).
- bool [not_in_limits](#) (dve_var_type_t var_type, all_values_t value)
Returns, whether value of 'value' is in bounds of type 'var_type'.
- void [set_property_with_synchro](#) (const bool contains_synchro)
Sets, whether the property process contains any synchronizaiton.
- void [set_system_synchro](#) (const system_synchronicity_t sync)
Sets a synchronicity of the [system](#).

Obligatory part of abstact interface

These methods have to implemented in each implementation of [system_t](#)

- virtual slong_int_t [from_string](#) (const std::string str)
Implements [system_t::from_string\(const std::string str\)](#) in DVE [system](#).
- virtual slong_int_t [read](#) (const char *const filename)
*Implements [system_t::read\(const char * const filename\)](#) in DVE [system](#).*
- virtual slong_int_t [read](#) (std::istream &ins=std::cin)
Implements [system_t::read\(std::istream & ins = std::cin\)](#) in DVE [system](#).
- virtual std::string [to_string](#) ()
Implements [system_t::to_string\(\)](#) in DVE [system](#).
- virtual void [write](#) (std::ostream &outs=std::cout)
Implements [system_t::write\(std::ostream & outs = std::cout\)](#) in DVE [system](#).
- virtual bool [write](#) (const char *const filename)
*Implements [system_t::write\(const char * const filename\)](#) in DVE [system](#).*

Methods working with processes

These methods are implemented and *can_processes()* returns true.

- virtual const [process_t](#) * [get_process](#) (const size_int_t id) const
Implements [system_t::get_process\(const size_int_t id\) const](#) in DVE system.
- virtual [process_t](#) * [get_process](#) (const size_int_t gid)
Implements [system_t::get_process\(const size_int_t gid\)](#) in DVE system.
- virtual size_int_t [get_process_count](#) () const
Implements [system_t::get_process_count\(\)](#) in DVE system.

Methods working with property process

These methods are implemented and *can_property_process()* returns true

- virtual size_int_t [get_property_gid](#) () const
Implements [system_t::get_property_gid\(\)](#) in DVE system.
- virtual const [process_t](#) * [get_property_process](#) () const
Implements [system_t::get_property_process\(\)](#) in DVE system.
- virtual [process_t](#) * [get_property_process](#) ()
Implements [system_t::get_property_process\(\)](#) in DVE system.
- virtual property_type_t [get_property_type](#) ()
Implements [system_t::get_property_type\(\)](#) in DVE system.
- virtual void [set_property_gid](#) (const size_int_t gid)
Implements [system_t::set_property_gid\(\)](#) in DVE system.

Methods to check the SCCs of property process graph

These methods are implemented only if *can_decompose_property()* returns true.

- virtual int [get_property_scc_count](#) () const
- virtual int [get_property_scc_id](#) (state_t &_state) const
- virtual int [get_property_scc_type](#) (state_t &_state) const
- virtual int [get_property_scc_type](#) (int _scc) const
- virtual bool [is_property_weak](#) () const

Methods working with transitions

These methods are implemented and *can_transitions()* returns true.

- virtual size_int_t [get_trans_count](#) () const
- virtual const [transition_t](#) * [get_transition](#) (size_int_t gid) const
Implements [system_t::get_transition\(size_int_t gid\) const](#) in DVE system.
- virtual [transition_t](#) * [get_transition](#) (size_int_t gid)
Implements [system_t::get_transition\(size_int_t gid\)](#) in DVE system.

Protected Member Functions

- all_values_t **eval_expr** (const dve_expression_t *const expr, bool &eval_err)
const
Evaluates an expression.
- all_values_t **fast_eval** (compacted_viewer_t *p_compacted, bool &eval_err)
const
Evaluates a compacted expression.

Protected Attributes

- size_int_t **channel_count**
- bool * **constants**
- all_values_t(* **eval_dot**)(const void *parameters, const dve_expression_t &subexpr, bool &eval_err)
- all_values_t(* **eval_dot_compact**)(const void *parameters, const dve_symbol_table_t *symbol_table, const size_int_t &gid, bool &eval_err)
- all_values_t(* **eval_id**)(const void *parameters, const dve_expression_t &expr, bool &eval_err)
- all_values_t(* **eval_id_compact**)(const void *parameters, const size_int_t &gid, bool &eval_err)
- all_values_t(* **eval_square_bracket**)(const void *parameters, const dve_expression_t &subexpr, const size_int_t &array_index, bool &eval_err)
- all_values_t(* **eval_square_bracket_compact**)(const void *parameters, const size_int_t &gid, const size_int_t &array_index, bool &eval_err)
- size_int_t **glob_var_count**
count of global variables
- size_int_t * **initial_states**
- SYS_initial_values_t * **initial_values**
initial values for some symbols (see union SYS_initial_values_t)
- size_int_t * **initial_values_counts**
additional information to 'initial_values'
- const size_int_t **MAX_INIT_VALUES_SIZE**
Constant set to initial_values_counts for scalar variables with initializer.
- void * **parameters**
- dve_process_t * **pproperty**
The pointer to the property process.
- size_int_t **process_field_size**
capacity of field 'processes'

- [array_t](#) < [dve_process_t](#) * > [processes](#)
field of processes (0th element is not used)
- [size_int_t](#) [prop_gid](#)
GID of the property process
- [bool](#) [property_has_synchronization](#)
Says, whether the property contains synchronization.
- [dve_symbol_table_t](#) * [psymbol_table](#)
"Symbol table"
- [size_int_t](#) [state_count](#)
- [size_int_t](#) * [state_lids](#)
- [system_synchronicity_t](#) [system_synchro](#)
- [size_int_t](#) [var_count](#)
- [dve_var_type_t](#) * [var_types](#)
types of [variables](#) (GID of [variable](#) is an index to this field)

12.32.1 Detailed Description

Class for a DVE system representation.

This class implements the abstract interface [system_t](#).

This implementation contains the **symbol table** ([dve_symbol_table_t](#)) (as its element). It can read in and write out DVE source files. It can also offers basic informations about the system represented read from DVE source file. It implements the full DiVinE abstract interface and the model of the system has the complete structure of DiVinE system - it means, that processes, transitions and expressions are supported.

Furthermore it provides the set of methods, which are purely DVE system specific.

The very DVE system specific properties: (1) System can be synchronous or asynchronous. These two types of system differ in runs which they can do. In the synchronous system all processes have to do the transition each round of run of the system. In the asynchronous system only one process does do the transition except for transitions which are synchronized and transitions of the property process. Therefore in the asynchronous system there can simultaneously work at most 3 processes (the second process is synchronized with the first process and the third process can be only the property process). Such a tuple of transitions which can be used in a given state of the asynchronous system is called [enabled transition](#).

(2) The system needs to know the declarations of channels, [variables](#), states and processes. For that reason it uses the class [dve_symbol_table_t](#) ([symbol table](#)) to store these declarations separatelly. Therefore [system_t](#) itself doesn't provide many informations about the declarations of symbols. To get these informations you should get the [symbol table](#) first. You can use the function `system_t::get_symbol_table()`.

12.32.2 Constructor & Destructor Documentation

12.32.2.1 dve_system_t (error_vector_t & *evect* = gerr)

A constructor.

Parameters:

evect = the **error vector**, that will be used by created instance of [system_t](#)

References [system_abilities_t::explicit_system_can_evaluate_expressions](#), [system_abilities_t::explicit_system_can_system_transitions](#), [system_t::get_abilities\(\)](#), [initial_values](#), [initial_values_counts](#), [parameters](#), [system_abilities_t::process_can_be_modified](#), [system_abilities_t::process_can_read](#), [system_abilities_t::process_can_transitions](#), [property_has_synchronization](#), [psymbol_table](#), [system_abilities_t::system_can_be_modified](#), [system_abilities_t::system_can_processes](#), [system_abilities_t::system_can_property_process](#), [system_abilities_t::system_can_transitions](#), [system_abilities_t::transition_can_be_modified](#), [system_abilities_t::transition_can_read](#), and [var_types](#).

12.32.3 Member Function Documentation

12.32.3.1 void add_process (process_t *const *process*) [virtual]

Implements [system_t::add_process\(\)](#) in DVE [system](#), but see also implementation specific notes below.

Important thing is, that this is the only permitted way of adding the process to the [system](#). Program should not use [dve_symbol_table_t::add_process\(\)](#) instead of this method.

Implements [system_t](#).

References [process_t::get_gid\(\)](#), [dve_symbol_table_t::get_process\(\)](#), [processes](#), [psymbol_table](#), and [array_t::push_back\(\)](#).

12.32.3.2 void DBG_print_all_initialized_variables ()

For debugging purposes.

Prints all [variables](#) with their initial values

References [SYS_initial_values_t::all_value](#), [dve_symbol_t::get_name\(\)](#), [dve_symbol_table_t::get_variable\(\)](#), [initial_values](#), [initial_values_counts](#), [dve_symbol_t::is_vector\(\)](#), and [psymbol_table](#).

12.32.3.3 all_values_t eval_expr (const dve_expression_t *const *expr*, bool & *eval_err*) const [protected]

Evaluates an expression.

Carries out complete expression evaluation. Evaluation of [variables](#) and state identifiers can be changed in the descendant by changing protected [variables](#) ‘eval_id’, ‘eval_square_bracket’, and ‘eval_dot’.

Parameters:

expr = pointer to structure representing expression

eval_err = boolean that says, whether evaluation passed correctly

References SYS_initial_values_t::all_value, SYS_initial_values_t::all_values, dve_expression_t::arity(), eval_dot, eval_id, eval_square_bracket, fast_eval(), dve_expression_t::get_ident_gid(), dve_expression_t::get_operator(), dve_expression_t::get_p_compact(), dve_expression_t::get_value(), initial_values, initial_values_counts, dve_expression_t::is_compacted(), dve_expression_t::left(), parameters, dve_expression_t::right(), and dve_expression_t::to_string().

Referenced by dve_explicit_system_t::eval_expr().

12.32.3.4 all_values_t fast_eval (compacted_viewer_t * p_compacted, bool & eval_err) const [protected]

Evaluates a compacted expression.

Carries out complete expression evaluation over compacted expression. Evaluation of [variables](#) and state identifiers can be changed in the descendant by changing protected [variables](#) ‘eval_id_compact’, ‘eval_square_bracket_compact’, and ‘eval_dot_compact’.

Parameters:

p_compacted = pointer to compacted expression

eval_err = boolean that says, whether evaluation passed correctly

References SYS_initial_values_t::all_value, SYS_initial_values_t::all_values, gerr, compacted_t::get_gid(), compacted_t::get_operator(), compacted_t::get_value(), initial_values, initial_values_counts, compacted_t::left(), parameters, psymbol_table, compacted_t::ptr, compacted_t::right(), and compacted_t::to_string().

Referenced by eval_expr(), and dve_explicit_system_t::eval_expr().

12.32.3.5 size_int_t get_channel_freq_ask (size_int_t ch_gid) const [inline]

Returns count of channel usage in processes in a synchronization with a question mark.

12.32.3.6 size_int_t get_channel_freq_exclaim (size_int_t ch_gid) const [inline]

Returns count of channel usage in processes in a synchronization with a exclamation mark.

12.32.3.7 `size_int_t get_global_variable_gid (const size_int_t i) const` `[inline]`

Returns a [GID](#) of global [variables](#).

Returns a [GID](#) of global [variables](#). *i* is an index to the list of global [variables](#) (in an interval 0..(`get_global_variable_count()`-1)). You can obtain this index by `dve_symbol_t::get_lid()`.

12.32.3.8 `virtual int get_property_scc_count () const` `[inline, virtual]`

Returns the number of SCCs in the decomposition.

References `gerr`.

12.32.3.9 `virtual int get_property_scc_id (state_t & _state) const` `[inline, virtual]`

Returns id of an SCC that the given state projects to.

References `gerr`, and `state_t::ptr`.

12.32.3.10 `virtual int get_property_scc_type (state_t & _state) const` `[inline, virtual]`

Returns type of an SCC that the given state projects to, where 0 means nonaccepting component, 1 means partially accepting component, and 2 means fully accepting component.

References `gerr`, and `state_t::ptr`.

12.32.3.11 `virtual int get_property_scc_type (int _scc) const` `[inline, virtual]`

Returns type of the given SCC, where 0 means nonaccepting component, 1 means partially accepting component, and 2 means fully accepting component.

References `gerr`.

12.32.3.12 `dve_symbol_table_t* get_symbol_table () const` `[inline]`

Returns pointer to the [symbol table](#).

[Symbol table](#) (= instance of class `dve_symbol_table_t`) is automatically created in creation time of `system_t` as its private data member. It contains all declarations of all named symbols (processes, channels, [variables](#) and process states).

See [symbol table](#) and `dve_symbol_table_t` for details.

Referenced by `dve_transition_t::get_symbol_table()`, `dve_process_t::get_symbol_table()`, `dve_expression_t::get_symbol_table()`, and `por_t::init()`.

12.32.3.13 `system_synchronicity_t get_system_synchro () const` [inline]

Returns a synchronicity of the [system](#).

Returns:

value of type `system_synchronicity_t`:

- `SYSTEM_ASYNC` iff system is defined as asynchronous
- `SYSTEM_SYNC` iff system is defined as synchronous

12.32.3.14 `virtual size_int_t get_trans_count () const` [inline, virtual]

Implements `system_t::get_trans_count()` in DVE [system](#), but see also implementation specific notes below

Returned count of transitions comprises also invalid transitions (transitions, where `transition_t::get_valid()` returns false)

Implements [system_t](#).

Referenced by `por_t::init()`, and `por_t::static_c3()`.

12.32.3.15 `virtual bool is_property_weak () const` [inline, virtual]

Returns whether the process has a weak graph.

References `gerr`.

12.32.3.16 `void set_property_with_synchro (const bool contains_synchro)` [inline]

Sets, whether the property process contains any synchronizaiton.

Parameters:

contains_synchro = true, iff the property process contains any synchronizaiton

12.32.4 Member Data Documentation

12.32.4.1 `all_values_t(* eval_dot)(const void *parameters, const dve_expression_t &subexpr, bool &eval_err)` [protected]

functional [variable](#) that points to the function, which should be used to evaluate state identifiers

Referenced by `eval_expr()`.

12.32.4.2 all_values_t(* eval_id)(const void *parameters, const dve_expression_t &expr, bool &eval_err) [protected]

functional [variable](#) that points to the function, which should be used to evaluate [variables](#)
Referenced by eval_expr().

12.32.4.3 all_values_t(* eval_square_bracket)(const void *parameters, const dve_expression_t &subexpr, const size_int_t &array_index, bool &eval_err) [protected]

functional [variable](#) that points to the function, which should be used to evaluate [variable](#) fields
Referenced by eval_expr().

12.32.4.4 size_int_t* initial_values_counts [protected]

additional information to 'initial_values'

'initial_values_counts' is the additional information to 'initial_values': initial_values_counts[i] =

- 0 => no initial value
- MAX_INIT_VALUES_SIZE => scalar, state or procname
- 1..(MAX_INIT_VALUES_SIZE-1) => vector

Referenced by DBG_print_all_initialized_variables(), dve_system_t(), eval_expr(), fast_eval(), and dve_explicit_system_t::get_initial_state().

12.32.4.5 void* parameters [protected]

parameters points to the piece of the memory where additional informations for 'eval_*' functions are stored.

Referenced by dve_system_t(), eval_expr(), fast_eval(), and dve_explicit_system_t::~dve_explicit_system_t().

The documentation for this class was generated from the following files:

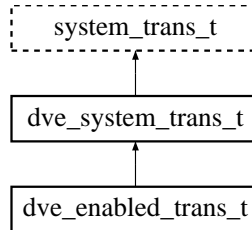
- [dve_system.hh](#)
- [dve_system.cc](#)

12.33 dve_system_trans_t Class Reference

Class implementing system transition in DVE [system](#).

```
#include <dve_system_trans.hh>
```

Inheritance diagram for dve_system_trans_t::



Public Member Functions

- [dve_system_trans_t](#) (const [dve_system_trans_t](#) &second)
A copy constructor.
- [dve_system_trans_t](#) ()
A constructor.
- virtual [system_trans_t](#) & [operator=](#) (const [system_trans_t](#) &second)
Implements [system_trans_t::operator=\(\)](#) in DVE system.
- virtual std::string [to_string](#) () const
Implements [system_trans_t::to_string\(\)](#) in DVE system.
- virtual void [write](#) (std::ostream &ostr) const
Implements [system_trans_t::write\(\)](#) in DVE system.
- virtual [~dve_system_trans_t](#) ()
A destructor.

Methods accessing transitions forming a system transition

These methods are implemented and [system_t::can_transitions\(\)](#) in the system that has generated the instance of this class returns true.

- virtual size_int_t [get_count](#) () const
Implements [system_trans_t::get_count\(\)](#) in DVE system.
- virtual [transition_t](#) *const & [operator\[\]](#) (const int i) const
Implements [system_trans_t::operator\[\]\(const int i\) const](#) in DVE system.

- virtual [transition_t](#) *[operator\[\]](#) (const int i)
Implements [system_trans_t::operator\[\]](#)(const int i).
- virtual void [set_count](#) (const size_int_t new_count)
Implements [system_trans_t::set_count\(\)](#) in DVE system.

Protected Member Functions

- void [copy_from](#) (const [system_trans_t](#) &second)
- void [create_from](#) (const [system_trans_t](#) &second)

12.33.1 Detailed Description

Class implementing system transition in DVE [system](#).

The documentation for this class was generated from the following files:

- [dve_system_trans.hh](#)
- [dve_system_trans.cc](#)

12.34 `dve_token_vector_t` Class Reference

Class used by [dve_symbol_table_t](#) to store the names of symbols.

```
#include <dve_token_vector.hh>
```

Public Member Functions

- `const char * save_token` (`const char *const token`)

12.34.1 Detailed Description

Class used by [dve_symbol_table_t](#) to store the names of symbols.

This is an internal type. Programmer of algorithms using DiVinE usually should not use this structure.

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

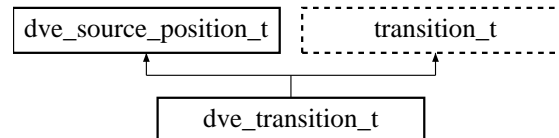
- [dve_token_vector.hh](#)

12.35 dve_transition_t Class Reference

Class representing a transition.

```
#include <dve_transition.hh>
```

Inheritance diagram for dve_transition_t::



Public Member Functions

- [dve_transition_t](#) ([system_t](#) *const system)
- [dve_transition_t](#) ()
A constructor (no system set - call [set_parent_system\(\)](#) after creation).
- virtual [~dve_transition_t](#) ()
A destructor.

DVE system specific methods

These methods are implemented only in DVE system and they cannot be run using an abstract interface of [transition_t](#).

- void [add_effect](#) ([dve_expression_t](#) *const effect)
Adds an effect to the list of effects contained in a transition.
- void [alloc_glob_mask](#) (const [size_int_t](#) count_of_glob_vars)
- [size_int_t](#) [get_channel_gid](#) () const
Returns a [GID](#) of channel used in a synchronisation of this transition.
- [dve_expression_t](#) * [get_effect](#) ([size_int_t](#) eff_nbr) const
Returns an `eff_nbr-ith` effect.
- [size_int_t](#) [get_effect_count](#) () const
Returns a count of effects in this transition.
- void [get_effect_expr_string](#) ([size_int_t](#) i, std::string &str) const
Returns the string representation of `i-th` effect in the argument str.
- const [bit_string_t](#) & [get_glob_mask](#) () const
Returns a bit mask of assigned variables.
- const [dve_expression_t](#) * [get_guard](#) () const

- `dve_expression_t * get_guard ()`
Returns a guard of this transition.
- `bool get_guard_string (std::string &str) const`
Sets the string representation of guard or returns false.
- `size_int_t get_partial_id () const`
- `size_int_t get_process_gid () const`
*Returns a *process GID* of process owning this transition.*
- `size_int_t get_state1_gid () const`
*Returns a *GID* of first (starting) state of a transition.*
- `size_int_t get_state1_lid () const`
*Returns a *LID* of first (starting) state of a transition.*
- `const char * get_state1_name () const`
Returns the name of the first state.
- `size_int_t get_state2_gid () const`
*Returns a *GID* of a second (finishing) state of a transition.*
- `size_int_t get_state2_lid () const`
*Returns a *LID* of a second (finishing) state of a transition.*
- `const char * get_state2_name () const`
Returns the name of the second state.
- `dve_symbol_table_t * get_symbol_table () const`
Returns a symbol table corresponding to this transition.
- `const char * get_sync_channel_name () const`
Returns the name of channel or 0 (see details).
- `const dve_expression_t * get_sync_expr_list_item (const size_int_t i) const`
- `dve_expression_t * get_sync_expr_list_item (const size_int_t i)`
Returns the 'i'-th expression in a synchronization written after '!' or '?'.
- `size_int_t get_sync_expr_list_size () const`
Returns a count of expressions written after '!' or '?'.
- `bool get_sync_expr_string (std::string &str) const`
- `sync_mode_t get_sync_mode () const`
Returns a synchronization mode of this transition.
- `bool get_valid () const`
Returns, whether this transition is valid or not.
- `bool is_sync_ask () const`
Returns true iff transition contains synchronisation of type 'question'.

- bool [is_sync_exclaim](#) () const
Returns true iff transition contains synchronisation of type 'exclaim'.
- bool [is_without_sync](#) () const
Returns true iff transition contains no synchronisation.
- void [set_channel_gid](#) (const size_int_t gid_of_channel)
Sets a [GID](#) of channel used in a synchronisation of this transition.
- void [set_glob_mark](#) (const size_int_t i, const bool mark)
Turns bit i-th bit in a global variables mask to 'mark'.
- void [set_guard](#) (dve_expression_t *const guard_expr)
Sets a guard of this transition.
- void [set_partial_id](#) (const size_int_t partial_id)
Sets [Partial ID](#) of this transition.
- void [set_process_gid](#) (const size_int_t gid)
Sets a [process GID](#) of process owning this transition.
- void [set_state1_gid](#) (const size_int_t state_gid)
Sets a [GID](#) of first (starting) state of a transition.
- void [set_state2_gid](#) (const size_int_t state_gid)
Sets a [GID](#) of a second (finishing) state of a transition.
- void [set_sync_expr_list_item](#) (const size_int_t i, [dve_expression_t](#) *const synchro_expr)
Sets the 'i'-th expression in a synchronization written after '!' of '??'.
- void [set_sync_expr_list_size](#) (const size_int_t size)
Sets a count of expression written after '!' or '??'.
- void [set_sync_mode](#) (const [sync_mode_t](#) synchro_mode)
Sets a [sync_mode](#) mode of this transition.
- void [set_valid](#) (const bool valid_value)
Sets the validity of the transition ('true' means valid).

Methods for reading a transition from a string representation

These methods are implemented only if [can_read\(\)](#) returns true.

- virtual int [from_string](#) (std::string &trans_str, const size_int_t process_gid=NO_ID)
Implements [transition_t::from_string\(\)](#) in DVE system.

- virtual int [read](#) (std::istream &istr, size_int_t process_gid=NO_ID)

Implements [transition_t::read\(\)](#) in DVE system.

Obligatory part of abstract interface

- virtual std::string [to_string](#) () const

Implements [transition_t::to_string\(\)](#) in DVE system.

- virtual void [write](#) (std::ostream &ostr) const

Implements [transition_t::write\(\)](#) in DVE system.

Protected Types

- typedef [array_t](#)< [dve_expression_t](#) * > **effects_container_t**

Protected Attributes

- size_int_t **channel_gid**
- [effects_container_t](#) **effects**
- [bit_string_t](#) **glob_mask**
- [dve_expression_t](#) * **guard**
- size_int_t **part_id**
- size_int_t **process_gid**
- size_int_t **state1_gid**
- size_int_t **state1_lid**
- size_int_t **state2_gid**
- size_int_t **state2_lid**
- [array_t](#)< [dve_expression_t](#) * > **sync_exprs**
- [sync_mode_t](#) **sync_mode**
- bool **valid**

12.35.1 Detailed Description

Class representing a transition.

This class implements the abstract interface [transition_t](#).

Notice, there is the set of methods, which are not virtual and they are already present in the class [transition_t](#) (e. g. [transition_t::get_gid\(\)](#), [transition_t::get_lid\(\)](#), ...) and there are also virtual methods with default implementation. They are not changed in [dve_transition_t](#) (e. g. [transition_t::set_gid\(\)](#), [transition_t::set_lid\(\)](#), ...).

It supports the full set of methods of an abstract interface and adds many DVE specific methods to this class, corresponding to the DVE point of view to transitions of processes.

Very DVE system specific feature: This class also contains a global variables mask that tells which global variables are used in this transition. This mask is maintained by [system](#) during its creation or consolidation. Functions for access to this mask are [get_glob_mask\(\)](#), [set_glob_mask\(\)](#) and [alloc_glob_mask\(\)](#).

There is a list of all transitions in [dve_system_t](#) (use functions [system_t::get_trans_count\(\)](#) and [system_t::get_transition\(\)](#)) and also each process has a list of his own transitions (functions [process_t::get_trans_count\(\)](#) and [process_t::get_transition\(\)](#))

12.35.2 Constructor & Destructor Documentation

12.35.2.1 dve_transition_t (system_t *const system) [inline]

A constructor (transition has to be created in a context of a [system](#)

- especially its [symbol table](#) given in a parameter ‘system’)

12.35.3 Member Function Documentation

12.35.3.1 void alloc_glob_mask (const size_int_t count_of_glob_vars) [inline]

Prepare bitmask of global variables - allocate as many bits as a count of global variables

12.35.3.2 size_int_t get_channel_gid () const [inline]

Returns a [GID](#) of channel used in a synchronisation of this transition.

Returns a [GID](#) of channel used in a synchronisation of this transition. Is has no reasonable meaning, when [get_sync_mode\(\)](#)==SYNC_NO_SYNC

Referenced by [dve_parser_t::check_restrictions_put_on_property\(\)](#), [dve_explicit_system_t::compute_enabled_stage2\(\)](#), [dve_explicit_system_t::get_async_enabled_trans_succ_without_property\(\)](#), and [por_t::init\(\)](#).

12.35.3.3 dve_expression_t* get_effect (size_int_t eff_nbr) const [inline]

Returns an `eff_nbr-ith` effect.

Parameters:

eff_nbr = number of effect (effects have numbers from 0 to [get_effect_count\(\)](#) - 1)

Referenced by [dve_explicit_system_t::apply_transition_effects\(\)](#), [dve_parser_t::check_restrictions_put_on_property\(\)](#), [por_t::init\(\)](#), and [write\(\)](#).

12.35.3.4 `const bit_string_t& get_glob_mask () const` `[inline]`

Returns a bit mask of assigned variables.

Returns an object of type `bit_string_t`, which represents the bit mask of assigned variables in effects of this transition

Referenced by `dve_explicit_system_t::not_in_glob_conflict()`.

12.35.3.5 `const dve_expression_t* get_guard () const` `[inline]`

Returns a pointer to the constant expression representing a guard of this transition

In case when there is no guard `get_guard()` returns 0.

12.35.3.6 `dve_expression_t* get_guard ()` `[inline]`

Returns a guard of this transition.

In case when there is no guard `get_guard()` returns 0.

Referenced by `por_t::init()`, and `dve_explicit_system_t::passed_through()`.

12.35.3.7 `bool get_guard_string (std::string & str) const`

Sets the string representation of guard or returns false.

Parameters:

str = variable to which to return the string repr. of guard

Returns:

true iff transition has a guard

References `dve_expression_t::to_string()`.

Referenced by `write()`.

12.35.3.8 `size_int_t get_partial_id () const` `[inline]`

Returns an index into vector of transitions of process with the same 'sync_mode'

Together with *sync_mode* forms an unique identifier of transition inside the process. To gain a single identifier use a function `get_id()`

12.35.3.9 `dve_symbol_table_t * get_symbol_table () const`

Returns a symbol table corresponding to this transition.

In this symbol table there are stored declarations of variables states, processes and channels, which are used in this transition.

References dve_system_t::get_symbol_table(), and transition_t::parent_system.

Referenced by get_state1_name(), get_state2_name(), get_sync_channel_name(), set_state1_gid(), and set_state2_gid().

12.35.3.10 const char * get_sync_channel_name () const

Returns the name of channel or 0 (see details).

If the transition is synchronised it returns the name of channel used for this synchronisation. Otherwise it returns 0.

References dve_symbol_table_t::get_channel(), dve_symbol_t::get_name(), get_symbol_table(), and SYNC_NO_SYNC.

Referenced by write().

12.35.3.11 const dve_expression_t* get_sync_expr_list_item (const size_int_t i) const [inline]

Returns the 'i'-th constant expression in a synchronization written after '!' or '?'

Returns the 'i'-th constant expression in a synchronization of this transition. In case when there is no synchronization * get_sync_expr() returns 0.

12.35.3.12 dve_expression_t* get_sync_expr_list_item (const size_int_t i) [inline]

Returns the 'i'-th expression in a synchronization written after '!' or '?'.

Returns the 'i'-th expression in a synchronization of this transition. In case when there is no synchronization * get_sync_expr() returns 0.

Referenced by dve_explicit_system_t::get_async_enabled_trans_succ_without_property(), and por_t::init().

12.35.3.13 bool get_sync_expr_string (std::string & str) const

Sets the string representation of expression in a synchronization or returns false

Parameters:

str = variable to which to return the string repr. of expression from synchronisation

Returns:

true iff transition has an expression in a synchronisation

References `array_t::size()`, and `to_string()`.

Referenced by `write()`.

12.35.3.14 `bool get_valid () const` [inline]

Returns, whether this transition is valid or not.

Returns, whether this transition is valid or not. When transition is not valid, it is not used, when we generate the states of system

12.35.3.15 `void set_partial_id (const size_int_t partial_id)` [inline]

Sets [Partial ID](#) of this transition.

Warning:

[Partial ID](#) should be set only by [process_t](#), when transition is added to the list of process's transitions

Referenced by `dve_process_t::add_transition()`.

12.35.3.16 `void set_process_gid (const size_int_t gid)` [inline]

Sets a [process GID](#) of process owning this transition.

Warning:

Use this method carefully. Do not try to cause inconsistencies

The documentation for this class was generated from the following files:

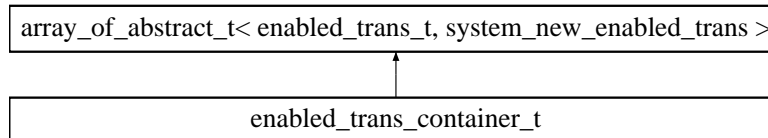
- [dve_transition.hh](#)
- [dve_transition.cc](#)

12.36 enabled_trans_container_t Class Reference

Container determined for storing enabled processes in one state.

```
#include <system_trans.hh>
```

Inheritance diagram for enabled_trans_container_t::



Public Member Functions

- void [clear](#) ()
Empties whole contents of container.
- [enabled_trans_container_t](#) (const [explicit_system_t](#) &system)
A constructor.
- size_int_t [get_begin](#) (const size_int_t process_gid) const
- size_int_t [get_count](#) (const size_int_t process_gid) const
Returns a count of transitions of process with GID 'process_gid'.
- const [enabled_trans_t](#) * [get_enabled_transition](#) (const size_int_t process_gid, const size_int_t index) const
The definition of [get_enabled_transition\(\)](#) for the case of constant.
- [enabled_trans_t](#) * [get_enabled_transition](#) (const size_int_t process_gid, const size_int_t index)
Returns a pointer to the enabled transition of process with GID.
- size_int_t [get_property_succ_count](#) () const
Returns a count of transitions enabled in a property process.
- void [set_next_begin](#) (const size_int_t process_gid, const size_int_t next_begin)
Sets, where the list of enabled transitions of next process begins.
- void [set_property_succ_count](#) (const size_int_t count)
Sets a count of transitions enabled in a property process.
- [~enabled_trans_container_t](#) ()
A destructor.

12.36.1 Detailed Description

Container determined for storing enabled processes in one state.

This container should be used in calls of [explicit_system_t::get_succs\(\)](#) and [explicit_system_t::get_enabled_trans\(\)](#) functions for storing the list of transitions enabled in a given state. Its constructor has an instance of [explicit_system_t](#) as a parameter, because it tries to guess the size of memory sufficient to store maximal count of transitions, which are enabled in one state of the system. This guess prevent the big count of reallocations at the beginning of the run.

If the system is with property process, then the `property_succ_count` must be set (by [set_property_succ_count\(\)](#) function) to correct usage of this container.

Note:

See [array_of_abstract_t](#) for details and more methods of this class

12.36.2 Member Function Documentation

12.36.2.1 `size_int_t get_begin (const size_int_t process_gid) const` [inline]

Returns the index of first enabled transition of process with GID '`process_gid`'.

Referenced by `por_t::ample_set_succs()`.

The documentation for this class was generated from the following files:

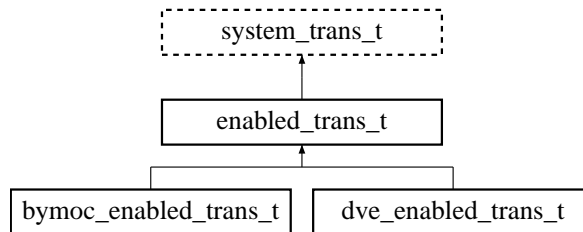
- [system_trans.hh](#)
- [system_trans.cc](#)

12.37 enabled_trans_t Class Reference

Class storing informations about one enabled transition.

```
#include <system_trans.hh>
```

Inheritance diagram for enabled_trans_t::



Public Member Functions

- [enabled_trans_t](#) ()
A constructor.
- bool [get_erroneous](#) () const
Returns, whether an enables transition is erroneous.
- virtual [enabled_trans_t](#) & [operator=](#) (const [enabled_trans_t](#) &second)=0
An assignment operator.
- void [set_erroneous](#) (const bool is_erroneous)
Sets, whether an enabled transition is erroneous.

12.37.1 Detailed Description

Class storing informations about one enabled transition.

Enabled transition = system transition + erroneous property

Enabled transitions can be produced by [explicit system](#) ([explicit_system_t](#)), if it can work with enabled transitions.

Each enabled transition represents a possible step of a [system](#) in a given state of the [system](#).

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

- [system_trans.hh](#)

12.38 `ERR_throw_t` Struct Reference

```
#include <error.hh>
```

Public Member Functions

- `ERR_throw_t` (const [ERR_type_t](#) type_arg, const [ERR_id_t](#) id_arg)

Public Attributes

- [ERR_id_t](#) id
- [ERR_type_t](#) type

12.38.1 Detailed Description

Integer type used to throw or catch exceptions with a parameter of [ERR_throw_t](#) type. This parameter is also called error type

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

- [error.hh](#)

12.39 ERR_triplet_t Struct Reference

```
#include <error.hh>
```

Public Member Functions

- **ERR_triplet_t** (const [ERR_char_string_t](#) mes, const [ERR_type_t](#) tt=ERR_UNKNOWN_TYPE, const [ERR_id_t](#) num=ERR_UNKNOWN_ID)

Public Attributes

- const [ERR_id_t](#) id
- const [ERR_char_string_t](#) message
- const [ERR_type_t](#) type

12.39.1 Detailed Description

Structure for storing (usually constant) error messages in a standardized form.

You can use this structure if you want to store a complete error message including its [error ID](#) and type.

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

- [error.hh](#)

12.40 `error_string_t` Class Reference

Class determined for storing error messages.

```
#include <error.hh>
```

Public Member Functions

- void **delete_content** ()
- **error_string_t** (**error_string_t** &s2)
- **error_string_t** (const **error_string_t** &s2)
- std::string & **operator*** ()
- std::string * **operator** → ()
- **error_string_t** & **operator**<< (const **error_string_t** second_str)
- **error_string_t** & **operator**<< (const ERR_std_string_t &second_str)
- **error_string_t** & **operator**<< (const ERR_char_string_t second_str)
- **error_string_t** & **operator**<< (const int i)
- **error_string_t** & **operator**<< (const unsigned int i)
- **error_string_t** & **operator**<< (const signed long int i)
- **error_string_t** & **operator**<< (const unsigned long int i)
- void **recreate** ()

Static Public Member Functions

- static divine::size_int_t **allocated_strings** ()

Friends

- const std::string & **operator*** (const **error_string_t** &errstr)
- std::ostream & **operator**<< (std::ostream &ostr, const **error_string_t** s)

12.40.1 Detailed Description

Class determined for storing error messages.

Class determined for storing error messages (1 error message in 1 instance of **error_string_t**).

This class should not be explicitly used in usual programs.

The documentation for this class was generated from the following files:

- [error.hh](#)
- [error.cc](#)

12.41 error_vector_t Class Reference

The main class in [error.hh](#) determined for storing.

```
#include <error.hh>
```

Public Member Functions

- void [clear](#) ()
Removes all error messages from a memory.
- [ERR_nbr_t](#) [count](#) ()
Returns a number of error messages in the vector.
- bool [empty](#) ()
Returns whether there exists any error message in a memory.
- [error_vector_t](#) ()
A constructor.
- void [flush](#) ()
Flushes all stored error messages.
- bool [get_silent](#) () const
Returns, whether it can write messages to STDERR (true) or not (false).
- [ERR_id_t](#) [id](#) (const [ERR_nbr_t](#) i)
- [ERR_id_t](#) [id_back](#) ()
Returns [error ID](#) of the error message from the end of the list of errors.
- [ERR_id_t](#) [id_front](#) ()
- [error_vector_t](#) & [operator<<](#) (const int i)
- [error_vector_t](#) & [operator<<](#) (const unsigned int i)
- [error_vector_t](#) & [operator<<](#) (const signed long int i)
- [error_vector_t](#) & [operator<<](#) (const unsigned long int i)
- [error_vector_t](#) & [operator<<](#) (const [error_string_t](#) second_str)
Appends second_str to the error message creating in the [error vector](#).
- [error_vector_t](#) & [operator<<](#) (const [ERR_std_string_t](#) &second_str)
Appends second_str to the error message creating in the [error vector](#).
- [error_vector_t](#) & [operator<<](#) (const [ERR_char_string_t](#) second_str)
Appends second_str to the error message creating in the [error vector](#).
- void [operator<<](#) (const [psh](#) &e)
Finishes creating of an error message and calls 'warning handling callback'.

- void `operator<<` (const `thr` &`e`)
Finishes creating of an error message and calls 'error handling callback'.
- void `perror` (`ERR_char_string_t` `mes`, const `ERR_nbr_t` `i`)
- void `perror` (const `ERR_nbr_t` `i`)
- void `perror_back` (`ERR_char_string_t` `mes`)
- void `perror_back` ()
- void `perror_front` (`ERR_char_string_t` `mes`)
- void `perror_front` ()
- void `pop` (const `ERR_nbr_t` `begin`, const `ERR_nbr_t` `end`)
Removes interval of errors.
- void `pop` (const `ERR_nbr_t` `index`)
Removes 'index'-th error.
- void `pop_back` ()
Removes 1 error message from the end of the list of errors.
- void `pop_back` (const `ERR_nbr_t` `n`)
Removes last n errors.
- void `pop_front` ()
Removes 1 error message from the beginning of the list of errors.
- void `pop_front` (const `ERR_nbr_t` `n`)
Removes first n errors.
- void `print` (`ERR_char_string_t` `mes`)
Prints mes to the standard error output.
- void `push` (`error_string_t` &`mes`, const `ERR_id_t` `id`=`ERR_UNKNOWN_ID`)
Inserts an error message to the end of a list of errors.
- void `set_push_callback` (const `ERR_psh_callback_t` `func`)
Sets a 'warning handling callback'.
- void `set_silent` (const bool `be_silent`)
Sets, whether it can write messages to STDERR (true) or not (false).
- void `set_throw_callback` (const `ERR_thr_callback_t` `func`)
Sets a 'error handling callback'.
- const `error_string_t` `string` (const `ERR_nbr_t` `i`)
- const `error_string_t` `string_back` ()
Returns the string of the error message from the end of the list of errors.

- `const error_string_t string_front ()`
- `void that (error_string_t &mes, const ERR_type_t err_type=ERR_UNKNOWN_TYPE, const ERR_id_t id=ERR_UNKNOWN_ID)`

Serves to store standardized error messages given by elements.

- `void that (const ERR_c_triplet_t &st)`
- `~error_vector_t ()`

A destructor.

12.41.1 Detailed Description

The main class in `error.hh` determined for storing.

This is the main class in `error.hh`. It is determined to store and handle all errors and warnings in your program. It is a replacement of standard error handling through throw & catch statements, but it defaultly uses throw & catch constructs to jump out of the problematic piece of a program code (that caused a possible error).

The main advantage of this class is a user friendly interface formed especially by '<<' operators, which permits to work with `error_vector_t` similarly as with `ostream` class.

The another advantage and the reason why this is a vector instead of single error buffer is that we distinguish between non-fatal errors (we call them 'warnings') and fatal errors (we call them 'errors').

The behaviour in a case of storing of error or warning can be selected or changed by functions `set_push_callback()` and `set_throw_callback()`.

For more informations about usage see [Error Handling Unit](#).

12.41.2 Constructor & Destructor Documentation

12.41.2.1 `error_vector_t ()` [inline]

A constructor.

Only initializes callback functions serving creation of error messages private attributes `throw_callback` and `push_callback`.

12.41.2.2 `~error_vector_t ()` [inline]

A destructor.

Only do some deallocation of dynamically allocated memory (in instances of `error_string_t`)

References `ERR_UNKNOWN_ID`, and `ERR_UNKNOWN_TYPE`.

12.41.3 Member Function Documentation

12.41.3.1 void flush ()

Flushes all stored error messages.

Prints all error messages stored in a memory to the standard error output and calls [clear\(\)](#) to empty the memory.

References [clear\(\)](#), [count\(\)](#), and [perror\(\)](#).

Referenced by [ERR_default_thr_callback\(\)](#).

12.41.3.2 ERR_id_t id (const ERR_nbr_t i)

Returns [error ID](#) of *i-th* error message from the beginning of the list of errors

12.41.3.3 ERR_id_t id_front ()

Returns [error ID](#) of the error message from the beginning of the list of errors

12.41.3.4 error_vector_t& operator<< (const int i) [inline]

Appends string representing *i* to the error message creating in the [error vector](#)

12.41.3.5 error_vector_t& operator<< (const unsigned int i) [inline]

Appends string representing *i* to the error message creating in the [error vector](#)

12.41.3.6 error_vector_t& operator<< (const signed long int i) [inline]

Appends string representing *i* to the error message creating in the [error vector](#)

12.41.3.7 error_vector_t& operator<< (const unsigned long int i) [inline]

Appends string representing *i* to the error message creating in the [error vector](#)

12.41.3.8 void operator<< (const psh & e)

Finishes creating of an error message and calls 'warning handling callback'.

Finishes creating of an error message and inserts prepared error message to the end of the list of errors. Then calls 'warning handling callback' (see [ERR_psh_callback_t](#))

References [psh::c](#), [push\(\)](#), and [psh::t](#).

12.41.3.9 void operator<< (const thr & e)

Finishes creating of an error message and calls 'error handling callback'.

Finishes creating of an error message and inserts prepared error message to the end of the list of errors. Then calls 'error handling callback' (see [ERR_thr_callback_t](#))

References thr::c, thr::t, and that().

12.41.3.10 void perror (ERR_char_string_t mes, const ERR_nbr_t i)

Prints *mes* and *i-th* message (from the beginning of the list of errors) to the standard error output

12.41.3.11 void perror (const ERR_nbr_t i)

Prints *i-th* message (from the beginning of the list of errors) to the standard error output

Referenced by flush(), por_t::generate_ample_sets(), and por_t::generate_composed_ample_sets().

12.41.3.12 void perror_back (ERR_char_string_t mes)

Prints *mes* and the first message from the end of the list of errors to the standard error output

12.41.3.13 void perror_back ()

Prints the first message from the end of the list of errors to the standard error output

Referenced by ERR_default_psh_callback().

12.41.3.14 void perror_front (ERR_char_string_t mes)

Prints *mes* and the first message from the beginning of the list of errors to the standard error output

12.41.3.15 void perror_front ()

Prints the first message from the beginning of the list of errors to the standard error output

12.41.3.16 void pop (const ERR_nbr_t begin, const ERR_nbr_t end)

Removes interval of errors.

Removes all error messages with ordering number from *begin* (inclusive) to *end* (exclusive) in a list of errors.

12.41.3.17 void pop (const ERR_nbr_t *index*)

Removes 'index'-th error.

Removes *index-th* error message from the beginning of the list of errors

12.41.3.18 void pop_back (const ERR_nbr_t *n*)

Removes last *n* errors.

Removes *n* error messages from the end of the list of errors. If *n* = 0, then removes all errors from a memory.

References pop_back().

Referenced by ERR_default_psh_callback().

12.41.3.19 void pop_front (const ERR_nbr_t *n*)

Removes first *n* errors.

Removes *n* error messages from the beginning of the list of errors. If *n* = 0, then removes all errors from a memory.

References pop_front().

12.41.3.20 void push (error_string_t & *mes*, const ERR_id_t *id* = ERR_UNKNOWN_ID)

Inserts an error message to the end of a list of errors.

Inserts an error message to the end of a list of errors. It does not call any of callback functions (= any of warning/error handling callbacks).

Note: parameter *id* is optional.

References error_string_t::recreate().

Referenced by operator<<(), and that().

12.41.3.21 const error_string_t string (const ERR_nbr_t *i*)

Returns the string of *i-th* error message from the beginning of the list of errors

12.41.3.22 const error_string_t string_front ()

Returns the string of the error message from the beginning of the list of errors

12.41.3.23 void that (error_string_t & mes, const ERR_type_t
err_type = ERR_UNKNOWN_TYPE, const ERR_id_t id =
ERR_UNKNOWN_ID)

Serves to store standardized error messages given by elements.

```
terr.that(message, id, type)
```

is functionally equivalent to

```
terr << message << thr(type, id)
```

Note: Parameters *err_type* and *id* are not obligatory.

References push().

12.41.3.24 void that (const ERR_c_triplet_t & st)

Serves to store standardized error messages (in triplets [ERR_triplet_t](#)).

```
terr.that(triplet)
```

is functionally equivalent to

```
terr << triplet.message << thr(triplet.type, triplet.id)
```

References error_string_t::delete_content(), ERR_triplet_t::id, ERR_triplet_t::message, push(), and ERR_triplet_t::type.

Referenced by operator<<().

The documentation for this class was generated from the following files:

- [error.hh](#)
- error.cc

12.42 ES_parameters_t Struct Reference

Structure determined for passing parameters to ES*_eval() functions.

```
#include <dve_explicit_system.hh>
```

Public Attributes

- `size_int_t * array_sizes`
- `size_int_t * initial_states`
- `SYS_initial_values_t * initial_values`
- `state_t state`
- `size_int_t * state_lids`
- `size_int_t * state_positions_proc`
- `size_int_t * state_positions_state`
- `size_int_t * state_positions_var`
- `dve_var_type_t * var_types`

12.42.1 Detailed Description

Structure determined for passing parameters to ES*_eval() functions.

[dve_explicit_system_t](#) defines its own structure for passing parameters to its own functions ES*_eval(), which are replacing SYS*_eval() functions used by [system_t](#).

This is really internal thing and the programmer of algorithms using DiVinE usually should not use this structure.

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

- [dve_explicit_system.hh](#)

12.43 explicit_storage_t Class Reference

explicit storage class

```
#include <explicit_storage.hh>
```

Public Member Functions

- void * [app_by_ref](#) (state_ref_t refer)
- void [delete_all_states](#) (bool leave_collision_lists=false)
- bool [delete_by_ref](#) (state_ref_t)
- [explicit_storage_t](#) (divine::error_vector_t &=gerr)
- template<class appendix_t>
bool [get_app_by_ref](#) (state_ref_t refer, appendix_t &result)
- size_t [get_coltables](#) ()
- size_t [get_ht_occupancy](#) ()
- size_t [get_max_coltable](#) ()
- size_t [get_mem_max_used](#) ()
- size_t [get_mem_used](#) ()
- size_t [get_states_max_stored](#) ()
- size_t [get_states_stored](#) ()
- void [init](#) ()
- void [insert](#) (state_t, state_ref_t &)
- void [insert](#) (state_t)
- bool [is_stored](#) (state_t, state_ref_t &)
- bool [is_stored](#) (state_t)
- bool [is_stored_if_not_insert](#) (state_t, state_ref_t &)
- [state_t](#) [reconstruct](#) (state_ref_t)
- template<class appendix_t>
bool [set_app_by_ref](#) (state_ref_t refer, appendix_t appen)
- template<class appendix_t>
void [set_appendix](#) (appendix_t)
- void [set_appendix_size](#) (size_t)
- void [set_col_init_size](#) (size_t)
- void [set_col_resize](#) (size_t)
- void [set_compression_method](#) (size_t)
- void [set_hash_function](#) (hash_functions_t)
- void [set_ht_size](#) (size_t)
- bool [set_mem_limit](#) (size_t)

Protected Member Functions

- void [mem_counting](#) (int)

Protected Attributes

- `size_t` **appendix_size**
- `size_t` **col_init_size**
- `size_t` **col_resize**
- `size_t` **compression_method**
- `compressor_t` **compressor**
- `error_vector_t` & **errvec**
- `hash_function_t` **hasher**
- `size_t` **ht_size**
- `bool` **initialized**
- `size_t` **mem_limit**
- `size_t` **mem_max_used**
- `size_t` **mem_used**
- `storage_t` **storage**

12.43.1 Detailed Description

explicit storage class

12.43.2 Constructor & Destructor Documentation

12.43.2.1 `explicit_storage_t` (`divine::error_vector_t` & = `gerr`)

Constructor. An `error_vector_t` may be specified.

12.43.3 Member Function Documentation

12.43.3.1 `void * app_by_ref` (`state_ref_t` *refer*)

Returns pointer to appendix stored at the referenced state.

References `state_ref_t::hres`, and `state_ref_t::id`.

12.43.3.2 `void delete_all_states` (`bool` *leave_collision_lists* = `false`)

Deletes all stored states and collision lists. Method has a voluntary parameter `leave_collision_lists`. If called with `leave_collision_lists` = `true`, then collision lists are only cleared and remain allocated.

12.43.3.3 `bool delete_by_ref` (`state_ref_t` *state_reference*)

Deletes the state that is referenced by given `state_ref_t`. Returns `false` if the delete operation was unsuccessful.

References `state_ref_t::hres`, and `state_ref_t::id`.

12.43.3.4 bool get_app_by_ref (state_ref_t *refer*, appendix_t & *result*)
[inline]

Returns appendix stored at the referenced state.

References state_ref_t::hres, and state_ref_t::id.

12.43.3.5 size_t get_coltables ()

Returns sum of sizes of collision tables (sum of lengths of all collision lists).

12.43.3.6 size_t get_ht_occupancy ()

Returns the number of occupied lines (lines with at least one state stored) in the hashtable.

12.43.3.7 size_t get_max_coltable ()

Returns maximum size of a single collision table (maximum length of a collision list).

12.43.3.8 size_t get_mem_max_used ()

Returns maximal (up to the time if this call) number of bytes allocated by storage and compressor instances. Note this differ from real memory consumption as no fragmentation is included.

12.43.3.9 size_t get_mem_used ()

Returns current number of bytes allocated by storage and compressor instances. Note this differ from real memory consumption as no fragmentation is included.

12.43.3.10 size_t get_states_max_stored ()

Returns maximum number of states stored in the set.

12.43.3.11 size_t get_states_stored ()

Returns current number of states stored in the set.

Referenced by logger_t::log_now().

12.43.3.12 void init ()

This member function performs initialization of the hash table. It must be called in order to use the class for storing states. This function call may be predecessed by calls of various set functions.

References `compressor_t::clear()`, and `compressor_t::init()`.

12.43.3.13 void insert (state_t state, state_ref_t & state_reference)

Insert a copy of the given state in the set of visited states. An exception is generated in the case of unsuccessfull insert. Also sets the given [state_ref_t](#) to reference the newly inserted state.

References `compressor_t::compress()`, `hash_function_t::get_hash()`, `state_ref_t::hres`, `state_ref_t::id`, `state_t::ptr`, and `state_t::size`.

12.43.3.14 void insert (state_t state)

Insert a copy of the given state in the set of visited states. An exception is generated in the case of unseccesfull insert.

12.43.3.15 bool is_stored (state_t state, state_ref_t & state_reference)

Returns true if the state is inserted in the set of visited states and if so then sets the [state_ref_t](#) to reference the state.

References `compressor_t::compress()`, `hash_function_t::get_hash()`, `state_ref_t::hres`, `state_ref_t::id`, `state_t::ptr`, and `state_t::size`.

12.43.3.16 bool is_stored (state_t state)

Returns true if the state is inserted in the set of visited states.

12.43.3.17 bool is_stored_if_not_insert (state_t state, state_ref_t & state_reference)

Tests whether given state is stored in the set of visited states. In the case the state is present in the set, the function returns true and sets the given [state_ref_t](#) to reference the state in the set, otherwise a copy of the state is inserted in the set, the given [state_ref_t](#) is set to the state that has been just inserted, and false is returned. In the latter case an exception is generated in the case of unseccesfull insert.

References `compressor_t::compress()`, `hash_function_t::get_hash()`, `state_ref_t::hres`, `state_ref_t::id`, `state_t::ptr`, and `state_t::size`.

12.43.3.18 state_t reconstruct (state_ref_t state_reference)

Reconstructs state referenced by given [state_ref_t](#). Invalid reference will cause warning and crash the application.

References compressor_t::decompress(), state_ref_t::hres, state_ref_t::id, state_t::ptr, and state_t::size.

12.43.3.19 bool set_app_by_ref (state_ref_t refer, appendix_t appen)
[inline]

Rewrites the appendix stored at referenced state with the given one.

References state_ref_t::hres, and state_ref_t::id.

12.43.3.20 void set_appendix (appendix_t) [inline]

Sets the size of appendix using an instance of an appendix strucute. Must be called before member function init.

12.43.3.21 void set_appendix_size (size_t appendix_size_in)

Sets the size of instance of an appendix structure. The call of this function must preceed the call of init member function.

12.43.3.22 void set_col_init_size (size_t coltable_init_size)

Sets initial size of a collision table (default=1). The call of this function must preceed the call of init member function.

12.43.3.23 void set_col_resize (size_t coltable_resize_by)

Sets the factor by which a collision table is enlarged if necessary. The call of this function must preceed the call of init member function.

12.43.3.24 void set_compression_method (size_t compression_method_id)

Sets compression method. Where the possibilities are: NO_COMPRESS (default) and HUFFMAN_COMPRESS. The call of this function must preceed the call of init member function.

12.43.3.25 void set_hash_function (hash_functions_t hf)

Sets hash function to be used.

References hash_function_t::set_hash_function().

12.43.3.26 void set_ht_size (size_t *hashtable_size*)

Sets the hash table size (default: $2^{16} = 65536$). The call of this function must precede the call of init member function. The call of this function must precede the call of init member function.

12.43.3.27 bool set_mem_limit (size_t *memory_limit*)

Sets a limit for the number of allocated bytes (not including memory fragmentation). The call of this function must precede the call of init member function.

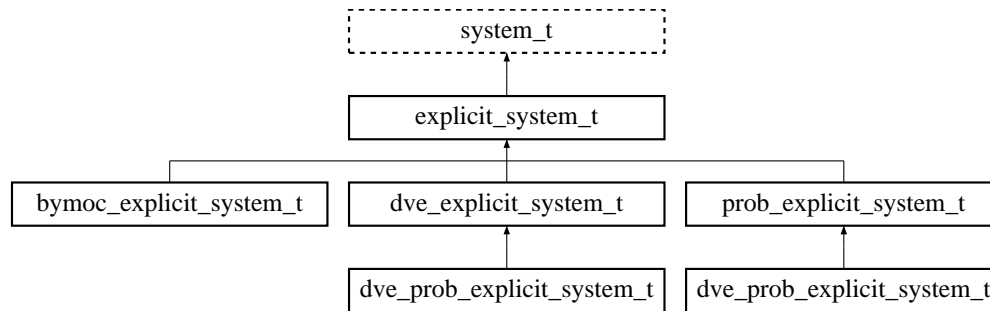
The documentation for this class was generated from the following files:

- [explicit_storage.hh](#)
- explicit_storage.cc

12.44 explicit_system_t Class Reference

```
#include <explicit_system.hh>
```

Inheritance diagram for explicit_system_t::



Public Member Functions

- bool [can_evaluate_expressions](#) ()
- bool [can_system_transitions](#) ()
- [explicit_system_t](#) ([error_vector_t](#) &evect)

A constructor.

- virtual [~explicit_system_t](#) ()

A destructor.

Methods for expression evaluation

These methods are implemented only if [can_evaluate_expressions\(\)](#) returns true

- virtual bool [eval_expr](#) (const [expression_t](#) *const expr, const [state_t](#) state, [data_t](#) &data) const =0

Evaluates an expression.

Methods working with system transitions and enabled transitions

These methods are implemented only if [can_system_transitions\(\)](#) returns true

- virtual int [get_enabled_ith_trans](#) (const [state_t](#) state, const size_int_t i, [enabled_trans_t](#) &enb_trans)=0

Computes the i-th enabled transition in a state 'state'.

- virtual int [get_enabled_trans](#) (const [state_t](#) state, [enabled_trans_container_t](#) &enb_trans)=0

Creates a list of enabled transitions in a state 'state'.

- virtual int `get_enabled_trans_count` (const `state_t` state, size_int_t &count)=0
Computes the count of transitions in a state 'state'.
- virtual bool `get_enabled_trans_succ` (const `state_t` state, const `enabled_trans_t` &enabled, `state_t` &new_state)=0
Generates a successor of 'state' using enabled transition 'enabled'.
- virtual bool `get_enabled_trans_succs` (const `state_t` state, `succ_container_t` &succs, const `enabled_trans_container_t` &enabled_trans)=0
- virtual int `get_succs` (`state_t` state, `succ_container_t` &succs, `enabled_trans_container_t` &etc)=0
Creates successors of state 'state'.
- virtual `enabled_trans_t` * `new_enabled_trans` () const =0
Creates an instance of enabled transition.

Obligatory part of abstract interface

A destructor.

These methods have to be implemented in each implementation of `explicit_system_t`

- virtual `state_t` `get_initial_state` ()=0
Returns an initial state of the system.
- virtual int `get_ith_succ` (`state_t` state, const int i, `state_t` &succ)=0
Computes i-th successor of 'state'.
- virtual size_int_t `get_preallocation_count` () const =0
Returns a count of successors to preallocate in successor container.
- virtual int `get_succs` (`state_t` state, `succ_container_t` &succs)=0
Creates successors of state 'state'.
- virtual bool `is_accepting` (`state_t` state, size_int_t acc_group=0, size_int_t pair_member=1)=0
- virtual bool `is_erroneous` (`state_t` state)=0
Returns, whether the state of the system is erroneous.
- virtual void `print_state` (`state_t` state, std::ostream &outs=std::cout)=0
Prints the standard text representation of a state of the system.
- virtual size_int_t `violated_assertion_count` (const `state_t` state) const =0
Returns a count of assertions violated in the state 'state'.
- virtual std::string `violated_assertion_string` (const `state_t` state, const size_int_t index) const =0
- virtual bool `violates_assertion` (const `state_t` state) const =0
Returns, whether any assertion is violated in the state 'state'.

12.44.1 Detailed Description

Abstract interface of a class representing a state generator based on the model of [system](#) stored in [system_t](#)

The role of [explicit_system](#) implemented by [explicit_system_t](#) is to provide an interface to the generator of states of a model of the system stored in the parent class [system_t](#).

The most important methods are [get_initial_state\(\)](#) and [get_succs\(\)](#). Values returned from [get_succs\(\)](#) can be analyzed by [succs_normal\(\)](#), [succs_error\(\)](#) and [succs_deadlock\(\)](#).

12.44.2 Constructor & Destructor Documentation

12.44.2.1 explicit_system_t (error_vector_t & *evect*) [inline]

A constructor.

Parameters:

evect = [error vector](#) used for reporting of error messages

12.44.3 Member Function Documentation

12.44.3.1 bool can_evaluate_expressions () [inline]

Tells whether the implementation of [explicit_system_t](#) interface can evaluate expressions

12.44.3.2 bool can_system_transitions () [inline]

Tells whether the implementation of [explicit_system_t](#) interface can work with [system_trans_t](#) and [enabled_trans_t](#)

12.44.3.3 virtual bool eval_expr (const expression_t *const *expr*, const state_t *state*, data_t & *data*) const [pure virtual]

Evaluates an expression.

Parameters:

expr = expression to evaluate

state = state of the system, in which we want to evaluate the expression *expr*

data = computed value of the expression

Returns:

false iff no error occurred during the evaluation of the expression

This method is implemented only if [can_evaluate_expressions\(\)](#) returns true.

Implemented in [bymoc_explicit_system_t](#), and [dve_explicit_system_t](#).

12.44.3.4 **virtual int get_enabled_ith_trans (const state_t state, const size_int_t i, enabled_trans_t & enb_trans)** [pure virtual]

Computes the i-th enabled transition in a state 'state'.

Computes the i-th enabled transition in a state *state*.

Parameters:

state = state of the system

i = the index of the enabled transition (0..get_async_enabled_trans_count)

enb_trans = the variable for storing computed transition

Returns:

bitwise OR of SUCC_NORMAL, SUCC_ERROR and SUCC_DEADLOCK (use functions [succs_normal\(\)](#), [succs_error\(\)](#) and [succs_deadlock\(\)](#) for testing)

This method is implemented only if [can_system_transitions\(\)](#) returns true.

Implemented in [bymoc_explicit_system_t](#), and [dve_explicit_system_t](#).

12.44.3.5 **virtual int get_enabled_trans (const state_t state, enabled_trans_container_t & enb_trans)** [pure virtual]

Creates a list of enabled transitions in a state 'state'.

Creates a list of enabled transitions in a state *state* and stores it to the container *enb_trans* (see [enabled_trans_container_t](#)).

Parameters:

state = state of the system

enb_trans = container for storing enabled transitions (see [Enabled transitions](#) for details)

Returns:

bitwise OR of SUCC_NORMAL, SUCC_ERROR and SUCC_DEADLOCK (use functions [succs_normal\(\)](#), [succs_error\(\)](#) and [succs_deadlock\(\)](#) for testing)

This method is implemented only if [can_system_transitions\(\)](#) returns true.

Implemented in [bymoc_explicit_system_t](#), and [dve_explicit_system_t](#).

12.44.3.6 virtual int get_enabled_trans_count (const state_t *state*, size_int_t & *count*) [pure virtual]

Computes the count of transitions in a state 'state'.

Computes the count of transitions in a state *state*.

Parameters:

state = state of the system

count = computed count of enabled transitions

Returns:

bitwise OR of SUCC_NORMAL, SUCC_ERROR and SUCC_DEADLOCK (use functions [succs_normal\(\)](#), [succs_error\(\)](#) and [succs_deadlock\(\)](#) for testing)

Note:

In fact the count of transitions in state 'state' is equal to the number of successors of state 'state'.

This method is implemented only if [can_system_transitions\(\)](#) returns true.

Implemented in [bymoc_explicit_system_t](#), and [dve_explicit_system_t](#).

12.44.3.7 virtual bool get_enabled_trans_succ (const state_t *state*, const enabled_trans_t & *enabled*, state_t & *new_state*) [pure virtual]

Generates a successor of 'state' using enabled transition 'enabled'.

Parameters:

state = state of system

enabled = enabled transition to use for generation of successor

new_state = state to rewrite by the successor of *state*

This method is implemented only if [can_system_transitions\(\)](#) returns true.

Implemented in [bymoc_explicit_system_t](#), and [dve_explicit_system_t](#).

12.44.3.8 virtual bool get_enabled_trans_succs (const state_t *state*, succ_container_t & *succs*, const enabled_trans_container_t & *enabled_trans*) [pure virtual]

Generates successors of 'state' using list of enabled transitions 'enabled_trans'

Parameters:

state = state of system

succs = container for storage of successors of *state*

enabled_trans = list of enabled transitions to use for generation of successors

This method is implemented only if [can_system_transitions\(\)](#) returns true.

Implemented in [bymoc_explicit_system_t](#), and [dve_explicit_system_t](#).

12.44.3.9 virtual state_t get_initial_state () [pure virtual]

Returns an initial state of the system.

This method takes a model of the [system](#) stored in [system_t](#) (parent of this class) and computes the initial state of the system.

Implemented in [bymoc_explicit_system_t](#), and [dve_explicit_system_t](#).

12.44.3.10 virtual size_int_t get_preallocation_count () const [pure virtual]

Returns a count of successors to preallocate in successor container.

[succ_container_t](#) uses this method to estimate an amount of memory to preallocate. Repeated reallocation is slow - therefore good estimation is useful, but not necessary.

Implemented in [bymoc_explicit_system_t](#), and [dve_explicit_system_t](#).

12.44.3.11 virtual int get_succs (state_t state, succ_container_t & succs, enabled_trans_container_t & etc) [pure virtual]

Creates successors of state 'state'.

Creates successors of state *state* and saves them to successor container (see [succ_container_t](#)). Furthermore it stores enabled transitions of the systems which are used to generate successors of *state*

Parameters:

state = state of the system

succs = successors container for storage of successors of *state*

etc = container of enabled transitions (see [enabled_trans_container_t](#) and [enabled_trans_t](#))

Returns:

bitwise OR of SUCC_NORMAL, SUCC_ERROR and SUCC_DEADLOCK (use functions [succs_normal\(\)](#), [succs_error\(\)](#) and [succs_deadlock\(\)](#) for testing)

This method is implemented only if [can_system_transitions\(\)](#) returns true.

Implemented in [bymoc_explicit_system_t](#), and [dve_explicit_system_t](#).

12.44.3.12 virtual int get_succs (state_t state, succ_container_t & succs) [pure virtual]

Creates successors of state 'state'.

Creates successors of state *state* and saves them to successor container (see [succ_container_t](#)).

Parameters:

state = state of the system

succs = successors container for storage of successors of *state*

Returns:

bitwise OR of SUCC_NORMAL, SUCC_ERROR and SUCC_DEADLOCK (use functions [succs_normal\(\)](#), [succs_error\(\)](#) and [succs_deadlock\(\)](#) for testing)

Implemented in [bymoc_explicit_system_t](#), and [dve_explicit_system_t](#).

12.44.3.13 virtual bool is_accepting (state_t state, size_int_t acc_group = 0, size_int_t pair_member = 1) [pure virtual]

Returns, whether the state is accepting in the specified accepting group of Buchi or generalized Buchi automata, or whether the state belongs to the first or second component of the specified accepting pair of Rabin or Streett automata.

Parameters:

state = state of the system

Returns:

whether *state* (i.e. its property automaton projection) belongs to specified accepting group of Buchi or generalized Buchi automaton, or whether it belongs to the first ('pair_member=1') or second ('pair_member=2') component of the specified accepting pair of Rabin and Streett automata.

Note:

If the system is specified without property process, `false` is returned.

Implemented in [bymoc_explicit_system_t](#), and [dve_explicit_system_t](#).

12.44.3.14 virtual bool is_erroneous (state_t state) [pure virtual]

Returns, whether the state of the system is erroneous.

An erroneous state is a special state that is unique in the whole system. It is reached by bad model specification (model that permits variable/index overflow synchronization collision etc.)

Parameters:

state = state of the system

Returns:

true iff any of processes is in the state 'error'.

Implemented in [bymoc_explicit_system_t](#), and [dve_explicit_system_t](#).

12.44.3.15 **virtual enabled_trans_t* new_enabled_trans () const** [pure virtual]

Creates an instance of enabled transition.

This method is needed by [enabled_trans_container_t](#), which cannot allocate enabled transitions itself, because it does not know their concrete type. The abstract type [enabled_trans_t](#) is not sufficient for creation because of its purely abstract methods.

{ Example - implementation of this method in DVE system: }

```
enabled_trans_t * dve_explicit_system_t::new_enabled_trans() const
{
    return (new dve_enabled_trans_t);
}
```

This method is implemented only if [can_system_transitions\(\)](#) returns true.

Implemented in [bymoc_explicit_system_t](#), and [dve_explicit_system_t](#).

12.44.3.16 **virtual size_int_t violated_assertion_count (const state_t state) const** [pure virtual]

Returns a count of assertions violated in the state 'state'.

If an implementation of [explicit_system_t](#) do not support assertions, this method can be implemented so that it returns always 0.

Implemented in [bymoc_explicit_system_t](#), and [dve_explicit_system_t](#).

12.44.3.17 **virtual std::string violated_assertion_string (const state_t state, const size_int_t index) const** [pure virtual]

Returns a string representation of index-th assertion violated in the state 'state'

If an implementation of [explicit_system_t](#) do not support assertions, this method can be implemented so that it returns always empty string.

Implemented in [bymoc_explicit_system_t](#), and [dve_explicit_system_t](#).

12.44.3.18 `virtual bool violates_assertion (const state_t state) const` `[pure virtual]`

Returns, whether any assertion is violated in the state ‘state’.

If an implementation of [explicit_system_t](#) do not support assertions, this method can be implemented so that it returns always `false`.

Implemented in [bymoc_explicit_system_t](#), and [dve_explicit_system_t](#).

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

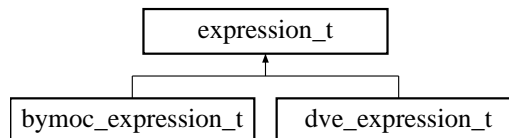
- [explicit_system.hh](#)

12.45 expression_t Class Reference

Abstract interface of a class representing an expression.

```
#include <expression.hh>
```

Inheritance diagram for expression_t::



Public Member Functions

- virtual void [assign](#) (const [expression_t](#) &expr)
Copies the contents of 'expr' into this expression (time $O(n)$).
- [expression_t](#) (const [expression_t](#) &second)
A copy constructor.
- [expression_t](#) ([system_t](#) *const system)
A constructor.
- [expression_t](#) ()
A constructor.
- virtual int [from_string](#) (std::string &expr_str, const size_int_t process_gid=NO_ID)=0
Reads in the expression from a string representation.
- [system_t](#) * [get_parent_system](#) () const
Returns a "parent" [system](#) of this expression.
- [expression_t](#) & [operator=](#) (const [expression_t](#) &second)
The assignment operator for expressions.
- virtual int [read](#) (std::istream &istr, size_int_t process_gid=NO_ID)=0
Reads in the expression from a string representation in stream.
- virtual void [set_parent_system](#) ([system_t](#) &system)
Sets a "parent" [system](#) of this expression.
- virtual void [swap](#) ([expression_t](#) &expr)
Swaps the content of 'expr' and this expression (time $O(1)$).

- virtual `std::string to_string () const =0`
Returns string representation of the expression.
- virtual `void write (std::ostream &ostr) const =0`
Writes a string representation of the expression to stream.
- virtual `~expression_t ()`
A destructor.

Protected Attributes

- `system_t * parent_system`

12.45.1 Detailed Description

Abstract interface of a class representing an expression.

Class `expression_t` represents expressions interpretable in a `system` given in a constructor `expression_t(system_t * const system)` or in a method `set_parent_system(system_t & system)`

Note:

Developer is responsible for correct setting of "parent" `system` (but expressions created during reading of the system from a file have the "parent" `system` set correctly).

12.45.2 Constructor & Destructor Documentation

12.45.2.1 `expression_t (system_t *const system) [inline]`

A constructor.

Parameters:

`system` = "parent" `system` of this expression

12.45.3 Member Function Documentation

12.45.3.1 `virtual int from_string (std::string & expr_str, const size_int_t process_gid = NO_ID) [pure virtual]`

Reads in the expression from a string representation.

Parameters:

`expr_str` = string containing source of an expression

process_gid = context of a process (default value NO_ID = global context)

Returns:

... 0 iff no error occurs, non-zero value in a case of error during a reading.

Implemented in [bymoc_expression_t](#), and [dve_expression_t](#).

12.45.3.2 `expression_t& operator= (const expression_t & second)` [inline]

The assignment operator for expressions.

It copies the content of *second* (the right argument of an operator) to the left argument of an operator.

It is implemented simply by [assign\(\)](#) method and you can also write `left_argument.assign(right_argument)` instead of `left_argument = right_argument`.

Returns:

The reference to this object (left argument of an operator after an assignment)

12.45.3.3 `virtual int read (std::istream & istr, size_int_t process_gid = NO_ID)` [pure virtual]

Reads in the expression from a string representation in stream.

Parameters:

istr = input stream containing source of expression

process_gid = context of process (default value NO_ID = global context)

Returns:

... 0 iff no error occurs, non-zero value in a case of error during a reading.

Implemented in [bymoc_expression_t](#), and [dve_expression_t](#).

12.45.3.4 `virtual std::string to_string () const` [pure virtual]

Returns string representation of the expression.

Implementation issue: This function is slower than [write\(\)](#), because it does a copying of potentially long string

Implemented in [bymoc_expression_t](#), and [dve_expression_t](#).

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

- [expression.hh](#)

12.46 hash_function_t Class Reference

Class that unifies hash functions in the library.

```
#include <hash_function.hh>
```

Public Member Functions

- `size_int_t` [get_hash](#) (unsigned char *, `size_int_t`, `size_int_t`=1)
Returns hash key for a given piece of memory. An optional parameter specify a seed or initvalue for the hash function.
- [hash_function_t](#) ()
Constructor.
- [hash_function_t](#) ([hash_functions_t](#))
Constructor.
- `size_int_t` [hash_state](#) ([state_t](#), `size_int_t`=1)
Returns hash key for a given state. An optional parameter specify a seed or initvalue for the hash function.
- void [set_hash_function](#) ([hash_functions_t](#))
Sets hash function to be used for hashing.
- [~hash_function_t](#) ()
Destructor.

Protected Attributes

- [hash_functions_t](#) `hf_id`

12.46.1 Detailed Description

Class that unifies hash functions in the library.

The documentation for this class was generated from the following files:

- [hash_function.hh](#)
- [hash_function.cc](#)

12.47 logger_t Class Reference

```
#include <logger.hh>
```

Public Member Functions

- void [init](#) (string tmpstr_in, int format_in=0)
- void [init](#) ([distributed_t](#) *divine_ptr_in, std::string basename, int format=0)
- void [log_now](#) ()
- [logger_t](#) ()
- void [register_double](#) (const log_double_t &func_ptr, std::string description="")
- void [register_double](#) (doublefunc_t func_ptr, std::string description="")
- void [register_int](#) (intfunc_t func_ptr, std::string description="")
- void [register_slong_int](#) (const log_slong_int_t &func_ptr, std::string description="")
- void [register_ulong_int](#) (const log_ulong_int_t &func_ptr, std::string description="")
- void [register_unsigned](#) (unsignedfunc_t func_ptr, std::string description="")
- void [set_storage](#) ([explicit_storage_t](#) *storage_ptr)
- void [stop_SIGALRM](#) ()
- void [use_SIGALRM](#) (unsigned int period=1)
- [~logger_t](#) ()

Public Attributes

- unsigned int [signal_period](#)

12.47.1 Detailed Description

Class [logger_t](#) is used to periodically log various values into logfiles. There are as many logfiles produced as there are workstations participating the computation. The logfiles may be further processed by plotlog script that uses gnuplot to create corresponding (e)ps graphs. The class can be used only in combination with class [distributed_t](#).

NOTE: Usage of the [logger_t](#) class slowsdowns the execution of the algorithm.

12.47.2 Constructor & Destructor Documentation

12.47.2.1 [logger_t](#) ()

Constructor does nothing. To initialize the instance of the class use the [init](#) member function.

12.47.2.2 [~logger_t](#) ()

Destructor prints footer to the logfile and closes the logfile.

12.47.3 Member Function Documentation

12.47.3.1 void init (string *tmpstr_in*, int *format_in* = 0)

The same as `init(distributed_t * divine_ptr_in, std::string basename, int format)`, but `divine_ptr_in = 0` implicitly.

References `init()`.

12.47.3.2 void init (distributed_t * *divine_ptr_in*, std::string *basename*, int *format* = 0)

This member function is called to initialize an instance of `logger_t` class. It accepts two obligatory parameters: a pointer to instance of `distributed_t` class and a base name of logfiles. (The base name is extended with .00, .01, .02, etc, by `logger_t` to distinguish individual logfiles.) The third parameter that gives the format of logfiles is optional (default value is 0, which corresponds to the format compatible with plotlog script).

The method opens the logfile and print header into it according the chosen format. The function should be called after the `set_storage` and all the register member functions were called.

divine_ptr_in can be zero value (network-related values will be then zero too).

Referenced by `init()`.

12.47.3.3 void log_now ()

Forces the logger to log current values now. If signal/alarm mechanism is involved, then this function is typically called just before the destructor to log final values. If no signal/alarm mechanism is used, then this function is supposed to be called periodically.

References `distributed_t::cluster_size`, `network_t::get_all_barriers_cnt()`, `network_t::get_all_buffers_flushes_cnt()`, `network_t::get_all_received_msgs_cnt()`, `network_t::get_all_sent_msgs_cnt()`, `network_t::get_sent_msgs_cnt_sent_to()`, `explicit_storage_t::get_states_stored()`, `distributed_t::network`, `vminfo_t::scan()`, and `vminfo_t::vmsize`.

12.47.3.4 void register_double (const log_double_t & *functor*, std::string *description* = "")

This function can be used to register functor that will be called by the `log_now` function to obtained double f. p. value to be logged into logfile. The second optional parameter is a short (below 8 characters) description of the value that could be (depending on format) printed in the header of the file.

This can be used to log the number of states stored in the queue of states waiting for exploration, for example.

References `gerr`.

12.47.3.5 void register_double (doublefunc_t func_ptr, std::string description = " ")

This function can be used to register function that will be called by the log_now function to obtained double value to be logged into logfile. The second optional parameter is a short (below 8 characters) description of the value that could be (depending on format) printed in the header of the file.

12.47.3.6 void register_int (intfunc_t func_ptr, std::string description = " ")

This function can be used to register function that will be called by the log_now function to obtained int value to be logged into logfile. The second optional parameter is a short (below 8 characters) description of the value that could be (depending on format) printed in the header of the file.

12.47.3.7 void register_slong_int (const log_slong_int_t &functor, std::string description = " ")

This function can be used to register functor that will be called by the log_now function to obtained signed int value to be logged into logfile. The second optional parameter is a short (below 8 characters) description of the value that could be (depending on format) printed in the header of the file.

This can be used to log the number of states stored in the queue of states waiting for exploration, for example.

References gerr.

12.47.3.8 void register_ulong_int (const log_ulong_int_t &functor, std::string description = " ")

This function can be used to register functor that will be called by the log_now function to obtained unsigned int value to be logged into logfile. The second optional parameter is a short (below 8 characters) description of the value that could be (depending on format) printed in the header of the file.

This can be used to log the number of states stored in the queue of states waiting for exploration, for example.

References gerr.

12.47.3.9 void register_unsigned (unsignedfunc_t func_ptr, std::string description = " ")

This function can be used to register function that will be called by the log_now function to obtained unsigned int value to be logged into logfile. The second optional parameter is a short (below 8 characters) description of the value that could be (depending on format) printed in the header of the file.

This can be used to log the number of states stored in the queue of states waiting for exploration, for example.

12.47.3.10 `void set_storage (explicit_storage_t * storage_ptr)`

If logging of the number of states kept in the storage is requested, this member function should be used to set the pointer to instance of `explicit_storage_t` class.

Should be called before `init` member function.

12.47.3.11 `void stop_SIGALRM ()`

Stops POSIX signal/alarm mechanism. (Starts ingoring incomming signals.)

12.47.3.12 `void use_SIGALRM (unsigned int period = 1)`

Starts POSIX signal/alarm mechanism to call `log_now` function periodically. The period is given in seconds. (One second is the shortest possible period.)

Should be called after the `init` member function.

The documentation for this class was generated from the following files:

- [logger.hh](#)
- [logger.cc](#)

12.48 message_t Class Reference

Class representing a data to send or receive using [network_t](#).

```
#include <message.hh>
```

Public Member Functions

- void [append_bool](#) (const bool flag)
Writes a flag to the message.
- void [append_byte](#) (const byte_t number)
Writes a number of type byte_t to the message.
- void [append_data](#) (const byte_t *const data_to_copy, const size_int_t size_to_copy)
Writes 'size_to_copy' bytes from 'data_to_copy' to the message.
- void [append_sbyte](#) (const sbyte_t number)
Writes a number of type sbyte_t to the message.
- void [append_size_int](#) (const size_int_t number)
Writes a number of type size_int_t to the message.
- void [append_slong_int](#) (const slong_int_t number)
Writes a number of type slong_int_t to the message.
- void [append_sshort_int](#) (const sshort_int_t number)
Writes a number of type sshort_int_t to the message.
- void [append_state](#) (const state_t state)
Writes a complete representation of state to the message.
- void [append_state_ref](#) (const state_ref_t state_ref)
Writes a state reference 'state_ref' to the message.
- void [append_ulong_int](#) (const ulong_int_t number)
Writes a number of type ulong_int_t to the message.
- void [append_ushort_int](#) (const ushort_int_t number)
Writes a number of type ushort_int_t to the message.
- size_int_t [get_allocated_size](#) () const
Returns a size of byte sequence representing a message.
- const byte_t * [get_data](#) () const

Returns a stored byte sequence representing a message.

- `byte_t * get_data ()`

Returns a stored byte sequence representing a message.

- `size_int_t get_written_size () const`
- `void load_data (byte_t *const new_data, const size_int_t new_allocated_size)`
Replaces a data in a message by given data (using memory copying).
- `message_t (const size_int_t number_of_preallocated_bytes=1024, const size_int_t reallocation_step=1024)`

A constructor.

- `void read_bool (bool &flag)`
Copies a flag from the message to 'flag'.
- `void read_byte (byte_t &number)`
Copies a number of type byte_t to 'number'.
- `void read_data (char *const data_to_copy, const std::size_t size_to_copy)`
Copies 'size_to_copy' bytes from the message to 'data_to_copy'.
- `void read_sbyte (sbyte_t &number)`
Copies a number of type sbyte_t to 'number'.
- `void read_size_int (size_int_t &number)`
Copies a number of type size_int_t to 'number'.
- `void read_slong_int (slong_int_t &number)`
Copies a number of type slong_int_t to 'number'.
- `void read_sshort_int (sshort_int_t &number)`
Copies a number of type sshort_int_t to 'number'.
- `void read_state (state_t &state)`
Copies a state stored in the message to 'state'.
- `void read_state_ref (state_ref_t &ref)`
Copies a reference to state stored in the message to 'ref'.
- `void read_ulong_int (ulong_int_t &number)`
Copies a number of type ulong_int_t to 'number'.
- `void read_ushort_int (ushort_int_t &number)`
Copies a number of type ushort_int_t to 'number'.

- void [rewind](#) ()
Moves both reading and writing heads to the beginning of the message.
- void [rewind_append](#) ()
Moves a writing head to the beginning of the message.
- void [rewind_read](#) ()
Moves a reading head to the beginning of the message.
- void [set_data](#) (byte_t *const new_data, const size_int_t new_allocated_size)
Sets a byte sequence stored in a message.
- void [set_data](#) (byte_t *const new_data, const size_int_t new_allocated_size, const size_int_t new_written_size)
Sets a byte sequence stored in a message.
- void [set_written_size](#) (const size_int_t new_size)
Sets a size of written part of message.
- [~message_t](#) ()
A destructor.

12.48.1 Detailed Description

Class representing a data to send or receive using [network_t](#).

This class is good for sending or receiving of messages consisting of several items (of possibly various types). It supports a transmission of basic integer types, states, state references and general sequences of bytes.

Warning:

This class implicitly allocates 4096 B. It is not good idea to have 1000000 instances of it or to create and destroy its instances many times. It is presumed, that in a program there will be only few "global" instances shared by all sending and receiving procedures.

12.48.2 Constructor & Destructor Documentation

12.48.2.1 [message_t](#) (const size_int_t *number_of_preallocated_bytes* = 1024, const size_int_t *reallocation_step* = 1024)

A constructor.

Parameters:

number_of_preallocated_bytes = a count of bytes to allocate for a byte sequence representing a message

reallocation_step = how much more bytes to allocate in case of reallocation

12.48.2.2 ~message_t() [inline]

A destructor.

It only deallocates a byte sequence stored in a message

12.48.3 Member Function Documentation

12.48.3.1 void append_bool (const bool *flag*)

Writes a flag to the message.

Warning:

Flag is stored to the single byte (not to sizeof(bool) bytes!)

References append_byte().

12.48.3.2 void append_data (const byte_t **data_to_copy*, const size_int_t *size_to_copy*)

Writes 'size_to_copy' bytes from 'data_to_copy' to the message.

Parameters:

data_to_copy = pointer to the sequence of bytes

size_to_copy = number of bytes to copy from the byte sequence given in *data_to_copy*

12.48.3.3 void append_state (const state_t *state*)

Writes a complete representation of state to the message.

Parameters:

state = state to copy to the message

Writes 'state.size' and the byte sequence referenced by 'state.ptr' to the message

References state_t::ptr, and state_t::size.

12.48.3.4 size_int_t get_written_size () const [inline]

Returns a size of part of the message, where something has been written using append_* methods

Referenced by network_t::send_message(), and network_t::send_urgent_message().

12.48.3.5 void load_data (byte_t *const new_data, const size_int_t new_allocated_size) [inline]

Replaces a data in a message by given data (using memory copying).

Writes a given byte sequence to the begin of the message. The reading head is rewound to the begin of the message, writing head is moved to the end of new message content.

Implemented simply as `this->rewind(); this->append_data(new_data, new_allocated_size);`

12.48.3.6 void read_data (char *const data_to_copy, const std::size_t size_to_copy)

Copies 'size_to_copy' bytes from the message to 'data_to_copy'.

Parameters:

data_to_copy = the pointer to the byte sequence at least *data_to_copy* bytes long

size_to_copy = count of bytes to copy from the message

12.48.3.7 void read_state (state_t & state)

Copies a state stored in the message to 'state'.

Parameters:

state = the state which will be rewritten by the copy of state stored in the message

Warning:

This method does not deallocate the byte sequence in state.ptr. This method allocated a new memory space and the user is responsible for a deallocation of written state.ptr.

12.48.3.8 void rewind () [inline]

Moves both reading and writing heads to the beginning of the message.

Then `get_written_size()` will return 0, `append_*` methods will write from the beginning of the message and `read_*` methods will read from the beginning of the message. Use this method if you do not care of the stored message and you want to write a new message to the same instance of this class.

12.48.3.9 void rewind_append () [inline]

Moves a writing head to the beginning of the message.

Then [get_written_size\(\)](#) will return 0 and `append_*` methods will write from the beginning of the message. Use this method if you do not care of the stored message and you want to write a new message to the same instance of this class.

12.48.3.10 void rewind_read () [inline]

Moves a reading head to the beginning of the message.

Then all `read_*` methods will read from the beginning of the message

12.48.3.11 void set_data (byte_t *const new_data, const size_int_t new_allocated_size)

Sets a byte sequence stored in a message.

The same as `message_t::set_data(new_data, new_allocated_size, new_allocated_size)`

Warning:

The byte sequence set by this method will be deallocated in a destructor! If you do not like this behavior, use a method [load_data\(\)](#).

12.48.3.12 void set_data (byte_t *const new_data, const size_int_t new_allocated_size, const size_int_t new_written_size)

Sets a byte sequence stored in a message.

Parameters:

new_data = byte sequence to store in a message

new_allocated_size = size in bytes of the memory referenced by *new_data*

new_written_size = the number of bytes, which are already written in the message (valid bytes)

Warning:

The byte sequence set by this method will be deallocated in a destructor! If you do not like this behavior, use a method [load_data\(\)](#).

Referenced by `network_t::receive_message()`.

12.48.3.13 void set_written_size (const size_int_t new_size) [inline]

Sets a size of written part of message.

It can be useful if you need to write a part of message directly to the memory referenced by [get_data\(\)](#).

Referenced by `network_t::receive_message()`.

The documentation for this class was generated from the following files:

- `message.hh`
- `message.cc`

12.49 network_t Class Reference

Network communication support class.

```
#include <network.hh>
```

Public Member Functions

- bool [abort](#) (void)
Abort computation.
- bool [all_gather](#) (char *sbuf, int ssize, char *rbuf, int rsize)
Similar to [gather\(\)](#), but the collected information are received on every workstation.
- bool [barrier](#) (void)
Finalizes network, essentially closes connections and frees buffers.
- bool [finalize](#) ()
Finalizes network, frees all allocated buffers.
- bool [flush_all_buffers](#) ()
All buffers flush.
- bool [flush_all_buffers_timed_out_only](#) ()
Only timed out buffers flush.
- bool [flush_buffer](#) (int dest)
One send buffer flush.
- bool [flush_some_buffers](#) ()
Some buffers flush.
- bool [gather](#) (char *sbuf, int ssize, char *rbuf, int rsize, int root)
Similar to [barrier\(\)](#) but allows to collect some data on selected workstation.
- bool [get_all_barriers_cnt](#) (int &cnt)
Get count of all [barrier\(\)](#) calls.
- bool [get_all_buffers_flushes_cnt](#) (int &cnt)
Get number of flushes of all send buffers.
- bool [get_all_received_msgs_cnt](#) (int &cnt)
Get all received messages count (including urgent messages).
- bool [get_all_sent_msgs_cnt](#) (int &cnt)
Get all sent messages count (including urgent messages).

- bool [get_all_total_pending_size](#) (int &size, bool test)
Get total size of pending (blocked) send buffers for all workstations.
- bool [get_buf_msgs_cnt_limit](#) (int &limit)
Get send buffer's message count limit.
- bool [get_buf_size_limit](#) (int &limit)
Get send buffer's size limit.
- bool [get_buf_time_limit](#) (int &limit_sec, int &limit_msec)
Get send buffer's time limit.
- bool [get_buffer_flushes_cnt](#) (int dest, int &cnt)
Get number of flushes of send buffer for workstation dest.
- bool [get_cluster_size](#) (int &size)
Retrieves the number of computers in the cluster.
- bool [get_comm_matrix_rnm](#) (pcomm_matrix_t &ret, int target)
Get communication matrix of received normal messages.
- bool [get_comm_matrix_rum](#) (pcomm_matrix_t &ret, int target)
Get communication matrix of received urgent messages.
- bool [get_comm_matrix_snm](#) (pcomm_matrix_t &ret, int target)
Get communication matrix of sent normal messages.
- bool [get_comm_matrix_sum](#) (pcomm_matrix_t &ret, int target)
Get communication matrix of sent urgent messages.
- bool [get_id](#) (int &id)
Retrieves the identifier of the calling workstation.
- bool [get_processor_name](#) (char *proc_name, int &length)
Retrieves the name of the calling workstation.
- int [get_rcv_msgs_cnt_rcv_from](#) (int from)
Get number of messages received from given workstation.
- bool [get_rcv_msgs_cnt_rcv_from](#) (int from, int &cnt)
Get number of messages received from given workstation.
- int [get_sent_msgs_cnt_sent_to](#) (int to)
Get number of messages sent to a given workstation.

- bool [get_sent_msgs_cnt_sent_to](#) (int to, int &cnt)
Get number of messages sent to a given workstation.
- bool [get_total_pending_size](#) (int dest, int &size, bool test)
Get size of pending (blocked) send buffer for workstation dest.
- bool [get_user_received_msgs_cnt](#) (int &cnt)
Get normal received messages count (only normal messages, without urgent messages).
- bool [get_user_sent_msgs_cnt](#) (int &cnt)
Get normal sent messages count (only normal messages, without urgent messages).
- bool [initialize_buffers](#) ()
Completes initialization, must be called after and only after [initialize_network\(\)](#).
- bool [initialize_network](#) (int &argc, char **&argv)
- bool [is_new_message](#) (int &size, int &src, int &tag, bool &flag)
Non-blocking message probe.
- bool [is_new_message_from_source](#) (int &size, int src, int &tag, bool &flag)
Non-blocking message from specific source probe.
- bool [is_new_urgent_message](#) (int &size, int &src, int &tag, bool &flag)
Non-blocking urgent message probe.
- bool [is_new_urgent_message_from_source](#) (int &size, int src, int &tag, bool &flag)
Non-blocking urgent message probe from specific source.
- [network_t](#) ([error_vector_t](#) &arg0=[gerr](#))
- bool [receive_message](#) ([message_t](#) &message, int &src, int &tag)
Blocking message receive.
- bool [receive_message](#) (char *buf, int &size, int &src, int &tag)
Blocking message receive.
- bool [receive_message_from_source](#) (char *buf, int &size, int src, int &tag)
Blocking message from source receive.
- bool [receive_message_non_exc](#) (char *&buf, int &size, int &src, int &tag)
Blocking message receive, which sets buf to point to internal buffer.
- bool [receive_urgent_message](#) (char *buf, int &size, int &src, int &tag)
Blocking message receive.

- bool [receive_urgent_message_from_source](#) (char *buf, int &size, int src, int &tag)
Blocking urgent message from source receive.
- bool [receive_urgent_message_non_exc](#) (char *&buf, int &size, int &src, int &tag)
Blocking urgent message receive, which sets buf to point to internal buffer.
- bool [send_message](#) (const [message_t](#) &message, int dest, int tag)
Blocking message send.
- bool [send_message](#) (char *buf, int size, int dest, int tag)
Blocking message send.
- bool [send_urgent_message](#) (const [message_t](#) &message, int dest, int tag)
Blocking urgent message send.
- bool [send_urgent_message](#) (char *buf, int size, int dest, int tag)
Blocking urgent message send.
- bool [set_buf_msgs_cnt_limit](#) (int limit)
Set send buffer's message count limit, see [get_buf_msgs_cnt_limit\(\)](#) doc for more info.
- bool [set_buf_size_limit](#) (int limit)
Set send buffer's size limit, see [get_buf_size_limit\(\)](#) doc for more info.
- bool [set_buf_time_limit](#) (int limit_sec, int limit_msec)
Set send buffer's time limit, see [get_buf_time_limit\(\)](#) doc for more info.
- void **stats_init** (statistics *stats)
- double **stats_max** (statistics *stats)
- double **stats_mean** (statistics *stats)
- double **stats_min** (statistics *stats)
- unsigned **stats_num** (statistics *stats)
- double **stats_sqrt_var** (statistics *stats)
- void **stats_update** (statistics *stats, double val)
- double **stats_var** (statistics *stats)
- [~network_t](#) ()
A destructor, frees allocated memory.

Public Attributes

- statistics **stats_Recv_bytes**
- statistics **stats_Sent_bytes_local**
- double **stats_slice_begin**
- double **stats_slice_end**
- double **stats_time_begin**

Protected Attributes

- [error_vector_t](#) & `errvec`

12.49.1 Detailed Description

Network communication support class.

This class mainly provides methods for buffered transmission of messages. Additionally barrier and gather methods are included. Methods for detailed statistics are a matter of course.

12.49.2 Constructor & Destructor Documentation

12.49.2.1 network_t (error_vector_t & *arg0* = *gerr*)

A constructor, doesn't initialize network!,
non-default error handling vector can be specified

12.49.3 Member Function Documentation

12.49.3.1 bool abort (void)

Abort computation.

Returns:

true if the function succeeds, **false** otherwise Use this function to terminate all processes participating in the distributed computation

References `ERR_triplet_t::message`, `NET_ERR_ABORT_FAILED`, `net_err_msgs`, and `NET_ERR_NOT_INITIALIZED`.

12.49.3.2 bool all_gather (char * *sbuf*, int *ssize*, char * *rbuf*, int *rsize*)

Similar to [gather\(\)](#), but the collected information are received on every workstation.

Differs from [gather\(\)](#) in that all workstations must have *rbuf* and *rsize* valid, not only one workstation.

References `ERR_triplet_t::message`, `NET_ERR_ALLGATHER_FAILED`, `net_err_msgs`, and `NET_ERR_NOT_INITIALIZED`.

12.49.3.3 bool barrier (void)

Finalizes network, essentially closes connections and frees buffers.

Returns:

true if initialization was successful, **false** otherwise

Use this function to finalize network, this works only if the network was previously initialized, otherwise error occurs. The current implementation does not allow to initialize the network more than once, so subsequent calls to [initialize_network\(\)](#) will cause error. Stops the computation until all workstation call [barrier\(\)](#)

Returns:

true if barrier was successful, **false** otherwise

Use this function to "synchronize" all workstation. The function blocks until all workstations call it.

References [ERR_triplet_t::message](#), [NET_ERR_BARRIER_FAILED](#), [net_err_msgs](#), and [NET_ERR_NOT_INITIALIZED](#).

Referenced by [distributed_t::synchronized\(\)](#).

12.49.3.4 bool flush_all_buffers ()

All buffers flush.

Returns:

true if the function succeeds, **false** otherwise

Use this function to flush all buffers (Technically, the function calls [flush_buffer\(\)](#) function for all workstations in the cluster.

References [flush_buffer\(\)](#), [ERR_triplet_t::message](#), [net_err_msgs](#), and [NET_ERR_NOT_INITIALIZED](#).

Referenced by [distributed_t::set_idle\(\)](#).

12.49.3.5 bool flush_all_buffers_timed_out_only ()

Only timed out buffers flush.

Returns:

true if the function succeeds, **false** otherwise

Use this function to flush all buffers that have the time limit exceeded (the time limit can be adjusted by the [set_buf_time_limit\(\)](#) function).

References [flush_buffer\(\)](#), [ERR_triplet_t::message](#), [net_err_msgs](#), and [NET_ERR_NOT_INITIALIZED](#).

Referenced by [distributed_t::process_messages\(\)](#).

12.49.3.6 bool flush_buffer (int *dest*)

One send buffer flush.

Parameters:

dest - identification of the buffer (id of the workstation the buffer is designated for) to be flushed

Returns:

true if the function succeeds, **false** otherwise

Use this function to send data in the buffer determined by the *dest* parameter.

References ERR_triplet_t::message, NET_ERR_INVALID_DESTINATION, net_err_msgs, NET_ERR_NOT_INITIALIZED, NET_ERR_SEND_MSG_FAILED, and NET_TAG_NORMAL.

Referenced by flush_all_buffers(), flush_all_buffers_timed_out_only(), and flush_some_buffers().

12.49.3.7 bool flush_some_buffers ()

Some buffers flush.

Returns:

true if the function succeeds, **false** otherwise

Use this function to flush "some" of the largest buffers (currently, the function only calls [flush_buffer\(\)](#) for the single workstation with the largest amount of work buffered, since this already appears to give good performance in practice).

References flush_buffer(), ERR_triplet_t::message, net_err_msgs, and NET_ERR_NOT_INITIALIZED.

Referenced by distributed_t::set_idle().

12.49.3.8 bool gather (char * *sbuf*, int *ssize*, char * *rbuf*, int *rsize*, int *root*)

Similar to [barrier\(\)](#) but allows to collect some data on selected workstation.

Parameters:

sbuf - pointer to buffer that contains the data to be collected

ssize - size of the data sbuf points to (in sizeof(char))

rbuf - pointer to buffer that receives the collected data (relevant only on selected workstation)

rsize - size of the collected data corresponding to one workstation, therefore *rsize* should be equal to *ssize* (relevant only on selected workstation)

root - id of the selected workstation that receives the collected data

Returns:

true if the function succeeds, **false** otherwise

Use this function to "synchronize" workstations (in the sense explained for [barrier\(\)](#) function) and collect some information at the same time. Each workstation must call `gather` with the same *ssize*, all workstations except for the selected one can pass **NULL** and **0** as *rbuf* and *rsiz*e parameters, respectively. Root parameter must be the same on all workstations. The selected workstation should pass the same value as *ssize* as *rsiz*e. The buffer to which *rbuf* points must be of size *rsiz*e * [number of computers in the cluster].

The collected data are in order imposed by workstation ids.

References `ERR_triplet_t::message`, `NET_ERR_GATHER_FAILED`, `net_err_msgs`, and `NET_ERR_NOT_INITIALIZED`.

12.49.3.9 bool get_all_barriers_cnt (int & cnt)

Get count of all [barrier\(\)](#) calls.

Parameters:

cnt - output parameter that receives the count of all [barrier\(\)](#) function calls on the calling workstation

Returns:

true if the function succeeds, **false** otherwise

References `ERR_triplet_t::message`, `net_err_msgs`, and `NET_ERR_NOT_INITIALIZED`.

Referenced by `logger_t::log_now()`.

12.49.3.10 bool get_all_buffers_flushes_cnt (int & cnt)

Get number of flushes of all send buffers.

Parameters:

cnt - output parameter that receives the count of all send buffer flushes on the calling workstation

Returns:

true if the function succeeds, **false** otherwise

References `get_buffer_flushes_cnt()`, `ERR_triplet_t::message`, `net_err_msgs`, and `NET_ERR_NOT_INITIALIZED`.

Referenced by `logger_t::log_now()`.

12.49.3.11 bool get_all_received_msgs_cnt (int & cnt)

Get all received messages count (including urgent messages).

Parameters:

cnt - output parameter that receives the count of all messages received on the calling workstation

Returns:

true if the function succeeds, **false** otherwise

References ERR_triplet_t::message, net_err_msgs, and NET_ERR_NOT_INITIALIZED.

Referenced by logger_t::log_now(), and distributed_t::process_messages().

12.49.3.12 bool get_all_sent_msgs_cnt (int & cnt)

Get all sent messages count (including urgent messages).

Parameters:

cnt - output parameter that receives the count of all messages sent by the calling workstation

Returns:

true if the function succeeds, **false** otherwise.

References ERR_triplet_t::message, net_err_msgs, and NET_ERR_NOT_INITIALIZED.

Referenced by logger_t::log_now(), and distributed_t::process_messages().

12.49.3.13 bool get_all_total_pending_size (int & size, bool test)

Get total size of pending (blocked) send buffers for all workstations.

Parameters:

size - output parameter that receives total size of memory occupied by all pending send buffers

test - if test = true, tests are performed and exact value is retrieved, if test = false, upper estimate is returned

Returns:

true if the function succeeds, **false** otherwise

References get_total_pending_size(), ERR_triplet_t::message, net_err_msgs, and NET_ERR_NOT_INITIALIZED.

12.49.3.14 bool get_buf_msgs_cnt_limit (int & limit)

Get send buffer's message count limit.

Parameters:

limit - the maximal count of messages send buffer can hold

Returns:

true if the function succeeds, **false** otherwise

Use this function to get the send buffer's message count limit. If the number of messages in any send buffer (on each workstation there is one send buffer for every workstation) is equal to this limit, the whole contents of the buffer is sent to the workstation the buffer is designated for.

100 messages is default.

This limit can be set by [set_buf_msgs_cnt_limit\(\)](#) function, but only before [initialize_buffers\(\)](#) function is called.

12.49.3.15 bool get_buf_size_limit (int & limit)

Get send buffer's size limit.

Parameters:

limit - the maximal size of the send buffer

Returns:

true if the function succeeds, **false** otherwise

Use this function to get the send buffer's size limit. If the size of any send buffer (on each workstation there is one send buffer for every workstation) is about to exceed or equal this limit, the whole contents of the buffer is sent to the workstation the buffer is designated for.

8192 chars is default.

This limit can be set by [set_buf_size_limit\(\)](#) function, but only before [initialize_buffers](#) function is called.

12.49.3.16 bool get_buf_time_limit (int & limit_sec, int & limit_msec)

Get send buffer's time limit.

Parameters:

limit_sec - together with *limit_msec* the "maximal" amount of time send buffer can hold data without flush (in seconds)

limit_msec - together with *limit_sec* the "maximal" amount of time send buffer can hold data without flush (in milliseconds)

Returns:

true if the function succeeds, **false** otherwise

Use this function to get the send buffer's time limit. If the time of any send buffer (on each workstation there is one send buffer for every workstation) is about to exceed or equal this limit, the whole contents of the buffer is sent to the workstation the buffer is designated for. The previous sentence is not completely true, because if nobody calls some function that checks the limit, the data can stay in the buffer for more than the limit. There is no "timer" that check the limit.

300 milliseconds is default.

This limit can be set by [set_buf_time_limit\(\)](#) function, but only before initialize buffers function is called.

Setting the limit to 0 seconds and 0 milliseconds turns timed flushing off.

12.49.3.17 bool get_buffer_flushes_cnt (int *dest*, int & *cnt*)

Get number of flushes of send buffer for workstation *dest*.

Parameters:

dest - identification of the send buffer

cnt - output parameter that receives the count of all times the send buffer designated for workstation with id *dest* has been flushed

Returns:

true if the function succeeds, **false** otherwise

References `ERR_triplet_t::message`, `NET_ERR_INVALID_DESTINATION`, `net_err_msgs`, and `NET_ERR_NOT_INITIALIZED`.

Referenced by `get_all_buffers_flushes_cnt()`.

12.49.3.18 bool get_cluster_size (int & *size*)

Retrieves the number of computers in the cluster.

Parameters:

size - output parameter that receives the retrieved value

Returns:

true if the function succeeds, **false** otherwise

Use this function to get the number of computers in the cluster. The variable that receives the actual size of the cluster is passed as the first parameter.

You must call [initialize_network\(\)](#) function before this function.

References `ERR_triplet_t::message`, `net_err_msgs`, and `NET_ERR_NOT_INITIALIZED`.

Referenced by `distributed_t::network_initialize()`.

12.49.3.19 `bool get_comm_matrix_rnm (pcomm_matrix_t & ret, int target)`

Get communication matrix of received normal messages.

Similar to [get_comm_matrix_snm\(\)](#), but the position (i, j) of the matrix contains the count of normal messages (sent by [send_message\(\)](#) function) received on workstation with id i from workstation with id j .

12.49.3.20 `bool get_comm_matrix_rum (pcomm_matrix_t & ret, int target)`

Get communication matrix of received urgent messages.

Similar to [get_comm_matrix_snm\(\)](#), but the position (i, j) of the matrix contains the count of urgent messages (sent by [send_urgent_message\(\)](#) function) received on workstation with id i from workstation with id j .

12.49.3.21 `bool get_comm_matrix_snm (pcomm_matrix_t & ret, int target)`

Get communication matrix of sent normal messages.

Parameters:

- ret* - output parameter, pointer to communication matrix (of type [comm_matrix_t](#)) that contains at position (i, j) the count of normal messages (sent by function [send_message\(\)](#)) sent by workstation with id i to workstation with id j
- target* - only one workstation gets valid pointer ret, this workstation is specified by *target* parameter, which must be the same on all workstations

Returns:

true if the function succeeds, **false** otherwise

Use this function to get the counts of messages sent by each workstation to each workstation. The function must be called in a way explained for the [barrier\(\)](#) function.

12.49.3.22 `bool get_comm_matrix_sum (pcomm_matrix_t & ret, int target)`

Get communication matrix of sent urgent messages.

Similar to [get_comm_matrix_snm\(\)](#), but the position (i, j) of the matrix contains the count of urgent messages (sent by [send_urgent_message\(\)](#) function) sent by workstation with id i to workstation with id j .

12.49.3.23 bool get_id (int & *id*)

Retrieves the identifier of the calling workstation.

Parameters:

id - output parameter that receives the workstation id

Returns:

true if the function succeeds, **false** otherwise

Use this function to get the unique identifier of the calling workstation. The identifier is in the range $[0..cluster_size - 1]$, where *cluster_size* is the number of computers in the cluster and can be obtained by calling the [get_cluster_size\(\)](#) function.

You must call [initialize_network\(\)](#) function before this function.

References `ERR_triplet_t::message`, `net_err_msgs`, and `NET_ERR_NOT_INITIALIZED`.

Referenced by `distributed_t::network_initialize()`.

12.49.3.24 bool get_processor_name (char * *proc_name*, int & *length*)

Retrieves the name of the calling workstation.

Parameters:

proc_name - output parameter that receives the workstation name

length - output parameter that receives the number of characters in the name

Returns:

true if the function succeeds, **false** otherwise

Use this function to get the name of the calling workstation.

You must call [initialize_network\(\)](#) function before this function.

References `ERR_triplet_t::message`, `net_err_msgs`, and `NET_ERR_NOT_INITIALIZED`.

Referenced by `distributed_t::network_initialize()`.

12.49.3.25 int get_rcv_msgs_cnt_rcv_from (int *from*)

Get number of messages received from given workstation.

Parameters:

from - specifies the workstation we are interested in

Returns:

the count of all message received on the calling workstation from the workstation specified by *from* parameter

References `ERR_triplet_t::message`, `NET_ERR_INVALID_WORKSTATION_NUMBER`, `net_err_msgs`, and `NET_ERR_NOT_INITIALIZED`.

12.49.3.26 `bool get_rcv_msgs_cnt_rcv_from (int from, int & cnt)`

Get number of messages received from given workstation.

Parameters:

from - specifies the workstation we are interested in

cnt - output parameter that receives the count of all message received on the calling workstation from the workstation specified by *from* parameter.

Returns:

true if the function succeeds, **false** otherwise

References `ERR_triplet_t::message`, `NET_ERR_INVALID_WORKSTATION_NUMBER`, `net_err_msgs`, and `NET_ERR_NOT_INITIALIZED`.

12.49.3.27 `int get_sent_msgs_cnt_sent_to (int to)`

Get number of messages sent to a given workstation.

Parameters:

to - specifies the workstation we are interested in

Returns:

the count of all messages sent by calling workstation to a workstation specified by *to* parameter

References `ERR_triplet_t::message`, `NET_ERR_INVALID_WORKSTATION_NUMBER`, `net_err_msgs`, and `NET_ERR_NOT_INITIALIZED`.

12.49.3.28 `bool get_sent_msgs_cnt_sent_to (int to, int & cnt)`

Get number of messages sent to a given workstation.

Parameters:

to - specifies the workstation we are interested in

cnt - output parameter that receives the count of all messages sent by calling workstation to a workstation specified by *to* parameter.

Returns:

true if the function succeeds, **false** otherwise

References ERR_triplet_t::message, NET_ERR_INVALID_WORKSTATION_NUMBER, net_err_msgs, and NET_ERR_NOT_INITIALIZED.

Referenced by logger_t::log_now().

12.49.3.29 bool get_total_pending_size (int dest, int & size, bool test)

Get size of pending (blocked) send buffer for workstation *dest*.

Parameters:

dest - identification of the send buffer

size - output parameter that receives total size of memory occupied by pending send buffer designated for workstation with id *dest*

test - if test = true, tests are performed and exact value is retrieved, if test = false, upper estimate is returned

Returns:

true if the function succeeds, **false** otherwise

References ERR_triplet_t::message, NET_ERR_INVALID_DESTINATION, net_err_msgs, and NET_ERR_NOT_INITIALIZED.

Referenced by get_all_total_pending_size().

12.49.3.30 bool get_user_received_msgs_cnt (int & cnt)

Get normal received messages count (only normal messages, without urgent messages).

Parameters:

cnt - output parameter that receives the count of all normal messages received on the calling workstation

Returns:

true if the function succeeds, **false** otherwise

References ERR_triplet_t::message, net_err_msgs, and NET_ERR_NOT_INITIALIZED.

12.49.3.31 `bool get_user_sent_msgs_cnt (int & cnt)`

Get normal sent messages count (only normal messages, without urgent messages).

Parameters:

cnt - output parameter that receives the count of all normal messages sent by the calling workstation

Returns:

true if the function succeeds, **false** otherwise.

References `ERR_triplet_t::message`, `net_err_msgs`, and `NET_ERR_NOT_INITIALIZED`.

12.49.3.32 `bool initialize_buffers ()`

Completes initialization, must be called after and only after [initialize_network\(\)](#).

Returns:

true if initialization was successful, **false** otherwise

Use this function to complete initialization process. It initializes buffers, thus enabling the send/receive functions, which use buffering.

References `ERR_triplet_t::message`, `net_err_msgs`, and `NET_ERR_NOT_INITIALIZED`.

Referenced by `distributed_t::initialize()`.

12.49.3.33 `bool initialize_network (int & argc, char **& argv)`

Initializes network, essentially establishes connections and gives all workstations the command line parameters

Parameters:

argc - number of command line parameters, passing the first parameter of `main()` is recommended

argv - command line parameters, passing the second parameter of `main()` is recommended

Returns:

true if initialization was successful, **false** otherwise

Use this function to initialize network. After successful initialization you can obtain workstation identification by calling [get_id\(\)](#) function, number of computers in cluster by calling [get_cluster_size\(\)](#) function and workstation name using [get_processor_name\(\)](#) function. You can also set some properties of build-in buffering scheme using set-functions.

References `ERR_triplet_t::message`, `NET_ERR_ALREADY_INITIALIZED`, `NET_ERR_INITIALIZATION_FAILED`, and `net_err_msgs`.

Referenced by `distributed_t::network_initialize()`.

12.49.3.34 `bool is_new_message (int & size, int & src, int & tag, bool & flag)`

Non-blocking message probe.

Parameters:

- size* - output parameter that receives the size of the message, if there is one
- src* - output parameter that receives the source workstation of the message, if there is one
- tag* - output parameter that receives the tag of the message, if there is one
- flag* - output parameter that determines if there is a message waiting to be received

Returns:

true if the function succeeds, **false** otherwise

Use this function to find out whether there is some unreceived message waiting. The function is non-blocking in the sense that it does not wait for messages to arrive, it only finds out whether there are some.

This function only looks for messages sent by [send_message\(\)](#) function.

Referenced by `distributed_t::process_messages()`.

12.49.3.35 `bool is_new_message_from_source (int & size, int src, int & tag, bool & flag)`

Non-blocking message from specific source probe.

Similar to [is_new_message\(\)](#), but the *src* parameter is input. It means that the function only looks for messages from speceified source workstation.

12.49.3.36 `bool is_new_urgent_message (int & size, int & src, int & tag, bool & flag)`

Non-blocking urgent message probe.

Similar to [is_new_message\(\)](#), but it looks only for messages sent by [send_urgent_message\(\)](#) function.

Referenced by `distributed_t::process_messages()`.

12.49.3.37 `bool is_new_urgent_message_from_source (int & size, int src, int & tag, bool & flag)`

Non-blocking urgent message probe from specific source.

Similar to `is_new_message()`, but it looks only for messages sent by `send_urgent_message()` function and only for those of them that are from source workstation specified by the input parameter `src`.

12.49.3.38 `bool receive_message (message_t & message, int & src, int & tag)`

Blocking message receive.

Parameters:

message - instance of `message_t`, where the data will be stored

src - output parameter that gets id of the source workstation of the received message

tag - output parameter that gets the tag of the received message

Returns:

true if the function succeeds, **false** otherwise

Use this function to receive a message from network. The receiving process is blocking in the sense that the function does not terminate until the message is completely received. Apart from the other things, It means that if there is no message to be received, the function will block the computation until one arrives.

This function only concerns messages sent by the `send_message()` function.

References `message_t::get_allocated_size()`, `message_t::get_data()`, `receive_message()`, `message_t::set_data()`, and `message_t::set_written_size()`.

12.49.3.39 `bool receive_message (char * buf, int & size, int & src, int & tag)`

Blocking message receive.

Parameters:

buf - pointer to memory used to store the received data

size - output parameter that receives the size of the received message

src - output parameter that gets id of the source workstation of the received message

tag - output parameter that gets the tag of the received message

Returns:

true if the function succeeds, **false** otherwise

Use this function to receive a message from network. The receiving process is blocking in the sense that the function does not terminate until the message is completely received. Apart from the other things, It means that if there is no message to be received, the function will block the computation until one arrives.

This function only concerns messages sent by the [send_message\(\)](#) function.

Referenced by `distributed_t::process_messages()`, and `receive_message()`.

12.49.3.40 `bool receive_message_from_source (char * buf, int & size, int src, int & tag)`

Blocking message from source receive.

Similar to [receive_message\(\)](#), but it receives only messages sent by `send_urgent_message` function.

12.49.3.41 `bool receive_message_non_exc (char *& buf, int & size, int & src, int & tag)`

Blocking message receive, which sets `buf` to point to internal buffer.

Similar to [receive_message\(\)](#), but the `buf` parameter is output and need not be initialized, because the function sets it to point to internal buffer containing the received data, which saves both memory and time. It is obvious that the memory `buf` points to must neither be freed nor rewritten.

Referenced by `distributed_t::process_messages()`.

12.49.3.42 `bool receive_urgent_message (char * buf, int & size, int & src, int & tag)`

Blocking message receive.

Similar to [receive_message\(\)](#), but it receives only messages sent by `send_urgent_message` function.

Referenced by `distributed_t::process_messages()`.

12.49.3.43 `bool receive_urgent_message_from_source (char * buf, int & size, int src, int & tag)`

Blocking urgent message from source receive.

Similar to `receive_message_from_source()`, but it receives only messages sent by [send_urgent_message\(\)](#) function.

12.49.3.44 **bool receive_urgent_message_non_exc** (char *& *buf*, int & *size*, int & *src*, int & *tag*)

Blocking urgent message receive, which sets *buf* to point to internal buffer.

Similar to [receive_message_non_exc\(\)](#), but it receives only messages sent by [send_urgent_message\(\)](#) function.

Referenced by `distributed_t::process_messages()`.

12.49.3.45 **bool send_message** (const message_t & *message*, int *dest*, int *tag*)

Blocking message send.

Parameters:

message - a message to send

dest - id of the workstation that receives the data

tag - additional information attached to the message, which is typically used to identify the type of the message

Returns:

true if the function succeeds, **false** otherwise

Use this function to send data to a workstation, the sending process is blocking in the sense that as soon as the function finishes the data which *buf* points to has already been copied to "lower level" and the original memory can be safely rewritten.

The function is connected to a buffering mechanism, it means that the messages are not sent instantly, but they are accumulated in a special buffer. There is one such buffer for each workstation. The data from a buffer are sent to the destination workstation as soon as one of the limits (set by functions [set_buf_msgs_cnt_limit\(\)](#), [set_buf_size_limit\(\)](#), [set_buf_time_limit\(\)](#)) is exceeded or [flush_buffer\(\)](#) function is called.

References `message_t::get_data()`, `message_t::get_written_size()`, and `send_message()`.

12.49.3.46 **bool send_message** (char * *buf*, int *size*, int *dest*, int *tag*)

Blocking message send.

Parameters:

buf - pointer to data that are to be sent

size - size of the data to be sent (in `sizeof(char)`)

dest - id of the workstation that receives the data

tag - additional information attached to the message, which is typically used to identify the type of the message

Returns:

true if the function succeeds, **false** otherwise

Use this function to send data to a workstation, the sending process is blocking in the sense that as soon as the function finishes the data which *buf* points to has already been copied to "lower level" and the original memory can be safely rewritten.

The function is connected to a buffering mechanism, it means that the messages are not sent instantly, but they are accumulated in a special buffer. There is one such buffer for each workstation. The data from a buffer are sent to the destination workstation as soon as one of the limits (set by functions [set_buf_msgs_cnt_limit\(\)](#), [set_buf_size_limit\(\)](#), [set_buf_time_limit\(\)](#)) is exceeded or [flush_buffer\(\)](#) function is called.

Referenced by [send_message\(\)](#).

12.49.3.47 bool send_urgent_message (const message_t & message, int dest, int tag)

Blocking urgent message send.

Similar to [send_message\(\)](#) function but of course there are differences. The function is not connected to buffering mechanism, so the messages are sent immediately. Different functions are used to receive messages sent by this function.

References [message_t::get_data\(\)](#), [message_t::get_written_size\(\)](#), and [send_urgent_message\(\)](#).

12.49.3.48 bool send_urgent_message (char * buf, int size, int dest, int tag)

Blocking urgent message send.

Similar to [send_message\(\)](#) function but of course there are differences. The function is not connected to buffering mechanism, so the messages are sent immediately. Different functions are used to receive messages sent by this function.

Referenced by [distributed_t::process_messages\(\)](#), and [send_urgent_message\(\)](#).

The documentation for this class was generated from the following files:

- [network.hh](#)
- [network.cc](#)

12.50 por_t Class Reference

Class for utilization of partial order reduction.

```
#include <por.hh>
```

Public Member Functions

- `int ample_set (state_t s, enabled_trans_container_t &enabled_trans, std::size_t &proc_gid)`
Generates all enabled transitions and proc_gid of ample set choosed by estimated heuristic.
- `int ample_set_succs (state_t s, succ_container_t &succs, std::size_t &proc_gid)`
Generates successors of ample set choosed by estimated heuristic.
- `int generate_ample_sets (state_t s, bool *ample_sets, enabled_trans_container_t &enabled_trans, std::size_t &le_proc_gid)`
Generates all enabled transitions and offers possible candidates to an ample set.
- `int generate_composed_ample_sets (state_t s, bool *ample_sets, enabled_trans_container_t **ample_trans, enabled_trans_container_t &all_enabled_trans)`
Generates all enabled transitions and offers possible candidates to an ample set composed of several processes.
- `void get_dep (bit_string_t *result)`
- `void get_pre (bit_string_t *result)`
Returns field of `bit_string_t` storing for all transition set of processes whose may enable this transition.
- `bit_string_t get_visibility ()`
Returns a `bit_string_t` structure (see `bit_string_t` class) containing information which transition is visible or not (`trans[i]==true` iff transition with `GID=i` is visible).
- `void init (explicit_system_t *S, list< expression_t * > *vis_list=NULL)`
*Initialization and static analysis of the system, **necessary** to call.*
- `por_t (void)`
A creator.
- `void set_choose_type (std::size_t type)`
Method for estimating of choosing specific ample set, parameter type is one of constants `POR_FIRST`, `POR_LAST`, `POR_SMALLEST` or `POR_FIND_ONLY`.
- `void set_dep (bit_string_t *new_dep)`

- void **set_pre** ([bit_string_t](#) *new_pre)
- void **set_visibility** ([bit_string_t](#) trans)
Method for changing of visibility relation (trans[i]==true iff transition with [GID](#)=i is visible).
- void **static_c3** ()
Adds to visible transitions so called sticky transitions in order to fulfill cycle condition (by static analysis).
- [~por_t](#) ()
A destructor (frees some used memory).

Public Attributes

- bool **count_approx_interruptions**
- unsigned * **dep_interrupted**
- [bit_string_t](#) * **full_dep**
- [bit_string_t](#) * **full_pre**
- [bit_string_t](#) **full_vis**
- unsigned * **pre_interrupted**
- unsigned * **vis_interrupted**

12.50.1 Detailed Description

Class for utilization of partial order reduction.

Class implements basic features necessary to using partial order reduction, i.e. static analysis finding dependency and visibility relations, or choosing appropriate candidates for ample-sets. **The cycle condition is not ensured by this class and it has to be treated externally.**

12.50.2 Member Function Documentation

12.50.2.1 **int ample_set** ([state_t](#) s, [enabled_trans_container_t](#) & *enabled_trans*, [std::size_t](#) & *proc_gid*)

Generates all enabled transitions and *proc_gid* of ample set choosed by estimated heuristic.

If *proc_gid*==[system_t::get_process_count\(\)](#), then the state must be fully expanded.

Parameters:

s = state for which we generate ample set

enabled_trans = container of all enabled transitions as generated by function [explicit_system_t::get_async_enabled_trans](#) (programmer hasn't generate them again, which would be time consumed)

proc_gid = [GID](#) of process whose (enabled) transitions can be taken as ample set. The process is choosed by estimated heuristic (see a creator of this class [por_t::por_t](#))

Returns:

bitwise OR of SUCC_NORMAL, SUCC_ERROR and SUCC_DEADLOCK according to [explicit_system_t::get_enabled_trans](#) (use functions [explicit_system_t::succs_normal\(\)](#), [explicit_system_t::succs_error\(\)](#) and [explicit_system_t::succs_deadlock\(\)](#) for testing)

References [generate_ample_sets\(\)](#), and [dve_system_t::get_process_count\(\)](#).

12.50.2.2 `int ample_set_succs (state_t s, succ_container_t & succs, std::size_t & proc_gid)`

Generates successors of ample set choosed by estimated heuristic.

Parameters:

s = state for which we generate ample set
succs = container of generated successors

Returns:

[GID](#) of process whose (enabled) transitions can be taken as ample set. The process is choosed by estimated heuristic (see a creator of this class [por_t::por_t](#)). If *proc_gid*==[system_t::get_process_count\(\)](#), then the state is fully expanded (and parameter `<it>succs</it>` contains all enabled successors).
 bitwise OR of SUCC_NORMAL, SUCC_ERROR and SUCC_DEADLOCK according to [explicit_system_t::get_enabled_trans](#) (use functions [explicit_system_t::succs_normal\(\)](#), [explicit_system_t::succs_error\(\)](#) and [explicit_system_t::succs_deadlock\(\)](#) for testing)

References [generate_ample_sets\(\)](#), [enabled_trans_container_t::get_begin\(\)](#), [enabled_trans_container_t::get_count\(\)](#), [dve_explicit_system_t::get_enabled_trans_succs\(\)](#), [dve_explicit_system_t::get_ith_succ\(\)](#), [dve_system_t::get_process_count\(\)](#), and [array_t::push_back\(\)](#).

12.50.2.3 `int generate_ample_sets (state_t s, bool * ample_sets, enabled_trans_container_t & enabled_trans, std::size_t & ample_proc_gid)`

Generates all enabled transitions and offers possible candidates to an ample set.

Parameters:

s = state for which we generate ample set

ample_sets = set of dimension equal to number of processes of the system. If *ample_sets[i]=true*, then the set of transitions of the i-th process is a candidate to ample set.

enabled_trans = container of all enabled transitions as generated by function *explicit_system_t::get_async_enabled_trans* (programmer don't have to generate them again, which would be time consumed)

Returns:

bitwise OR of SUCC_NORMAL, SUCC_ERROR and SUCC_DEADLOCK according to *explicit_system_t::get_enabled_trans* (use functions *explicit_system_t::succs_normal()*, *explicit_system_t::succs_error()* and *explicit_system_t::succs_deadlock()* for testing)

References *error_vector_t::clear()*, *error_vector_t::count()*, *gerr*, *bit_string_t::get_bit()*, *enabled_trans_container_t::get_count()*, *dve_explicit_system_t::get_enabled_trans()*, *enabled_trans_container_t::get_enabled_transition()*, *process_t::get_gid()*, *transition_t::get_gid()*, *dve_system_t::get_process_count()*, *dve_transition_t::get_process_gid()*, *dve_system_t::get_property_gid()*, *dve_explicit_system_t::get_receiving_trans()*, *dve_explicit_system_t::get_sending_or_normal_trans()*, *dve_explicit_system_t::get_state_of_process()*, *dve_system_t::get_transition()*, *system_t::get_with_property()*, *error_vector_t::perror()*, *array_of_abstract_t::size()*, and SUCC_ERROR.

Referenced by *ample_set()*, and *ample_set_succs()*.

12.50.2.4 int generate_composed_ample_sets (state_t s, bool * ample_sets, enabled_trans_container_t ** ample_trans, enabled_trans_container_t & all_enabled_trans)

Generates all enabled transitions and offers possible candidates to an ample set composed of several processes.

Extension of classic *ample_set* notion: ample set can composed of several processes here.

Parameters:

s = state for which we generate ample set

ample_sets = set of dimension equal to number of processes of the system. If *ample_set[i]=true*, then there exists an ample set A containing process i, where A is a proper subset of all enabled conditions.

ample_trans = array of *enabled_trans_container_t* of dimension equal to number of processes of the system. if *ample_set[i]=true*, then *ample_trans[i]* contains (enabled) transitions of an ample set which covers transitions of the i-th process.

Returns:

bitwise OR of SUCC_NORMAL, SUCC_ERROR and SUCC_DEADLOCK according to *explicit_system_t::get_enabled_trans* (use functions *explicit_system_t::succs_normal()*, *explicit_system_t::succs_error()* and *explicit_system_t::succs_deadlock()* for testing)

References `array_of_abstract_t::back()`, `enabled_trans_container_t::clear()`, `error_vector_t::clear()`, `error_vector_t::count()`, `array_of_abstract_t::extend()`, `gerr`, `bit_string_t::get_bit()`, `enabled_trans_container_t::get_count()`, `dve_explicit_system_t::get_enabled_trans()`, `enabled_trans_container_t::get_enabled_transition()`, `process_t::get_gid()`, `transition_t::get_gid()`, `dve_transition_t::get_process_gid()`, `dve_system_t::get_property_gid()`, `enabled_trans_container_t::get_property_succ_count()`, `dve_explicit_system_t::get_receiving_trans()`, `dve_explicit_system_t::get_sending_or_normal_trans()`, `dve_explicit_system_t::get_state_of_process()`, `dve_system_t::get_transition()`, `system_t::get_with_property()`, `error_vector_t::perror()`, `dve_explicit_system_t::print_state()`, `state_t::ptr`, `enabled_trans_container_t::set_next_begin()`, `enabled_trans_container_t::set_property_succ_count()`, `array_of_abstract_t::size()`, `state_t::size`, and `SUCC_ERROR`.

12.50.2.5 `void init (explicit_system_t * S, list< expression_t * > * vis_list = NULL)`

Initialization and static analysis of the system, **necessary** to call.

If the system is without property, visibility is computed from the list of expressions given in the argument `vis_list`. For the systems with the property the parameter is omitted.

References `bit_string_t::alloc_mem()`, `bit_string_t::clear()`, `bit_string_t::DBG_print()`, `bit_string_t::enable_bit()`, `bit_string_t::get_bit()`, `dve_transition_t::get_channel_gid()`, `dve_transition_t::get_effect()`, `dve_transition_t::get_effect_count()`, `transition_t::get_gid()`, `dve_system_t::get_global_variable_count()`, `dve_transition_t::get_guard()`, `transition_t::get_lid()`, `dve_symbol_table_t::get_process()`, `dve_system_t::get_process()`, `dve_system_t::get_process_count()`, `dve_transition_t::get_process_gid()`, `dve_system_t::get_property_process()`, `dve_transition_t::get_state1_gid()`, `dve_transition_t::get_state1_lid()`, `dve_transition_t::get_state2_gid()`, `dve_symbol_table_t::get_state_count()`, `dve_system_t::get_symbol_table()`, `dve_transition_t::get_sync_expr_list_item()`, `dve_transition_t::get_sync_expr_list_size()`, `dve_transition_t::get_sync_mode()`, `process_t::get_trans_count()`, `dve_system_t::get_trans_count()`, `process_t::get_transition()`, `dve_system_t::get_transition()`, `dve_symbol_table_t::get_variable()`, `dve_symbol_t::get_vector_size()`, `system_t::get_with_property()`, `dve_symbol_t::is_vector()`, `dve_expression_t::left()`, `dve_expression_t::right()`, `bit_string_t::set_bit()`, `SYNC_ASK`, `SYNC_ASK_BUFFER`, `SYNC_EXCLAIM`, `SYNC_EXCLAIM_BUFFER`, and `SYNC_NO_SYNC`.

The documentation for this class was generated from the following files:

- [por.hh](#)
- [por.cc](#)

12.51 `prob_and_property_trans_t` Struct Reference

```
#include <prob_explicit_system.hh>
```

Public Member Functions

- `bool operator!=` (const `prob_and_property_trans_t` &second)
An operator of inequality.
- `bool operator==` (const `prob_and_property_trans_t` &second)
An operator of equality.
- `prob_and_property_trans_t` (const `size_int_t` init_prob_trans_gid, const `size_int_t` init_property_trans_gid)
A constructor.
- `prob_and_property_trans_t` ()
A default constructor (set "uninitialized" values to both items).

Public Attributes

- `size_int_t prob_trans_gid`
GID of probabilistic transition
- `size_int_t property_trans_gid`

12.51.1 Detailed Description

Duple of transitions GIDs denoting probabilistic transition of the system multiplied with transitions of the property process

12.51.2 Member Data Documentation

12.51.2.1 `size_int_t property_trans_gid`

`GID` of transition of the property process

Referenced by `operator!=()`, and `operator==()`.

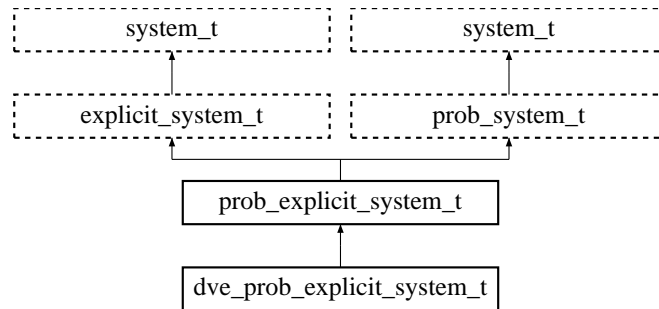
The documentation for this struct was generated from the following file:

- `prob_explicit_system.hh`

12.52 prob_explicit_system_t Class Reference

```
#include <prob_explicit_system.hh>
```

Inheritance diagram for prob_explicit_system_t:



Public Member Functions

- virtual int [get_succs](#) ([state_t](#) state, [prob_succ_container_t](#) &succs, [enabled_trans_container_t](#) &etc)=0
Creates probabilistic successors of state 'state'.
- virtual int [get_succs](#) ([state_t](#) state, [prob_succ_container_t](#) &succs)=0
Creates probabilistic successors of state 'state'.
- [prob_explicit_system_t](#) ([error_vector_t](#) &evect)
A constructor.

12.52.1 Detailed Description

Abstract interface of a class representing a state generator based on the model of [system](#) stored in [prob_system_t](#)

This class works like [explicit_system_t](#), but in addition to [explicit_system_t](#) it also provides methods:

- [get_succs](#)([state_t](#) state, [prob_succ_container_t](#) & succs, [enabled_trans_container_t](#) & etc)
- [get_succs](#)([state_t](#) state, [prob_succ_container_t](#) & succs), which

These methods return successors with their weight and [GID](#) of used probabilistic transition. All these informations are also possible to find out using information about enabled transitions (parameter *etc* of [get_succs\(\)](#)) and methods [prob_system_t::get_prob_trans_of_trans\(\)](#), [get_index_of_trans_in_prob_trans\(\)](#) and interface of [prob_transition_t](#), but parameter *etc* is voluntary and thus [get_succs](#)([state_t](#) state, [prob-](#)

`succ_container_t` & `succs`) provides a minimal interface to do a probabilistic state space generation.

12.52.2 Constructor & Destructor Documentation

12.52.2.1 `prob_explicit_system_t (error_vector_t & evec)` `[inline]`

A constructor.

Parameters:

evec = `error vector` used for reporting of error messages

12.52.3 Member Function Documentation

12.52.3.1 `virtual int get_succs (state_t state, prob_succ_container_t & succs, enabled_trans_container_t & etc)` `[pure virtual]`

Creates probabilistic successors of state '*state*'.

Creates probabilistic successors of state *state*. In addition to `get_succs(state_t state, prob_succ_container_t & succs)` this method also creates a piece of information about enabled transitions used for successor generation.

Together with methods `prob_system_t::get_prob_trans_of_trans()`, `prob_system_t::get_index_of_trans_in_prob_trans()` and methods of `prob_transition_t` it is possible to extract all additional information (and even more) that is stored in `prob_succ_container_t`.

Implemented in `dve_prob_explicit_system_t`.

12.52.3.2 `virtual int get_succs (state_t state, prob_succ_container_t & succs)` `[pure virtual]`

Creates probabilistic successors of state '*state*'.

Creates probabilistic successors of state *state* and saves them to successor container *succs* (see `prob_succ_container_t`).

Parameters:

state = state of the system

succs = successors container for storage of successors of *state*

Returns:

bitwise OR of `SUCC_NORMAL`, `SUCC_ERROR` and `SUCC_DEADLOCK` (use functions `succs_normal()`, `succs_error()` and `succs_deadlock()` for testing)

Implemented in `dve_prob_explicit_system_t`.

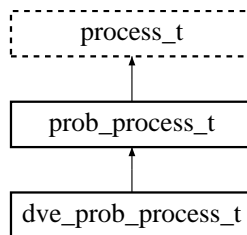
The documentation for this class was generated from the following file:

- [prob_explicit_system.hh](#)

12.53 prob_process_t Class Reference

```
#include <prob_process.hh>
```

Inheritance diagram for prob_process_t:



Public Member Functions

- virtual void [add_prob_transition](#) ([prob_transition_t](#) *const transition)=0
- [prob_system_t](#) * [get_parent_prob_system](#) () const
Returns the "parent" probabilistic system of this process.
- virtual [size_int_t](#) [get_prob_trans_count](#) () const =0
Returns a number of probabilistic transitions contained in a process.
- virtual [prob_transition_t](#) * [get_prob_transition](#) (const [size_int_t](#) prob_trans_lid)=0
*Returns a pointer to the probabilistic transition with **LID** prob_trans_lid.*
- virtual const [prob_transition_t](#) * [get_prob_transition](#) (const [size_int_t](#) prob_trans_lid) const =0
- [prob_process_t](#) ([prob_system_t](#) *const prob_system)
A constructor.
- [prob_process_t](#) ()
A constructor.
- virtual void [remove_prob_transition](#) (const [size_int_t](#) prob_trans_lid)=0
- virtual void [set_parent_prob_system](#) ([prob_system_t](#) &system)
Sets the "parent" probabilistic system of this process.
- virtual [~prob_process_t](#) ()
A destructor.

Protected Attributes

- [prob_system_t](#) * [parent_prob_system](#)

Protected data item storing a parent probabilistic system.

12.53.1 Detailed Description

Abstract interface of a class representing a probabilistic process of a probabilistic system

A "parent" probabilistic system is set in a constructor [prob_process_t\(prob_system_t * const system\)](#) or using the method [set_parent_system\(\)](#).

Note:

Developer is responsible for correct setting of corresponding "parent" [system](#) (but he/she rarely needs to create own processes - they are usually created automatically during reading of a source of the system from a file)

12.53.2 Constructor & Destructor Documentation

12.53.2.1 [prob_process_t](#) ([prob_system_t](#) *const *prob_system*) [inline]

A constructor.

Parameters:

system = "parent" [system](#) of this process

12.53.2.2 [virtual](#) ~[prob_process_t](#) () [inline, virtual]

A destructor.

A destructor

12.53.3 Member Function Documentation

12.53.3.1 [virtual void](#) [add_prob_transition](#) ([prob_transition_t](#) *const *transition*) [pure virtual]

Adds the probabilistic transition 'transition' to the process and sets its [LID](#)

Implemented in [dve_prob_process_t](#).

12.53.3.2 [virtual const](#) [prob_transition_t*](#) [get_prob_transition](#) (const [size_int_t](#) *prob_trans_lid*) const [pure virtual]

Returns a pointer to the constant probabilistic transition with [LID](#) *prob_trans_lid*

Implemented in [dve_prob_process_t](#).

Referenced by `prob_system_t::consolidate()`.

12.53.3.3 `virtual void remove_prob_transition (const size_int_t prob_trans_lid)` [pure virtual]

Removes the probabilistic transition with LID ‘`prob_trans_lid`’ from the process

Implemented in [dve_prob_process_t](#).

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

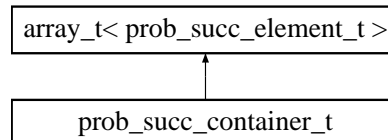
- [prob_process.hh](#)

12.54 prob_succ_container_t Class Reference

Class determined to store probabilistic successors of some state.

```
#include <prob_explicit_system.hh>
```

Inheritance diagram for prob_succ_container_t::



Public Member Functions

- [prob_succ_container_t](#) (const [explicit_system_t](#) &system)
- [prob_succ_container_t](#) ()
- [~prob_succ_container_t](#) ()

A destructor.

12.54.1 Detailed Description

Class determined to store probabilistic successors of some state.

It is the descendant of [array_t<prob_succ_element_t>](#) and it differs only in a constructor. Its constructor has a parameter of type [explicit_system_t](#), which is used to set the sufficient size of this container.

The main reason for this class is a better efficiency.

12.54.2 Constructor & Destructor Documentation

12.54.2.1 [prob_succ_container_t](#) () [inline]

A constructor (only calls a constructor of [array_t<state_t>](#) with parameters 4096 (pre-allocation) and 16 (allocation step).

12.54.2.2 [prob_succ_container_t](#) (const [explicit_system_t](#) & system) [inline]

A constructor (needs only 'system' to guess the preallocation needed for lists of successors).

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

- [prob_explicit_system.hh](#)

12.55 `prob_succ_element_t` Struct Reference

Single element to store in [prob_succ_container_t](#).

```
#include <prob_explicit_system.hh>
```

Public Member Functions

- **`prob_succ_element_t`** ([state_t](#) init_state, const `ulong_int_t` init_weight, const `ulong_int_t` init_sum, const [prob_and_property_trans_t](#) init_prob_and_property_trans)

Public Attributes

- [prob_and_property_trans_t](#) `prob_and_property_trans`
`GID` of probabilistic transition and `GID` of property transition used for generation of a successor
- [state_t](#) `state`
A state (successor).
- `ulong_int_t` `sum`
Sum of weights of transitions contained in probabilistic transition with `GID` 'prob_and_property_trans.prob_trans_gid'.
- `ulong_int_t` `weight`
Weight of a transition used for generation of a successor.

12.55.1 Detailed Description

Single element to store in [prob_succ_container_t](#).

The structure consists of a state (successor in a state generation), a weight of a transition and a used probabilistic transition.

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

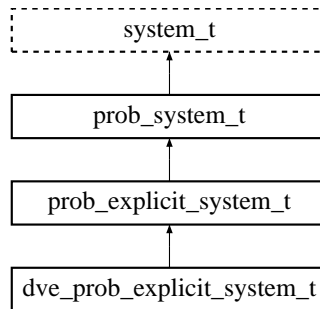
- [prob_explicit_system.hh](#)

12.56 prob_system_t Class Reference

Abstract interface of a class representing a model of a system.

```
#include <prob_system.hh>
```

Inheritance diagram for prob_system_t::



Public Member Functions

- `size_int_t` [get_index_of_trans_in_prob_trans](#) (const `size_int_t` trans_gid) const
- virtual `size_int_t` [get_prob_trans_count](#) () const

Returns a count of probabilistic transitions.

- `size_int_t` [get_prob_trans_of_trans](#) (const `size_int_t` trans_gid) const
- virtual const `prob_transition_t` * [get_prob_transition](#) (size_int_t gid) const

Returns the pointer to the constant probabilistic transition given by [GID](#).

- virtual `prob_transition_t` * [get_prob_transition](#) (size_int_t gid)

Returns the pointer to the probabilistic transition given by [GID](#).

- `prob_system_t` (error_vector_t &evect=[gerr](#))

A constructor.

- virtual `~prob_system_t` ()

A destructor.

Protected Member Functions

- void [consolidate](#) ()

Protected method called after reading of input.

12.56.1 Detailed Description

Abstract interface of a class representing a model of a system.

This class provides an interface to the model of system. Depending on abilities (see [get_abilities\(\)](#)) you can more or less deeply analyse the model.

Furthermore you can generate states of the system using child class `explicit_system_t` (and its children)

12.56.2 Constructor & Destructor Documentation

12.56.2.1 `prob_system_t (error_vector_t & evec = gerr) [inline]`

A constructor.

Parameters:

evec = the **error vector**, that will be used by created instance of [system_t](#)

12.56.3 Member Function Documentation

12.56.3.1 `size_int_t get_index_of_trans_in_prob_trans (const size_int_t trans_gid) const [inline]`

Returns the index of transition with [GID](#) '*trans_gid*' in a probabilistic transition

Parameters:

trans_gid = [GID](#) of transition

Returns:

index of transition with [GID](#) *trans_gid* in a probabilistic transition, which contains it. If there is no such a probabilistic transition, it returns `NO_ID`

Referenced by `dve_prob_explicit_system_t::get_succs()`.

12.56.3.2 `size_int_t get_prob_trans_of_trans (const size_int_t trans_gid) const [inline]`

Returns [GID](#) of probabilistic transition containing transition with [GID](#) '*trans_gid*' or returns `NO_ID`

Parameters:

trans_gid = [GID](#) of transition

Returns:

[GID](#) of probabilistic transition containing transition with [GID](#) *trans_gid*. If there is no such a probabilistic transition, it returns `NO_ID`

Referenced by `dve_prob_explicit_system_t::get_succs()`.

The documentation for this class was generated from the following files:

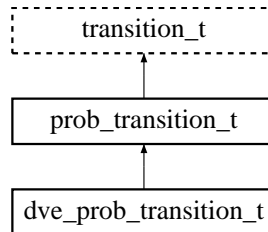
- [prob_system.hh](#)
- `prob_system.cc`

12.57 prob_transition_t Class Reference

Abstract interface of a class representing a probabilistic transition.

```
#include <prob_transition.hh>
```

Inheritance diagram for prob_transition_t:



Public Member Functions

- `size_int_t get_trans_count () const`
- `const transition_t * get_transition (const size_int_t i) const`
- `transition_t * get_transition (const size_int_t i)`
Returns a pointer to the 'i'-th transition of the probabilistic transition.
- `ulong_int_t get_weight (const size_int_t i) const`
Returns a weight of 'i'-th transition.
- `ulong_int_t get_weight_sum () const`
- `prob_transition_t (prob_system_t *const system)`
A constructor.
- `prob_transition_t ()`
A constructor.
- `void set_trans_count (const size_int_t size)`
Sets a number of transitions contained in the probabilistic transition.
- `void set_transition_and_weight (const size_int_t i, transition_t *const p_trans, const ulong_int_t weight)`
Sets 'i'-th transition and its weight.

Protected Member Functions

- `void initialize ()`
A protected initializer.

Protected Attributes

- `array_t< ulong_int_t > prob_weights`
- `array_t< transition_t * > trans`
- `ulong_int_t weight_sum`

12.57.1 Detailed Description

Abstract interface of a class representing a probabilistic transition.

This class represents a probabilistic transition. It is a part of the model of a probabilistic system.

This class is derived from `transition_t`, although, in fact, it is really different from the ordinary system transition:

- it consists of several ordinary transitions
- each ordinary transition contained in a probabilistic transition has a weight (probability of the *i*-th ordinary transition can be computed as `prob_trans.get_weight(i)/prob_trans.get_weight_sum()`)

12.57.2 Member Function Documentation

12.57.2.1 `size_int_t get_trans_count () const` [inline]

Returns a number of transition, which the probabilistic transition consists of

Referenced by `prob_system_t::consolidate()`, and `dve_prob_process_t::write()`.

12.57.2.2 `const transition_t* get_transition (const size_int_t i) const` [inline]

Returns a pointer to the constant '*i*'-th transition of the probabilistic transition

12.57.2.3 `ulong_int_t get_weight_sum () const` [inline]

Returns a sum of weights of all transitions contained in the probabilistic transition

Note:

Usage of `get_weight_sum()` is faster than computation of the sum explicitly using repeated call of `get_weight()`. `get_weight_sum()` returns precomputed value.

Referenced by `dve_prob_explicit_system_t::get_succs()`.

12.57.2.4 `void set_trans_count (const size_int_t size)`

Sets a number of transitions contained in the probabilistic transition.

Note:

Existing transitions are not lost by setting new count of transitions. Do not forget to delete all transitions, which you do not plan to use.

References `array_t::resize()`.

Referenced by `dve_parser_t::prob_trans_create()`.

12.57.2.5 `void set_transition_and_weight (const size_int_t i, transition_t *const p_trans, const ulong_int_t weight)`

Sets 'i'-th transition and its weight.

Warning:

If there has already been set 'i'-th transition before, this transition is not deleted automatically. Thus, do not forget to call `delete (pr_trans.get_transition(i))` before rewriting it by the pointer to another transition.

Referenced by `dve_parser_t::prob_trans_create()`.

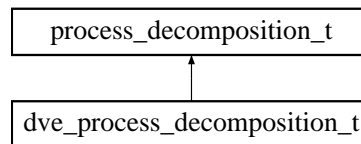
The documentation for this class was generated from the following files:

- `prob_transition.hh`
- `prob_transition.cc`

12.58 process_decomposition_t Class Reference

```
#include <process_decomposition.hh>
```

Inheritance diagram for process_decomposition_t:



Public Member Functions

- virtual int [get_process_scc_id](#) (state_t &)=0
- virtual int [get_process_scc_type](#) (state_t &)=0
- virtual int [get_scc_count](#) ()=0
- virtual int [get_scc_type](#) (int)=0
- virtual bool [is_weak](#) ()=0
- virtual void [parse_process](#) (std::size_t)=0

12.58.1 Detailed Description

This class is used to decompose a graph of a single process specified in a dve file into SCCs. The decomposition is not accesible directly but with member functions returning for a given state SCC id and type.

12.58.2 Member Function Documentation

12.58.2.1 virtual int [get_process_scc_id](#) (state_t &) [pure virtual]

Returns id of an SCC that the given local state of the process belongs to.

Implemented in [dve_process_decomposition_t](#).

12.58.2.2 virtual int [get_process_scc_type](#) (state_t &) [pure virtual]

Returns type of an SCC that the given local state belongs to. Returned values: 0 means nonaccepting component, 1 means partially accepting component, and 2 means fully accepting component.

Implemented in [dve_process_decomposition_t](#).

12.58.2.3 virtual int [get_scc_count](#) () [pure virtual]

Returns the number of SCCs in the decomposition.

Implemented in [dve_process_decomposition_t](#).

12.58.2.4 virtual int get_scc_type (int) [pure virtual]

Returns type of the given SCC, where 0 means nonaccepting component, 1 means partially accepting component, and 2 means fully accepting component.

Implemented in [dve_process_decomposition_t](#).

12.58.2.5 virtual bool is_weak () [pure virtual]

Returns whether the process has a weak graph.

Implemented in [dve_process_decomposition_t](#).

12.58.2.6 virtual void parse_process (std::size_t) [pure virtual]

Performs the decomposition of a process with a given (global) id.

Implemented in [dve_process_decomposition_t](#).

Referenced by `dve_explicit_system_t::read()`.

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

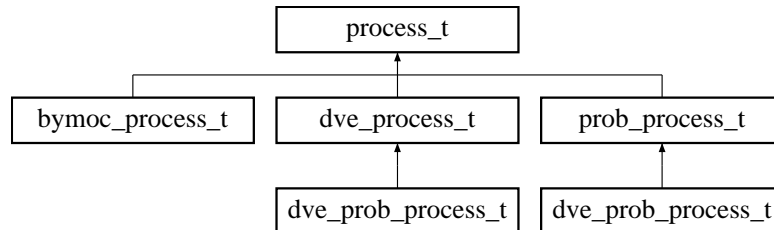
- [process_decomposition.hh](#)

12.59 process_t Class Reference

Abstract interface of a class representing a process of a [system](#).

```
#include <process.hh>
```

Inheritance diagram for process_t::



Public Member Functions

- bool [can_be_modified](#) () const
Tells, whether processes can be modified.
- bool [can_read](#) () const
Tells, whether process can be read from a string representation.
- bool [can_transitions](#) () const
Tells, whether process can work with transitions and contains them.
- size_int_t [get_gid](#) () const
Returns a [GID](#) of this process.
- [system_t](#) * [get_parent_system](#) () const
Returns a "parent" [system](#) of this process.
- [process_t](#) ([system_t](#) *const system)
A constructor.
- [process_t](#) ()
A constructor.
- virtual void [set_parent_system](#) ([system_t](#) &system)
Sets a "parent" [system](#) of this process.
- virtual [~process_t](#) ()
A destructor.

Methods modifying a process

These methods are implemented only if `can_transitions()` returns true.

- virtual void `add_transition` (`transition_t *const transition`)=0
Adds new transition to the process.
- virtual void `remove_transition` (`const size_int_t transition_gid`)=0
Removes a transition from the process.
- virtual void `set_gid` (`const size_int_t new_gid`)
Sets a `GID` of this process.

Methods for reading a process from a string representation

These methods are implemented only if `can_read()` returns true.

- virtual int `from_string` (`std::string &proc_str`)=0
Reads in the process from the string representation.
- virtual int `read` (`std::istream &istr`)=0
Reads in the process from the string representation in stream.

Methods working with transitions of a process

These methods are implemented only if `can_transitions()` returns true.

- virtual `size_int_t` `get_trans_count` () const =0
Returns a count of transitions of this process.
- virtual const `transition_t *` `get_transition` (`const size_int_t lid`) const =0
Returns a pointer to the constant transition with `LID` 'lid'.
- virtual `transition_t *` `get_transition` (`const size_int_t lid`)=0
Returns a pointer to the transition with `LID` 'lid'.

Obligatory part of abstract interface

- virtual `std::string` `to_string` () const =0
Returns a string representation of the process.
- virtual void `write` (`std::ostream &ostr`) const =0
Writes a string representation of the process to a stream.

Static Protected Member Functions

- static `error_vector_t &` `get_error_vector` ()
- static void `set_error_vector` (`error_vector_t &evect`)

Protected Attributes

- `size_int_t gid`
Protected data item storing [GID](#) of this process.
- `system_t * parent_system`
Protected data item storing a parent [system](#).

Static Protected Attributes

- static `error_vector_t * pproc_terr = &gerr`

12.59.1 Detailed Description

Abstract interface of a class representing a process of a [system](#).

This class represents a process of a [system](#). Its "parent" [system](#) is given in a constructor `process_t(system_t * const system)` or a method `set_parent_system()`.

Note:

Developer is responsible for correct setting of corresponding "parent" [system](#) (but he rarely needs to create own processes - they are usually created automatically during reading of a source of the system from a file)

12.59.2 Constructor & Destructor Documentation

12.59.2.1 `process_t (system_t *const system)` [inline]

A constructor.

Parameters:

system = "parent" [system](#) of this process

12.59.2.2 `virtual ~process_t ()` [inline, virtual]

A destructor.

A destructor

12.59.3 Member Function Documentation

12.59.3.1 `virtual void add_transition (transition_t *const transition)` [pure virtual]

Adds new transition to the process.

Parameters:

transition = pointer to the transition to add

This method modifies the added transition because it has to set [transition LID](#) and [Partial ID](#).

Implemented in [bymoc_process_t](#), and [dve_process_t](#).

12.59.3.2 virtual int from_string (std::string & proc_str) [pure virtual]

Reads in the process from the string representation.

Parameters:

proc_str = string to read from

Returns:

... 0 iff no error occurs, non-zero value in a case of error during a reading.

Implemented in [bymoc_process_t](#), [dve_prob_process_t](#), and [dve_process_t](#).

12.59.3.3 static error_vector_t& get_error_vector () [inline, static, protected]

Protected static method returning , which will be used in case of any error message

12.59.3.4 virtual size_int_t get_trans_count () const [pure virtual]

Returns a count of transitions of this process.

Then [LID](#) of transitions can be from 0 to ([get_trans_count\(\)](#)-1)

Note:

The sum of returned values obtained by calling this method for all processes of the system is equal to the value obtained by method [system_t::get_trans_count\(\)](#)

Implemented in [bymoc_process_t](#), and [dve_process_t](#).

Referenced by [dve_parser_t::check_restrictions_put_on_property\(\)](#), and [por_t::init\(\)](#).

12.59.3.5 virtual const transition_t* get_transition (const size_int_t lid) const [pure virtual]

Returns a pointer to the constant transition with [LID](#) 'lid'.

[LID](#) of transitions can be from 0 to ([get_trans_count\(\)](#)-1)

Implemented in [bymoc_process_t](#), and [dve_process_t](#).

12.59.3.6 `virtual transition_t* get_transition (const size_int_t lid)` [pure virtual]

Returns a pointer to the transition with LID 'lid'.

LID of transitions can be from 0 to (`get_trans_count()`-1)

Implemented in `bymoc_process_t`, and `dve_process_t`.

Referenced by `dve_parser_t::check_restrictions_put_on_property()`, and `por_t::init()`.

12.59.3.7 `virtual int read (std::istream & istr)` [pure virtual]

Reads in the process from the string representation in stream.

Parameters:

istr = input stream containing source of the process

Returns:

... 0 iff no error occurs, non-zero value in a case of error during a reading.

Implemented in `bymoc_process_t`, `dve_prob_process_t`, and `dve_process_t`.

12.59.3.8 `virtual void remove_transition (const size_int_t transition_gid)` [pure virtual]

Removes a transition from the process.

Parameters:

transition_gid = GID of removed transition

Implemented in `bymoc_process_t`, and `dve_process_t`.

12.59.3.9 `static void set_error_vector (error_vector_t & evecf)` [inline, static, protected]

Protected static method setting , which will be used in case of any error message

12.59.4 Member Data Documentation

12.59.4.1 `error_vector_t * pproc_terr = &gerr` [static, protected]

Static protected data item storing an used in case of any error messages

Referenced by `dve_prob_process_t::add_prob_transition()`.

The documentation for this class was generated from the following files:

- [process.hh](#)
- process.cc

12.60 psh Struct Reference

Structure determined for causing storing/printing error messages.

```
#include <error.hh>
```

Public Member Functions

- [psh](#) ([ERR_type_t](#) tt=[ERR_UNKNOWN_TYPE](#), [ERR_id_t](#) i=[ERR_UNKNOWN_ID](#))

Public Attributes

- [ERR_id_t](#) c
- [ERR_type_t](#) t

12.60.1 Detailed Description

Structure determined for causing storing/printing error messages.

This structure should be used as follows:

```
terr << "My warning mess" << "age" << psh(3,13565)
```

where "My warning message" is an example of error message sent into [error_vector_t](#) 'terr', 3 is an example of so called error type (see [ERR_type_t](#)) and 13565 is an example of [error ID](#).

Sending [psh\(\)](#) to an instance of [error_vector_t](#) through an operator '<< ' causes the end of assigning of a message. Message is then stored at the end of a list of error messages. Then 'warning handling callback' is called (callback that can be set by [error_vector_t::set_push_callback\(\)](#)).

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

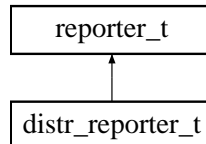
- [error.hh](#)

12.61 reporter_t Class Reference

Class for measuring and reporting.

```
#include <reporter.hh>
```

Inheritance diagram for reporter_t::



Public Member Functions

- double [get_time](#) ()
- void [print](#) ()
- void [print](#) (std::ostream &out)
- void [set_alg_name](#) (std::string s)
- void [set_file_name](#) (std::string s)
- void [set_info](#) (std::string s, double a, const std::string &long_name)
- void [set_info](#) (std::string s, double a)
- void [set_obligatory_keys](#) (std::string _alg_name, std::string _file_name, std::string _problem, size_int_t _states_stored, size_int_t _succs_calls)
- void [set_problem](#) (std::string s)
- void [set_states_stored](#) (size_int_t stored)
- void [set_succs_calls](#) (size_int_t calls)
- void [start_timer](#) ()
- void [stop_timer](#) ()

Protected Attributes

- std::string **alg_name**
- std::string **file_name**
- double **init_time**
- std::string **problem**
- std::map< std::string, double > **specific_info**
- std::map< std::string, std::string > **specific_long_name**
- size_int_t **states_stored**
- size_int_t **succs_calls**
- double **time**
- [timeinfo_t](#) **time_info**
- [vminfo_t](#) **vm_info**

12.61.1 Detailed Description

Class for measuring and reporting.

Class `reporter_t` is used to measure and report (in standard format) statistics about the computation of an algorithm.

Todo

Currently it prints a memory consumption only in the end of the run of the program. It should print a maximum during a run of the program

12.61.2 Member Function Documentation

12.61.2.1 `double get_time ()` [inline]

Returns the time (to be used after `stop_time`); useful e.g. for calculating speed

12.61.2.2 `void print ()`

Prints the report into the file with 'standard name' (at the moment `file_name.report`).

12.61.2.3 `void print (std::ostream & out)`

Prints the report into the given ostream.

12.61.2.4 `void set_alg_name (std::string s)` [inline]

Sets name of the algorithm.

12.61.2.5 `void set_file_name (std::string s)` [inline]

Sets name of the file which is the input of the algorithm.

12.61.2.6 `void set_info (std::string s, double a)` [inline]

Sets any specific information which we want to report (e.g., `set_info("states", states_num)`);

12.61.2.7 `void set_obligatory_keys (std::string _alg_name, std::string _file_name, std::string _problem, size_int_t _states_stored, size_int_t _succs_calls)` [inline]

Sets all obligatory keys (algorithm name, input file name, problem, number of states and number of calling successor function)

12.61.2.8 void set_problem (std::string *s*) [inline]

Sets the description of the problem that the algorithm solves (e.g., 'reachability x==3', 'ltl GF hungry == 0')

12.61.2.9 void set_states_stored (size_int_t *stored*) [inline]

Sets the number of stored states

12.61.2.10 void set_succs_calls (size_int_t *calls*) [inline]

Sets the number of calling get_succs() function

12.61.2.11 void start_timer () [inline]

Starts the timer which measure the (real) time taken by computation. Convention: This should be called after inicializations.

12.61.2.12 void stop_timer () [inline]

Stops the timer.

The documentation for this class was generated from the following files:

- [reporter.hh](#)
- reporter.cc

12.62 state_ref_t Class Reference

State reference class.

```
#include <explicit_storage.hh>
```

Public Member Functions

- void [invalidate](#) ()
- bool [is_valid](#) ()
- std::string [to_string](#) ()

Public Attributes

- size_t [hres](#)
- size_t [id](#)

12.62.1 Detailed Description

State reference class.

This class is a constant-sized short representation of state stored in an instance of [explicit_storage_t](#).

[explicit_storage_t](#) guaranties that this reference is a unique identifier of a state for each instance of [explicit_storage_t](#).

Operators ==, !=, <, <= and > and >= are defined for this class.

References are also useful in distributed environment in a connection with the identifier of computer that keeps the referenced state. It suffices to send a reference and computer ID instead of relatively long explicit representation of a state.

12.62.2 Member Function Documentation

12.62.2.1 void invalidate ()

Invalidates the reference, i.e. make it hold an invalid value.

12.62.2.2 bool is_valid ()

Tests whether the reference holds a valid value or not.

The documentation for this class was generated from the following files:

- [explicit_storage.hh](#)
- [explicit_storage.cc](#)

12.63 state_t Struct Reference

Structure representing the state of the system.

```
#include <state.hh>
```

Public Member Functions

- [state_t\(\)](#)
A constructor (only sets ptr to 0).

Public Attributes

- [char * ptr](#)
Pointer to the piece of memory, where the state is stored in.
- [std::size_t size](#)
Variable that stores the size of the state.

12.63.1 Detailed Description

Structure representing the state of the system.

There are many functions that can work with states. The list and the description of them can be found in a manual page of file [state.hh](#)

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

- [state.hh](#)
- [state.cc](#)

12.64 `static_info_t` Class Template Reference

Class used to collect data during synchronization.

```
#include <distributed.hh>
```

Inherits `abstract_info_t`.

Public Member Functions

- virtual void `get_data_ptr` (char *&ptr)
- virtual void `get_data_size` (int &size)

12.64.1 Detailed Description

```
template<class static_data_t> class static_info_t< static_data_t >
```

Class used to collect data during synchronization.

Instances of this class are passed to [distributed_t::synchronized\(abstract_info_t &info\)](#) function.

The *static_data_t* type should be struct with simple types in it (no dynamic arrays, strings, etc.) - the struct holds the collected data.

The word "static" means that each workstation cannot change the data being collected during the synchronization process. For more information see [distributed_t::synchronized\(abstract_info_t &info\)](#) function.

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

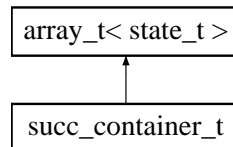
- [distributed.hh](#)

12.65 succ_container_t Class Reference

Class determined to store successors of some state.

```
#include <explicit_system.hh>
```

Inheritance diagram for succ_container_t:



Public Member Functions

- [succ_container_t](#) (const [explicit_system_t](#) &system)
- [succ_container_t](#) ()
- [~succ_container_t](#) ()

A destructor.

12.65.1 Detailed Description

Class determined to store successors of some state.

It is the descendant of [array_t<state_t>](#) and it differs only in a constructor. Its constructor has a parameter of type [explicit_system_t](#), which is used to set the sufficient size of this container.

The main reason for this class is a better efficiency.

12.65.2 Constructor & Destructor Documentation

12.65.2.1 succ_container_t () [inline]

A constructor (only calls a constructor of [array_t<state_t>](#) with parameters 4096 (pre-allocation) and 16 (allocation step).

12.65.2.2 succ_container_t (const explicit_system_t & system) [inline]

A constructor (needs only 'system' to guess the preallocation needed for lists of successors).

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

- [explicit_system.hh](#)

12.66 SYS_initial_values_t Union Reference

Internal type for storing initial values of various identifiers.

```
#include <dve_system.hh>
```

Public Attributes

- `all_values_t all_value`
for scalar [variable](#) (initial value)
- `all_values_t * all_values`
for vector [variable](#) (initial values)
- `size_int_t size_t_value`
procname symbols (initial state number)

12.66.1 Detailed Description

Internal type for storing initial values of various identifiers.

Type that can contain either initial value of scalar [variable](#) or field of initial values of vector [variable](#) or initial process state for state of procname symbols.

This is an internal type. Programmer of algorithms using DiVinE usually should not use this structure.

12.66.2 Member Data Documentation

12.66.2.1 size_int_t size_t_value

procname symbols (initial state number)

for state symbols (state number) and

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

- [dve_system.hh](#)

12.67 SYS_parameters_t Struct Reference

```
#include <dve_system.hh>
```

Public Member Functions

- [SYS_parameters_t](#) ([SYS_initial_values_t](#) *const initial_vals, size_int_t *const init_st, size_int_t *const initial_vals_counts, size_int_t *const st_lids)

A constructor.

Public Attributes

- size_int_t * [initial_states](#)
pointer to field initial_states
- [SYS_initial_values_t](#) * [initial_values](#)
pointer to field initial_values
- size_int_t * [initial_values_counts](#)
initial_values_counts
- size_int_t * [state_lids](#)

12.67.1 Detailed Description

Structure for passing parameters to 'eval_*' functions (parameters are pointers to some necessary private data of class [system_t](#))

12.67.2 Member Data Documentation

12.67.2.1 size_int_t* initial_values_counts

[initial_values_counts](#)

pointer to field

12.67.2.2 size_int_t* state_lids

pointer to field [state_lids](#)

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

- [dve_system.hh](#)

12.68 system_abilities_t Struct Reference

Structure storing abilities of [system](#) and its subordinate components.

```
#include <system_abilities.hh>
```

Public Member Functions

- [system_abilities_t \(\)](#)

A constructor.

Public Attributes

- bool [explicit_system_can_evaluate_expressions](#)
= true iff *explicit system* can evaluate expressions
- bool [explicit_system_can_system_transitions](#)
= true iff *explicit system* can work with system transitions and enabled transitions
- bool [process_can_be_modified](#)
= true iff process can be modified
- bool [process_can_read](#)
= true iff process can be read from a string representation
- bool [process_can_transitions](#)
= true iff process can work (and contains) transitions
- bool [system_can_be_modified](#)
= true iff *system* can be modified
- bool [system_can_decompose_property](#)
= true iff *system* can decompose property process graph
- bool [system_can_processes](#)
= true iff *system* can work with processes
- bool [system_can_property_process](#)
= true iff *system* can work with property process
- bool [system_can_transitions](#)
= true iff *system* can work with transitions
- bool [transition_can_be_modified](#)
= true iff transition can be modified

- bool `transition_can_read`
= *true iff transition can be read from a string representation*

12.68.1 Detailed Description

Structure storing abilities of `system` and its subordinate components.

An instance of this method is returned by `system_t::get_abilities()`. Values stored in data items differ according to methods which are implemented in a given representation of a system (e. g. in a system created from DVE source or a system created from Promela bytecode).

This "system of abilities" has been created to preserve rich abstract interfaces for a price of testing of their abilities - but without a need of typecasting.

12.68.2 Constructor & Destructor Documentation

12.68.2.1 `system_abilities_t()` `[inline]`

A constructor.

Sets all abilities to false (`system` is defaultly as dumb as possible)

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

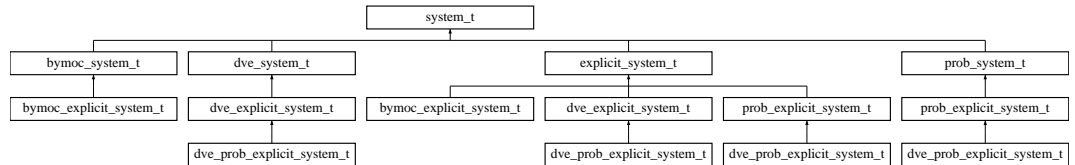
- `system_abilities.hh`

12.69 system_t Class Reference

Abstract interface of a class representing a model of a system.

```
#include <system.hh>
```

Inheritance diagram for system_t::



Public Member Functions

- bool [can_be_modified](#) () const
Tells, whether current [system](#) can be modified.
- bool [can_decompose_property](#) () const
Tells, whether current [system](#) is able to work with processes.
- bool [can_processes](#) () const
Tells, whether current [system](#) is able to work with processes.
- bool [can_property_process](#) () const
Tells, whether current [system](#) is able to work with property process.
- bool [can_transitions](#) () const
Tells, whether current [system](#) is able to work with transitions.
- const [system_abilities_t](#) & [get_abilities](#) () const
Returns a structure storing a list of abilities of current system.
- [system_abilities_t](#) & [get_abilities](#) ()
Returns a structure storing a list of abilities of current system.
- const [error_vector_t](#) & [get_error_vector](#) () const
Returns a constant reference to the [error_vector_t](#) inside of [system_t](#).
- [error_vector_t](#) & [get_error_vector](#) ()
Returns a reference to the [error_vector_t](#) inside of [system_t](#).
- bool [get_with_property](#) () const
Returns, whether a property process is specified or not.

- void [set_with_property](#) (const bool is_with)
Sets, wheter [system](#) is the [system](#) with a property process.
- [system_t](#) ([error_vector_t](#) &evect=[gerr](#))
A constructor.
- virtual [~system_t](#) ()
A destructor.

Methods modifying a system

These methods are implemented only if [can_be_modified\(\)](#) returns true.

- virtual void [add_process](#) ([process_t](#) *const process)=0
Method for adding of process to [system](#).
- virtual void [remove_process](#) (const size_int_t process_id)=0
Method for removing of process from [system](#).

Obligatory part of abstact interface

These methods have to implemented in each implementation of [system_t](#)

- virtual slong_int_t [from_string](#) (const std::string str)=0
Method for reading a model of a [system](#) from a source stored in a string.
- virtual property_type_t [get_property_type](#) ()=0
- virtual slong_int_t [read](#) (const char *const filename)=0
Method for reading a model of a [system](#) from a file.
- virtual slong_int_t [read](#) (std::istream &ins=std::cin)=0
Method for reading a model of a [system](#) from a stream.
- virtual std::string [to_string](#) ()=0
- virtual void [write](#) (std::ostream &outs=std::cout)=0
- virtual bool [write](#) (const char *const filename)=0
Method for writing currently stored model of a [system](#) to a file.

Methods working with processes

These methods are implemented only if [can_processes\(\)](#) returns true.

- virtual const [process_t](#) * [get_process](#) (const size_int_t gid) const =0
Returns a constant instance of a process with [process GID](#) 'gid'.
- virtual [process_t](#) * [get_process](#) (const size_int_t gid)=0
Returns a process with [process GID](#) 'gid'.
- virtual size_int_t [get_process_count](#) () const =0

Returns a count of processes in the [system](#).

Methods to check the SCCs of property process graph

- virtual [process_decomposition_t](#) * [get_property_decomposition](#) ()
Returns property decomposition, or 0, if subsystem is not available.

Methods working with property process

These methods are implemented only if [can_property_process\(\)](#) returns true.

- virtual [size_int_t](#) [get_property_gid](#) () const =0
Returns [GID](#) of property process.
- virtual const [process_t](#) * [get_property_process](#) () const =0
Returns a pointer to the constant instance of property process.
- virtual [process_t](#) * [get_property_process](#) ()=0
Returns a pointer to the property process.
- virtual void [set_property_gid](#) (const [size_int_t](#) gid)=0
Sets, which process is a property process.

Methods working with transitions

These methods are implemented only if [can_transitions\(\)](#) returns true.

- virtual [size_int_t](#) [get_trans_count](#) () const =0
Returns the count of all transitions in the [system](#).
- virtual const [transition_t](#) * [get_transition](#) ([size_int_t](#) gid) const =0
Returns a constant instance of transition with [transition GID](#) = 'gid'.
- virtual [transition_t](#) * [get_transition](#) ([size_int_t](#) gid)=0
Returns a transition with [transition GID](#) = 'gid'.

Static Public Attributes

- static const [ERR_id_t](#) [ERR_FILE_NOT_OPEN](#) = 65134
- static const [ERR_type_t](#) [ERR_TYPE_SYSTEM](#) = 100

Protected Member Functions

- void [copy_private_part](#) (const [system_t](#) &second)

Protected Attributes

- `system_abilities_t` `abilities`
- `error_vector_t` & `terr`
- `bool` `with_property`

12.69.1 Detailed Description

Abstract interface of a class representing a model of a system.

This class provides an interface to the model of system. Depending on abilities (see `get_abilities()`) you can more or less deeply analyse the model.

Furthermore you can generate states of the system using child class `explicit_system_t` (and its children)

12.69.2 Constructor & Destructor Documentation

12.69.2.1 `system_t (error_vector_t & evec = gerr)` `[inline]`

A constructor.

Parameters:

evec = the **error vector**, that will be used by created instance of `system_t`

12.69.3 Member Function Documentation

12.69.3.1 `virtual void add_process (process_t *const process)` `[pure virtual]`

Method for adding of process to `system`.

This method is implemented only if `can_be_modified()` returns true.

Implemented in `bymoc_system_t`, and `dve_system_t`.

12.69.3.2 `bool can_be_modified () const` `[inline]`

Tells, whether current `system` can be modified.

Generally `thr system` can be modified using `add_process()` and `remove_process()` together with methods modifying processes

12.69.3.3 `virtual slong_int_t from_string (const std::string str)` `[pure virtual]`

Method for reading a model of a `system` from a source stored in a string.

Parameters:

str = string to read

Returns:

... 0 iff no error occurs, non-zero value in a case of error during a reading.

Implemented in [bymoc_system_t](#), and [dve_system_t](#).

12.69.3.4 const system_abilities_t& get_abilities () const [inline]

Returns a structure storing a list of abilities of current system.

It also stores the list of abilities of processes and transitions, which are a part of a [system](#) and abilities of [explicit system](#)

12.69.3.5 system_abilities_t& get_abilities () [inline]

Returns a structure storing a list of abilities of current system.

It also stores the list of abilities of processes and transitions, which are a part of a [system](#) and abilities of [explicit system](#)

Referenced by [bymoc_explicit_system_t::bymoc_explicit_system_t\(\)](#), [transition_t::can_be_modified\(\)](#), [process_t::can_be_modified\(\)](#), [transition_t::can_read\(\)](#), [process_t::can_read\(\)](#), [process_t::can_transitions\(\)](#), [dve_explicit_system_t::dve_explicit_system_t\(\)](#), and [dve_system_t::dve_system_t\(\)](#).

12.69.3.6 virtual const process_t* get_process (const size_int_t gid) const
[pure virtual]

Returns a constant instance of a process with [process GID](#) 'gid'.

Parameters:

gid = [process GID](#) of a process (can be any from an interval 0..([get_process_count\(\)](#)-1))

Returns:

process (represented by a pointer to [process_t](#))

This method is implemented only if [can_processes_true_methods\(\)](#) returns true.

Implemented in [bymoc_system_t](#), and [dve_system_t](#).

12.69.3.7 virtual process_t* get_process (const size_int_t gid) [pure virtual]

Returns a process with [process GID](#) 'gid'.

Parameters:

gid = [process GID](#) of a process (can be any from an interval 0..([get_process_count\(\)](#)-1))

Returns:

process (represented by a pointer to [process_t](#))

This method is implemented only if [can_processes_true_methods\(\)](#) returns true.

Implemented in [bymoc_system_t](#), and [dve_system_t](#).

Referenced by [prob_system_t::consolidate\(\)](#).

12.69.3.8 virtual size_int_t get_process_count () const [pure virtual]

Returns a count of processes in the [system](#).

This method is implemented only if [can_processes_true_methods\(\)](#) returns true.

Implemented in [bymoc_system_t](#), and [dve_system_t](#).

Referenced by [prob_system_t::consolidate\(\)](#), and [enabled_trans_container_t::enabled_trans_container_t\(\)](#).

12.69.3.9 virtual size_int_t get_property_gid () const [pure virtual]

Returns [GID](#) of property process.

This method is implemented only if [can_property_process\(\)](#) returns true.

Implemented in [bymoc_system_t](#), and [dve_system_t](#).

12.69.3.10 virtual const process_t* get_property_process () const [pure virtual]

Returns a pointer to the constant instance of property process.

This method is implemented only if [can_property_process\(\)](#) returns true.

Implemented in [bymoc_system_t](#), and [dve_system_t](#).

12.69.3.11 virtual process_t* get_property_process () [pure virtual]

Returns a pointer to the property process.

This method is implemented only if [can_property_process\(\)](#) returns true.

Implemented in [bymoc_system_t](#), and [dve_system_t](#).

12.69.3.12 virtual property_type_t get_property_type () [pure virtual]

Method for identifying type of the accepting condition of the property process. Possible types are NONE, BUCHI, GENBUCHI, MULLER, RABIN, STREETT.

Implemented in [bymoc_explicit_system_t](#), [bymoc_system_t](#), and [dve_system_t](#).

12.69.3.13 virtual size_int_t get_trans_count () const [pure virtual]

Returns the count of all transitions in the [system](#).

This method is implemented only if [can_transitions\(\)](#) returns true.

Note:

Returned value is the sum of returned values obtained by calling [process_t::get_trans_count\(\)](#) for all processes of the system

Implemented in [bymoc_system_t](#), and [dve_system_t](#).

Referenced by [prob_system_t::consolidate\(\)](#).

12.69.3.14 virtual const transition_t* get_transition (size_int_t gid) const [pure virtual]

Returns a constant instance of transition with [transition GID](#) = 'gid'.

This method is implemented only if [can_transitions\(\)](#) returns true.

Implemented in [bymoc_system_t](#), and [dve_system_t](#).

12.69.3.15 virtual transition_t* get_transition (size_int_t gid) [pure virtual]

Returns a transition with [transition GID](#) = 'gid'.

This method is implemented only if [can_transitions\(\)](#) returns true.

Implemented in [bymoc_system_t](#), and [dve_system_t](#).

12.69.3.16 virtual slong_int_t read (const char *const filename) [pure virtual]

Method for reading a model of a [system](#) from a file.

Parameters:

filename = path to a source of a model

Returns:

0 iff successfully opens the file and successfully parses it. Otherwise it returns non-zero value. If the error was caused by the impossibility to open a file, it returns

`system_t::ERR_FILE_NOT_OPEN`.

Implemented in [bymoc_system_t](#), [dve_explicit_system_t](#), [dve_prob_explicit_system_t](#), and [dve_system_t](#).

12.69.3.17 `virtual slong_int_t read (std::istream & ins = std::cin)` [pure virtual]

Method for reading a model of a [system](#) from a stream.

Parameters:

ins = input stream containing a source of a model

Returns:

... 0 iff no error occurs, non-zero value in a case of error during a reading.

Implemented in [bymoc_system_t](#), [dve_prob_explicit_system_t](#), and [dve_system_t](#).

12.69.3.18 `virtual void remove_process (const size_int_t process_id)` [pure virtual]

Method for removing of process from [system](#).

This method is implemented only if [can_be_modified\(\)](#) returns true.

Implemented in [bymoc_system_t](#), and [dve_system_t](#).

12.69.3.19 `virtual void set_property_gid (const size_int_t gid)` [pure virtual]

Sets, which process is a property process.

Parameters:

gid = [GID](#) of new property process

This method is implemented only if [can_property_process\(\)](#) returns true.

Implemented in [bymoc_system_t](#), and [dve_system_t](#).

12.69.3.20 `virtual std::string to_string ()` [pure virtual]

Method for writing source of actually stored model of a [system](#) to the string

Implemented in [bymoc_system_t](#), and [dve_system_t](#).

12.69.3.21 **virtual void write (std::ostream & *outs* = std::cout) [pure virtual]**

Method for writing currently stored model of a [system](#) to a stream. (parameter ‘outs’)

Implemented in [bymoc_system_t](#), and [dve_system_t](#).

12.69.3.22 **virtual bool write (const char *const *filename*) [pure virtual]**

Method for writing currently stored model of a [system](#) to a file.

Parameters:

filename = path to a file

Returns:

true iff successfully opens the file

Implemented in [bymoc_system_t](#), and [dve_system_t](#).

The documentation for this class was generated from the following files:

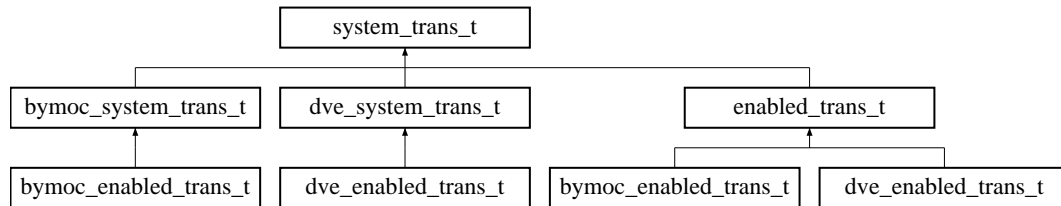
- [system.hh](#)
- [system.cc](#)

12.70 system_trans_t Class Reference

Class storing informations about one system transition.

```
#include <system_trans.hh>
```

Inheritance diagram for system_trans_t::



Public Member Functions

- virtual [system_trans_t](#) & [operator=](#) (const [system_trans_t](#) &second)=0

An assignment operator.

- [system_trans_t](#) ()

A constructor.

- virtual std::string [to_string](#) () const =0

Returns a string representation of enabled transition.

- virtual void [write](#) (std::ostream &ostr) const =0

- virtual [~system_trans_t](#) ()

A destructor.

Methods accessing transitions forming a system transition

These methods are implemented only if [system_t::can_transitions\(\)](#) in the system that has generated the instance of this class returns true.

- virtual size_int_t [get_count](#) () const =0
- virtual [transition_t](#) *const & [operator\[\]](#) (const int i) const =0

Returns 'i'-th transition forming this transition of the system.

- virtual [transition_t](#) *& [operator\[\]](#) (const int i)=0

Returns 'i'-th transition forming this transition of the system.

- virtual void [set_count](#) (const size_int_t new_count)=0

Sets a count of transitions of processes forming this enabled transition.

12.70.1 Detailed Description

Class storing informations about one system transition.

System transition consists of several transitions (of type [transition_t](#), iff [system](#) can work with transitions)

System transition represent the step of the entire [system](#) (compared to the [transition_t](#), which represents the step of a single process).

Developer will usually use [system_trans_t](#) as its child [enabled_trans_t](#) which adds the "erroneous" property to this class.

12.70.2 Member Function Documentation

12.70.2.1 `virtual size_int_t get_count () const` [pure virtual]

Returns a count of transitions of processes forming this enabled transition

Implemented in [bymoc_system_trans_t](#), and [dve_system_trans_t](#).

Referenced by [dve_explicit_system_t::get_property_trans\(\)](#), [dve_explicit_system_t::get_receiving_trans\(\)](#), and [dve_explicit_system_t::get_sync_enabled_trans_succ\(\)](#).

12.70.2.2 `virtual system_trans_t& operator= (const system_trans_t & second)` [pure virtual]

An assignment operator.

Makes a hard copy of system transition => takes a time O(second.size())

Implemented in [bymoc_system_trans_t](#), and [dve_system_trans_t](#).

12.70.2.3 `virtual void write (std::ostream & ostr) const` [pure virtual]

Prints a string representation of enabled transition to output stream 'ostr'

Implemented in [bymoc_system_trans_t](#), and [dve_system_trans_t](#).

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

- [system_trans.hh](#)

12.71 thr Struct Reference

Structure determined for causing storing/printing error messages.

```
#include <error.hh>
```

Public Member Functions

- `thr` ([ERR_type_t](#) tt=[ERR_UNKNOWN_TYPE](#), [ERR_id_t](#) i=[ERR_UNKNOWN_ID](#))

Public Attributes

- [ERR_id_t](#) c
- [ERR_type_t](#) t

12.71.1 Detailed Description

Structure determined for causing storing/printing error messages.

This structure should be used as follows:

```
terr << "My error mess" << "age" << thr(3,13565)
```

where "My error message" is an example of error message sent into [error_vector_t](#) 'terr', 3 is an example of so called error type (see [ERR_type_t](#)) and 13565 is an example of [error ID](#).

Sending `thr()` to an instance of [error_vector_t](#) through an operator '`<<`' causes the end of assigning of a message. Message is then stored at the end of a list of error messages. Then 'error handling callback' is called (callback that can be set by [error_vector_t::set_throw_callback\(\)](#)).

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

- [error.hh](#)

12.72 `timeinfo_t` Class Reference

Class for a time measuring.

```
#include <sysinfo.hh>
```

Public Member Functions

- double **gettime** ()
- void **print** ()
- void **reset** ()
- void **settimeout** (double)
- void **settimeout** (long, long)
- bool **testtimeout** (timeval &)
- bool **testtimeout** ()
- **timeinfo_t** (long, long)
- **timeinfo_t** (double)

Protected Member Functions

- void **print_time** (long, long)

Protected Attributes

- timeval **stored**
- long **tv_sec**
- long **tv_usec**

12.72.1 Detailed Description

Class for a time measuring.

The documentation for this class was generated from the following files:

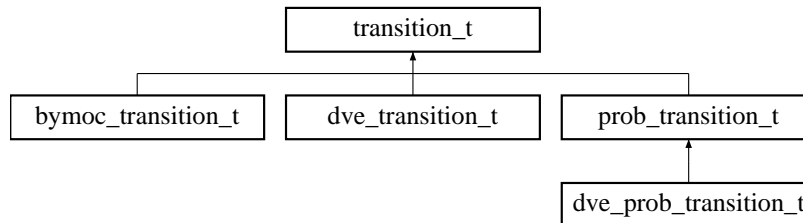
- sysinfo.hh
- sysinfo.cc

12.73 transition_t Class Reference

Abstract interface of a class representing a transition.

```
#include <transition.hh>
```

Inheritance diagram for transition_t::



Public Member Functions

- bool [can_be_modified](#) () const
Tells, whether this transition can be modified.
- bool [can_read](#) () const
Tells, whether this transition can be modified.
- size_int_t [get_gid](#) () const
Returns [transition GID](#) of this transition.
- size_int_t [get_lid](#) () const
Returns [transition LID](#) of this transition.
- [system_t](#) * [get_parent_system](#) () const
Returns a "parent" [system](#) of this transition.
- virtual void [set_parent_system](#) ([system_t](#) &system)
Sets a "parent" [system](#) of this transition.
- [transition_t](#) ([system_t](#) *const system)
A constructor.
- [transition_t](#) ()
A constructor.
- virtual [~transition_t](#) ()
A destructor.

Methods for reading a transition from a string representation

These methods are implemented only if *can_read()* returns true.

- virtual int [from_string](#) (std::string &trans_str, const size_int_t process_gid=NO_ID)=0
Reads in the transition from a string representation.
- virtual int [read](#) (std::istream &istr, size_int_t process_gid=NO_ID)=0
Reads in the transition from a string representation in stream.

Methods for modifying a transition

These methods are implemented only if *can_be_modified()* returns true.

- virtual void [set_gid](#) (const size_int_t gid)
Sets a [GID](#) of this transition.
- virtual void [set_lid](#) (const size_int_t id)
Sets a [LID](#) of this transition.

Obligatory part of abstract interface

These methods have to implemented in each implementation of *transition_t* (if transitions are supported by the system).

- virtual std::string [to_string](#) () const =0
Returns a string representation of the transition.
- virtual void [write](#) (std::ostream &ostr) const =0
Writes a string representation of the transition to stream.

Static Protected Member Functions

- static [error_vector_t](#) & [get_error_vector](#) ()
- static void [set_error_vector](#) ([error_vector_t](#) &evect)

Protected Attributes

- size_int_t [global_id](#)
Protected data item containing [GID](#).
- size_int_t [local_id](#)
Protected data item containing [LID](#).
- [system_t](#) * [parent_system](#)
Protected data item containing "parent" [system](#).

Static Protected Attributes

- static `error_vector_t * ptrans_err = &gerr`

12.73.1 Detailed Description

Abstract interface of a class representing a transition.

This class represents a transition. It is a part of the model of a system. The corresponding `system` is given as a parameter of constructor `transition_t(system_t * const system)` or by method `set_parent_system()`. Developer is responsible for correct setting of this "parent" `system` (but developer rarely creates new transitions - they are automatically created by `system` during source file reading).

12.73.2 Constructor & Destructor Documentation

12.73.2.1 `transition_t (system_t *const system) [inline]`

A constructor.

Parameters:

`system` = "parent" `system` of this transition

12.73.3 Member Function Documentation

12.73.3.1 `bool can_be_modified () const`

Tells, whether this transition can be modified.

This method uses "parent" `system` given in a constructor or a method `set_parent_system()`. Therefore "parent" `system` has to be set before the first call of this method.

References `system_t::get_abilities()`, `parent_system`, and `system_abilities_t::transition_can_be_modified`.

12.73.3.2 `bool can_read () const`

Tells, whether this transition can be modified.

This method uses "parent" `system` given in a constructor or a method `set_parent_system()`. Therefore "parent" `system` has to be set before the first call of this method.

References `system_t::get_abilities()`, `parent_system`, and `system_abilities_t::transition_can_read`.

12.73.3.3 `virtual int from_string (std::string & trans_str, const size_int_t process_gid = NO_ID) [pure virtual]`

Reads in the transition from a string representation.

Parameters:

trans_str = string containing a source of a transition

process_gid = context of a process (default value NO_ID = global context)

Returns:

... 0 iff no error occurs, non-zero value in a case of error during a reading.

Implemented in [bymoc_transition_t](#), [dve_prob_transition_t](#), and [dve_transition_t](#).

12.73.3.4 `static error_vector_t& get_error_vector () [inline, static, protected]`

Protected static method returning , which will be used in case of any error message

12.73.3.5 `size_int_t get_gid () const [inline]`

Returns [transition GID](#) of this transition.

Note:

This method is not virtual, because it should be really fast

Referenced by `prob_system_t::consolidate()`, `por_t::generate_ample_sets()`, `por_t::generate_composed_ample_sets()`, `dve_prob_explicit_system_t::get_succs()`, and `por_t::init()`.

12.73.3.6 `size_int_t get_lid () const [inline]`

Returns [transition LID](#) of this transition.

Note:

This method is not virtual, because it should be really fast

Referenced by `por_t::init()`, `dve_system_trans_t::write()`, and `dve_prob_process_t::write()`.

12.73.3.7 `virtual int read (std::istream & istr, size_int_t process_gid = NO_ID) [pure virtual]`

Reads in the transition from a string representation in stream.

Parameters:

istr = input stream containing a source of transition

process_gid = context of process (default value NO_ID = global context)

Returns:

... 0 iff no error occurs, non-zero value in a case of error during a reading.

Implemented in [bymoc_transition_t](#), [dve_prob_transition_t](#), and [dve_transition_t](#).

12.73.3.8 static void set_error_vector (error_vector_t & *evect*) [inline, static, protected]

Protected static method setting , which will be used in case of any error message

12.73.3.9 virtual void set_gid (const size_int_t *gid*) [inline, virtual]

Sets a [GID](#) of this transition.

Warning:

See the description of concrete implementation. This method often cannot be used by developer from consistency reasons (even if it is implemented).

Referenced by [prob_system_t::consolidate\(\)](#).

12.73.3.10 virtual void set_lid (const size_int_t *id*) [inline, virtual]

Sets a [LID](#) of this transition.

Warning:

See the description of concrete implementation. This method often cannot be used by developer from consistency reasons (even if it is implemented).

Referenced by [dve_prob_process_t::add_prob_transition\(\)](#), and [dve_process_t::add_transition\(\)](#).

12.73.3.11 virtual std::string to_string () const [pure virtual]

Returns a string representation of the transition.

If [system](#) can work with transitions (according to [system_t::can_transitions\(\)](#)), this method is obligatory to implement.

Implemented in [bymoc_transition_t](#), [dve_prob_transition_t](#), and [dve_transition_t](#).

12.73.3.12 `virtual void write (std::ostream & ostr) const` [pure virtual]

Writes a string representation of the transition to stream.

If [system](#) can work with transitions (according to [system_t::can_transitions\(\)](#)), this method is obligatory to implement.

Implemented in [bymoc_transition_t](#), [dve_prob_transition_t](#), and [dve_transition_t](#).

Referenced by [dve_prob_process_t::write\(\)](#).

12.73.4 Member Data Documentation

12.73.4.1 `error_vector_t * ptrans_terr = &gerr` [static, protected]

Protected static data item containing , which will be used in case of any error message

Referenced by [dve_transition_t::read\(\)](#), [dve_prob_transition_t::read\(\)](#), and [dve_prob_transition_t::write\(\)](#).

The documentation for this class was generated from the following files:

- [transition.hh](#)
- [transition.cc](#)

12.74 updateable_info_t Class Template Reference

Class used to collect data during synchronization.

```
#include <distributed.hh>
```

Inherits `abstract_info_t`.

Public Member Functions

- virtual void **get_data_ptr** (char *&ptr)
- virtual void **get_data_size** (int &size)
- virtual void **update** (void)

Public Attributes

- constant_data_t [const_data](#)
- updateable_data_t [data](#)

Structure that stores collected data.

12.74.1 Detailed Description

```
template<class updateable_data_t, class constant_data_t = no_const_data_type_  
t> class updateable_info_t< updateable_data_t, constant_data_t >
```

Class used to collect data during synchronization.

Instances of this class are passed to [distributed_t::synchronized\(abstract_info_t &info\)](#) function.

The *updateable_data_t* type should be struct with simple types in it (no dynamic arrays, strings, etc.) - the struct holds the collected data. It must have a "void update(void)" function, which will be called on every workstation.

The word "updateable" means that each workstation can change the data being collected during the synchronization process (by calling the `update()` function). For more information see [distributed_t::synchronized\(abstract_info_t &info\)](#) function.

12.74.2 Member Data Documentation

12.74.2.1 constant_data_t const_data

Structure that stores local constant part of info (usually needed for computation of the collected part)

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

- [distributed.hh](#)

12.75 `updateable_info_t< updateable_data_t, no_const_data_type_t >` Class Template Reference

Class used to collect data during synchronization.

```
#include <distributed.hh>
```

Inherits `abstract_info_t`.

Public Member Functions

- virtual void **get_data_ptr** (char *&ptr)
- virtual void **get_data_size** (int &size)
- virtual void **update** (void)

Public Attributes

- `updateable_data_t` [data](#)
Structure that stores collected data.

12.75.1 Detailed Description

template<class `updateable_data_t`> class `updateable_info_t< updateable_data_t, no_const_data_type_t >`

Class used to collect data during synchronization.

Instances of this class are passed to [distributed_t::synchronized\(abstract_info_t &info\)](#) function.

The `updateable_data_t` type should be struct with simple types in it (no dynamic arrays, strings, etc.) - the struct holds the collected data. It must have a "void update(void)" function, which will be called on every workstation.

The word "updateable" means that each workstation can change the data being collected during the synchronization process (by calling the `update()` function). For more information see [distributed_t::synchronized\(abstract_info_t &info\)](#) function.

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

- [distributed.hh](#)

12.76 vminfo_t Class Reference

Class for a memory consumption measuring.

```
#include <sysinfo.hh>
```

Public Member Functions

- int **getvmdata** ()
- int **getvmrss** ()
- int **getvmsize** ()
- void **print** ()
- void **scan** ()

Public Attributes

- int **vmdata**
- int **vmexe**
- int **vmck**
- int **vmlib**
- int **vmrss**
- int **vmsize**
- int **vmstk**

Protected Attributes

- char **filename** [100]
- int **lineskip**
- int **pid**
- bool **statm**

12.76.1 Detailed Description

Class for a memory consumption measuring.

The documentation for this class was generated from the following files:

- sysinfo.hh
- sysinfo.cc

Chapter 13

File Documentation

13.1 array.hh File Reference

Classes

- class [array_t](#)
Simple resizable container representing 1-dimensional array.

Functions

- `template<class T>`
`T * default_new_field_of_objects (const size_int_t count)`

13.1.1 Detailed Description

This header file contains a complete definition of the [array_t](#) class template

This class template is commonly used (e. g. in [system_t](#) and [explicit_system_t](#)). It is used as a replacement of slow containers from STL.

Author:

Pavel Simecek

13.2 bit_string.hh File Reference

Classes

- class [bit_string_t](#)
Class for impementation of field of bits.

Variables

- static const `ulong_int_t` **bit_values** [32]

13.2.1 Detailed Description

Unit implementing a simple vector of bits using class [bit_string_t](#)

Author:

Pavel Simecek

13.2.2 Variable Documentation

13.2.2.1 `const ulong_int_t bit_values[32]` `[static]`

Initial value:

```
{1, 2, 4, 8, 16, 32, 64, 128, 256, 1<<9, 1<<10, 1<<11, 1<<12, 1<<13, 1<<14,  
1<<15, 1<<16, 1<<17, 1<<18, 1<<19, 1<<20, 1<<21, 1<<22, 1<<23, 1<<24,  
1<<25, 1<<26, 1<<27, 1<<28, 1<<29, 1<<30, 1<<31}
```

13.3 bymoc_explicit_system.hh File Reference

Classes

- class [bymoc_explicit_system_t](#)

13.3.1 Detailed Description

The main contribution of this file is the class [bymoc_explicit_system_t](#)

13.4 bymoc_expression.hh File Reference

Classes

- class [bymoc_expression_t](#)
Class representing an expression in BYMOC [system](#).

13.4.1 Detailed Description

The main contribution of this file is the class [bymoc_expression_t](#).

13.5 bymoc_process.hh File Reference

Classes

- class [bymoc_process_t](#)
Class representing a process in BYMOC system.

13.5.1 Detailed Description

The main contribution of this file is the class [bymoc_process_t](#).

13.6 bymoc_process_decomposition.hh File Reference

13.6.1 Detailed Description

Process graph decomposition into SCCs including types

13.7 bymoc_system.hh File Reference

Classes

- class [bymoc_system_t](#)
Class for Promela/Bytecode system representation.

13.7.1 Detailed Description

The main contribution of this file is the class [bymoc_system_t](#)

13.8 bymoc_system_trans.hh File Reference

Classes

- class [bymoc_enabled_trans_t](#)
Class implementing enabled trasition in BYMOC [system](#).
- class [bymoc_system_trans_t](#)
Class implementing system trasition in BYMOC [system](#).

13.8.1 Detailed Description

This file contains definitions of classes [bymoc_system_trans_t](#) and [bymoc_enabled_trans_t](#).

13.9 bymoc_transition.hh File Reference

Classes

- class [bymoc_transition_t](#)
Class representing a transition in BYMOC [system](#).

13.9.1 Detailed Description

The main contribution of this file is the class [bymoc_transition_t](#).

13.10 compressor.hh File Reference

Namespaces

- namespace **divine**

Classes

- class [compressor_t](#)
Compression implementation class.

Defines

- #define **HUFFMAN_COMPRESS** 12
- #define **NO_COMPRESS** 11

13.10.1 Detailed Description

Unit for compression of explicit states of system. Implemented by class `compressor_t`.

Author:

Jiri Barnat

13.11 data.hh File Reference

Classes

- class [data_t](#)

Class representing a general data type.

Variables

- const ushort_int_t **DATA_TYPE_BYTE** = 1
- const ushort_int_t **DATA_TYPE_SBYTE** = 0
- const ushort_int_t **DATA_TYPE_SIZE_INT** = 6
- const ushort_int_t **DATA_TYPE_SLONG_INT** = 4
- const ushort_int_t **DATA_TYPE_SSHORT_INT** = 2
- const ushort_int_t **DATA_TYPE_UBYTE** = 1
- const ushort_int_t **DATA_TYPE_ULONG_INT** = 5
- const ushort_int_t **DATA_TYPE_UNKNOWN** = MAX_USHORT_INT
- const ushort_int_t **DATA_TYPE_USHORT_INT** = 3

13.11.1 Detailed Description

This file contains a definition of the class [data_t](#) - the general container of data computed in [system](#) or [explicit system](#)

13.12 distr_reporter.hh File Reference

Classes

- class [distr_reporter_t](#)

Variables

- const size_t **REPORTER_AVG** = 2
- const size_t [REPORTER_MASTER](#) = 0
These are constants for set_info.
- const size_t **REPORTER_MAX** = 4
- const size_t **REPORTER_MIN** = 1
- const size_t [REPORTER_OUTPUT_LONG](#) = 3
Long detailed output with values on all workstations.
- const size_t [REPORTER_OUTPUT_NORMAL](#) = 2
Normal output, with avg/max/min... etc, but without the list of values.
- const size_t [REPORTER_OUTPUT_SHORT](#) = 1
Short output, compatible with sequential one.
- const size_t **REPORTER_SUM** = 3

13.12.1 Detailed Description

This file contains a definition of [distr_reporter_t](#) - class for reporting and measuring during distributed computations

Author:

Radek Pelanek

13.13 distributed.hh File Reference

Classes

- class `distributed_t`
Main distributed support class.
- class `static_info_t`
Class used to collect data during synchronization.
- class `updateable_info_t`
Class used to collect data during synchronization.
- class `updateable_info_t< updateable_data_t, no_const_data_type_t >`
Class used to collect data during synchronization.

Variables

- const int `DIVINE_TAG_SYNC_COMPLETION` = 3
Constant used internally, of no relevance to user of `distributed_t`.
- const int `DIVINE_TAG_SYNC_ONE` = 1
Constant used internally, of no relevance to user of `distributed_t`.
- const int `DIVINE_TAG_SYNC_READY` = 0
Constant used internally, of no relevance to user of `distributed_t`.
- const int `DIVINE_TAG_SYNC_TWO` = 2
Constant used internally, of no relevance to user of `distributed_t`.
- const int `DIVINE_TAG_USER` = 144
All messages sent by user must have tags greater or equal to this constant.
- const int `NETWORK_HASH_SEED` = 0xbafba99
Default hash seed for partition function.
- const int `NETWORK_ID_MANAGER` = 0
Id of the workstation that initiated the distributed computation.

13.13.1 Detailed Description

Distributed computing support unit header

13.14 dve_commonparse.hh File Reference

Classes

- struct [dve_position_t](#)

Structure for storing of position in a source code.

Defines

- #define [MAXLEN](#) 64

!!DO NEVER INCLUDE ONLY gramsymb.hh

- #define [YYLTYPE](#) [dve_position_t](#)

Functions

- void [dve_eeerror](#) (const char *msg)

- int [dve_eeparse](#) ()

Bison's parser of DVE expressions.

- void [dve_pperror](#) (const char *msg)

- int [dve_ppparse](#) ()

Bison's parser of DVE processes.

- void [dve_tterror](#) (const char *msg)

- int [dve_ttparse](#) ()

Bison's parser of DVE transactions.

- void [dve_yyerror](#) (const char *msg)

- int [dve_yyparse](#) ()

Bison's parser of DVE files.

13.14.1 Detailed Description

Auxiliary header file used in parser. You should not need to use it explicitly during an implementation of program based on this library

Author:

Pavel Simecek

13.14.2 Function Documentation

13.14.2.1 void dve_eeerror (const char * *msg*)

Method called by Bison's parser in case of unrecorvable syntax error in expression (of DVE syntax)

13.14.2.2 void dve_pperror (const char * *msg*)

Method called by Bison's parser in case of unrecorvable syntax error in process (of DVE syntax)

13.14.2.3 void dve_tterror (const char * *msg*)

Method called by Bison's parser in case of unrecorvable syntax error in transition (of DVE syntax)

13.14.2.4 void dve_yyerror (const char * *msg*)

Method called by Bison's parser in case of unrecorvable syntax error in DVE file

13.15 dve_explicit_system.hh File Reference

Classes

- class [dve_explicit_system_t](#)
Class in DVE system interpretation.
- struct [dve_explicit_system_t::state_creator_t](#)
- struct [ES_parameters_t](#)
*Structure determined for passing parameters to `ES*_eval()` functions.*

Typedefs

- typedef `ushort_int_t` [dve_state_int_t](#)
- typedef [ES_parameters_t](#) * [ES_p_parameters_t](#)
Pointer to [ES_parameters_t](#).

Variables

- const `size_int_t` [ES_FMT_DIVIDE_PROCESSES_BY_CR](#) = 8
Print 'new-line' character after each process./*
- const `size_int_t` [ES_FMT_PRINT_ALL_NAMES](#)
- const `size_int_t` [ES_FMT_PRINT_PROCESS_NAMES](#) = 4
Print names of processes before printing of its state and [variables](#).
- const `size_int_t` [ES_FMT_PRINT_STATE_NAMES](#) = 1
Print names of states instead of state IDs.
- const `size_int_t` [ES_FMT_PRINT_VAR_NAMES](#) = 2
Print names of [variables](#) before printing of their values.

13.15.1 Detailed Description

The main contribution of this file is the class [dve_explicit_system_t](#)

13.15.2 Typedef Documentation

13.15.2.1 `ushort_int_t` [dve_state_int_t](#)

[dve_state_int_t](#) is an interger type that is used to store state ID

13.15.3 Variable Documentation

13.15.3.1 `const size_int_t ES_FMT_PRINT_ALL_NAMES`

Initial value:

```
ES_FMT_PRINT_VAR_NAMES |
                        ES_FMT_PRINT_STATE_NAMES |
                        ES_FMT_PRINT_PROCESS_NAMES |
                        ES_FMT_DIVIDE_PROCESSES_BY_CR
```

Print all the things given by `ES_FMT_PRINT_STATE_NAMES`, `ES_FMT_PRINT_VAR_NAMES`, `ES_FMT_PRINT_PROCESS_NAMES` and `ES_FMT_DIVIDE_PROCESSES_BY_CR`

13.16 dve_expression.hh File Reference

Classes

- struct [compacted_t](#)
Class to access compacted expressions.
- struct [compacted_viewer_t](#)
Structure of a single compacted expression. In the memory block, this initial structure is followed by the left subexpression (if present), and then by right subexpression (if present). `r_offset` is an offset to the right subexpression.
- class [dve_expression_t](#)
Class representing an expression in DVE [system](#).

Variables

- `const size_int_t DVE_MAXIMAL_ARITY = 2`

13.16.1 Detailed Description

The main contribution of this file is the class [dve_expression_t](#).

13.17 dve_grammar.hh File Reference

Functions

- void [dve_expr_init_parsing](#) ([dve_parser_t](#) *const, [error_vector_t](#) *, std::istream &mystream)
Initializes a Bison's parser of DVE expressions before parsing.
- void [dve_init_parsing](#) ([dve_parser_t](#) *const, [error_vector_t](#) *, std::istream &mystream)
Initializes a Bison's parser of DVE files before parsing.
- void [dve_proc_init_parsing](#) ([dve_parser_t](#) *const, [error_vector_t](#) *, std::istream &mystream)
Initializes a Bison's parser of DVE processes before parsing.
- void [dve_trans_init_parsing](#) ([dve_parser_t](#) *const, [error_vector_t](#) *, std::istream &mystream)
Initializes a Bison's parser of DVE transitions before parsing.

13.17.1 Detailed Description

This file only contains function `*init_parsing()` which are further used in [system_t::read\(\)](#), [process_t::read\(\)](#), [transition_t::read\(\)](#) and [expression_t::read\(\)](#).

13.18 dve_parser.hh File Reference

Classes

- class [dve_parser_t](#)
Class that provides an interface to the parsing of DVE sources.

Variables

- const size_int_t [DVE_MAX_ARRAY_SIZE](#) = 2147483647
maximal bound for array in DVE source

13.18.1 Detailed Description

The main contribution of this file is class [dve_parser_t](#), that serves as a symbol table and the interface for parsing and analysing DVE sources.

Author:

Pavel Simecek

13.19 dve_prob_explicit_system.hh File Reference

Classes

- class [dve_prob_explicit_system_t](#)
Class in DVE system interpretation.

13.19.1 Detailed Description

The main contribution of this file is the class [dve_prob_explicit_system_t](#)

13.20 dve_prob_process.hh File Reference

Classes

- class [dve_prob_process_t](#)
Class representing a DVE process in probabilistic system.

13.20.1 Detailed Description

The main contribution of this file is the abstract interface [dve_prob_process_t](#).

13.21 dve_prob_system.hh File Reference

13.21.1 Detailed Description

The main contribution of this file is the class `dve_prob_system_t`

13.22 dve_prob_transition.hh File Reference

Classes

- class [dve_prob_transition_t](#)
Class representing a DVE transition in probabilistic system.

13.22.1 Detailed Description

The main contribution of this file is the class [dve_prob_transition_t](#).

13.23 dve_process.hh File Reference

Classes

- class [dve_process_t](#)
Class representing a process.

Variables

- const size_int_t [DVE_PROCESS_ALLOC_STEP](#) = 20

13.23.1 Detailed Description

The main contribution of this file is the class [dve_process_t](#).

13.23.2 Variable Documentation

13.23.2.1 const size_int_t DVE_PROCESS_ALLOC_STEP = 20

Alloctaion steps for [array_t](#) used in an implementation of [dve_process_t](#)

- quite internal thing

13.24 dve_process_decomposition.hh File Reference

Classes

- class [dve_process_decomposition_t](#)

13.24.1 Detailed Description

Process graph decomposition into SCCs including types

13.25 dve_source_position.hh File Reference

Classes

- class [dve_source_position_t](#)
Class for storage of position in a source code.

13.25.1 Detailed Description

The main contribution of this file is the class [dve_source_position_t](#).

13.26 dve_symbol_table.hh File Reference

Classes

- class [dve_symbol_table_t](#)

Class which stores the declaration of symbols (see [dve_symbol_t](#)).

13.26.1 Detailed Description

The main contribution of this file is a definition of class [dve_symbol_table_t](#).

13.27 dve_system.hh File Reference

Classes

- class [dve_system_t](#)
Class for a DVE system representation.
- union [SYS_initial_values_t](#)
Internal type for storing initial values of various identifiers.
- struct [SYS_parameters_t](#)

Typedefs

- typedef [SYS_parameters_t](#) * [SYS_p_parameters_t](#)

Enumerations

- enum [system_synchronicity_t](#) { [SYSTEM_SYNC](#) = 0, [SYSTEM_ASYNC](#) = 1 }

13.27.1 Detailed Description

The main contribution of this file is the class [dve_system_t](#)

13.27.2 Enumeration Type Documentation

13.27.2.1 enum system_synchronicity_t

Type of system - [SYSTEM_SYNC](#) = synchrononous system, [SYSTEM_ASYNC](#) = asynchronous system

13.28 dve_system_trans.hh File Reference

Classes

- class [dve_enabled_trans_t](#)
Class implementing enabled trasition in DVE [system](#).
- class [dve_system_trans_t](#)
Class implementing system trasition in DVE [system](#).

13.28.1 Detailed Description

This file contains definitions of classes [dve_system_trans_t](#) and [dve_enabled_trans_t](#).

13.29 dve_token_vector.hh File Reference

Classes

- class [dve_token_vector_t](#)

Class used by [dve_symbol_table_t](#) to store the names of symbols.

13.29.1 Detailed Description

This file contains class [dve_token_vector_t](#) that can store a long vector of strings and serves as a deallocator of the memory used for strings.

This class is used in class `table_t` for storing names of identifiers.

13.30 dve_transition.hh File Reference

Classes

- class [dve_transition_t](#)
Class representing a transition.

Enumerations

- enum [sync_mode_t](#) {
 [SYNC_NO_SYNC](#) = 0, [SYNC_EXCLAIM](#) = 1, [SYNC_ASK](#) = 2, [SYNC_EXCLAIM_BUFFER](#) = 3,
 [SYNC_ASK_BUFFER](#) = 4 }
Synchronization mode of a transition.

Variables

- const size_int_t [TR_effects_alloc_step](#) = 20
- const size_int_t [TR_effects_default_alloc](#) = 10

13.30.1 Detailed Description

The main contribution of this file is the class [dve_transition_t](#).

13.30.2 Enumeration Type Documentation

13.30.2.1 enum sync_mode_t

Synchronization mode of a transition.

Enumerator:

- SYNC_NO_SYNC*** transition is not synchronized
- SYNC_EXCLAIM*** transition is synchronized and sends a value to another process
- SYNC_ASK*** transition is synchronized and receives a value from another process
- SYNC_EXCLAIM_BUFFER*** transition is asynchronous and a message is stored into a buffer
- SYNC_ASK_BUFFER*** transition is asynchronous and a message is taken from a buffer

13.30.3 Variable Documentation

13.30.3.1 `const size_int_t TR_effects_alloc_step = 20`

Allocation step of [array_t](#) used in an implementation of [dve_transition_t](#)

- quite internal thing

13.30.3.2 `const size_int_t TR_effects_default_alloc = 10`

Default allocation size of [array_t](#) used in an implementation of [dve_transition_t](#) - quite internal thing

13.31 error.hh File Reference

Classes

- struct [ERR_throw_t](#)
- struct [ERR_triplet_t](#)
- class [error_string_t](#)
Class determined for storing error messages.
- class [error_vector_t](#)
The main class in [error.hh](#) determined for storing.
- struct [psh](#)
Structure determined for causing storing/printing error messages.
- struct [thr](#)
Structure determined for causing storing/printing error messages.

Defines

- `#define UNIMPLEMENTED(type)`

Typedefs

- typedef const [ERR_triplet_t](#) [ERR_c_triplet_t](#)
Definition of constant [ERR_triplet_t](#).
- typedef const char * [ERR_char_string_t](#)
Header file for error handling //.
- typedef int [ERR_id_t](#)
Integer type used for identifiers of errors.
- typedef unsigned int [ERR_nbr_t](#)
Integer type used for index of an error in an [error vector](#).
- typedef void(* [ERR_psh_callback_t](#))([error_vector_t](#) &terr, const [ERR_throw_t](#))
Callback function type (for handling of '<< psh()').
- typedef const std::string [ERR_std_string_t](#)
- typedef void(* [ERR_thr_callback_t](#))([error_vector_t](#) &terr, const [ERR_throw_t](#))
Callback function type (for '<< thr()').

- typedef int [ERR_type_t](#)
Integer type used for identifiers of errors.

Functions

- void [ERR_default_psh_callback](#) ([error_vector_t](#) &terr, const [ERR_throw_t](#))
Default 'warning handling callback'.
- void [ERR_default_thr_callback](#) ([error_vector_t](#) &terr, const [ERR_throw_t](#))
Default 'error handling callback'.
- template<typename T>
T **noreturn** ()

Variables

- const [ERR_id_t](#) [ERR_UNKNOWN_ID](#) = 1
- const [ERR_type_t](#) [ERR_UNKNOWN_TYPE](#) = 1
- [error_vector_t](#) gerr
Global [error vector](#) 'gerr'.

13.31.1 Detailed Description

Header file of a handling unit. This unit is independent of DiVinE and can be used in an arbitrary project.

Extern variable gerr is the global [error vector](#) that can be used in arbitrary file that includes [error.hh](#)

For more informations see [Error Handling Unit](#) page

13.31.2 Define Documentation

13.31.2.1 #define UNIMPLEMENTED(type)

Value:

```
do { \
    gerr << "unimplemented " << __FILE__ << ":" << __LINE__ << thr(); \
    return noreturn<type>(); \
} while(0)
```

13.31.3 Typedef Documentation

13.31.3.1 `typedef const char* ERR_char_string_t`

Header file for error handling `//`.

String, which is field of characters finished by `''`

13.31.3.2 `typedef void(* ERR_psh_callback_t)(error_vector_t &terr, const ERR_throw_t)`

Callback function type (for handling of `'<< psh()'`).

Type of callback function for handling commands like

```
terr << psh();
```

where `'terr'` is an instance of `error_vector_t`.

We will call functions of this type `'warning handling callbacks'`.

13.31.3.3 `typedef void(* ERR_thr_callback_t)(error_vector_t &terr, const ERR_throw_t)`

Callback function type (for `'<< thr()'`).

Type of callback function for handling commands like

```
terr << psh();
```

where `'terr'` is an instance of `error_vector_t`.

We will call functions of this type `'error handling callback'`.

13.31.4 Function Documentation

13.31.4.1 `void ERR_default_psh_callback (error_vector_t &terr, const ERR_throw_t)`

Default `'warning handling callback'`.

This is a default `'warning handling callback'` in `error_vector_t`.

To exchange this function by another function in an instance of `error_vector_t` you can call `error_vector_t::set_push_callback()`.

This default callback function has the following behaviour:

1. Prints the last kept message on the standard error output.
2. Erases printed message from a memory.

References `error_vector_t::perror_back()`, and `error_vector_t::pop_back()`.

13.31.4.2 void ERR_default_thr_callback (error_vector_t &terr, const ERR_throw_t)

Default 'error handling callback'.

This is a default 'error handling callback'.

To exchange this function by another function in an instance of [error_vector_t](#) you can call `error_vector_t::set_thr_callback()`.

This default callback function has the following behaviour:

1. Flushes all messages stored in a memory to the standard error output.
2. Clears the memory of messages.
3. Throws exception of type [ERR_throw_t](#) (parameter *error* type).

References `error_vector_t::flush()`.

13.31.5 Variable Documentation

13.31.5.1 const ERR_id_t ERR_UNKNOWN_ID = 1

Constant that is substituted instead of [error ID](#), if you don't specify any [error ID](#) to the function that has [error ID](#) as a parameter.

Referenced by `error_vector_t::~~error_vector_t()`.

13.31.5.2 const ERR_type_t ERR_UNKNOWN_TYPE = 1

Constant that is substituted instead of of integer sent by throwing, if you don't specify any [error ID](#) to the function that has [error ID](#) as a parameter.

Referenced by `error_vector_t::~~error_vector_t()`.

13.31.5.3 error_vector_t gerr

Global [error vector](#) 'gerr'.

Global [error vector](#) *gerr*. You can use it in your programs, but you can also declare your own [error vector](#) and use it instead of *gerr*. This global object is proper because of functions which are not wrapped in a structure or class with own [error vector](#).

Referenced by `bymoc_process_t::add_transition()`, `dve_parser_t::assertion_create()`, `bymoc_expression_t::assign()`, `dve_expression_t::dve_expression_t()`, `dve_system_t::fast_eval()`, `bymoc_transition_t::from_string()`, `bymoc_process_t::from_string()`, `bymoc_expression_t::from_string()`, `por_t::generate_ample_sets()`, `por_t::generate_composed_ample_sets()`, `bymoc_system_trans_t::get_count()`, `dve_system_t::get_property_scc_count()`, `dve_system_t::get_property_scc_id()`, `dve_system_t::get_property_scc_type()`, `bymoc_process_t::get_trans_count()`, `bymoc_process_t::get_transition()`, `dve_system_t::is_property_weak()`, `bymoc_enabled_trans_t::operator=()`,

bymoc_system_trans_t::operator=(), bymoc_system_trans_t::operator[](), array_t<dve_symbol_t * >::operator[](), bymoc_transition_t::read(), bymoc_process_t::read(), bymoc_expression_t::read(), logger_t::register_double(), logger_t::register_slong_int(), logger_t::register_ulong_int(), bymoc_process_t::remove_transition(), bymoc_system_trans_t::set_count(), distr_reporter_t::set_global_info(), bymoc_expression_t::swap(), dve_parser_t::system_property(), dve_parser_t::system_synchronicity(), bymoc_transition_t::to_string(), bymoc_system_trans_t::to_string(), bymoc_process_t::to_string(), bymoc_expression_t::to_string(), dve_parser_t::trans_sync(), dve_expression_t::write(), bymoc_transition_t::write(), bymoc_system_trans_t::write(), bymoc_process_t::write(), and bymoc_expression_t::write().

13.32 explicit_storage.hh File Reference

Classes

- class `explicit_storage_t`
explicit storage class
- class `state_ref_t`
State reference class.

Defines

- `#define EXPLICIT_STORAGE_ERR_TYPE 11`
- `#define EXPLICIT_STORAGE_HASH_SEED 0xBA9A9ABA`

Functions

- static bool `operator!=` (const `state_ref_t` &r1, const `state_ref_t` &r2)
Operator for non-equality of references.
- static bool `operator<` (const `state_ref_t` &r1, const `state_ref_t` &r2)
Operator "<" for references.
- `std::ostream & operator<<` (std::ostream &ostr, const `state_ref_t` &ref)
Prints a reference representation to the given stream.
- static bool `operator<=` (const `state_ref_t` &r1, const `state_ref_t` &r2)
Operator "<=" for references.
- static bool `operator==` (const `state_ref_t` &r1, const `state_ref_t` &r2)
Operator for equality of references.
- static bool `operator>` (const `state_ref_t` &r1, const `state_ref_t` &r2)
Operator ">" for references.
- static bool `operator>=` (const `state_ref_t` &r1, const `state_ref_t` &r2)
Operator ">=" for references.

13.32.1 Detailed Description

state hashing support

Author:

Jiri Barnat

13.33 explicit_system.hh File Reference

Classes

- class [explicit_system_t](#)
- class [succ_container_t](#)

Class determined to store successors of some state.

Functions

- bool [succs_deadlock](#) (const int succs_result)

Returns, whether return value of `get_succs()` comprises `SUCC_DEADLOCK`.

- bool [succs_error](#) (const int succs_result)

Returns, whether return value of `get_succs()` comprises `SUCC_ERROR`.

- bool [succs_normal](#) (const int succs_result)

Returns, whether `get_succs()` method has returned `SUCC_NORMAL`.

Variables

- const int [SUCC_DEADLOCK](#) = 2
- const int [SUCC_ERROR](#) = 1
- const int [SUCC_NORMAL](#) = 0

13.33.1 Detailed Description

This file contains:

- constants `SUCC_NORMAL`, `SUCC_ERROR` and `SUCC_DEADLOCK`
- functions [succs_normal\(\)](#), [succs_error\(\)](#), [succs_deadlock\(\)](#) ... for analyses of return values of `explicit_system_t::get_succs()` and similar methods
- definition of the class [succ_container_t](#) ... container of generated successors of states of the [system](#)
- definition of the abstract interface [explicit_system_t](#) ... the most important part of this unit

13.33.2 Function Documentation

13.33.2.1 `bool succs_deadlock (const int succs_result) [inline]`

Returns, whether return value of `get_succs()` comprises `SUCC_DEADLOCK`.

Parameters:

succs_result = return value of `get_succs()` method

Returns:

true iff *succs_results* bitwise contains `SUCC_DEADLOCK` (*succs_results* is a bitwise sum of `SUCC_NORMAL`, `SUCC_ERROR` and `SUCC_DEADLOCK`)

See [Results of methods for creating successors](#) for details.

References `SUCC_DEADLOCK`.

13.33.2.2 `bool succs_error (const int succs_result) [inline]`

Returns, whether return value of `get_succs()` comprises `SUCC_ERROR`.

Parameters:

succs_result = return value of `get_succs()` method

Returns:

true iff *succs_results* bitwise contains `SUCC_ERROR` (*succs_results* is a bitwise sum of `SUCC_NORMAL`, `SUCC_ERROR` and `SUCC_DEADLOCK`)

See [Results of methods for creating successors](#) for details.

References `SUCC_ERROR`.

13.33.2.3 `bool succs_normal (const int succs_result) [inline]`

Returns, whether `get_succs()` method has returned `SUCC_NORMAL`.

Parameters:

succs_result = return value of `get_succs()` method

Returns:

true iff *succs_results* == `SUCC_NORMAL`

See [Results of methods for creating successors](#) for details.

References `SUCC_NORMAL`.

13.33.3 Variable Documentation

13.33.3.1 `const int SUCC_DEADLOCK = 2`

Constant that is a possible return value of `get_succs()` functions. See [Results of methods for creating successors](#) for details

Referenced by `dve_explicit_system_t::get_async_enabled_trans()`, `bymoc_explicit_system_t::get_succs()`, `dve_explicit_system_t::get_sync_succs_internal()`, and `succs_deadlock()`.

13.33.3.2 `const int SUCC_ERROR = 1`

Constant that is a possible return value of `get_succs()` functions. See [Results of methods for creating successors](#) for details

Referenced by `por_t::generate_ample_sets()`, `por_t::generate_composed_ample_sets()`, `dve_explicit_system_t::get_async_enabled_trans()`, `dve_explicit_system_t::get_async_succs_internal()`, `dve_explicit_system_t::get_ith_succ()`, `dve_explicit_system_t::get_sync_succs_internal()`, and `succs_error()`.

13.33.3.3 `const int SUCC_NORMAL = 0`

Constant that is a possible return value of `get_succs()` functions. See [Results of methods for creating successors](#) for details

Referenced by `dve_explicit_system_t::get_async_enabled_trans()`, `dve_explicit_system_t::get_sync_succs_internal()`, and `succs_normal()`.

13.34 expression.hh File Reference

Classes

- class [expression_t](#)

Abstract interface of a class representing an expression.

13.34.1 Detailed Description

The main contribution of this file is the abstract interface [expression_t](#).

13.35 hash_function.hh File Reference

Classes

- class [hash_function_t](#)

Class that unifies hash functions in the library.

Enumerations

- enum [hash_functions_t](#) { **DEFAULT**, **JENKINS**, **DIVINE**, **HC4** }

Type to identify hash functions.

13.35.1 Detailed Description

This header file contains the definition of class [hash_function_t](#). The class is instantiated by [explicit_storage_t](#) and [distributed_t](#) instances.

Author:

Jiri Barnat

13.36 huffman.hh File Reference

Variables

- `huff_sym_t sym` [256]

13.36.1 Detailed Description

File containing table for Huffman coding. Each symbol is coded by code `sym[symbol][0]` in `sym[symbol][1]` bits

13.37 inttostr.hh File Reference

Functions

- `template<typename T>`
 static char * **create_string_from** (T i)
- void **dispose_string** (char *const str)

13.37.1 Detailed Description

This file contains the functions which can convert some integer types to the string representation (e. g. convert 2345 to "2345").

It also contains the function `dispose_string` that only calls `free()` on a given argument.

These function are only used in [error.hh](#) ([Error Handling Unit](#)) for converting numbers on the error input to strings.

13.38 logger.hh File Reference

Classes

- class [logger_t](#)

13.38.1 Detailed Description

This file contains declaration of [logger_t](#) class.

13.39 mcr12_explicit_system.hh File Reference

13.39.1 Detailed Description

The main contribution of this file is the class `mcr12_explicit_system_t`

13.40 mcrl2_system.hh File Reference

Defines

- `#define MCRL2_ATEM_INIT(x, y) do{ }while(0)`

13.40.1 Detailed Description

The main contribution of this file is the class `mcrl2_system_t`

13.41 network.hh File Reference

Classes

- class [comm_matrix_t](#)
Communication matrix.
- class [network_t](#)
Network communication support class.

Defines

- #define **OPT_STATS**
- #define **OPT_STATS_MPI_RATE**
- #define **STATS_INTERVAL** 3.0

Typedefs

- typedef auto_ptr< [comm_matrix_t](#) > [pcomm_matrix_t](#)
Communication matrix auto pointer.

Variables

- const int [NET_ERR_ABORT_FAILED](#) = 13
Network abort function failed.
- const int [NET_ERR_ALLGATHER_FAILED](#) = 15
Function allgather failed.
- const int [NET_ERR_ALREADY_INITIALIZED](#) = 1
Trying to initialize, when network is already initialized.
- const int [NET_ERR_BARRIER_FAILED](#) = 12
Barrier function failed.
- const int [NET_ERR_FINALIZATION_FAILED](#) = 4
Finalization of network failed.
- const int [NET_ERR_GATHER_FAILED](#) = 14
Function gather failed.
- const int [NET_ERR_GET_MSG_SIZE_FAILED](#) = 11
Getting message size from network failed.

- const int `NET_ERR_INITIALIZATION_FAILED` = 3
Initialization of network failed.
- const int `NET_ERR_INVALID_DESTINATION` = 6
Trying to send message to or aquire information about a non-existing destination.
- const int `NET_ERR_INVALID_MSG_SIZE` = 5
Trying to send message whose size exceeds the buffer size.
- const int `NET_ERR_INVALID_SOURCE` = 7
Trying to get message from or aquire information about a non-existing source.
- const int `NET_ERR_INVALID_WORKSTATION_NUMBER` = 16
Trying aquire information about a non-existing workstation.
- const int `NET_ERR_MSG_PROBE_FAILED` = 9
Probe for messages failed.
- const `ERR_triplet_t net_err_msgs` [17]
Array of error descriptions.
- const int `NET_ERR_NOT_INITIALIZED` = 2
Trying to do something that requires initialized network, when network is not initialized.
- const int `NET_ERR_RECEIVE_MSG_FAILED` = 10
Receiving of message failed.
- const int `NET_ERR_SEND_MSG_FAILED` = 8
Sending of message failed.
- const int `NET_NO_ERROR` = 0
Everything's ok.
- const int `NET_TAG_NORMAL` = 0
- const int `NET_TAG_URGENT` = 1
- const int `NETWORK_ERR_TYPE` = 1729
Identifier of exceptions raised by `network_t`.

13.41.1 Detailed Description

Network support unit header

13.41.2 Typedef Documentation

13.41.2.1 `typedef auto_ptr<comm_matrix_t> pcomm_matrix_t`

Communication matrix auto pointer.

Points to communication matrix. Variables of this type are outputs of some statistical methods of [network_t](#). The only important thing you need to know about auto pointers is that if auto pointer is destroyed (or rewritten), then the object it points to is also destroyed (unless some other pointer points to the object).

13.41.3 Variable Documentation

13.41.3.1 `const int NET_TAG_NORMAL = 0`

MPI tag for normal messages

This tag is used internally by [network_t](#) and has nothing to do with tags passed as parameters to [network_t](#) methods. Programmers using [network_t](#) do not have direct access to MPI, hence this constant is irrelevant to them.

Referenced by `network_t::flush_buffer()`.

13.41.3.2 `const int NET_TAG_URGENT = 1`

MPI tag for urgent messages

This tag is used internally by [network_t](#) and has nothing to do with tags passed as parameters to [network_t](#) methods. Programmers using [network_t](#) do not have direct access to MPI, hence this constant is irrelevant to them.

13.42 path.hh File Reference

Defines

- #define `PATH_CYCLE_SEPARATOR` "====="

13.42.1 Detailed Description

This file contains definition of `path_t` class

13.42.2 Define Documentation

13.42.2.1 #define `PATH_CYCLE_SEPARATOR` "====="

Class `path_t` is used to construct and print the representation of path in the state space. For building a 'normal' path, just keep adding states from the beginning or end of the path. For building a path with cycle, start from one end, keep adding states on path and mark the state where the cycle starts. You add this state only one time.

Programmer should take care that states are added correctly (such that transitions between them exists)

13.43 por.hh File Reference

Classes

- class `por_t`
Class for utilization of partial order reduction.

Variables

- `const std::size_t POR_FIND_ONLY = 4`
Defines the type of choosing ample set - only finds processes whose transitions can be taken as ample set.
- `const std::size_t POR_FIRST = 1`
*Defines the type of choosing ample set - transitions from the **first** proper process will be taken.*
- `const std::size_t POR_LAST = 2`
*Defines the type of choosing ample set - transitions from the **last** proper process will be taken.*
- `const std::size_t POR_SMALLEST = 3`
*Defines the type of choosing ample set - transitions from the proper process with **smallest** number of transitions will be taken.*

13.43.1 Detailed Description

* This file contains the class `por_t` for utilization of partial order reduction.

13.44 prob_explicit_system.hh File Reference

Classes

- struct [prob_and_property_trans_t](#)
- class [prob_explicit_system_t](#)
- class [prob_succ_container_t](#)

Class determined to store probabilistic successors of some state.

- struct [prob_succ_element_t](#)

Single element to store in [prob_succ_container_t](#).

13.44.1 Detailed Description

This file contains:

- definition of the abstract interface [prob_explicit_system_t](#) the most important part of this unit

13.45 prob_process.hh File Reference

Classes

- class [prob_process_t](#)

13.45.1 Detailed Description

The main contribution of this file is the abstract interface [prob_process_t](#).

13.46 prob_system.hh File Reference

Classes

- class [prob_system_t](#)

Abstract interface of a class representing a model of a system.

13.46.1 Detailed Description

The main contribution of this file is the abstract interface [prob_system_t](#) - the class representing a model of the propabilistic system

13.47 process.hh File Reference

Classes

- class [process_t](#)
Abstact interface of a class representing a process of a [system](#).

13.47.1 Detailed Description

The main contribution of this file is the abstact interface [process_t](#).

13.48 process_decomposition.hh File Reference

Classes

- class [process_decomposition_t](#)

13.48.1 Detailed Description

Process graph decomposition into SCCs including types

13.49 reporter.hh File Reference

Classes

- class [reporter_t](#)
Class for measuring and reporting.

13.49.1 Detailed Description

This file contains definition of [reporter_t](#) class (class for runtime statistics and reports)

Author:

Radek Pelanek

13.50 state.hh File Reference

Classes

- struct `state_t`
Structure representing the state of the system.

Functions

- void `byte_to_state_pos` (const `state_t` state, const `std::size_t` pos, const `byte_t` value)
• void `clear_state` (`state_t`)
Fills a memory representing the state with zeros.
- void `delete_state` (`state_t` &state)
Deletes the state.
- `state_t` `duplicate_state` (`state_t` state)
Creates a copy of state 'state' and returns a pointer to it.
- void `int_to_state_pos` (const `state_t` state, const `std::size_t` pos, const `short_int_t` value)
• `state_t` `new_state` (char *const state_memory, const `std::size_t` size)
Creates a new state and returns a pointer to it.
- `state_t` `new_state` (const `std::size_t` size)
Creates a new state and returns a pointer to it.
- bool `operator!=` (const `state_t` &arg1, const `state_t` &arg2)
Returns whether state 'arg1' is different from 'arg2' using function memcmp().
- bool `operator<` (const `state_t` &arg1, const `state_t` &arg2)
Returns whether state 'arg1' is smaller than 'arg2' using function memcmp().
- bool `operator==` (const `state_t` &arg1, const `state_t` &arg2)
Returns whether state 'arg1' is the same as 'arg2' using function memcmp().
- bool `operator>` (const `state_t` &arg1, const `state_t` &arg2)
Returns whether state 'arg1' is larger than 'arg2' using function memcmp().
- void `realloc_state` (`state_t` &state, `size_t` new_size)
Realloc state. Existing data block is deleted, and replaced with a new one of given size.

- `template<class T>`
`void set_to_state_pos (const state_t state, const std::size_t pos, const T value)`
Sets the value of type 'T' to 'state' at the position 'pos'.
- `template<>`
`void set_to_state_pos< byte_t > (const state_t state, const std::size_t pos, const byte_t value)`
- `template<>`
`void set_to_state_pos< sshort_int_t > (const state_t state, const std::size_t pos, const sshort_int_t value)`
- `template<>`
`void set_to_state_pos< ulong_int_t > (const state_t state, const std::size_t pos, const ulong_int_t value)`
- `template<>`
`void set_to_state_pos< ushort_int_t > (const state_t state, const std::size_t pos, const ushort_int_t value)`
- `template<class T>`
`T state_pos_to (const state_t state, const std::size_t pos)`
Returns the value of type 'T' stored in 'state' at the position 'pos'.
- `template<>`
`byte_t state_pos_to< byte_t > (const state_t state, const std::size_t pos)`
- `template<>`
`sshort_int_t state_pos_to< sshort_int_t > (const state_t state, const std::size_t pos)`
- `template<>`
`ulong_int_t state_pos_to< ulong_int_t > (const state_t state, const std::size_t pos)`
- `template<>`
`ushort_int_t state_pos_to< ushort_int_t > (const state_t state, const std::size_t pos)`
- `byte_t state_pos_to_byte (const state_t state, const std::size_t pos)`
- `sshort_int_t state_pos_to_int (const state_t state, const std::size_t pos)`
- `ushort_int_t state_pos_to_uint (const state_t state, const std::size_t pos)`
- `ulong_int_t state_pos_to_ulong_int (const state_t state, const std::size_t pos)`
- `void uint_to_state_pos (const state_t state, const std::size_t pos, const ushort_int_t value)`
- `void ulong_int_to_state_pos (const state_t state, const std::size_t pos, const ulong_int_t value)`

13.50.1 Detailed Description

The main contribution of this file is a class `state_t` and methods for the work with it (especially `new_state()`, `delete_state()`, `state_pos_to_*`() functions `*_to_state_pos()` functions and functions for comparing states.

13.50.2 Function Documentation

13.50.2.1 `state_t new_state (char *const state_memory, const std::size_t size)`

Creates a new state and returns a pointer to it.

Parameters:

state_memory = pointer to the memory representing a state of the system - the content will be copied to the

size = size of the memory referenced by 'state_memory' in bytes

References state_t::ptr, and state_t::size.

13.50.2.2 `void set_to_state_pos (const state_t state, const std::size_t pos, const T value) [inline]`

Sets the value of type 'T' to 'state' at the position 'pos'.

Sets the value of type *T* to *state* at position *pos*. It can be instantiated by the following types *T* :

- `byte_t`
- `sshort_int_t`
- `ushort_int_t`
- `ulong_int_t`

13.50.2.3 `T state_pos_to (const state_t state, const std::size_t pos) [inline]`

Returns the value of type 'T' stored in 'state' at the position 'pos'.

Returns the value of type *T* stored in *state* at the position *pos*. It can be instantiated by the following types *T* :

- `byte_t`
- `sshort_int_t`
- `ushort_int_t`
- `ulong_int_t`

13.51 sysopen.hh File Reference

13.51.1 Detailed Description

This header file declare class that should be used for opening models, printing help and version in divine-cluster tools.

Author:

Jiri Barnat

13.52 system.hh File Reference

Classes

- class [system_t](#)

Abstract interface of a class representing a model of a system.

Enumerations

- enum **property_type_t** {
 NONE, BUCHI, GENBUCHI, MULLER,
 RABIN, STREETT }

13.52.1 Detailed Description

The main contribution of this file is the abstract interface [system_t](#) - the class representing a model of the [system](#).

13.53 system_abilities.hh File Reference

Classes

- struct [system_abilities_t](#)

Structure storing abilities of [system](#) and its subordinate components.

13.53.1 Detailed Description

The main contribution of this file is the definition of structure [system_abilities_t](#).

13.54 `system_trans.hh` File Reference

Classes

- class `enabled_trans_container_t`
Container determined for storing enabled processes in one state.
- class `enabled_trans_t`
Class storing informations about one enabled transition.
- class `system_trans_t`
Class storing informations about one system transition.

Functions

- `enabled_trans_t * system_new_enabled_trans` (const void *params)

13.54.1 Detailed Description

This file contains:

- definition of abstract interface `system_trans_t`, which represents a transition of the `system` (composed from one or more `transition_t` - synchronized transitions)
- simple extension of `system_trans_t` called `enabled_trans_t`, which is used to represent a system transition, which is enabled (it is only extended by the possibility of being erroneous).
- definition of a container for storing of enabled transitions (based on `array_of_abstract_t`).

Index

- ~bymoc_explicit_system_t
 - bymoc_explicit_system_t, [55](#)
- ~dve_expression_t
 - dve_expression_t, [113](#)
- ~error_vector_t
 - error_vector_t, [189](#)
- ~logger_t
 - logger_t, [214](#)
- ~message_t
 - message_t, [221](#)
- ~prob_process_t
 - prob_process_t, [256](#)
- ~process_t
 - process_t, [270](#)
- abort
 - network_t, [229](#)
- accept_genbuchi_muller_set_complete
 - dve_parser_t, [121](#)
- accept_rabin_streett_first_complete
 - dve_parser_t, [121](#)
- accept_rabin_streett_pair_complete
 - dve_parser_t, [121](#)
- accept_type
 - dve_parser_t, [121](#)
- add_assertion
 - dve_process_t, [140](#)
- add_prob_transition
 - prob_process_t, [256](#)
- add_process
 - dve_symbol_table_t, [156](#)
 - dve_system_t, [165](#)
 - system_t, [289](#)
- add_state
 - dve_process_t, [140](#)
- add_transition
 - bymoc_process_t, [60](#)
 - dve_process_t, [140](#)
 - process_t, [270](#)
- add_variable
 - dve_process_t, [141](#)
- all_gather
 - network_t, [229](#)
- alloc_glob_mask
 - dve_transition_t, [177](#)
- alloc_mem
 - bit_string_t, [50](#)
- ample_set
 - por_t, [247](#)
- ample_set_succs
 - por_t, [248](#)
- app_by_ref
 - explicit_storage_t, [196](#)
- append_bool
 - message_t, [221](#)
- append_data
 - message_t, [221](#)
- append_new_enabled
 - dve_explicit_system_t, [98](#)
- append_new_enabled_prop_sync
 - dve_explicit_system_t, [98](#)
- append_state
 - message_t, [221](#)
- arity
 - dve_expression_t, [114](#)
- array.hh, [309](#)
- array_of_abstract_t, [35](#)
 - array_of_abstract_t, [38](#)
 - array_of_abstract_t, [38](#)
 - assign_from, [38](#)
 - begin, [38](#)
 - clear, [39](#)
 - const_iterator, [38](#)
 - end, [39](#)
 - extend, [39](#)
 - extend_to, [39](#)
 - get_alloc_step, [39](#)
 - get_allocated, [40](#)
 - iterator, [38](#)
 - last, [40](#)
 - pop_back, [40](#)
 - push_back, [40](#)

- resize, 40
- set_alloc_step, 41
- shrink_to, 41
- swap, 41
- array_t, 42
 - array_t, 45
 - array_t, 45
 - assign_from, 45
 - begin, 45
 - clear, 45
 - const_iterator, 44
 - end, 46
 - extend, 46
 - extend_to, 46
 - get_alloc_step, 46
 - get_allocated, 46
 - iterator, 44
 - last, 47
 - pop_back, 47
 - push_back, 47
 - resize, 47
 - set_alloc_step, 48
 - shrink_to, 48
 - swap, 48
- assign_from
 - array_of_abstract_t, 38
 - array_t, 45
- barrier
 - network_t, 229
- begin
 - array_of_abstract_t, 38
 - array_t, 45
- bit_string.hh, 310
 - bit_values, 310
- bit_string_t, 49
 - alloc_mem, 50
 - bit_string_t, 50
 - bit_string_t, 50
 - clear, 51
 - DBG_print, 51
 - get_allocated_4bytes_count, 51
- bit_values
 - bit_string.hh, 310
- bymoc_enabled_trans_t, 52
- bymoc_explicit_system.hh, 311
- bymoc_explicit_system_t, 53
 - ~bymoc_explicit_system_t, 55
 - bymoc_explicit_system_t, 55
 - bymoc_explicit_system_t, 55
- get_preallocation_count, 55
- is_erroneous, 55
- print_state, 55
- violated_assertion_count, 56
- violated_assertion_string, 56
- violates_assertion, 56
- bymoc_expression.hh, 312
- bymoc_expression_t, 57
 - swap, 58
- bymoc_process.hh, 313
- bymoc_process_decomposition.hh, 314
- bymoc_process_t, 59
 - add_transition, 60
- bymoc_system.hh, 315
- bymoc_system_t, 61
 - bymoc_system_t, 63
 - bymoc_system_t, 63
- bymoc_system_trans.hh, 316
- bymoc_system_trans_t, 64
- bymoc_transition.hh, 317
- bymoc_transition_t, 66
- can_be_modified
 - system_t, 289
 - transition_t, 301
- can_evaluate_expressions
 - explicit_system_t, 203
- can_read
 - transition_t, 301
- can_system_transitions
 - explicit_system_t, 203
- channel_content_count
 - dve_explicit_system_t, 98
- CHANNEL_UNUSED
 - dve_symbol_t, 153
- check_restrictions_put_on_property
 - dve_parser_t, 121
- clear
 - array_of_abstract_t, 39
 - array_t, 45
 - bit_string_t, 51
 - divine::compressor_t, 76
- collect_and_print
 - distr_reporter_t, 80
- comm_matrix_t, 67
 - comm_matrix_t, 68
 - comm_matrix_t, 68
 - getcolcount, 68
 - getrowcount, 68
 - operator*, 69

- operator(), 68
- operator+, 69
- operator-, 69, 70
- compacted_t, 71
 - create_gid, 72
 - create_val, 72
 - first, 72
 - get_arity, 72
 - get_gid, 72
 - get_operator, 73
 - get_value, 73
 - join, 73
 - last, 73
 - left, 73
 - right, 73
 - to_string, 74
- compacted_viewer_t, 75
- compress
 - divine::compressor_t, 76
- compressor.hh, 318
- compute_enabled_of_property
 - dve_explicit_system_t, 98
- compute_enabled_stage1
 - dve_explicit_system_t, 99
- compute_enabled_stage2
 - dve_explicit_system_t, 99
- compute_successors_without_sync
 - dve_explicit_system_t, 99
- const_data
 - updateable_info_t, 305
- const_iterator
 - array_of_abstract_t, 38
 - array_t, 44
- create_error_state
 - dve_explicit_system_t, 100
- create_gid
 - compacted_t, 72
- create_init_expr_field
 - dve_symbol_t, 151
- create_val
 - compacted_t, 72
- data.hh, 319
- data_t, 78
- DBG_print
 - bit_string_t, 51
- DBG_print_all_initialized_variables
 - dve_system_t, 165
- DBG_print_state
 - dve_explicit_system_t, 100
- DBG_print_state_CR
 - dve_explicit_system_t, 100
- decompress
 - divine::compressor_t, 76
- delete_all_states
 - explicit_storage_t, 196
- delete_by_ref
 - explicit_storage_t, 196
- distr_reporter.hh, 320
- distr_reporter_t, 79
 - collect_and_print, 80
 - set_info, 80
- distributed.hh, 321
- distributed_t, 81
 - distributed_t, 83
 - distributed_t, 83
 - get_all_received_sync_msgs_cnt, 83
 - get_all_sent_sync_msgs_cnt, 83
 - get_all_sync_barriers_cnt, 83
 - get_comm_matrix_rsm, 83
 - get_comm_matrix_ssm, 84
 - get_proc_msgs_buf_excludeve_ - mem, 84
 - is_manager, 84
 - network_initialize, 84
 - partition_function, 85
 - process_messages, 85
 - process_user_message, 87
 - set_busy, 86
 - set_idle, 86
 - set_proc_msgs_buf_exclusive_mem, 86
 - synchronized, 86
- divine::compressor_t, 76
 - clear, 76
 - compress, 76
 - decompress, 76
 - init, 77
- dve_parser_t
 - EXPRESSION, 121
 - PROB_TRANSITION, 120
 - PROCESS, 120
 - SYSTEM, 120
 - TRANSITION, 121
- dve_transition.hh
 - SYNC_ASK, 340
 - SYNC_ASK_BUFFER, 340
 - SYNC_EXCLAIM, 340
 - SYNC_EXCLAIM_BUFFER, 340
 - SYNC_NO_SYNC, 340

- dve_commonparse.hh, 322
 - dve_eerror, 323
 - dve_perror, 323
 - dve_tterror, 323
 - dve_yyerror, 323
- dve_eerror
 - dve_commonparse.hh, 323
- dve_enabled_trans_t, 89
 - dve_enabled_trans_t, 89
 - dve_enabled_trans_t, 89
 - operator=, 89
- dve_explicit_system.hh, 324
 - dve_state_int_t, 324
 - ES_FMT_PRINT_ALL_NAMES, 325
- dve_explicit_system_t, 91
 - append_new_enabled, 98
 - append_new_enabled_prop_sync, 98
 - channel_content_count, 98
 - compute_enabled_of_property, 98
 - compute_enabled_stage1, 99
 - compute_enabled_stage2, 99
 - compute_successors_without_sync, 99
 - create_error_state, 100
 - DBG_print_state, 100
 - DBG_print_state_CR, 100
 - dve_explicit_system_t, 97
 - dve_explicit_system_t, 97
 - eval_expr, 101
 - get_async_enabled_trans_succ, 101
 - get_async_enabled_trans_succ_ - without_property, 101
 - get_enabled_ith_trans, 102
 - get_enabled_trans_count, 102
 - get_ith_succ, 102
 - get_property_trans, 103
 - get_receiving_trans, 103
 - get_sending_or_normal_trans, 103
 - get_state_creator_value_of_var_ - type, 104
 - get_state_of_process, 104
 - get_succs, 104
 - get_sync_enabled_trans_succ, 105
 - go_to_error, 105
 - is_accepting, 105
 - not_in_glob_conflict, 106
 - passed_through, 106
 - read, 106
 - set_state_creator_value, 107
 - set_state_creator_value_extended, 107
 - set_state_creator_value_of_var_ - type, 107
 - set_var_value, 108
- dve_explicit_system_t::state_creator_t, 109
- dve_expression.hh, 326
- dve_expression_t, 110
 - ~dve_expression_t, 113
 - arity, 114
 - dve_expression_t, 113
 - dve_expression_t, 113
 - get_ident_gid, 114
 - get_operator, 114
 - get_value, 114
 - set_arity, 114
 - set_ident_gid, 114
 - set_operator, 115
 - set_value, 115
- dve_grammar.hh, 327
- dve_parser.hh, 328
- dve_parser_t, 116
 - accept_genbuchi_muller_set_ - complete, 121
 - accept_rabin_streett_first_complete, 121
 - accept_rabin_streett_pair_complete, 121
 - accept_type, 121
 - check_restrictions_put_on_property, 121
 - expr_array_mem, 121
 - expr_assign, 122
 - expr_bin, 122
 - expr_false, 122
 - expr_id, 122
 - expr_nat, 122
 - expr_parenthesis, 122
 - expr_state_of_process, 123
 - expr_true, 123
 - expr_unary, 123
 - expr_var_of_process, 123
 - get_expression, 123
 - get_mode, 123
 - parser_mode_t, 120
 - set_fpos, 123
 - set_lpos, 124
 - state_accept, 124

- state_genbuchi_muller_accept, 124
- state_list_done, 124
- state_rabin_streett_accept, 124
- trans_effect_part, 124
- type_is_const, 124
- type_list_clear, 124
- var_decl_array_size, 125
- var_decl_begin, 125
- var_decl_cancel, 125
- var_decl_create, 125
- var_decl_done, 125
- var_init_field_part, 125
- var_init_is_field, 125
- dve_position_t, 127
- dve_pperror
 - dve_commonparse.hh, 323
- dve_prob_explicit_system.hh, 329
- dve_prob_explicit_system_t, 128
 - dve_prob_explicit_system_t, 129
 - dve_prob_explicit_system_t, 129
 - get_succs, 129
 - read, 130
- dve_prob_process.hh, 330
- dve_prob_process_t, 131
- dve_prob_system.hh, 331
- dve_prob_transition.hh, 332
- dve_prob_transition_t, 133
- dve_process.hh, 333
 - DVE_PROCESS_ALLOC_STEP, 333
- DVE_PROCESS_ALLOC_STEP
 - dve_process.hh, 333
- dve_process_decomposition.hh, 334
- dve_process_decomposition_t, 135
 - get_process_scc_id, 135
 - get_process_scc_type, 135
 - get_scc_count, 135
 - get_scc_type, 136
 - is_weak, 136
 - parse_process, 136
- dve_process_t, 137
 - add_assertion, 140
 - add_state, 140
 - add_transition, 140
 - add_variable, 141
 - dve_process_t, 140
 - dve_process_t, 140
 - get_acceptance, 141
 - get_accepting_group_count, 141
 - get_assertion, 141
 - get_trans_count, 141
 - get_transition, 142
 - set_acceptance, 142
 - set_initial_state, 143
- dve_source_position.hh, 335
- dve_source_position_t, 144
 - get_source_pos, 145
 - set_source_pos, 145
- dve_state_int_t
 - dve_explicit_system.hh, 324
- dve_symbol_t, 147
 - CHANNEL_UNUSED, 153
 - create_init_expr_field, 151
 - dve_symbol_t, 151
 - dve_symbol_t, 151
 - get_channel_item_count, 151
 - get_channel_type_list_item, 151
 - get_channel_type_list_size, 151
 - get_init_expr, 151, 152
 - get_lid, 152
 - get_process_gid, 152
 - is_byte, 152
 - is_int, 152
 - no_init_expr_field, 153
 - set_channel_item_count, 153
 - set_channel_type_list_item, 153
 - set_channel_type_list_size, 153
- dve_symbol_table.hh, 336
- dve_symbol_table_t, 154
 - add_process, 156
 - dve_symbol_table_t, 156
 - dve_symbol_table_t, 156
 - find_global_symbol, 156
 - find_symbol, 156
 - find_visible_symbol, 157
 - found_global_symbol, 157
 - found_symbol, 157
 - get_channel_count, 158
 - get_process_count, 158
 - get_state_count, 158
 - get_symbol, 158
 - get_symbol_count, 158
 - get_variable_count, 159
 - save_token, 159
- dve_system.hh, 337
 - system_synchronicity_t, 337
- dve_system_t, 160
 - add_process, 165
 - DBG_print_all_initialized_variables, 165

- dve_system_t, 165
- dve_system_t, 165
- eval_dot, 168
- eval_expr, 165
- eval_id, 168
- eval_square_bracket, 169
- fast_eval, 166
- get_channel_freq_ask, 166
- get_channel_freq_exclaim, 166
- get_global_variable_gid, 166
- get_property_scc_count, 167
- get_property_scc_id, 167
- get_property_scc_type, 167
- get_symbol_table, 167
- get_system_synchro, 168
- get_trans_count, 168
- initial_values_counts, 169
- is_property_weak, 168
- parameters, 169
- set_property_with_synchro, 168
- dve_system_trans.hh, 338
- dve_system_trans_t, 170
- dve_token_vector.hh, 339
- dve_token_vector_t, 172
- dve_transition.hh, 340
 - sync_mode_t, 340
 - TR_effects_alloc_step, 341
 - TR_effects_default_alloc, 341
- dve_transition_t, 173
 - alloc_glob_mask, 177
 - dve_transition_t, 177
 - dve_transition_t, 177
 - get_channel_gid, 177
 - get_effect, 177
 - get_glob_mask, 177
 - get_guard, 178
 - get_guard_string, 178
 - get_partial_id, 178
 - get_symbol_table, 178
 - get_sync_channel_name, 179
 - get_sync_expr_list_item, 179
 - get_sync_expr_string, 179
 - get_valid, 180
 - set_partial_id, 180
 - set_process_gid, 180
- dve_tterror
 - dve_commonparse.hh, 323
- dve_yyerror
 - dve_commonparse.hh, 323
- enabled_trans_container_t, 181
 - get_begin, 182
- enabled_trans_t, 183
- end
 - array_of_abstract_t, 39
 - array_t, 46
- ERR_char_string_t
 - error.hh, 344
- ERR_default_psh_callback
 - error.hh, 344
- ERR_default_thr_callback
 - error.hh, 344
- ERR_psh_callback_t
 - error.hh, 344
- ERR_thr_callback_t
 - error.hh, 344
- ERR_throw_t, 184
- ERR_triplet_t, 185
- ERR_UNKNOWN_ID
 - error.hh, 345
- ERR_UNKNOWN_TYPE
 - error.hh, 345
- error.hh, 342
 - ERR_char_string_t, 344
 - ERR_default_psh_callback, 344
 - ERR_default_thr_callback, 344
 - ERR_psh_callback_t, 344
 - ERR_thr_callback_t, 344
 - ERR_UNKNOWN_ID, 345
 - ERR_UNKNOWN_TYPE, 345
 - gerr, 345
 - UNIMPLEMENTED, 343
- error_string_t, 186
- error_vector_t, 187
 - ~error_vector_t, 189
 - error_vector_t, 189
 - error_vector_t, 189
 - flush, 190
 - id, 190
 - id_front, 190
 - operator<<, 190
 - perror, 191
 - perror_back, 191
 - perror_front, 191
 - pop, 191
 - pop_back, 192
 - pop_front, 192
 - push, 192
 - string, 192
 - string_front, 192

- that, [192, 193](#)
- ES_FMT_PRINT_ALL_NAMES
 - dve_explicit_system.hh, [325](#)
- ES_parameters_t, [194](#)
- eval_dot
 - dve_system_t, [168](#)
- eval_expr
 - dve_explicit_system_t, [101](#)
 - dve_system_t, [165](#)
 - explicit_system_t, [203](#)
- eval_id
 - dve_system_t, [168](#)
- eval_square_bracket
 - dve_system_t, [169](#)
- explicit_storage.hh, [347](#)
- explicit_storage_t, [195](#)
 - app_by_ref, [196](#)
 - delete_all_states, [196](#)
 - delete_by_ref, [196](#)
 - explicit_storage_t, [196](#)
 - explicit_storage_t, [196](#)
 - get_app_by_ref, [196](#)
 - get_coltables, [197](#)
 - get_ht_occupancy, [197](#)
 - get_max_coltable, [197](#)
 - get_mem_max_used, [197](#)
 - get_mem_used, [197](#)
 - get_states_max_stored, [197](#)
 - get_states_stored, [197](#)
 - init, [197](#)
 - insert, [198](#)
 - is_stored, [198](#)
 - is_stored_if_not_insert, [198](#)
 - reconstruct, [198](#)
 - set_app_by_ref, [199](#)
 - set_appendix, [199](#)
 - set_appendix_size, [199](#)
 - set_col_init_size, [199](#)
 - set_col_resize, [199](#)
 - set_compression_method, [199](#)
 - set_hash_function, [199](#)
 - set_ht_size, [199](#)
 - set_mem_limit, [200](#)
- explicit_system.hh, [348](#)
 - SUCC_DEADLOCK, [350](#)
 - SUCC_ERROR, [350](#)
 - SUCC_NORMAL, [350](#)
 - succs_deadlock, [349](#)
 - succs_error, [349](#)
 - succs_normal, [349](#)
- explicit_system_t, [201](#)
 - can_evaluate_expressions, [203](#)
 - can_system_transitions, [203](#)
 - eval_expr, [203](#)
 - explicit_system_t, [203](#)
 - explicit_system_t, [203](#)
 - get_enabled_ith_trans, [204](#)
 - get_enabled_trans, [204](#)
 - get_enabled_trans_count, [204](#)
 - get_enabled_trans_succ, [205](#)
 - get_enabled_trans_succs, [205](#)
 - get_initial_state, [206](#)
 - get_preallocation_count, [206](#)
 - get_succs, [206](#)
 - is_accepting, [207](#)
 - is_erroneous, [207](#)
 - new_enabled_trans, [208](#)
 - violated_assertion_count, [208](#)
 - violated_assertion_string, [208](#)
 - violates_assertion, [208](#)
- expr_array_mem
 - dve_parser_t, [121](#)
- expr_assign
 - dve_parser_t, [122](#)
- expr_bin
 - dve_parser_t, [122](#)
- expr_false
 - dve_parser_t, [122](#)
- expr_id
 - dve_parser_t, [122](#)
- expr_nat
 - dve_parser_t, [122](#)
- expr_parenthesis
 - dve_parser_t, [122](#)
- expr_state_of_process
 - dve_parser_t, [123](#)
- expr_true
 - dve_parser_t, [123](#)
- expr_unary
 - dve_parser_t, [123](#)
- expr_var_of_process
 - dve_parser_t, [123](#)
- EXPRESSION
 - dve_parser_t, [121](#)
- expression.hh, [351](#)
- expression_t, [210](#)
 - expression_t, [211](#)
 - expression_t, [211](#)
 - from_string, [211](#)
 - operator=, [212](#)

- read, 212
- to_string, 212
- extend
 - array_of_abstract_t, 39
 - array_t, 46
- extend_to
 - array_of_abstract_t, 39
 - array_t, 46
- fast_eval
 - dve_system_t, 166
- find_global_symbol
 - dve_symbol_table_t, 156
- find_symbol
 - dve_symbol_table_t, 156
- find_visible_symbol
 - dve_symbol_table_t, 157
- first
 - compacted_t, 72
- flush
 - error_vector_t, 190
- flush_all_buffers
 - network_t, 230
- flush_all_buffers_timed_out_only
 - network_t, 230
- flush_buffer
 - network_t, 230
- flush_some_buffers
 - network_t, 231
- found_global_symbol
 - dve_symbol_table_t, 157
- found_symbol
 - dve_symbol_table_t, 157
- from_string
 - expression_t, 211
 - process_t, 271
 - system_t, 289
 - transition_t, 301
- gather
 - network_t, 231
- generate_ample_sets
 - por_t, 248
- generate_composed_ample_sets
 - por_t, 249
- gerr
 - error.hh, 345
- get_abilities
 - system_t, 290
- get_acceptance
 - dve_process_t, 141
- get_accepting_group_count
 - dve_process_t, 141
- get_all_barriers_cnt
 - network_t, 232
- get_all_buffers_flushes_cnt
 - network_t, 232
- get_all_received_msgs_cnt
 - network_t, 232
- get_all_received_sync_msgs_cnt
 - distributed_t, 83
- get_all_sent_msgs_cnt
 - network_t, 233
- get_all_sent_sync_msgs_cnt
 - distributed_t, 83
- get_all_sync_barriers_cnt
 - distributed_t, 83
- get_all_total_pending_size
 - network_t, 233
- get_alloc_step
 - array_of_abstract_t, 39
 - array_t, 46
- get_allocated
 - array_of_abstract_t, 40
 - array_t, 46
- get_allocated_4bytes_count
 - bit_string_t, 51
- get_app_by_ref
 - explicit_storage_t, 196
- get_arity
 - compacted_t, 72
- get_assertion
 - dve_process_t, 141
- get_async_enabled_trans_succ
 - dve_explicit_system_t, 101
- get_async_enabled_trans_succ_without_
 - property
 - dve_explicit_system_t, 101
- get_begin
 - enabled_trans_container_t, 182
- get_buf_msgs_cnt_limit
 - network_t, 233
- get_buf_size_limit
 - network_t, 234
- get_buf_time_limit
 - network_t, 234
- get_buffer_flushes_cnt
 - network_t, 235
- get_channel_count
 - dve_symbol_table_t, 158

- get_channel_freq_ask
 - dve_system_t, 166
- get_channel_freq_exclaim
 - dve_system_t, 166
- get_channel_gid
 - dve_transition_t, 177
- get_channel_item_count
 - dve_symbol_t, 151
- get_channel_type_list_item
 - dve_symbol_t, 151
- get_channel_type_list_size
 - dve_symbol_t, 151
- get_cluster_size
 - network_t, 235
- get_coltables
 - explicit_storage_t, 197
- get_comm_matrix_rnm
 - network_t, 236
- get_comm_matrix_rsm
 - distributed_t, 83
- get_comm_matrix_rum
 - network_t, 236
- get_comm_matrix_snm
 - network_t, 236
- get_comm_matrix_ssm
 - distributed_t, 84
- get_comm_matrix_sum
 - network_t, 236
- get_count
 - system_trans_t, 296
- get_effect
 - dve_transition_t, 177
- get_enabled_ith_trans
 - dve_explicit_system_t, 102
 - explicit_system_t, 204
- get_enabled_trans
 - explicit_system_t, 204
- get_enabled_trans_count
 - dve_explicit_system_t, 102
 - explicit_system_t, 204
- get_enabled_trans_succ
 - explicit_system_t, 205
- get_enabled_trans_succs
 - explicit_system_t, 205
- get_error_vector
 - process_t, 271
 - transition_t, 302
- get_expression
 - dve_parser_t, 123
- get_gid
 - compact_t, 72
 - transition_t, 302
- get_glob_mask
 - dve_transition_t, 177
- get_global_variable_gid
 - dve_system_t, 166
- get_guard
 - dve_transition_t, 178
- get_guard_string
 - dve_transition_t, 178
- get_ht_occupancy
 - explicit_storage_t, 197
- get_id
 - network_t, 236
- get_ident_gid
 - dve_expression_t, 114
- get_index_of_trans_in_prob_trans
 - prob_system_t, 261
- get_init_expr
 - dve_symbol_t, 151, 152
- get_initial_state
 - explicit_system_t, 206
- get_ith_succ
 - dve_explicit_system_t, 102
- get_lid
 - dve_symbol_t, 152
 - transition_t, 302
- get_max_coltable
 - explicit_storage_t, 197
- get_mem_max_used
 - explicit_storage_t, 197
- get_mem_used
 - explicit_storage_t, 197
- get_mode
 - dve_parser_t, 123
- get_operator
 - compact_t, 73
 - dve_expression_t, 114
- get_partial_id
 - dve_transition_t, 178
- get_preallocation_count
 - bymoc_explicit_system_t, 55
 - explicit_system_t, 206
- get_prob_trans_of_trans
 - prob_system_t, 261
- get_prob_transition
 - prob_process_t, 256
- get_proc_msgs_buf_excluseve_mem
 - distributed_t, 84
- get_process

- system_t, 290
- get_process_count
 - dve_symbol_table_t, 158
 - system_t, 291
- get_process_gid
 - dve_symbol_t, 152
- get_process_scc_id
 - dve_process_decomposition_t, 135
 - process_decomposition_t, 266
- get_process_scc_type
 - dve_process_decomposition_t, 135
 - process_decomposition_t, 266
- get_processor_name
 - network_t, 237
- get_property_gid
 - system_t, 291
- get_property_process
 - system_t, 291
- get_property_scc_count
 - dve_system_t, 167
- get_property_scc_id
 - dve_system_t, 167
- get_property_scc_type
 - dve_system_t, 167
- get_property_trans
 - dve_explicit_system_t, 103
- get_property_type
 - system_t, 291
- get_receiving_trans
 - dve_explicit_system_t, 103
- get_rcv_msgs_cnt_rcv_from
 - network_t, 237, 238
- get_scc_count
 - dve_process_decomposition_t, 135
 - process_decomposition_t, 266
- get_scc_type
 - dve_process_decomposition_t, 136
 - process_decomposition_t, 267
- get_sending_or_normal_trans
 - dve_explicit_system_t, 103
- get_sent_msgs_cnt_sent_to
 - network_t, 238
- get_source_pos
 - dve_source_position_t, 145
- get_state_count
 - dve_symbol_table_t, 158
- get_state_creator_value_of_var_type
 - dve_explicit_system_t, 104
- get_state_of_process
 - dve_explicit_system_t, 104
- get_states_max_stored
 - explicit_storage_t, 197
- get_states_stored
 - explicit_storage_t, 197
- get_succs
 - dve_explicit_system_t, 104
 - dve_prob_explicit_system_t, 129
 - explicit_system_t, 206
 - prob_explicit_system_t, 253
- get_symbol
 - dve_symbol_table_t, 158
- get_symbol_count
 - dve_symbol_table_t, 158
- get_symbol_table
 - dve_system_t, 167
 - dve_transition_t, 178
- get_sync_channel_name
 - dve_transition_t, 179
- get_sync_enabled_trans_succ
 - dve_explicit_system_t, 105
- get_sync_expr_list_item
 - dve_transition_t, 179
- get_sync_expr_string
 - dve_transition_t, 179
- get_system_synchro
 - dve_system_t, 168
- get_time
 - reporter_t, 276
- get_total_pending_size
 - network_t, 239
- get_trans_count
 - dve_process_t, 141
 - dve_system_t, 168
 - prob_transition_t, 264
 - process_t, 271
 - system_t, 292
- get_transition
 - dve_process_t, 142
 - prob_transition_t, 264
 - process_t, 271
 - system_t, 292
- get_user_received_msgs_cnt
 - network_t, 239
- get_user_sent_msgs_cnt
 - network_t, 239
- get_valid
 - dve_transition_t, 180
- get_value
 - compact_t, 73
 - dve_expression_t, 114

- get_variable_count
 - dve_symbol_table_t, 159
- get_weight_sum
 - prob_transition_t, 264
- get_written_size
 - message_t, 221
- getcolcount
 - comm_matrix_t, 68
- getrowcount
 - comm_matrix_t, 68
- go_to_error
 - dve_explicit_system_t, 105
- hash_function.hh, 352
- hash_function_t, 213
- huffman.hh, 353
- id
 - error_vector_t, 190
- id_front
 - error_vector_t, 190
- init
 - divine::compressor_t, 77
 - explicit_storage_t, 197
 - logger_t, 215
 - por_t, 250
- initial_values_counts
 - dve_system_t, 169
 - SYS_parameters_t, 283
- initialize_buffers
 - network_t, 240
- initialize_network
 - network_t, 240
- insert
 - explicit_storage_t, 198
- inttostr.hh, 354
- invalidate
 - state_ref_t, 278
- is_accepting
 - dve_explicit_system_t, 105
 - explicit_system_t, 207
- is_byte
 - dve_symbol_t, 152
- is_erroneous
 - bymoc_explicit_system_t, 55
 - explicit_system_t, 207
- is_int
 - dve_symbol_t, 152
- is_manager
 - distributed_t, 84
- is_new_message
 - network_t, 241
- is_new_message_from_source
 - network_t, 241
- is_new_urgent_message
 - network_t, 241
- is_new_urgent_message_from_source
 - network_t, 241
- is_property_weak
 - dve_system_t, 168
- is_stored
 - explicit_storage_t, 198
- is_stored_if_not_insert
 - explicit_storage_t, 198
- is_valid
 - state_ref_t, 278
- is_weak
 - dve_process_decomposition_t, 136
 - process_decomposition_t, 267
- iterator
 - array_of_abstract_t, 38
 - array_t, 44
- join
 - compact_t, 73
- last
 - array_of_abstract_t, 40
 - array_t, 47
 - compact_t, 73
- left
 - compact_t, 73
- load_data
 - message_t, 221
- log_now
 - logger_t, 215
- logger.hh, 355
- logger_t, 214
 - ~logger_t, 214
 - init, 215
 - log_now, 215
 - logger_t, 214
 - logger_t, 214
 - register_double, 215
 - register_int, 216
 - register_slong_int, 216
 - register_ulong_int, 216
 - register_unsigned, 216
 - set_storage, 217
 - stop_SIGALRM, 217

- use_SIGALRM, 217
- mcr12_explicit_system.hh, 356
- mcr12_system.hh, 357
- message_t, 218
 - ~message_t, 221
 - append_bool, 221
 - append_data, 221
 - append_state, 221
 - get_written_size, 221
 - load_data, 221
 - message_t, 220
 - message_t, 220
 - read_data, 222
 - read_state, 222
 - rewind, 222
 - rewind_append, 222
 - rewind_read, 223
 - set_data, 223
 - set_written_size, 223
- NET_TAG_NORMAL
 - network.hh, 360
- NET_TAG_URGENT
 - network.hh, 360
- network.hh, 358
 - NET_TAG_NORMAL, 360
 - NET_TAG_URGENT, 360
 - pcomm_matrix_t, 360
- network_initialize
 - distributed_t, 84
- network_t, 225
 - abort, 229
 - all_gather, 229
 - barrier, 229
 - flush_all_buffers, 230
 - flush_all_buffers_timed_out_only, 230
 - flush_buffer, 230
 - flush_some_buffers, 231
 - gather, 231
 - get_all_barriers_cnt, 232
 - get_all_buffers_flushes_cnt, 232
 - get_all_received_msgs_cnt, 232
 - get_all_sent_msgs_cnt, 233
 - get_all_total_pending_size, 233
 - get_buf_msgs_cnt_limit, 233
 - get_buf_size_limit, 234
 - get_buf_time_limit, 234
 - get_buffer_flushes_cnt, 235
 - get_cluster_size, 235
 - get_comm_matrix_rnm, 236
 - get_comm_matrix_rum, 236
 - get_comm_matrix_snm, 236
 - get_comm_matrix_sum, 236
 - get_id, 236
 - get_processor_name, 237
 - get_rcv_msgs_cnt_rcv_from, 237, 238
 - get_sent_msgs_cnt_sent_to, 238
 - get_total_pending_size, 239
 - get_user_received_msgs_cnt, 239
 - get_user_sent_msgs_cnt, 239
 - initialize_buffers, 240
 - initialize_network, 240
 - is_new_message, 241
 - is_new_message_from_source, 241
 - is_new_urgent_message, 241
 - is_new_urgent_message_from_source, 241
 - network_t, 229
 - network_t, 229
 - receive_message, 242
 - receive_message_from_source, 243
 - receive_message_non_exc, 243
 - receive_urgent_message, 243
 - receive_urgent_message_from_source, 243
 - receive_urgent_message_non_exc, 243
 - send_message, 244
 - send_urgent_message, 245
- new_enabled_trans
 - explicit_system_t, 208
- new_state
 - state.hh, 371
- no_init_expr_field
 - dve_symbol_t, 153
- not_in_glob_conflict
 - dve_explicit_system_t, 106
- operator<<
 - error_vector_t, 190
- operator*
 - comm_matrix_t, 69
- operator()
 - comm_matrix_t, 68
- operator+
 - comm_matrix_t, 69
- operator-

- comm_matrix_t, 69, 70
- operator=
 - dve_enabled_trans_t, 89
 - expression_t, 212
 - system_trans_t, 296
- parameters
 - dve_system_t, 169
- parse_process
 - dve_process_decomposition_t, 136
 - process_decomposition_t, 267
- parser_mode_t
 - dve_parser_t, 120
- partition_function
 - distributed_t, 85
- passed_through
 - dve_explicit_system_t, 106
- path.hh, 361
 - PATH_CYCLE_SEPARATOR, 361
- PATH_CYCLE_SEPARATOR
 - path.hh, 361
- pcomm_matrix_t
 - network.hh, 360
- perror
 - error_vector_t, 191
- perror_back
 - error_vector_t, 191
- perror_front
 - error_vector_t, 191
- pop
 - error_vector_t, 191
- pop_back
 - array_of_abstract_t, 40
 - array_t, 47
 - error_vector_t, 192
- pop_front
 - error_vector_t, 192
- por.hh, 362
- por_t, 246
 - ample_set, 247
 - ample_set_succs, 248
 - generate_ample_sets, 248
 - generate_composed_ample_sets, 249
 - init, 250
- pproc_terr
 - process_t, 272
- print
 - reporter_t, 276
- print_state
 - bymoc_explicit_system_t, 55
- PROB_TRANSITION
 - dve_parser_t, 120
- prob_and_property_trans_t, 251
 - property_trans_gid, 251
- prob_explicit_system.hh, 363
- prob_explicit_system_t, 252
 - get_succs, 253
 - prob_explicit_system_t, 253
 - prob_explicit_system_t, 253
- prob_process.hh, 364
- prob_process_t, 255
 - ~prob_process_t, 256
 - add_prob_transition, 256
 - get_prob_transition, 256
 - prob_process_t, 256
 - prob_process_t, 256
 - remove_prob_transition, 257
- prob_succ_container_t, 258
 - prob_succ_container_t, 258
 - prob_succ_container_t, 258
- prob_succ_element_t, 259
- prob_system.hh, 365
- prob_system_t, 260
 - get_index_of_trans_in_prob_trans, 261
 - get_prob_trans_of_trans, 261
 - prob_system_t, 261
 - prob_system_t, 261
- prob_transition_t, 263
 - get_trans_count, 264
 - get_transition, 264
 - get_weight_sum, 264
 - set_trans_count, 264
 - set_transition_and_weight, 265
- PROCESS
 - dve_parser_t, 120
- process.hh, 366
- process_decomposition.hh, 367
- process_decomposition_t, 266
 - get_process_scc_id, 266
 - get_process_scc_type, 266
 - get_scc_count, 266
 - get_scc_type, 267
 - is_weak, 267
 - parse_process, 267
- process_messages
 - distributed_t, 85
- process_t, 268
 - ~process_t, 270

- add_transition, [270](#)
- from_string, [271](#)
- get_error_vector, [271](#)
- get_trans_count, [271](#)
- get_transition, [271](#)
- pproc_terr, [272](#)
- process_t, [270](#)
- process_t, [270](#)
- read, [272](#)
- remove_transition, [272](#)
- set_error_vector, [272](#)
- process_user_message
 - distributed_t, [87](#)
- property_trans_gid
 - prob_and_property_trans_t, [251](#)
- psh, [274](#)
- ptrans_terr
 - transition_t, [304](#)
- push
 - error_vector_t, [192](#)
- push_back
 - array_of_abstract_t, [40](#)
 - array_t, [47](#)
- read
 - dve_explicit_system_t, [106](#)
 - dve_prob_explicit_system_t, [130](#)
 - expression_t, [212](#)
 - process_t, [272](#)
 - system_t, [292](#), [293](#)
 - transition_t, [302](#)
- read_data
 - message_t, [222](#)
- read_state
 - message_t, [222](#)
- receive_message
 - network_t, [242](#)
- receive_message_from_source
 - network_t, [243](#)
- receive_message_non_exc
 - network_t, [243](#)
- receive_urgent_message
 - network_t, [243](#)
- receive_urgent_message_from_source
 - network_t, [243](#)
- receive_urgent_message_non_exc
 - network_t, [243](#)
- reconstruct
 - explicit_storage_t, [198](#)
- register_double
 - logger_t, [215](#)
- register_int
 - logger_t, [216](#)
- register_slong_int
 - logger_t, [216](#)
- register_ulong_int
 - logger_t, [216](#)
- register_unsigned
 - logger_t, [216](#)
- remove_prob_transition
 - prob_process_t, [257](#)
- remove_process
 - system_t, [293](#)
- remove_transition
 - process_t, [272](#)
- reporter.hh, [368](#)
- reporter_t, [275](#)
 - get_time, [276](#)
 - print, [276](#)
 - set_alg_name, [276](#)
 - set_file_name, [276](#)
 - set_info, [276](#)
 - set_obligatory_keys, [276](#)
 - set_problem, [276](#)
 - set_states_stored, [277](#)
 - set_succs_calls, [277](#)
 - start_timer, [277](#)
 - stop_timer, [277](#)
- resize
 - array_of_abstract_t, [40](#)
 - array_t, [47](#)
- rewind
 - message_t, [222](#)
- rewind_append
 - message_t, [222](#)
- rewind_read
 - message_t, [223](#)
- right
 - compacted_t, [73](#)
- save_token
 - dve_symbol_table_t, [159](#)
- send_message
 - network_t, [244](#)
- send_urgent_message
 - network_t, [245](#)
- set_acceptance
 - dve_process_t, [142](#)
- set_alg_name
 - reporter_t, [276](#)

- set_alloc_step
 - array_of_abstract_t, 41
 - array_t, 48
- set_app_by_ref
 - explicit_storage_t, 199
- set_appendix
 - explicit_storage_t, 199
- set_appendix_size
 - explicit_storage_t, 199
- set_arity
 - dve_expression_t, 114
- set_busy
 - distributed_t, 86
- set_channel_item_count
 - dve_symbol_t, 153
- set_channel_type_list_item
 - dve_symbol_t, 153
- set_channel_type_list_size
 - dve_symbol_t, 153
- set_col_init_size
 - explicit_storage_t, 199
- set_col_resize
 - explicit_storage_t, 199
- set_compression_method
 - explicit_storage_t, 199
- set_data
 - message_t, 223
- set_error_vector
 - process_t, 272
 - transition_t, 303
- set_file_name
 - reporter_t, 276
- set_fpos
 - dve_parser_t, 123
- set_gid
 - transition_t, 303
- set_hash_function
 - explicit_storage_t, 199
- set_ht_size
 - explicit_storage_t, 199
- set_ident_gid
 - dve_expression_t, 114
- set_idle
 - distributed_t, 86
- set_info
 - distr_reporter_t, 80
 - reporter_t, 276
- set_initial_state
 - dve_process_t, 143
- set_lid
 - transition_t, 303
- set_lpos
 - dve_parser_t, 124
- set_mem_limit
 - explicit_storage_t, 200
- set_obligatory_keys
 - reporter_t, 276
- set_operator
 - dve_expression_t, 115
- set_partial_id
 - dve_transition_t, 180
- set_problem
 - reporter_t, 276
- set_proc_msgs_buf_exclusive_mem
 - distributed_t, 86
- set_process_gid
 - dve_transition_t, 180
- set_property_gid
 - system_t, 293
- set_property_with_synchro
 - dve_system_t, 168
- set_source_pos
 - dve_source_position_t, 145
- set_state_creator_value
 - dve_explicit_system_t, 107
- set_state_creator_value_extended
 - dve_explicit_system_t, 107
- set_state_creator_value_of_var_type
 - dve_explicit_system_t, 107
- set_states_stored
 - reporter_t, 277
- set_storage
 - logger_t, 217
- set_succs_calls
 - reporter_t, 277
- set_to_state_pos
 - state.hh, 371
- set_trans_count
 - prob_transition_t, 264
- set_transition_and_weight
 - prob_transition_t, 265
- set_value
 - dve_expression_t, 115
- set_var_value
 - dve_explicit_system_t, 108
- set_written_size
 - message_t, 223
- shrink_to
 - array_of_abstract_t, 41
 - array_t, 48

- size_t_value
 - SYS_initial_values_t, 282
- start_timer
 - reporter_t, 277
- state.hh, 369
 - new_state, 371
 - set_to_state_pos, 371
 - state_pos_to, 371
- state_accept
 - dve_parser_t, 124
- state_genbuchi_muller_accept
 - dve_parser_t, 124
- state_lids
 - SYS_parameters_t, 283
- state_list_done
 - dve_parser_t, 124
- state_pos_to
 - state.hh, 371
- state_rabin_streett_accept
 - dve_parser_t, 124
- state_ref_t, 278
 - invalidate, 278
 - is_valid, 278
- state_t, 279
- static_info_t, 280
- stop_SIGALRM
 - logger_t, 217
- stop_timer
 - reporter_t, 277
- string
 - error_vector_t, 192
- string_front
 - error_vector_t, 192
- succ_container_t, 281
 - succ_container_t, 281
 - succ_container_t, 281
- SUCC_DEADLOCK
 - explicit_system.hh, 350
- SUCC_ERROR
 - explicit_system.hh, 350
- SUCC_NORMAL
 - explicit_system.hh, 350
- succs_deadlock
 - explicit_system.hh, 349
- succs_error
 - explicit_system.hh, 349
- succs_normal
 - explicit_system.hh, 349
- swap
 - array_of_abstract_t, 41
 - array_t, 48
 - bymoc_expression_t, 58
- SYNC_ASK
 - dve_transition.hh, 340
- SYNC_ASK_BUFFER
 - dve_transition.hh, 340
- SYNC_EXCLAIM
 - dve_transition.hh, 340
- SYNC_EXCLAIM_BUFFER
 - dve_transition.hh, 340
- SYNC_NO_SYNC
 - dve_transition.hh, 340
- sync_mode_t
 - dve_transition.hh, 340
- synchronized
 - distributed_t, 86
- SYS_initial_values_t, 282
 - size_t_value, 282
- SYS_parameters_t, 283
 - initial_values_counts, 283
 - state_lids, 283
- sysopen.hh, 372
- SYSTEM
 - dve_parser_t, 120
- system.hh, 373
- system_abilities.hh, 374
- system_abilities_t, 284
 - system_abilities_t, 285
 - system_abilities_t, 285
- system_synchronicity_t
 - dve_system.hh, 337
- system_t, 286
 - add_process, 289
 - can_be_modified, 289
 - from_string, 289
 - get_abilities, 290
 - get_process, 290
 - get_process_count, 291
 - get_property_gid, 291
 - get_property_process, 291
 - get_property_type, 291
 - get_trans_count, 292
 - get_transition, 292
 - read, 292, 293
 - remove_process, 293
 - set_property_gid, 293
 - system_t, 289
 - system_t, 289
 - to_string, 293
 - write, 293, 294

- system_trans.hh, 375
- system_trans_t, 295
 - get_count, 296
 - operator=, 296
 - write, 296
- that
 - error_vector_t, 192, 193
- thr, 297
- timeinfo_t, 298
- to_string
 - compact_t, 74
 - expression_t, 212
 - system_t, 293
 - transition_t, 303
- TR_effects_alloc_step
 - dve_transition.hh, 341
- TR_effects_default_alloc
 - dve_transition.hh, 341
- trans_effect_part
 - dve_parser_t, 124
- TRANSITION
 - dve_parser_t, 121
- transition_t, 299
 - can_be_modified, 301
 - can_read, 301
 - from_string, 301
 - get_error_vector, 302
 - get_gid, 302
 - get_lid, 302
 - ptrans_terr, 304
 - read, 302
 - set_error_vector, 303
 - set_gid, 303
 - set_lid, 303
 - to_string, 303
 - transition_t, 301
 - transition_t, 301
 - write, 303
- type_is_const
 - dve_parser_t, 124
- type_list_clear
 - dve_parser_t, 124
- UNIMPLEMENTED
 - error.hh, 343
- updateable_info_t, 305
 - const_data, 305
- updateable_info_t < updateable_data_t,
no_const_data_type_t >, 306
- use_SIGALRM
 - logger_t, 217
- var_decl_array_size
 - dve_parser_t, 125
- var_decl_begin
 - dve_parser_t, 125
- var_decl_cancel
 - dve_parser_t, 125
- var_decl_create
 - dve_parser_t, 125
- var_decl_done
 - dve_parser_t, 125
- var_init_field_part
 - dve_parser_t, 125
- var_init_is_field
 - dve_parser_t, 125
- violated_assertion_count
 - bymoc_explicit_system_t, 56
 - explicit_system_t, 208
- violated_assertion_string
 - bymoc_explicit_system_t, 56
 - explicit_system_t, 208
- violates_assertion
 - bymoc_explicit_system_t, 56
 - explicit_system_t, 208
- vminfo_t, 307
- write
 - system_t, 293, 294
 - system_trans_t, 296
 - transition_t, 303