

Telelogic  
**StateMate®**

**Documentor Reference Guide**



**IBM®**



*Statemate*®

**Documentor Reference Guide**



Before using the information in this manual, be sure to read the “Notices” section of the Help or the PDF file available from **Help > List of Books**.

This edition applies to Telelogic Statemate 4.5 and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 1997, 2008.

US Government Users Restricted Rights—Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

---

<b>Overview of Documentor</b> .....	<b>1</b>
<b>Basic Concepts</b> .....	<b>1</b>
Document Generation Language (DGL) .....	3
DGL Template .....	3
DGL Segments .....	3
Document Assembly .....	3
<b>Designing a Document Using Templates</b> .....	<b>4</b>
Generating the Document .....	5
Formatting Commands .....	5
Sample Template .....	5
Template Sections .....	6
Executing the Template .....	7
Output Files .....	7
Final Assembly .....	8
Reusing Templates .....	8
Include Files .....	8
File Access .....	9
<b>Documentor Interface with Formatting Systems</b> .....	<b>10</b>
Embedding Formatting Instructions .....	10
Predefined Reports .....	10
Plots .....	10
Invoking a Formatter from Within the Documentor .....	11
<b>Using Documentor</b> .....	<b>13</b>
<b>Document Production Process</b> .....	<b>13</b>
<b>Starting Documentor</b> .....	<b>14</b>
<b>Producing the Document Template</b> .....	<b>15</b>
Creating and Manipulating Templates .....	15
Creating a Template .....	16
Editing a Template .....	17
Deleting a Template .....	17
Copying a Template .....	18
Exporting a Template .....	19

---

Compiling a Template . . . . .	20
Printing a Template . . . . .	21
<b>Using Include Files . . . . .</b>	<b>21</b>
Creating and Manipulating Include Files . . . . .	21
Creating an Include File . . . . .	22
Editing an Include File . . . . .	22
Deleting an Include File . . . . .	23
Copying an Include File . . . . .	23
Exporting an Include File . . . . .	23
Printing an Include File . . . . .	23
<b>Producing the Document Segments . . . . .</b>	<b>24</b>
Creating and Manipulating Documents . . . . .	24
Creating a Document . . . . .	25
Editing a Document . . . . .	28
Deleting a Document . . . . .	28
Regenerating Document Segments . . . . .	29
<b>Producing the Final Document . . . . .</b>	<b>31</b>
Printing a Document . . . . .	31
Exporting a Document . . . . .	32
Formatting a Document . . . . .	33
<b>Working with Different Formatters . . . . .</b>	<b>33</b>
Using nroff and troff . . . . .	34
Using Interleaf . . . . .	34
<b>Document Templates . . . . .</b>	<b>35</b>
<b>Principles of DGL . . . . .</b>	<b>35</b>
DGL Template Structure . . . . .	35
DGL Syntax Rules . . . . .	37
Special Features of DGL . . . . .	39
Extensions to Conventional Programming Constructs . . . . .	39
Database Extractions . . . . .	39
Overview of DGL Statements . . . . .	40
<b>Data-types and Expressions . . . . .</b>	<b>41</b>
Data-types . . . . .	41
Conventional Types found in Other Programming Languages . . . . .	41
Statement Element Types . . . . .	42
LIST OF simple_type . . . . .	43
Expressions . . . . .	44
Numeric Expressions . . . . .	44
Boolean Expressions . . . . .	45

---

String Expressions . . . . .	45
Statement Element Expressions . . . . .	46
List Expressions . . . . .	46
Enumerated Types—Predefined Constants . . . . .	47
<b>DGL Statements. . . . .</b>	<b>48</b>
Structure Statements . . . . .	48
TEMPLATE Statement . . . . .	48
SEGMENT Statement . . . . .	48
PROCEDURE Statement . . . . .	49
BEGIN/END Statement . . . . .	49
Comment Statement . . . . .	49
Declaration Statements . . . . .	50
PARAMETER Statement . . . . .	50
CONSTANT Statement . . . . .	51
VARIABLE Statement . . . . .	52
Assignment Statement . . . . .	53
File Handling Statements . . . . .	53
OPEN Statement . . . . .	53
CLOSE Statement . . . . .	54
READ Statement . . . . .	54
Output Statements . . . . .	55
Verbatim Statement . . . . .	55
WRITE Statement . . . . .	56
Using WRITE to Produce Messages . . . . .	58
INCLUDE Statement . . . . .	58
EXECUTE Statement . . . . .	59
Include Reports Statement . . . . .	59
Include Plots Statement . . . . .	64
Include Table Statement . . . . .	67
Control Flow Statements . . . . .	69
IF/THEN/ELSE Statement . . . . .	69
SELECT/WHEN Statement . . . . .	69
FOR/LOOP Statement . . . . .	71
WHILE/LOOP Statement . . . . .	72
EXIT Statement . . . . .	72
STOP Statement . . . . .	73

---

<b>Documentor Functions</b> .....	<b>75</b>
<b>Overview of the Extraction Functions</b> .....	<b>75</b>
Function Structure .....	76
Using Database Extraction Functions .....	76
<b>Calling Conventions</b> .....	<b>77</b>
Function Names .....	77
Element Type Abbreviations .....	77
Arrow Elements .....	79
Function Input Arguments .....	79
Status Codes .....	79
Function Return Values .....	80
Return Values of Type ELEMENT .....	80
Return Values of Filename .....	81
Return Values of Enumerated Types .....	81
<b>Model Templates</b> .....	<b>83</b>
<b>Properties</b> .....	<b>83</b>
Properties Template Structure .....	86
Properties Initiation Section .....	86
Declaration Part .....	86
Body .....	87
Properties Segment Section .....	87
Generating the Report Heading .....	88
Iteration: Using the FOR/LOOP Statement .....	88
Generating the Entry Heading .....	89
Extracting and Printing Information from the Data-Item Form .....	89
Description and Synonym .....	89
Using the SELECT/WHEN Construct .....	89
Using Nested FOR Loops to Extract Attribute Names and Values .....	90
Using Keywords to Write Portions of the Long Description .....	91
Final Output for Data-item Properties .....	92
<b>Activity Interface Report</b> .....	<b>94</b>
act_interface Template .....	95
Activity Interface Report Initiation Section .....	97
Activity Interface Report Segment Section .....	97
Declarations .....	97
Producing the Headings .....	98
Building Element Lists .....	98
Alphabetizing and Sorting the List .....	98
Writing the Input Elements .....	99
Final Output for Act_Interface Report .....	100



---

<b>Template for nroff</b> .....	<b>100</b>
Template with nroff Commands .....	101
Initiation Section (nroff) .....	102
Segment 1: Heading and Report Overview (nroff) .....	103
Including Global Declarations .....	103
Producing the Heading and Overview (nroff) .....	103
Segment 2: Activity-Chart Plot and Property Report (nroff) .....	104
<b>Template for Interleaf</b> .....	<b>105</b>
Initiation Section (Interleaf) .....	106
Segment 1: Heading and Report Overview (Interleaf) .....	106
Producing the Heading and Overview (Interleaf) .....	107
Segment 2: Activity-Chart Plot and Property Report (Interleaf) .....	108
<b>DGL Statement Reference</b> .....	<b>111</b>
<b>ASSIGNMENT</b> .....	<b>114</b>
<b>BEGIN</b> .....	<b>115</b>
<b>CLOSE</b> .....	<b>117</b>
<b>COMMENT</b> .....	<b>118</b>
<b>CONSTANT</b> .....	<b>119</b>
<b>END</b> .....	<b>121</b>
<b>EXECUTE</b> .....	<b>122</b>
<b>EXIT</b> .....	<b>123</b>
<b>FOR/LOOP</b> .....	<b>124</b>
<b>IF/THEN/ELSE</b> .....	<b>126</b>
<b>INCLUDE</b> .....	<b>128</b>
<b>OPEN</b> .....	<b>130</b>
<b>PARAMETER</b> .....	<b>132</b>
<b>PROCEDURE</b> .....	<b>134</b>
<b>READ</b> .....	<b>138</b>
<b>REPORT</b> .....	<b>139</b>
Attribute Report .....	141
Property Report .....	141
Interface Report .....	142
List Report .....	143
N2 Chart Report .....	143
Protocol Report .....	144
Resolution Report .....	145

---

Structure Report .....	145
Tree Report .....	146
<b>SEGMENT</b> .....	<b>147</b>
<b>SELECT/WHEN</b> .....	<b>148</b>
<b>STOP</b> .....	<b>151</b>
<b>TABLE</b> .....	<b>152</b>
<b>TEMPLATE</b> .....	<b>155</b>
<b>VARIABLE</b> .....	<b>156</b>
<b>VERBATIM</b> .....	<b>158</b>
<b>WHILE/LOOP</b> .....	<b>160</b>
<b>WRITE</b> .....	<b>162</b>
<b>Single-Element Functions</b> .....	<b>165</b>
<b>Calling Single-Element Functions</b> .....	<b>166</b>
Single-Element Function Input Arguments .....	167
Examples of Single-Element Function Calls .....	168
Single-Element Function Example 1 .....	168
Single-Element Function Example 2 .....	168
Single-Element Function Example 3 .....	169
<b>List of Functions</b> .....	<b>169</b>
stm_r_xx .....	175
stm_r_sb_action_lang .....	177
stm_r_sb_action_lang_expression .....	178
stm_r_sb_action_lang_local_data .....	179
stm_r_actual_parameter_exp .....	180
stm_r_actual_parameter_type .....	181
stm_r_elem_in_ddb_list .....	182
stm_r_sb_ada_user_code .....	183
stm_r_sb_ansi_c_user_code .....	184
stm_r_st_combinationals .....	185
stm_r_xx_array_lindex .....	186
stm_r_xx_array_rindex .....	187
stm_r_xx_attr_enforced .....	188
stm_r_xx_attr_name .....	190
stm_r_xx_attr_val .....	192
stm_r_xx_bit_array_lindex .....	195
stm_r_xx_bit_array_rindex .....	196
stm_r_xx_cbk_binding .....	197
stm_r_xx_cbk_binding_enable .....	198
stm_r_xx_cbk_binding_expression .....	200

---

stm_r_xx_cbk_binding_expression_hyper	201
stm_r_tt_cell	202
stm_r_tt_cell_type	203
stm_r_changes_log	205
stm_r_xx_chart	206
stm_r_xx_combinationals	208
stm_r_sb_connected_chart	209
stm_r_xx_containing_fields	210
stm_r_ch_creation_date	211
stm_r_ch_creator	213
stm_r_xx_data_type	215
stm_r_rt_date	216
stm_r_xx_definition_type	217
stm_r_xx_desc_file	220
stm_r_xx_description	221
stm_r_design_attr	223
stm_r_xx_displayed_name	224
stm_r_ddb_list_names	226
stm_r_element_type	227
stm_r_xx_expr_hyper	230
stm_r_xx_expression	231
stm_r_xx_ext_link	233
stm_r_uc_ext_point_def	235
stm_r_formal_parameter_names	236
stm_r_sb_global_data	237
stm_r_global_interface_report	238
stm_r_xx_cbk_binding_expression_hyper	239
stm_r_xx_graphic	240
stm_r_hyper_key	242
stm_r_md_implementation	243
stm_r_included_gds	244
stm_r_msg_included_in_ord_insig	245
stm_r_cd_info	246
stm_r_inherited_gds	247
stm_r_xx_instance_name	248
stm_r_xx_keyword	250
stm_r_sb_kr_c_user_code	253
stm_r_xx_labels	254
stm_r_xx_labels_hyper	256
stm_r_local_interface_report	257
stm_r_xx_longdes	258
stm_r_lookup_table_header	261
stm_r_xx_max_val	262
stm_r_xx_min_val	263

---

stm_r_xx_mini_spec . . . . .	264
stm_r_ac_mini_spec_hyper . . . . .	265
stm_r_xx_mode . . . . .	266
stm_r_ch_modification_date . . . . .	267
stm_r_xx_name . . . . .	269
stm_r_next_msg . . . . .	272
stm_r_xx_note . . . . .	273
stm_r_xx_notes . . . . .	274
stm_r_tt_num_of_col . . . . .	275
stm_r_tt_num_of_in . . . . .	276
stm_r_tt_num_of_out . . . . .	277
stm_r_tt_num_of_row . . . . .	278
stm_r_uc_num_of_scen . . . . .	279
stm_r_xx_number_of_bits . . . . .	280
stm_r_xx_of_enum_type . . . . .	281
stm_r_xx_of_enum_type_name_type . . . . .	282
stm_r_ord_insig_defined_in_ch . . . . .	284
stm_r_parameter_binding . . . . .	285
stm_r_parameter_mode . . . . .	286
stm_r_sb_parameters . . . . .	288
stm_r_en_parent . . . . .	289
stm_r_previous_msg . . . . .	290
stm_r_sb_proc_sch_local_data . . . . .	291
stm_r_md_purpose . . . . .	292
stm_r_xx_reactions . . . . .	293
stm_r_param_binding_hyper . . . . .	295
stm_r_param_binding_id . . . . .	296
stm_r_sb_return_type . . . . .	297
stm_r_sb_return_user_type . . . . .	298
stm_r_sb_return_user_type_name . . . . .	299
stm_r_tt_row . . . . .	300
stm_r_uc_scen . . . . .	301
stm_r_uc_scen_attr_name . . . . .	302
stm_r_uc_scen_attr_val . . . . .	303
stm_r_sd_scope . . . . .	305
stm_r_xx_select_implementation . . . . .	306
stm_r_st_static_reactions . . . . .	308
stm_r_st_static_reactions_hyper . . . . .	309
stm_r_xx_string_length . . . . .	310
stm_r_xx_structure_type . . . . .	311
stm_r_ac_subroutine_bind . . . . .	313
stm_r_ac_subroutine_bind_enable . . . . .	314
stm_r_ac_subroutine_bind_expr . . . . .	315
stm_r_xx_synonym . . . . .	316

---

stm_r_ac_termination . . . . .	318
stm_r_xx_truth_table . . . . .	320
stm_r_xx_truth_table_expression . . . . .	321
stm_r_sb_truth_table_local_data . . . . .	322
stm_r_xx_type . . . . .	323
stm_r_xx_type_expression . . . . .	327
stm_r_xx_uniquename . . . . .	328
stm_r_ch_usage_type . . . . .	331
stm_r_xx_user_type . . . . .	332
stm_r_xx_user_type_name_type . . . . .	333
stm_r_ch_version . . . . .	334
stm_r_gds_visibility_mode . . . . .	335
stm_r_msg_where_tc_begins . . . . .	336
stm_r_msg_where_tc_ends . . . . .	337
stm_r_sb_connected_statechart . . . . .	338
stm_r_sb_connected_flowchart . . . . .	339
stm_r_sb_proc_fch_local_data . . . . .	340
stm_r_xx_des_attr_val . . . . .	341
stm_r_xx_des_attr_name . . . . .	343
stm_r_tt_cell_hyper . . . . .	345
stm_r_tt_row_hyper . . . . .	346
stm_r_xx_default_val . . . . .	347
stm_r_component_param_binding . . . . .	348
stm_r_component_param_mode . . . . .	349
stm_r_stubs_names . . . . .	350
stm_r_information_stub_names . . . . .	351
stm_r_sb_connected_statechart . . . . .	352
stm_r_sb_connected_flowchart . . . . .	353

**Query Functions . . . . . 355**

**Calling Query Functions . . . . . 356**

By Attributes . . . . .	356
By Structure Type . . . . .	357
Name and Synonym Patterns . . . . .	358

**Query Function Input Arguments . . . . . 359**

**Examples of Query Functions . . . . . 360**

Query Function Example 1 . . . . .	360
Query Function Example 2 . . . . .	360
Query Function Example 3 . . . . .	361

---

List of Query Functions	361
Activities (ac)	362
Input List Type: ac	362
Input List Type: af	369
Input List Type: ch	371
Input List Type: ds	372
Input List Type: md	372
Input List Type: mx	373
Input List Type: router	373
Input List Type: st	374
A-Flow-Lines (af, ba, laf)	375
Output List Type: af	375
Output List Type: laf	380
Actions (an)	382
Input List Type: an	382
Input List Type: ch	384
Charts (ch)	385
Input List Type: ac	385
Input List Type: an	385
Input List Type: ch	386
Input List Type: co	388
Input List Type: di	389
Input List Type: ds	389
Input List Type: dt	389
Input List Type: ev	390
Input List Type: fd	390
Input List Type: if	390
Input List Type: md	391
Input List Type: mx	392
Input List Type: router	392
Input List Type: sb	393
Input List Type: st	393
Connectors (cn)	394
Input List Type: cn	394
Input List Type: st	395
Input List Type:bm	395
Input List Type: tr	396
Conditions (co)	396
Input List Type: af	396
Input List Type: ch	397
Input List Type: co	397
Input List Type: di	399
Input List Type: if	399
Input List Type: mf	399

---

Data-Items (di) . . . . .	400
Input List Type: af . . . . .	400
Input List Type: ch . . . . .	400
Input List Type: co . . . . .	401
Input List Type: di . . . . .	401
Input List Type: fd . . . . .	406
Input List Type: if . . . . .	407
Input List Type: mf . . . . .	407
Data-Stores (ds) . . . . .	408
Input List Type: ac . . . . .	408
Input List Type: af . . . . .	408
Input List Type: ch . . . . .	409
Input List Type: ds . . . . .	409
Input List Type: md . . . . .	410
User-Defined Types (dt) . . . . .	411
Input List Type: ch . . . . .	411
Input List Type: dt . . . . .	412
Input List Type: fd . . . . .	417
Events (ev) . . . . .	418
Input List Type: af . . . . .	418
Input List Type: ch . . . . .	418
Input List Type: ev . . . . .	419
Input List Type: if . . . . .	420
Input List Type: mf . . . . .	421
Fields (fd) . . . . .	421
Input List Type: ch . . . . .	421
Input List Type: di . . . . .	421
Input List Type: dt . . . . .	422
Input List Type: fd . . . . .	422
Input List Type: mx . . . . .	426
Functions (fn) . . . . .	427
Input List Type: ch . . . . .	427
Information-Flows (if) . . . . .	428
Input List Type: af . . . . .	428
Input List Type: ch . . . . .	429
Input List Type: co . . . . .	429
Input List Type: di . . . . .	430
Input List Type: ev . . . . .	430
Input List Type: if . . . . .	431
Input List Type: mf . . . . .	432

---

M-Flow-Lines (bf, bm, lmf, mf) . . . . .	433
Output List Type: bf . . . . .	433
Output List Type: bm . . . . .	436
Output List Type: lmf . . . . .	437
Output List Type: mf . . . . .	438
Modules (md) . . . . .	443
Input List Type: ac . . . . .	443
Input List Type: ch . . . . .	443
Input List Type: ds . . . . .	444
Input List Type: md . . . . .	445
Input List Type: mf . . . . .	449
Input List Type: router . . . . .	449
Mixed (mx) . . . . .	450
Input List Type: af . . . . .	450
Input List Type: ac . . . . .	451
Input List Type: actor . . . . .	452
Input List Type: an . . . . .	453
Input List Type: bb . . . . .	454
Input List Type: bt . . . . .	454
Input List Type: bm . . . . .	456
Input List Type: ch . . . . .	457
Input List Type: co . . . . .	459
Input List Type: di . . . . .	460
Input List Type: ds . . . . .	461
Input List Type: dt . . . . .	461
Input List Type: ev . . . . .	462
Input List Type: fd . . . . .	463
Input List Type: fn . . . . .	464
Input List Type: if . . . . .	464
Input List Type: md . . . . .	465
Input List Type: mf . . . . .	466
Input List Type: msg . . . . .	466
Input List Type: mx . . . . .	467
Input List Type: router . . . . .	475
Input List Type: sb . . . . .	476
Input List Type: st . . . . .	476
Input List Type: tr . . . . .	478
Input List Type: uc . . . . .	479
Routers (router) . . . . .	479
Input List Type: ac . . . . .	479
Input List Type: af . . . . .	480
Input List Type: ch . . . . .	480
Input List Type: md . . . . .	481
Input List Type: router . . . . .	481



---

Subroutines (sb) . . . . .	482
Input List Type: ch . . . . .	482
Input List Type: sb . . . . .	483
States (st) . . . . .	488
Input List Type: ac . . . . .	488
Input List Type: ch . . . . .	489
Input List Type: cn . . . . .	490
Input List Type: mx . . . . .	490
Input List Type: st . . . . .	491
Input List Type: tr . . . . .	494
Timing Constraint (tc) . . . . .	495
Input List Type: ch . . . . .	495
Transitions (tr) . . . . .	495
Input List Type: cn . . . . .	495
Input List Type: mx . . . . .	496
Input List Type: st . . . . .	497
Input List Type: tr . . . . .	497

## **Utility Functions . . . . . 499**

### **Calling Utility Functions . . . . . 500**

Contains Element . . . . .	500
List Extraction by Type . . . . .	501
List Extraction by Chart . . . . .	502
Location of Pattern in a String . . . . .	502
Extract Portion of a String . . . . .	503

### **Utility Function Input Arguments . . . . . 504**

### **Examples of Utility Functions . . . . . 505**

Utility Functions Example 1 . . . . .	505
Utility Functions Example 2 . . . . .	505
Utility Functions Example 3 . . . . .	506

### **List of Utility Functions . . . . . 506**

stm_action_of_reaction . . . . .	509
stm_delete_file . . . . .	510
stm_dispose_memory . . . . .	511
stm_index . . . . .	512
stm_int . . . . .	513
stm_int_to_string . . . . .	513
stm_r_is_state . . . . .	514
stm_list_contains_element . . . . .	514
stm_list_contains_string . . . . .	516
stm_list_extraction . . . . .	517
stm_list_extraction_by_chart . . . . .	518

---

stm_list_extraction_by_chart_id	519
stm_list_extraction_by_type	520
stm_list_first_element	522
stm_list_last_element	523
stm_list_length	524
stm_list_next_element	526
stm_plot_ext	528
stm_list_previous_element	532
stm_list_sort	534
stm_list_sort_by_attr_value	535
stm_list_sort_by_branches	537
stm_list_sort_by_chart	538
stm_list_sort_by_levels	539
stm_list_sort_by_name	541
stm_list_sort_by_synonym	543
stm_list_sort_by_type	545
stm_multiline_to_one	546
stm_multiline_to_strings	546
stm_plot	547
stm_plot_hyper_exp	548
stm_plot_with_autonumber	551
stm_plot_with_break	554
stm_plot_with_headerline	558
stm_replace_string	561
stm_replace_word	562
stm_set_rpt_formator	563
stm_string_retain	564
stm_str_list_first_element	565
stm_str_list_last_element	566
stm_str_list_length	567
stm_str_list_next_element	568
stm_str_list_previous_element	569
stm_str_list_to_str	571
stm_str_to_list	572
stm_string_extract	573
stm_string_free	574
stm_string_retain	575
stm_string_to_int	576
stm_strlen	577

---

<b>Project Management</b> .....	<b>579</b>
stm_r_pm_member_workareas .....	580
stm_r_pm_operator_projects .....	581
stm_r_pm_project_databank .....	582
stm_r_pm_project_manager .....	583
stm_r_pm_project_members .....	584
stm_r_pm_projects .....	585
<b>Function Status Codes</b> .....	<b>587</b>
<b>DGL Reserved Words</b> .....	<b>593</b>
<b>BNF Syntax</b> .....	<b>595</b>
<b>BNF Structure and Conventions</b> .....	<b>595</b>
Symbol Types .....	595
BNF Notations .....	596
<b>BNF for DGL Statements</b> .....	<b>596</b>
<b>Index</b> .....	<b>601</b>



# Overview of Documentor

---

You can use the Documentor to design and produce documentation for the system you are designing. With this tool, you can:

- ◆ Design your document format.
- ◆ Determine what kinds of information to include in it.

Your documents can include textual and graphical information from a variety of sources, including your project database and external files and programs outside your workarea.

You can use the Documentor in conjunction with the Reports tool. Whereas the Reports tool enables you to produce predefined reports, the Documentor enables you to produce reports customized for your specific needs.

This section presents an overview of the Documentor tool. The topics are as follows:

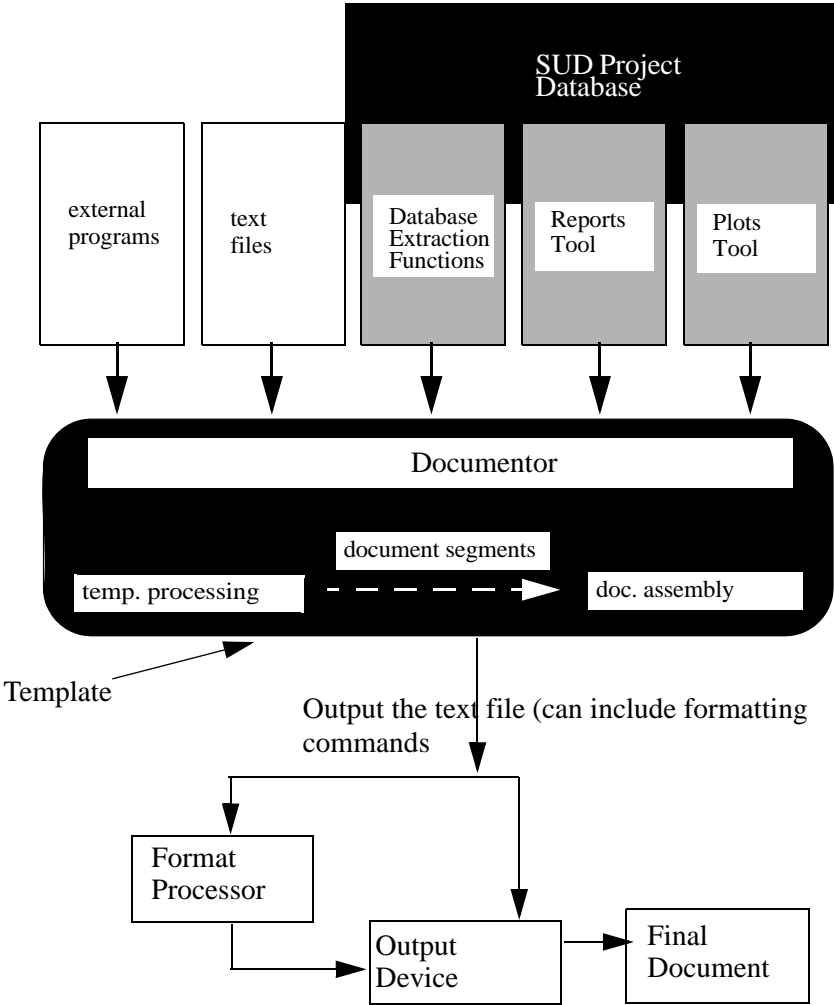
- ◆ [Basic Concepts](#)
- ◆ [Designing a Document Using Templates](#)
- ◆ [Documentor Interface with Formatting Systems](#)

## Basic Concepts

You can use the Documentor to design formatted reports that combine information from the following sources:

- ◆ System-under-design (SUD) database retrievals
- ◆ Reports produced by the Reports tool
- ◆ Plots of charts produced by the Plots tool
- ◆ External files (referred to as *include files*) that can contain text, tables, figures, and so on
- ◆ Information produced by external programs

The following figure illustrates internal and external sources, and the process of final document production. Note that non-shaded areas represent facilities *outside* of the system.



## Document Generation Language (DGL)

You design your documents by writing a program using the Document Generation Language (DGL). This language provides you with great flexibility in designing your document. Among other features, DGL enables you to extract information from your project database (workarea).

### DGL Template

The document design program that you write using DGL is called a *template*. The template contains instructions as to what information is to be included in the document. It can also include formatting instructions to be passed to a document format processing system, called a *formatter*. Formatting instructions specify components within your document, including:

- ◆ Text width
- ◆ Margins
- ◆ Headers and footers
- ◆ Pagination

### DGL Segments

The template is divided into sections called *segments*. To generate the document, you execute the template. Each segment of the template produces a separate text file called a *document segment*.

Creating a document in segments is more efficient than producing an entire document. By using segments, you can produce the document in stages, updating or editing the segments as needed.

#### Note

---

The division of the template into segments does not necessarily have to correspond to the final document divisions (chapters, sections, and so on). However, it is a good idea to have the template segments correspond with document divisions wherever possible.

### Document Assembly

The Documentor assembles the generated document segments, then does one of the following:

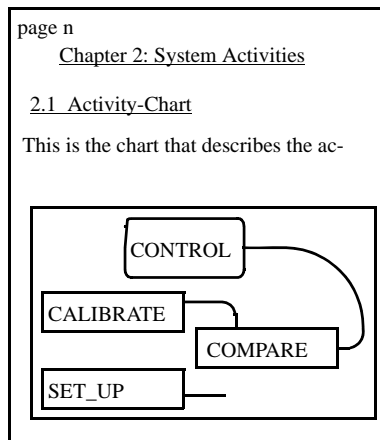
- ◆ Exports the segments to an external output device or file.
- ◆ Sends the segments to a formatting system to produce the final document.

## Designing a Document Using Templates

The following example shows a document that consists of the following:

- ◆ Title
- ◆ Overview, which contains introductory information from an external text file
- ◆ Plot of an activity-chart
- ◆ Property information on subactivities shown in the chart, consisting of a property report from the Reports tool

The following figure shows the formatted report.





## Generating the Document

The first step is to create the template needed to generate the desired document using DGL. Because the document contains information about a particular activity, you can use a DGL parameter called *act\_name*. You specify the value of the parameter (in this case, the activity name for which you are producing the report) when you are ready to generate the document.

### Formatting Commands

Some of the commands in the template consist of formatting instructions to be interpreted by the formatter used to produce the final document. This example uses an abstract formatter, rather than a specific one.

The following table lists the commands used by this example.

Command	Description
<b>.skip</b> <n>	Outputs <i>n</i> blank lines
<b>.center</b> <title>	Centers the specified title on the line
<b>.chapter</b> <title>	Starts a new chapter with the specified title
<b>.section</b> <title>	Starts a new section with the specified title
<b>.page</b>	Starts a new page
<b>.literal</b> and <b>.end literal</b>	Outputs the block of text specified between the keywords <b>.literal</b> and <b>.end literal</b>

### Sample Template

In a template, statements beginning with “- -” are comments and are not interpreted by the tool. Formatting instructions and other text written between the */@* and *@/* markers are passed to the output files *verbatim*.

The example uses the following template:

```
TEMPLATE example;

-- Initiation (global) section
PARAMETER
  STRING  act_name;  -- activity for which
                   -- the report is written
VARIABLE
  ACTIVITY act_id;  -- id of "act_name"
  INTEGER  st;      -- status return code

BEGIN
  act_id := stm_r_ac (act_name, st);
END;

SEGMENT seg1;      -- contains chapter 1 of the document
```

```
BEGIN
  /@
  .page
  @/
  WRITE ('.center Description of ',act_name);
  /@
  .chapter An Overview
  @/
  INCLUDE ('sys_overview'); -- an include file containing
                           -- text with formatting commands
END;

SEGMENT seg2; -- contains chapter 2 of the document

VARIABLE
  LIST OF ACTIVITY ac_list;

BEGIN
  /@
  .page
  .chapter System Activities
  .section Activity-chart
  .skip
  This is the chart that describes the activities
  of the system:
  @/
  .....
  stm_plt(act_id, ....); -- Activity-chart plot
  .....
  /@
  .page
  .section Activities' description
  .skip
  Detailed description of each activity in the chart:
  @/
  ac_list:=stm_r_ac_logical_sub_of_ac({act_id},st);
  stm_rpt_dictionary(ac_list,....); -- property report
END;
```

## Template Sections

The example template is divided into three parts:

- ◆ **Initiation section**—Extracts the activity ID number (*act\_id*) from the database. This number is used later in the template to call the appropriate activity-chart plot and to extract the subactivities for the property report.
- ◆ **SEGMENT seg1**—Contains instructions to produce the title of the document and chapter 1. In this case, chapter 1 is text contained in an include file, *sys\_overview*.
- ◆ **SEGMENT seg2**—Contains instructions for producing a plot of an activity-chart and a property report of subactivities.

## Executing the Template

After the template is created, you execute it. As part of this process, you enter the value for the parameter *act\_name* in a special form provided by the tool. (In this example, the value for the parameter is the activity's name, MEASURE.) As a result of specifying the activity name, the appropriate information for the activity MEASURE will be included in the document.

The Documentor produces separate text files, corresponding to the segment sections specified in the template. These files include:

- ◆ Information generated from the project database (the activity-chart plot and property report).
- ◆ Text from an include file.
- ◆ Formatting instructions.

Formatting instructions and other text written between the */@* and *@/* markers are passed to the output files verbatim.

If you have made any errors in your template, error messages are displayed during the execution phase. Correct the template and then re-execute it.

## Output Files

After executing the template for the activity MEASURE, the resulting output files (document segments) resemble the following:

```
DOCUMENT SEGMENT SEG1:

.page
.center Description of MEASURE
.chapter An Overview
  text and formatting commands as written in the file 'sys_overview'

DOCUMENT SEGMENT SEG2:

.page
.chapter System Activities
.section Activity-chart
.skip
This is the chart that describes the activities of the system:
.literal
  plot information to be sent to the printer
.end literal
.page
.section Activities' description
.skip
Detailed description of each activity in the chart:
Output of Report Tool for the Property Report, including formatting commands for
the specific formatter.
.
.
.
```

## Final Assembly

Once the segments are generated, you can use various Documentor options to edit or regenerate them.

When you are satisfied with the segments, you can copy them to an external file or format them. Both of these operations automatically assemble the segments into one file.

The **Format** option assembles the segments and passes them to the format processor that you specify when you create the template. The format processor interprets the formatting statements in the file and produces a final document that is completely formatted.

## Reusing Templates

You can generate different documents from the same template using parameters within the template. By changing the values of these parameters, you can change the information that is written, while maintaining the overall structure and format of the document.

For example, the template produces a plot of an activity-chart and a property report for the activity `MEASURE`. Instead of writing the name `MEASURE` directly into the template, the template uses the parameter `act_name`. This way, you can specify the name of the activity in a form at the time of template execution.

You can use the template repeatedly, specifying a different activity name each time you execute the template. This enables you to produce uniform reports for as many activities as desired.

## Include Files

You can include files outside of your project database within your documents. In the template example, the introductory `sys_overview` section is not part of the database, but is an external file. You can create such files while you work with the Documentor, or you can copy them from outside your workarea. In the example, this file is assumed to already be in the workarea. Therefore, the file path is not specified—only the file name. The file is accessed via the `INCLUDE` statement within the template and inserted in the segment files when the template is executed.

The same include files can be used in a number of different documents, and can be called from a number of different templates.

The figure illustrates the relationship between templates, include files, and parameters, and the segments that are produced after the templates are executed. Template A contains parameters and is generated twice—once using set 1 of the parameter values and once using set 2.

Using the first set of parameter values, the segments of document A1 are generated. In addition, template A contains a statement calling an external include file, `F1`, into the generated document segments of A1.

When the second set of parameters is used, the segments of document A2 are generated. Again, the include file F1 is included in the output segments.

Template B is used to generate the segments of document B1. These segments contain two include files: F1, which was also included in the segments of document A1, and a separate include file, F2.

### **File Access**

The Documentor files, templates, and include files

## Documentor Interface with Formatting Systems

When you create a template, you can assign a formatter to it from a predefined list (for example, Interleaf).

### Embedding Formatting Instructions

You can embed formatting instructions in a template. Upon execution, these instructions are passed verbatim to the output segments. When these segments are assembled and sent to a formatting processor, the instructions are interpreted and a final, formatted document is produced.

You can embed formatting instructions for any kind of formatting system or word processor. Some systems are interactive in nature; that is, you do not directly see the formatting instructions used by the system when working with it.

If you are working with such a system, you must first determine the particular language (set of instructions) that can be used by the system in *batch* mode. These instructions can then be embedded in the template in the same way as for any other formatting system.

### Predefined Reports

In many documents, you might want to include a report generated from the Reports tool. The formatting of reports and plots depends on the formatter used and whether this formatter is supported by the system.

Reports from the Reports tool are textual. They have predefined formats determined by embedded formatting instructions. For example, assume that you write a template that calls a report (such as the property report in the sample template), and that you are working with a supported formatter. When you execute the template, the Reports tool automatically embeds the formatting instructions appropriate for the attached formatter within the generated report. This means that the report that appears in the resulting output segments contains embedded formatting commands. Passing the files to the designated formatter results in a document displaying the report in its predefined format. For more information on reports, refer to [INCLUDE](#).

### Plots

The Plots tool can generate graphical instructions in several languages. These instructions can be addressed to a printing (plotting) device or to a specific formatter that supports graphics.

For example, the plot in the sample document was included in the document by using a function call `stm_plt`. One of the parameters of this function is the graphical output language. You can pass the attached formatter's language as a parameter to a plot function as long as the formatter can handle a graphical language. In this case, the formatting system processes the plot as part of the entire document.

If you are working with a formatter that cannot process graphical information, you can generate the plot in the language of the output device (printer or plotter) and instruct the formatter to pass the information without processing it. Alternatively, you can produce a separate plot file and later merge the plot into your final document. For more information on plots, refer to [INCLUDE](#).

#### Note

---

Plots created using the Word format in the Output Device dialog box are RTF files.

## Invoking a Formatter from Within the Documentor

There are two ways to send segment files to a formatter for final processing:

- ◆ Export the files to an external file (the export operation automatically assembles the segments into one file). You then invoke the formatter on the file that you want to format as you would for any other text file.
- ◆ For several formatting systems, you can choose the **Format** option. This option automatically assembles the segments and sends them to the formatter without requiring you to exit from the Documentor.

#### Note

---

The **Format** option is available only for specific formatters, such as nroff. Interleaf, which is an interactive formatting system, cannot be activated from within the Documentor tool.





# Using Documentor

---

This section describes how to use Documentor in detail. It includes:

- ◆ Descriptions of Documentor menus and dialog boxes
- ◆ Step-by-step procedures for each Documentor option

The first part of the chapter provides an overview of operations, including how to start the Documentor and the connection between the stages of document production and the different tool options. Use this section to locate the menus that you need to perform a particular operation.

The second part of the chapter explains how to perform Documentor operations.


## Document Production Process

There are four stages of document production:

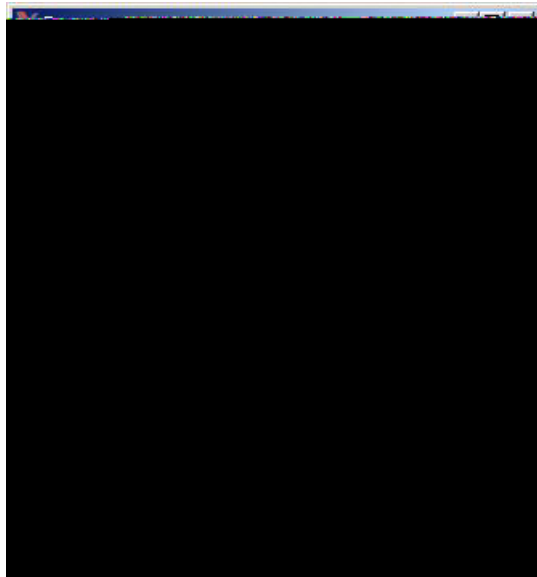
Document Production	Menu Option
<b>Writing the template</b> Creating new templates and editing, deleting, copying, exporting, compiling, and printing existing templates.	<b>Edit &gt; Templates</b>
<b>Preparing include files</b> Creating new include files and editing, deleting, copying, exporting, and printing existing include files.	<b>Edit &gt; Include File</b>
<b>Producing document segments</b> Generating new document segments. The Documentor enables you to regenerate particular segments without having to regenerate the entire document. If you are not satisfied with a segment, you can edit it before producing the final document.	<b>Edit &gt; Documents</b>
<b>Producing the final document</b> Regenerating and editing document segments. Deleting, exporting, printing, and formatting existing documents. If the segments do not contain formatting instructions for a particular formatter, you can print the files directly. If the segments contain formatting instructions, you send them to a formatter. This involves either sending the segment files directly to a formatter, or copying them to an external file and formatting them outside of the system.	<b>Edit &gt; Documents</b>

## Starting Documentor

To start the Documentor from within the system, follow these steps:

1. Start a session and open a project.
2. Click the **Documentor**  icon in the Statemate main window. The Document Management window opens.

In addition, the Document Management dialog box opens simultaneously so you can easily manage your documents (refer to [Creating and Manipulating Documents](#)).



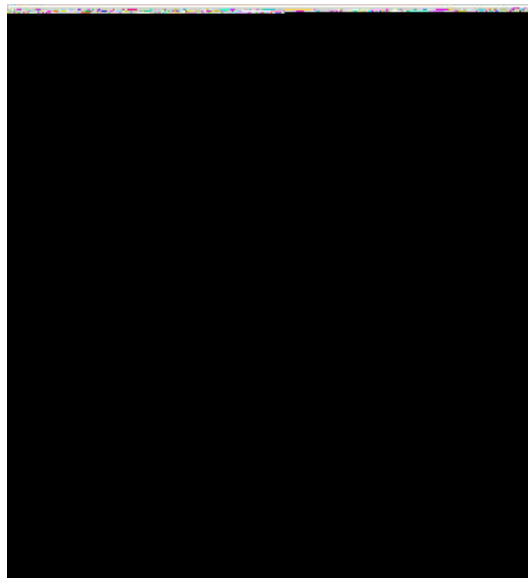
## Producing the Document Template

A document template consists of statements (instructions) written in DGL. A template is a text file containing instructions for document generation. In principle, you can create a template in the same way that you create a source file for any other programming language. For simplicity, template files are handled from within the system by standardized storage functions.

### Creating and Manipulating Templates

To work with templates, select the **Template**  icon in the toolbar or **Edit > Template** from the main menu. The Template Management dialog box opens. In the dialog box, the **Templates** table lists the available templates in your workarea.

The following figure shows the Template Management dialog box.



## Creating a Template

To create a template, follow these steps:

1. Click **New**. The **New Template** dialog box opens.



2. Enter a name for the new template in the **Name** field. The template name must begin with a letter, and can contain only alphanumeric characters or underscores.

The name must be unique in this project. If you select different formatters, you can use the same name for multiple templates because the system appends the formatter type to the name you enter, thus making the names unique.

For example, if you enter the name PAGER and select FrameMaker as the formatter, the system names the template PAGER\_FRM. The following table lists the extension for each formatter.

If a template with this name already exists, the Documentor displays a warning. You can cancel or confirm your choice to overwrite the old template. To modify an existing name, click on the cascade button and select a name from the list of templates. This places the name in the **Name** field so you can then modify it.

FrameMaker	_FRM
Glyph	_GLP
Interleaf	_IFF
troff	_TFF
Word	_RTF
Undefined	_OTH

3. Select a formatter from the list of supported format processors.

The format processor operates on the formatting instructions entered into the template. (Refer to [Overview of Documentor](#) for more information.) When you copy an existing template, the formatter type for the new template is the same as for the original, unless you specify otherwise.

4. Click **OK**. The text editor opens.
5. Write your template using DGL statements.
6. To save your template, select **File > Save**, then **File > Exit**. to close the text editor. You return to the Template Management dialog box.

You can also create a new template by copying an existing template and editing it as needed.

## Editing a Template

To edit an existing template, follow these steps:

1. Select the template from the list in the Template Management dialog box.
2. Click **Edit**. The template opens in the system's text editor.
3. Make your changes, then select **File > Save** and **File > Exit** in the text editor.

## Deleting a Template

To delete a template, follow these steps:

1. Select the template from the list in the Template Management dialog box.

To delete multiple templates, hold down the CTRL key when making your selections.

2. Click **Delete**.
3. The Documentor prompts you to confirm the deletion. Answer **Yes**.

The tool first checks for documents that were generated using the template. If it finds any, the template is not deleted and a list of documents that were generated from the template is displayed in the main window. The Documentor tool assumes that you need to keep the template sources from which existing documents were made. You must delete those documents before their associated templates can be deleted.

## Copying a Template

To copy a template, complete the following:

1. Select the template from the list in the Template Management dialog box.
2. Click **Copy**. The Copy Template dialog box opens with the name of the template in the title bar.

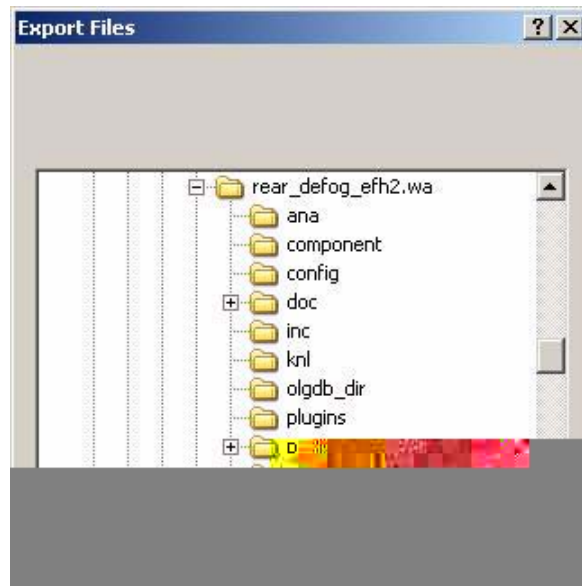


3. Type a new name for the template copy.
4. Select **OK**. A copy of the template is displayed in the Template Management dialog box.

## Exporting a Template

To export a template, complete the following:

1. Select the template from the list in the Template Management dialog box.
2. Click **Export**. The Export Files dialog box opens, as shown in the following figure.



3. Navigate to the location to which you want to export the template.
4. Click **OK** to export the template and dismiss the dialog box.

## Compiling a Template

You can check a template for adherence to DGL syntax rules using the **Compile** option. This is the same as program compilation in other languages. Use this option to see whether a template is executable, without actually performing the execution. Any errors are logged so you can execute templates unattended.

### Note

---

If you do not compile your template, the tool checks the template for errors upon execution (refer to [Creating a Document](#)).

To compile a template, follow these steps:

1. Select the template from the list in the Template Management dialog box.
2. Click **Compile**. A message is displayed in the main Documentor window that notifies you whether the compilation was successful.

If there are compilation errors, a message notifies you that errors were found. The error messages are written into the template in the form of comments. A compiler error message appears as close as possible to the line where the problem was found. The following example shows a template with error messages:

```
.
.
name := stm_r_st_name (state_chart, status;
--%DOC (E1352) Missing ')'
name := name + 5;
--%DOC (E1142) Argument mismatch
name := nm;
--%DOC (E1211) Identifier NM not declared
.
.
FOR sub_state IN sub_list LOOP
i:=i+1
--%DOC (E2001) Missing ';'
.
.
```

You must correct template errors before re-executing the template, but you do not need to remove them from the template file. The Documentor automatically removes them the next time the template is compiled or executed.



## Printing a Template

To print a template, follow these steps:

1. Select the template from the list in the Template Management dialog box.
2. Click **Print**.

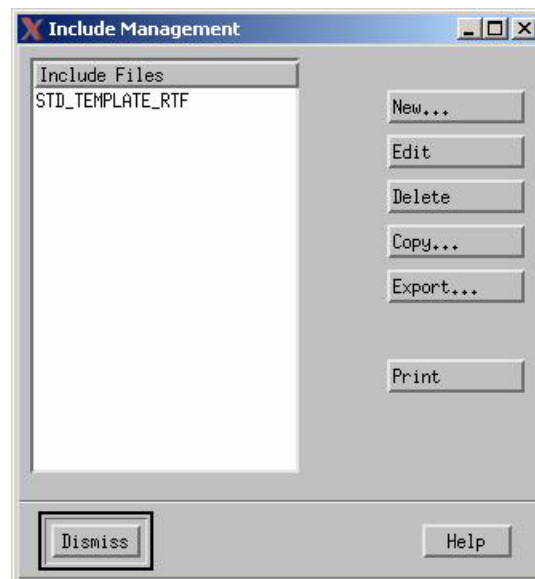
To print multiple templates, hold down the CTRL key when making your selections.

## Using Include Files

Your document might include files external to the specification database. Such include files consist of textual explanations, diagrams, pictures, and so on. Insert these files in your document using the INCLUDE statement in the template. When you execute the template, the include files are copied into the resulting document segments.

## Creating and Manipulating Include Files

To add include files, click the **Include Files**  icon in the main Documentor window, or select **Edit > Include File**. The Include Management dialog box opens, as shown in the following figure.

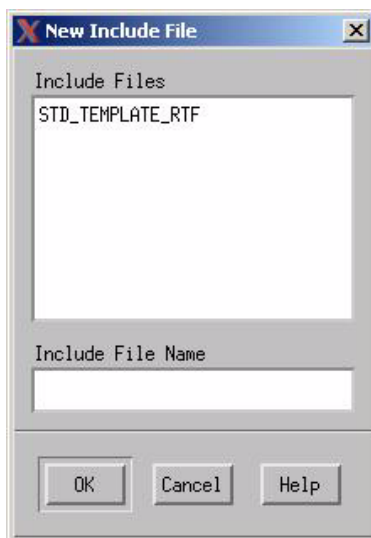


The dialog box lists the include files in your workarea.

## Creating an Include File

To create an include file, follow these steps:

1. Click **New**. The **New Include File** dialog opens.



2. Enter a name for the new include file in the text box. The include file name must begin with a letter, and can contain only alphanumeric characters or underscores.

The name must be unique in this project. If an include file with this name already exists, the system displays a warning. You can cancel or confirm your choice to overwrite the old include file. To modify an existing name, select a name from the list and modify it in the text box.

3. Click **OK** to invoke the system's text editor where you can create a new include file.

Alternatively, you can create a new include file by copying an existing include file and editing it as needed.

## Editing an Include File

To edit an include file, follow these steps:

1. Select the include file from the list in the **Include Management** dialog box.
2. Click **Edit**. The include file is displayed in the system's text editor.
3. Make your changes, then select **File > Save** and **File > Exit** in the text editor.

## Deleting an Include File

To delete an include file, follow these steps:

1. Select the include file from the list in the Include Management dialog box.

To delete multiple include files, hold down the **CTRL** key when making your selections.

2. Click **Delete**.
3. The Documentor prompts you to confirm the deletion. Answer **Yes**.

## Copying an Include File

To copy an include file, follow these steps:

1. Select the include file from the list in the Include Management dialog box.
2. Click **Copy**. The Copy Include dialog opens with the name of the include file in the title bar.
3. Type a new name for the include file copy.
4. Click **OK**. A copy of the include file is displayed in the Include Management dialog box.

## Exporting an Include File

To export an include file, follow these steps:

1. Select the include file from the list in the Include Management dialog box.
2. Click **Export**. The Export Files dialog box opens.
3. Type the directory and file name where you want to export the include file.
4. Click **OK** to apply your changes and dismiss the dialog box.

## Printing an Include File

To print an include file, follow these steps:

1. Select the include file from the list in the Include Management dialog box.
2. Click **Print**.

To print multiple include files, hold down the **CTRL** key when making your selections.

## Producing the Document Segments

After writing the template and preparing any include files to be included in the document, you are ready to execute the template and generate the *unformatted* document segments. Each segment file contains information from various sources that you specified through the DGL statements in your template.

The Documentor permits file operations on segments independent of the sources from which they were generated. These operations work in the same way for both for templates and include files.

This section describes how to use the following options in the Document Management dialog box:

- ◆ **New**—Generates segments for a new document
- ◆ **Edit**—Modifies any of the generated segments
- ◆ **Delete**—Deletes a document
- ◆ **Regenerate**—Regenerates specific segments of the document without re-executing the entire document.

For information on the **Export**, **Print**, and **Format** options, refer to [Producing the Final Document](#).

## Creating and Manipulating Documents

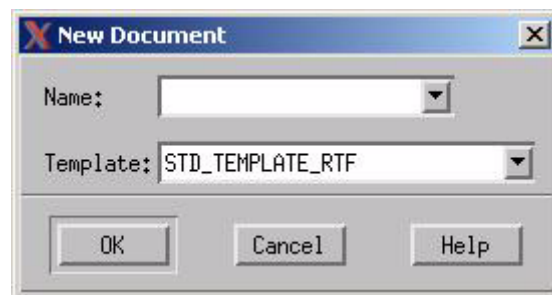
To work with documents, click the **Documents**  icon in the toolbar or select **Edit > Documents** from the main menu. The Document Management dialog box lists the documents in your workarea.



## Creating a Document

To create a document, follow these steps:

1. Click **New**. The **New Document** dialog box opens.



2. Enter a name for the new document in the text box. The name must begin with a letter, and can contain only alphanumeric characters or underscores.

The name must be unique in this project. If a document with this name already exists, the system displays a warning. You can cancel or confirm your choice to overwrite the old document. To modify an existing name, select a name from the drop-down list and modify it in the text box.

3. Enter the name of the template you want to execute, or select it from the list of templates.

4. Click **OK**.

The Documentor checks the template for errors. If there are compilation errors, the tool displays a warning. You must correct all the errors before you can generate any segments. Refer to [Creating a Template](#) for more information. When all of the errors have been resolved, click **New** again.

If there were no errors, the Generate Document dialog box opens, as shown in the figure. This dialog box enables you to select all or some of the template segments and enter values for the parameters to be used when generating the document.

The Generate Document dialog box contains two tables:

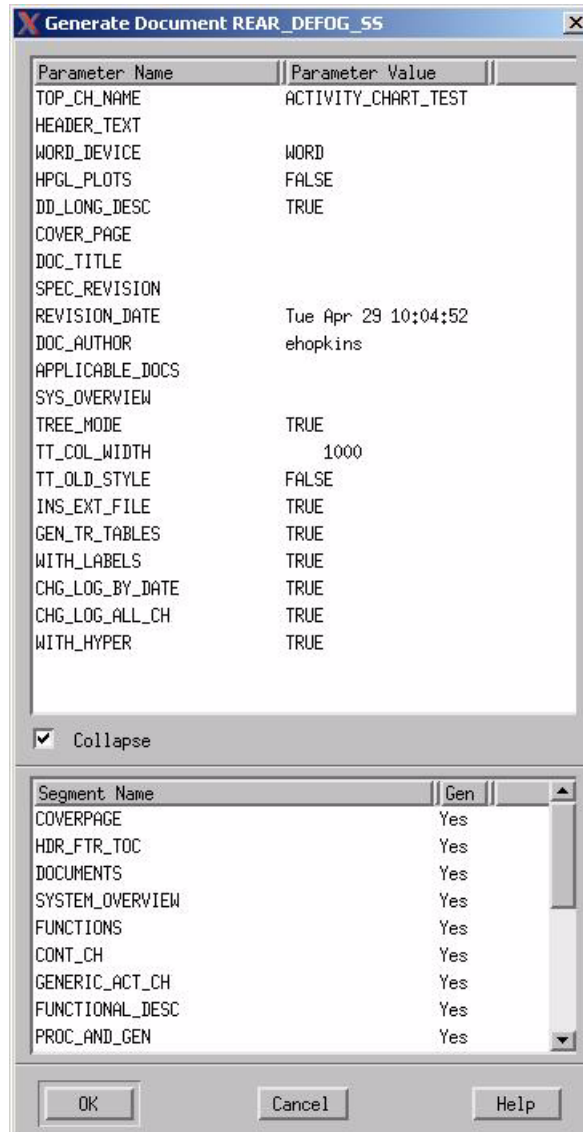
- a. **Parameter Name**—Displays the current parameter values. If you assigned initial values to the parameters in the template, these values are displayed in the **Parameter Value** column. If you change these values, the new values are stored and displayed prior to any regeneration of the document segments. (For more information, see [Regenerating Document Segments](#).)
- b. **Segment Name**—Lists the document segments, all of which will be generated by default (see the Yes value in the **Gen** column). If you do not want to generate a specific segment, click on the Yes next to that segment and the field changes to No.

By default, the view is collapsed so the segment information is not displayed. Click **Expand** to expand the view.

Regardless of the template segments that you select for execution, the initiation section of the template is always executed. Refer to [Overview of Documentor](#) for an explanation of template segments.

5. Enter values for the relevant parameters shown in the form. (For more information on template parameters, refer to [Reusing Templates](#).)
6. Click **OK** to apply your changes and dismiss the dialog box.

The tool generates the segments and the new document name is included in the list in the Document Management dialog box.

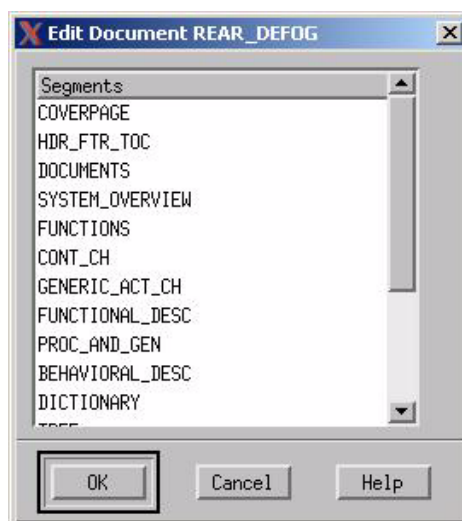
**Generate Document Dialog Box (shown expanded)**

## Editing a Document

After generating the document, you can edit any of the generated segments. The **Edit** option enables you to make minor changes in a generated document segment, without having to edit and re-execute the entire document template.

To edit a document segment, complete the following:

1. Select the document from the list in the Document Management dialog box.
2. Click **Edit**. The **Edit Document** dialog box opens, as shown in the following figure.



3. Select the segment you want to edit.
4. Click **OK**. The segment opens in the system's text editor.
5. Make your changes, then select **File > Save** and **File > Exit** in the text editor.

## Deleting a Document

To delete a document, complete the following:

1. Select the document from the list in the Document Management dialog box.  
To delete multiple documents, hold down the CTRL key when making your selections.
2. Click **Delete**.
3. The Documentor prompts you to confirm the deletion. Answer **Yes**.



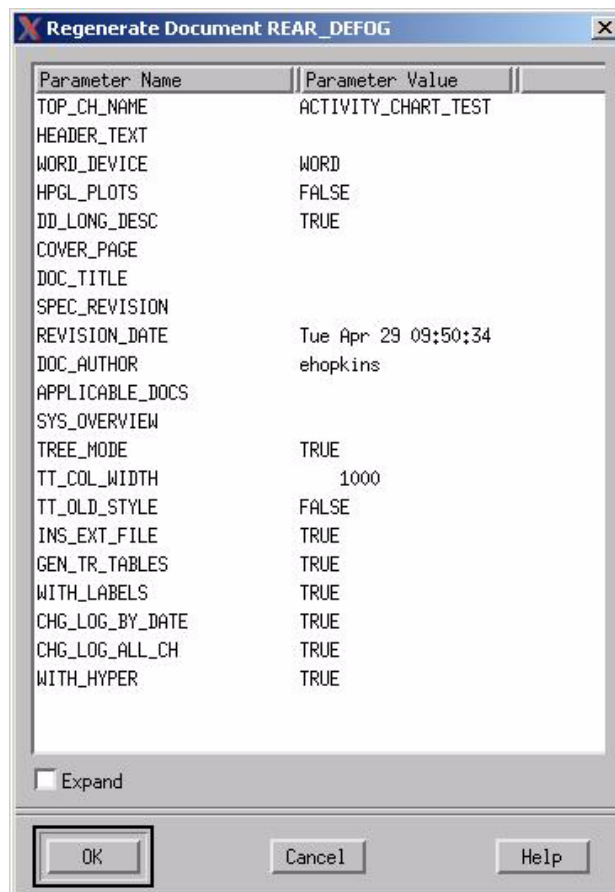
## Regenerating Document Segments

The Documentor enables you to regenerate particular segments of your document without re-executing the entire template. After generating some of the document's segments, you might want to go back and generate other segments that you did not originally select. Perhaps you altered the template or changed your specification. You will need to regenerate those segments affected by the modifications.

Using the **New** option for this task is inefficient. When you create a new document, any previous segments for the document are erased, then the specified segments are generated. This is why Documentor provides an additional document operation, **Regenerate**. This option allows you to redo particular segments, without affecting existing segments of your document.

To regenerate a document segment, complete the following:

1. Select the document from the list in the Document Management dialog box.
2. Click **Regenerate**. The Regenerate Document dialog box opens.



3. By default, all segments are selected. Unselect the segments that you do *not* want to regenerate by clicking on the Yes next to it; the field changes to No.

You can change parameter values. However, changing the parameter values can be problematic when regenerating segments. It is usually undesirable to have some segments generated with one set of parameter values, and other segments created with a different parameter set. If you alter some parameters and regenerate a portion of your document, the Documentor asks you to confirm the regeneration.

4. Click **OK** to apply your changes and dismiss the dialog box.

---

## Producing the Final Document

After generating the document segments, you output them as an assembled finished document. If the document segments contain formatting instructions, you send them to a formatter to produce the final document. This procedure also depends on whether the Documentor can invoke the formatter. For example, Interleaf cannot be activated from within the Documentor.

Select one of the following three ways to produce the final document depending on your formatter:

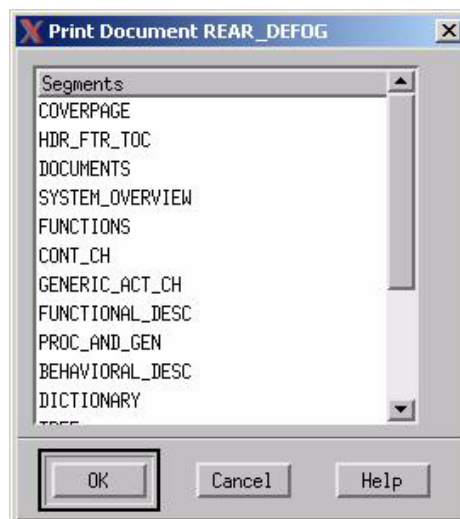
- ◆ For ASCII and PostScript files, use **Print**.
- ◆ For FrameMaker, Interleaf, Word, and troff files, use **Export**.
- ◆ For Glyph files, use **Format**.

### Printing a Document

Use **Print** for document segments that do *not* contain formatting instructions (such as ASCII and PostScript) that you want to send directly to your local printer. This option is also useful for shipping a document in a machine-readable format to another site.

To print a document, follow these steps:

1. Select the document from the list in the Document Management dialog box.
2. Select **Print**. The **Print Document** dialog box opens, as shown in the following figure.



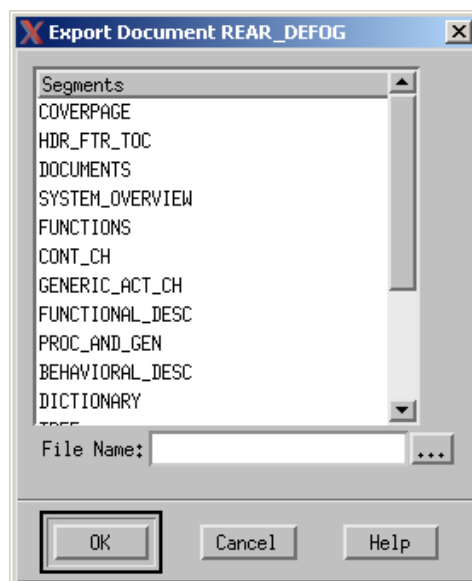
3. Select the segments you want to print:
  - ◆ Select one segment by clicking on it.
  - ◆ Select multiple segments by holding the CTRL key.
  - ◆ Do not select any segments to print all the segments.
4. Click **OK** to dismiss the dialog box and print the document.

## Exporting a Document

Use **Export** for document segments that contain formatting instructions for formatters that cannot be activated from within the Documentor (such as FrameMaker, Interleaf, Word, and troff).

To export a document to an external file, complete the following:

1. Select the document you want to export from the list in the Document Management dialog box.
2. Click **Export**. The Export Document dialog box opens with the name of the document you selected in the title bar. It lists all the generated segments for the selected document.



3. Select the segments you want to export:
  - ◆ Select one segment by clicking on it.
  - ◆ Select multiple segments by holding the CTRL key.
  - ◆ Do not select any segments to export all the segments.

4. Click **OK** to dismiss the dialog box.
5. Enter the output file path name (using the operating system conventions for your host computer). This is the file destination for the output.

**Note:** You can also enter a new name by clicking on the ellipsis (...) to display the **Export Files** dialog box.

6. Click **OK**. The tool assembles the segments in the correct order and writes them to the specified destination.
7. Invoke your formatter on that external file.

## Formatting a Document

Use **Format** for document segments that contain formatting instructions for formatters that can be activated directly from within the Documentor (such as Glyph).

To format a document, follow these steps:

1. Select the document from the list in the Document Management dialog box.
2. Click **Format**. The Format Document dialog box opens.
3. Specify the field values:
  - ♦ Select the segments that you want in your document.
  - ♦ Optionally, select additional output devices **Terminal**, **Printer**, or both.
  - ♦ Specify the output file path name. This is the file that contains the final document. The file is produced by the formatting system.
4. Click **OK**. The Documentor assembles the segments, then activates the formatter to produce the document.

## Working with Different Formatters

[Overview of Documentor](#) explains the relationship between the Documentor and formatters. To use any formatter, it must be present on your host computer. You can edit or generate Documentor files on any computer running Documentor, but you can use the **Format** option only on a host that actually runs the target formatting system.

## Using nroff and troff

nroff and troff are UNIX-based formatting systems.

nroff is a textual formatting system that prepares output for standard ASCII devices such as terminals, disk files, and printers. You can activate nroff directly from the Documentor using the **Format** option on templates attached to the troff formatter.

troff is a graphical formatting system that prepares output for laser printing devices. Despite the difference, these systems are compatible in that troff can use nroff input files to produce final output. troff cannot be activated directly from the Documentor.

### Note

---

- ◆ nroff input requires preprocessing for commands used with equations and tables. If you use such commands in your template or include files, you should *not* activate nroff directly from the Documentor. The Documentor does *not* support preprocessing of output before sending it to nroff.
- ◆ If you run nroff externally, predefined reports generated by the Reports tool use the macros library `me`.

For information on troff and nroff, refer to the UNIX system documentation for your computer.

## Using Interleaf

Interleaf is a documentation preparation system available on various computer systems. Because it is interactive, Interleaf cannot be activated directly from the Documentor. In addition, you must make sure to include the file `interleaf_glob` in the first segment of your template. The file is supplied as an include file in the databank. You must load this include file into your workarea for Documentor to access it. This file contains global definitions that are used in Statemate reports.

For more information on Interleaf, refer to the Interleaf documentation provided with the formatter.

# Document Templates

---

A template is a text file consisting of instructions for generating a document. The instructions are written in the Document Generation Language (DGL). This section, divided into three parts, explains the fundamentals of DGL:

- ◆ Principles of DGL
- ◆ Data-items and expressions
- ◆ Detailed explanation of DGL statements

Part one covers the basic concepts of DGL and shows you the general principles of template writing. In part two, we discuss identifiers and the types of data they can represent. We also detail the various kinds of expressions that you can use in DGL statements. In part three, we present each DGL statement.

In your template, you can call functions that extract the information stored in the specification database, and write this information into your document. The use of Database Extraction Functions is covered in [Overview of the Extraction Functions](#).

## Principles of DGL

DGL is a structured programming language and displays features typical of other structured languages such as Pascal. Among these are: declarations for identifiers, the use of variables, parameters and constants, and control flow statements. In this section, the principles and conventions of DGL are introduced.

## DGL Template Structure

Every template consists of two main parts:

1. Initiation section—Contains declarations and statements that pertain to the overall template. It may not include output statements.
2. Template segments—Sections of the program, each of which pertains to a particular portion of the document.

**TEMPLATE example****VARIABLE**

```
CHART    ch_id;  
INTEGER  status;
```

**BEGIN**

```
ch_id :=stm_r_ch ('CH1',status) ;
```

**SEGMENT seg 1;****VARIABLE**

```
STRING  fn:='/tmp/my_file';
```

**BEGIN**

```
INCLUDE ( fn ) ;
```

**Initiation section**

contains global declarations and

**segment**

contains local declarations and

**TEMPLATE example****VARIABLE**

```
CHART    ch_id;  
INTEGER  status;
```

**BEGIN**

```
ch_id :=stm_r_ch ('CH1',status) ;
```

**SEGMENT seg 1;****VARIABLE**

```
STRING  fn:='/tmp/my_file';
```

**BEGIN**

```
INCLUDE ( fn ) ;
```

**Initiation section**

contains global declarations and

**segment**

contains local declarations and

Template segments are the basic divisions of the document and, when executed, produce distinct output files called *document segments*; these are later assembled into the final document. The segments do not necessarily correspond to the document divisions (i.e. chapters, sections, etc.). However, we recommend that you divide the segments according to such divisions. This gives you the ability to generate separate sections or chapters individually; the initiation section, however, is executed whenever any segment is executed. Refer to the discussion on [Template Sections](#) for more information.



The initiation section and template segments all have the same overall structure:

1. **Identifier line** - (Required) Identifies the entire template or template segment by name.
2. **Declaration part** - (Optional) Contains the definitions of any constants, or variables used by the template section during execution. Identifiers declared in the initiation section are global, i.e., they may be used throughout the template; in the initiation section (only) you may include parameter declarations, as well as variables and constants.
3. **Body** - (Required) Contains execution statements. The body is composed of a BEGIN/END statement that delineates the section's statements. Any number of statements may be found in the body. The initiation section may **not** include output statements.

<pre> <b>SEGMENT seg 1;</b>  <b>VARIABLE</b>    <b>CHART      ch_id;</b>   <b>INTEGER    status;</b>  <b>BEGIN</b> <b>ch_id:=stm_r_ch('CHI',status);stm_plot</b> <b>(ch_id, 'my_file', ...);</b> . . . </pre>	<p><b>identifier line</b></p> <p><b>declaration part</b> contains local declarations</p> <p><b>body</b> contains DGL statements</p>
---	---

## DGL Syntax Rules

DGL, like most programming languages, has a particular syntax that must be obeyed. The full syntax of each DGL statement is given in [DGL Statement Reference](#). A more formal DGL syntax, written in BNF, is provided in [BNF Syntax](#).

The following is a list of general syntax rules that apply to all DGL statements:

1. DGL is not case sensitive. The exception to this rule is literal strings (inside apostrophe marks). These are utilized exactly as they appear in the template.
2. DGL statements terminate with a semicolon.
3. Multiple statements on a single line are permitted.
4. A single statement may span several lines.

5. DGL has a set of reserved words and syntactical constructs. Each of these has a special meaning and can be used only in the context for which it has been designed. The DGL reserved words are listed in [DGL Reserved Words](#).
6. An identifier (a name you assign to an object for identification purposes) can be any string, beginning with a letter and consisting of any of the following characters: A-Z, a-z, 0-9, `_`. Reserved words may not be used as identifiers.
7. Literal strings may be assigned by enclosing them within apostrophe marks (`'`). Any character sequence can be used inside literal strings.
8. Comments are preceded by a double dash (`--`) symbol. This symbol can be placed anywhere in the line except within a literal string. After the comment symbol, all other characters on the line are ignored.
9. Extra blank spaces within or between statements are ignored.

```
SEGMENT seg 1;
VARIABLE
  chArt      ch_9;
  INTE-     status;
  ch_9 := stm_r_ch
  ('CH1' , status);
BEGIN
WRITE ( ' This is a Plot ' );
STM_PLOT (ch_9, 'my_file', ...);
END;
-- this keyword ends
-- this section
```

not case sensitive

statements terminate with a semicolon.

identifier can be any string beginning with a letter and consisting of characters A-Z, 0-9, `_`

single statement may span two lines.

literal strings enclosed in apostrophes.

comment lines preceded by a double dash.

extra blank spaces within statements or between statements are ignored.

## Special Features of DGL

We mentioned before that DGL resembles other structured programming languages. DGL also includes special constructs and features important for the document generation process.

### Extensions to Conventional Programming Constructs

Some of the features unique to DGL are:

- ◆ **Verbatim Inclusion**—Text may be included in a template and passed “as is”, i.e. literally to the output document segments. This is particularly useful for passing formatting commands to the formatter.
- ◆ **Include File**—Text from another file may be copied to the output document segments.
- ◆ **Calling External Programs**—External programs (e.g., operating system services) may be called from within the template.
- ◆ **Include Statemate Reports and Plots**—Statemate predefined reports and graphical plots may be included in the document.

### Database Extractions

A set of functions can be used to extract database information regarding specific elements and produce lists of elements according to specified criteria. Database extraction functions and their use are explained in [Documentor Functions](#).

There are several kinds of functions:

- ◆ **Single-Element Functions**—Return element details (as strings, or numbers). Such functions can be used to return an element name, synonym, type, definition, short description, or attribute values. Parts of an element’s long description may be retrieved through the use of keywords.
- ◆ **Query Functions**—Retrieve a list of elements having a certain relationship with other specified elements or having a specified attribute value. For example, you may retrieve all descendants of a given activity and store them in a list.
- ◆ **Utility Functions**—Perform operations and manipulations (e.g. sort) on lists, single integers and strings.

## Overview of DGL Statements

Here is a brief overview of DGL statements. Subsequent sections provide you with the full range of available statements and their syntax. DGL is written as statements, each of which is a specific command to the Documentor Tool. The types of statements vary, and most are similar to constructs in other languages.

DGL statements consist of the following types:

- ◆ **Structure statements**—Define the structure of the template.
- ◆ **Declaration statements**—Declare the type of identifier for variables, constants and parameters.
- ◆ **Assignment statement**—Assign a value to a variable.
- ◆ **File handling statements**—Open or close files or the dialog area, and read data from files.
- ◆ **Output statements**—Pass information to output document segments, or to the dialog area.
- ◆ **Control flow statements**—For conditional and iterating execution of statements.

<b>SEGMENT seg 1;</b>	structure statement
<b>VARIABLE</b>	declaration statement
<b>CHART        ch_id;</b>	
<b>INTEGER      status;</b>	
<b>BEGIN</b>	structure statement
<b>ch_id :=        stm_r_ch ('CH1' ,status) ;</b>	database extraction function in an assignment statement
<b>IF status = stm_success</b>	control-flow statement
<b>THEN</b>	output statement
<b>WRITE ('This is a Plot');</b>	output statement (plot statement)
<b>STM_PLOT(ch_id, 'my_file', ...);</b>	control-flow statement
<b>ELSE</b>	output statement
<b>WRITE('Retrieval failed' );</b>	
<b>END IF;</b>	structure statement
<b>END;</b>	

## Data-types and Expressions

Identifiers are names that are used in a template and can represent constants, variables or parameters. The differences between these are noted here:

- ◆ **Constants** are identifiers whose values are constant and cannot be changed in DGL statements.
- ◆ **Variables** are identifiers whose values can change in DGL statements.
- ◆ **Parameters** are variables whose values may be assigned in a special form before executing the template.

Before you use an identifier, you must declare whether it is a constant, variable or parameter. Furthermore, you must declare the particular *type* of value that can be assigned to the identifier; we call this type a *data-type*.

For instance, an identifier (whether constant, variable, or parameter) may be declared to hold *integer* values, *float* values, *string* values, etc.

Identifiers can be combined to construct expressions; these can be used in various kinds of statements and database extraction functions. Various kinds of expressions can be constructed, depending on the data-types of the identifiers of which they are constituted.

For instance, you may construct numeric or string expressions as well as Statemate element expressions.

This section deals with both data-types and the expressions you can construct from them. In addition, *enumerated types*; variables or return values of a function that take a restricted number of discrete values, are described.

### Data-types

You declare identifiers and their data-types in declaration statements; these are described in later sections. Below we list the various data types that are recognized in DGL declaration statements.

#### Conventional Types found in Other Programming Languages

INTEGER (numeric)  
FLOAT (numeric)  
Boolean  
STRING  
FILE

## StateMate Element Types

These data-types are StateMate elements. Typically, you use identifiers of this type in StateMate database extraction functions. For example:

```
ACTIVITY    ac;  
STRING      ac_name;  
.  
.  
ac := stm_r_ac (ac_name , st);  
WRITE('activity synonym is ',  
      stm_r_ac_synonym(ac,st));
```

The variable `ac` is declared here to be of type `ACTIVITY`.

A database extraction function assigns a value to `ac` (the value is an activity whose name is the value of the variable `ac_name`). In the `WRITE` statement that follows, `ac` is used as an argument in another function that returns the synonym for the activity represented by `ac`.

There are two kinds of type declarations for StateMate elements:

- ◆ **ELEMENT** - this declaration allows you to assign any StateMate element to the identifier.
- ◆ **Specific StateMate element Types** - The following types are recognized:

```
ACTION  
ACTIVITY  
A_FLOW_LINE  
CHART  
CONDITION  
CONNECTOR  
DATA_ITEM  
DATA_STORE  
DATA_TYPE  
EVENT  
FIELD  
FUNCTION  
INFORMATION_FLOW  
M_FLOW_LINE  
MODULE  
REQUIREMENT  
STATE  
TRANSITION
```

The variables having StateMate element data-type hold an ID number which is used for internal representation.

You may use identifiers of type `ELEMENT` in place of using identifiers with specific type declarations. For instance, in the previous example, we could have legitimately declared the variable `ac` to be of type `ELEMENT` instead of type `ACTIVITY`.

However, there are advantages to using specific element type declarations. When you use a specific element type declaration, identifier assignments and function parameters are checked to ensure that values are of the proper type. This is of particular importance for assignments by database extraction functions. For instance, in this above example, the function `stm_r_ac` returns an activity ID that is assigned to `ac`; if we use `ac` with a function call that returns a module, the Documenter detects this as an error during template compilation. If, however, we had declared `ac` to be of type `ELEMENT`, no syntax error would have been detected.

In some cases, you must declare an identifier to be of type `ELEMENT` instead of being of a specific element type. This case comprises identifiers that are assigned values from a list of elements of mixed type (i.e., of more than one element type). For an illustration of this, see the example below for type `LIST OF ELEMENT`.

### **LIST OF `simple_type`**

Identifiers declared as `LIST OF simple-type` can be assigned values of a list of items of any data-type. Simple-type may be any of the above types (e.g. `LIST OF STRING`, `LIST OF ACTIVITY`, etc.).

The following example shows a typical use of identifiers of this type; in this case, we have declared an identifier of type `LIST OF STATE`.

```
VARIABLE
  LIST OF STATE      sub_st;
  STATE              st_id;
  INTEGER            st;
BEGIN
  st_id := stm_r_st ('S1' , st) ;
  sub_st := stm_r_st_logical_sub_of_st({st_id},st) ;
END;
```

In this example, the variable `st_id` is assigned the ID of the state `S1`.

This variable is then used with another database extraction function to extract all of the substates of `st_id` and assign their values to the variable `sub_st` of type `LIST OF STATE`.

## Expressions

Parameters, variables and constants are used to build expressions, such as  $a + 5$ .

Expressions such as these may be used in assignment statements, in calls to predefined functions, and in Boolean expressions (comparisons). In addition, functions themselves may participate in expressions.

Expressions can be constructed for any of the data-types:

- ◆ **NUMERIC:** for example,  $a*(b+3.2)$
- ◆ **STRING:** for example, 'DATA-ITEM DICTIONARY'
- ◆ **Boolean:** for example,  $A > B$
- ◆ **Stemate element:** for example, `stm_r_ac ( 'AA' , st)`
- ◆ **LIST:** for example,  $a\_list + b\_list$

In the following sections, we present a more detailed explanation of the expression types that you can construct.

### Numeric Expressions

These are integer and real numbers. Their constant values are the same as in conventional programming languages.

Binary operations (  $+$  ,  $-$  ,  $*$  ,  $/$  ,  $**$  ) and unary operations (  $-$  ,  $+$  ) follow conventional precedence rules.

Numeric expressions may mix integer or real operands.

Parenthesis may be used to change the precedence of operations.



## Boolean Expressions

The basic Boolean expressions are comparisons between expressions of other types. Note that not all comparisons are legal for all data-types.

Operator	Meaning	Allowed types
=	Equality	all
<>	inequality	all
<	Less than	numeric and lists
<=	Less than or equal to	numeric and lists
>	Greater than	numeric and lists
>=	Greater than or equal to	numeric and lists

For LIST types, Boolean comparisons are understood in terms of inclusion. For example, for lists A and B,  $A < B$  if B contains all elements of A and also other elements.

The Boolean operations NOT, OR, and AND are also supported.

In addition there are two predefined Boolean constants: TRUE and FALSE.

## String Expressions

String literals are written within apostrophes, for example: 'ABC'

Spaces within literal strings are always considered. For example, 'A BC' is different from 'ABC'. String literals may contain formatting characters using the backslash character (\) character:

- ◆ \n: inserts a new line in the string.
- ◆ \t: inserts a tab in the string.
- ◆ \: the backslash, when followed by any other character, includes that character literally into the string. This is not intended for use with alphanumeric characters but for including special characters in the string - especially \ and '.

String expressions may include string constants, string variables, and functions that return strings.

Concatenation of strings is supported; it is indicated by the "+" sign. For example: 'str1' + 'abc' results in 'str1abc'.

## Statestate Element Expressions

Expressions of this type may be either declared variables or function calls that return a Statestate element (refer to [Documentor Functions](#)). For example, `md := stm_r_md (md_name , st);` returns a value of type MODULE and assigns it to the variable `md` in the assignment statement.

As another example, consider the following:

```
stm_r_md_synonym (stm_r_md ('M1' , st1) , st2)
```

Here we use the module extracted by the function `stm_r_md` as an argument in another function.

There are no constants for Statestate elements. The variables of the above types get their values via the Statestate predefined functions, and are used as arguments in other predefined functions to retrieve additional information. Statestate element ID numbers are used as values of these expressions.

## List Expressions

This type is used to handle collections of items of any of the DGL types. A list is created either by explicitly enumerating the items in the list, or as a result of a function call. Explicit enumeration is written as:

```
{ list_item , list_item , ... }
```

as in the string list: `{ 'abc' , 'def' , 'xyz' } .`

A list item must be an expression of the list type.

All items in the list must be of the same type. Lists can be created from other lists using the operations: *union* (+), *subtraction* (-), *intersection* (\*), and *concatenation* (&).

This last operation differs from *union* when two identical lists are used, for instance:

```
{ 'Alpha' } + { 'Alpha' } produces { 'Alpha' },  
while { 'Alpha' } & { 'Alpha' } produces { 'Alpha' , 'Alpha' } .
```

A list expression can also be built as a result of a function. For example,

```
stm_r_md_name_of_md ( 'M*' , st) .
```

The value of this list expression is all modules whose names begin with `M`.

## Enumerated Types—Predefined Constants

A variable or return value of a function is considered of *enumerated type* if it may take a restricted number of discrete values. For example, a variable that represents a day of the week may have only seven values, Sunday through Saturday.

DGL does not directly support enumerated types. The way to deal with variables of such a type is to declare them as INTEGER, and to define constants that are equal to the specific possible values. In order to make the template clearer, you may use meaningful names for these constants. For instance, you may define constants for the days of the week: SUNDAY:= 1, MONDAY:= 2, and so on.

The Documentor has several sets of *predefined* constants used as enumerated types. The names for these constants always begin with the prefix *stm\_*.

For example, a very useful enumerated type is Element Type. The possible values for this type are *stm\_state*, *stm\_event*, etc; each of these identifiers has a unique integer value.

You may use predefined constants in your template without even knowing their numerical values. However, you must make sure that variables that are to be assigned enumerated values are declared as INTEGER.

For example, another widely used predefined enumerated type is the function *return status code* (refer to [Documentor Functions](#)). Status codes have a restricted number of values, each value denoting some information about the operation of the function being used.

For example, *stm\_success*, denoting the successful completion of the function operation, corresponds to the value 0. You do not have to explicitly use the value 0 when writing this status code into your template; for instance, you can write:

```
VARIABLE
  INTEGER    status ;
  .
  .
  md_id := stm_r_md ('M1' , status);
  IF status = stm_success THEN
  .
  .
```

Predefined enumerated types are listed in their relevant sections.

For instance, an enumerated type that is returned by a certain function is listed in the description of that function.

## DGL Statements

This section presents the DGL statements and their syntax, along with examples of their use.

### Structure Statements

These statements define the structure of the template. There are four statements that define the structure of your template:

- ◆ **TEMPLATE statement**—First statement of the template.
- ◆ **SEGMENT statement**—Starts a new segment section.
- ◆ **PROCEDURE statement**—Starts a new procedure section.
- ◆ **BEGIN/END statement**—Marks the boundaries of a template section.

In addition, Comment statements can be used to indicate program comments.

Next is a more detailed discussion of each structure statement.

#### TEMPLATE Statement

##### Statement Syntax:

```
TEMPLATE template_name ;
```

The **TEMPLATE** statement is the first statement in the template, and assigns an identifying name to the template. This name is used for internal documentation purposes only and does not have to correspond to the name you use to designate the template in the *Create Template* form.

#### SEGMENT Statement

##### Statement Syntax:

```
SEGMENT segment_name ;
```

The **SEGMENT** statement starts a new segment section. It is the first statement of a segment, and assigns an identifying name to it. The identifier is limited to 16 characters maximum. The segment name is used by the tool in its operation forms to identify the output segments.

## PROCEDURE Statement

### Statement Syntax:

```
PROCEDURE procedure_name [RETURN type] ;
```

The PROCEDURE statement begins a procedure section. It assigns an identifying name to the procedure and defines the return type, if the function returns a value. The procedure name is limited to 16 characters maximum.

## BEGIN/END Statement

### Statement Syntax:

```
BEGIN
    statements
END;
```

The BEGIN/END statement delineates the *body* of a template section. Recall that a section *body* contains DGL statements to be executed by the program. You may included any number of statements between BEGIN and END.

## Comment Statement

### Statement Syntax:

```
-- free text
```

DGL may include programmer comment lines in the template. These are lines of free text that are not interpreted or handled by the Documentor during execution. Such comments are useful for documentation purposes.

Comments are preceded by two dashes. The comment symbol may start anywhere in a line, except within a literal string. All characters after the comment symbol until the end of the line are ignored.

An example of a comment:

```
-- This line contains program comments
```

## Declaration Statements

You declare identifiers and their data-types in declaration statements in the declaration part of each template section.

Declarations for global identifiers and parameters are made in the initiation section - these identifiers can then be used throughout the template.

Declarations for local identifiers, i.e., identifiers to be used only in the relevant segment, are made after each segment identifier line. Procedures start with declarations of parameters and local identifiers.

### PARAMETER Statement

#### Statement Syntax:

```
PARAMETER data-type identifier [:= value] , ... ;
```

#### For Example:

```
PARAMETER STRING activity_name ;
```

There are “template parameters” for the entire template and “procedure parameters” for procedures. Template parameters are variables whose value may be changed interactively when the template is executed.

Declaration of template parameters is allowed only in the initiation section. The keyword `PARAMETER` appears only once in the declaration section, before the data-type assignments for parameters. Each data-type statement may be followed by as many identifiers of the same type as you want to define.

#### For Example:

```
PARAMETER STRING activity_name, state_name, event_name;
```

As many type statements as desired may follow the `PARAMETER` keyword.

#### For Example:

```
PARAMETER  
    STRING activity_name;  
    FLOAT a:=3.243;
```

Template parameters may **not** hold a *Stemate* element, while it is legal for a procedure parameter to be of this type. Procedure parameters are In/Out parameters.

Value assignments for PARAMETER statements are optional and allowed only for template parameters (we made one such assignment in the above example). If it is assigned, it represents the default value of the parameter at the first generation of a particular document. The value may only be a *literal* constant, not a constant identifier or an expression.

### Note

---

Avoid changing template parameters within the template. It may cause confusion and can create inconsistent results when parameters are changed within segments.

## CONSTANT Statement

### Statement Syntax:

```
CONSTANT data-type identifier := value , ... ;
```

Constants are identifiers that have a defined value that cannot be changed in DGL statements.

The keyword CONSTANT appears only once in the declaration section, before the data-type assignments for constants. Each data-type statement may be followed by as many identifiers of the same type as you want to define.

### For Example:

```
CONSTANT integer a:=1, b:=2, c:=3;
```

As many type statements as desired may follow the CONSTANT keyword.

### For Example:

```
CONSTANT  
  STRING activity_name:='Print';  
  FLOAT a:=3.243;  
  INTEGER c:=6;
```

The constant type may **not** be a *Stemate element*, or a *list of type*.

The identifiers in the CONSTANT statement must have their values assigned in the statement. The value may be any expression not containing variables or parameters.

## VARIABLE Statement

### Statement Syntax:

```
VARIABLE data-type identifier [:= value],... ;
```

Variables are identifiers whose values may be changed in other DGL statements.

The keyword VARIABLE appears only once in the declaration section, before the data-type assignments for variables. Each data-type statement may be followed by as many identifiers of the same type as you wish to define.

### For Example:

```
VARIABLE STRING act_name, act_syn, act_desc;
```

As many type statements as desired may follow the VARIABLE keyword. For example:

```
VARIABLE
    string activity_name;
    float a:=3.243;
    activity act_id;
```

Value assignments are optional (we made one such assignment in the above example). If they are assigned, they represent the default value of the variable at the first generation of a particular document. The value may be any expression that does not contain other variables or parameters.

Variables that are declared as *Statement elements* and *list of items* may not be assigned initial values.



## Assignment Statement

### Statement Syntax:

```
variable := expression;
```

This statement should be interpreted as follows: the variable on the left-hand side of the statement is assigned the value of the expression on the right-hand side.

The expression and the variable must be of the same or compatible type. If the expression is of type STATE, ACTIVITY, etc., the variable is of the same type, or of type ELEMENT.

Here is an example of a template with a declaration section, followed by a section containing an assignment statement.

```
VARIABLE
  LIST OF STATE      st_list;
  LIST OF ACTIVITY   act_list;
  LIST OF ELEMENT    el_list;
BEGIN
  .
  .
  el_list := st_list + act_list;
  .
  .
END;
```

## File Handling Statements

These statements open and close files or the dialog area, to and from which you can pass text or messages, through the use of WRITE and READ statements. The READ statement is also included in the file handling statements, whereas the WRITE is considered to be an output statement.

### OPEN Statement

#### Statement Syntax:

```
OPEN (f1, file_name, mode [, status]);
```

where *f1* is an identifier of type FILE and *mode* is either INPUT or OUTPUT.

This statement is used with *mode*=OUTPUT to open a file or the dialog area so that a subsequent WRITE statement can pass text to it, and with *mode*=INPUT for subsequent READ statements. The statement assigns a value to *f1*.

The dialog area is frequently opened to pass run-time messages to it. To open the dialog area, use the string 'DIALOG' (with the apostrophes) for the *file\_name*.

Optionally, you may include the status function code *status* which returns the value *stm\_success* upon successful execution of the statement.

If the file opened for output does not exist, this statement creates a new file.

If the file exists, it is initialized by the OPEN statement; i.e., the written information overwrites the existing contents of the file.

### **CLOSE Statement**

#### **Statement Syntax:**

```
CLOSE (f1);
```

where `f1` is an identifier of type FILE. This statement closes a file that was previously opened with the OPEN statement.

### **READ Statement**

#### **Statement Syntax:**

```
READ(f1, variable1, variable2, ...) ;
```

where `f1` is an identifier of type FILE that points to a file that was opened using an OPEN statement, in INPUT mode. The variables are identifiers of type *integer*, *float*, or *string*.

Each READ statement reads a line from the file. The numeric elements in the input line are separated by blank or tabs. Reading to a string variable reads the rest of the line.

#### **For Example:**

```
READ(fd, i, str)
```

where `i` is an integer and `str` is a string. This statement, when applied to an input line: 12May1991, results in: `i=12, str='May 1991'`.

The READ statement may operate as a function that returns either *stm\_success*, *stm\_cannot\_read\_file*, or *stm\_end\_of\_file*.

## Output Statements

Output statements pass text to the output segment files. The text can originate from a number of sources: the template itself, user text files, database retrieval functions, Statemate reports and plots, external programs, and evaluated expressions. Output statements cannot be put in the initiation section.

Below, we detail the various types of output statements.

### Verbatim Statement

#### Statement Syntax:

```
/@ verbatim text @/
```

When the verbatim symbols `/@` and `@/` frame text in the template file, the text is passed literally (without interpretation) to the output segment file. Comments inside the frame, rather than being ignored, are passed literally as well. The end-of-statement character, `“;”`, is not required following the concluding verbatim symbol.

Verbatim statements may be used to pass the following to the output file:

- ◆ Formatting commands applicable to a specific formatter.
- ◆ Short text passages such as titles, opening remarks, etc.  
In spite of the absence of any length restriction on verbatim text, longer text passages are usually passed using the `INCLUDE` file statement.

For example, when this verbatim section of the template is executed:

```
SEGMENT section1;
BEGIN      /@
.title ACTIVITY-SPEC
.skip 2
.center; AN ACTIVITY SPECIFICATION
-- This section will describe the
-- purpose of the activities.
@/
-- this line will not appear in the output
END ;
```

it results in the following output:

```
title ACTIVITY-SPEC
.skip 2
.center; AN ACTIVITY SPECIFICATION
-- This section will describe the
-- purpose of the activities.
```

## WRITE Statement

Statement Syntax:

```
WRITE ( [f1,] write_expression , ... ) ;
```

The WRITE statement to write expression values to any of the following

- ◆ The document output segment
- ◆ Another file
- ◆ The dialog area of the tool window

You may write a numeric or string expression that is evaluated in the template, or a literal piece of text. The WRITE statement can also be used to write information retrieved from the database, such as element names.

Using the WRITE statement to write to a file or to the dialog area is particularly useful if you want to write messages (error messages, run-time messages, etc.). When writing to a file or to the dialog area, you must include the `f1` identifier. In such cases you must also precede the WRITE statement with an OPEN statement.

The WRITE statement is commonly used to write lines that include text (string literals) together with expression values.

For example: `WRITE('NAME:', di_name);`  
results in the following being written in the output segment file `NAME: KUKU`, where *KUKU* is a value of `di_name`.

From the examples you can see that there can be more than one write expression. When there are multiple expressions they are separated by commas.

Lines of pure text are more suitably handled using the Verbatim statement.

Literal strings may include the formatting characters:

```
\n - new-line  
\t - tab
```

For example, `WRITE('\n', alpha);`  
would write the value of `alpha` at the beginning of the next line in the output segment file.

Optionally, you can specify the minimum number of characters to be written in the output file. You do this by using the following syntax for the write expression:

```
expression : num
```

where `expression` can be either a numeric or a string expression, and `num` is an integer constant or integer expression that represents the minimum number of characters that `expression` will occupy. `expression` and `num` may involve operands, operations, and function calls.

**For Example:**

```
WRITE(act_name: 10, ', ' , act_synonym);
```

results in the string value for `act_name` being written in the output file to a length of at least 10 characters; if the name has less than this number, blanks are added to achieve the specified string length - as in:

```
COMP      , SET
```

In this example, spaces have been added to “COMP” to give it a length of 10 characters.

The use of `num` determines the minimum number of characters to be written in the output file, as follows:

- ◆ For a string, the length of the string is the minimum number of output characters. When specified, and where `num` is greater than the string length, blanks are padded to the right of the string to achieve a total string length of `num`.
- ◆ For an integer, when `num` is specified, and where `num` is greater than the number of digits in the integer, blanks are padded to the left of the number to achieve a total output length of `num`.
- ◆ For a real number, when `num` is **not** specified, the value is output to no more than 8 decimal places. Thereafter, the number is automatically rounded. When specified, and where `num` is greater than the digits output according to the default above, blanks are padded to the left of the number to achieve a total output length of `num`. Where `num` is less than the digits output according to the default above, the decimal portion of the number may be rounded to arrive at a specified output length of `num`. However, in no case will the integer portion of the real number be truncated.

**For Example:**

```
WRITE ('Name':8, name, '\n', 'Value':8, v) ;
```

Assume that `name` contains “Xfactor” and `v` is an integer that equals 5105. This writes the following lines to the output file:

```
Name:   Xfactor
Value:  5105
```

**Note**

---

The WRITE command may not access a *list* variable directly; if you attempt to write a variable which refers to a list, an error message is displayed. To output a list, use a control flow construct such as a loop - writing one list item at a time.

## Using WRITE to Produce Messages

A WRITE statement may also be used to write information to a file or to the dialog area, instead of to the document itself. For this, you must first open a file in OUTPUT mode using the OPEN statement.

To write a string message to the file use the following syntax:

```
WRITE (f1, write_expression);
```

where `f1` is the file pointer to which you want to write the messages.

## INCLUDE Statement

### Statement Syntax:

```
INCLUDE (file_description [, status]);
```

where `file_description` is either a *file-name* or an identifier of type FILE.

This statement copies the text of a specified file to an output segment file. The text is passed to the file verbatim and may contain formatting commands for the format processor that is used to produce the formatted document.

When the `file_description` is a *file\_name*. It can be any string expression; it may be a literal string inside quotes (e.g. 'ABC') or an evaluated expression that produces a file name (e.g., a variable that contains a file name).

### Note

---

The *file\_name* may include the directory pathname of the file, using the file name conventions of the host operating system. If you do not specify the pathname, the Workarea is searched for the file. When the `file_description` is of type FILE, the file from which you are copying text must first be opened with an OPEN statement in OUTPUT mode.

When using the INCLUDE statement you may optionally include the status function code `status` which, upon successful execution of the statement, returns a value of *stm\_success*.

## EXECUTE Statement

### Statement Syntax:

```
EXECUTE (calling_sequence);
```

This statement invokes a program that is external to Statemate. The Documentor searches for the program name using the regular system search path. The tool then invokes the program and sends its standard output to the document's segment file.

The `calling_sequence` is a string expression containing the name of a program and any arguments. This is exactly the calling sequence used to invoke the program from the operating system.

The calling sequence can be a literal string in quotes or an expression that evaluates to a string.

### For Example:

```
EXECUTE ('DATE') ;
```

calls the operating system function `DATE` and prints the date in the output file.

The `EXECUTE` statement function operates as a function that returns either *stm\_success* or *stm\_error*.

## Include Reports Statement

### Statement Syntax:

```
stm_rpt_report_name ( report_parameters ) ;
```

This statement generates and writes a Statemate predefined report such as a Tree, Structure, Interface etc. into the output file. It activates the Reports Tool, generating the specified report with the given parameters.

For example, the statement:

```
stm_rpt_tree(list,5);
```

produces a Tree Report for the items in a list represented by the variable `list` to a depth of 5 in the hierarchy. The report is included in the output file.

The output from the Reports Tool contains formatting commands applicable to the format processor attached to the template. If no formatter is specified, the Reports Tool cannot be invoked and an error message is issued.

Each of the predefined reports is invoked for a list of elements. The input parameter that represents this list is denoted by a variable name that must be of type *list* of one of the Statemate element types. This variable, along with all other identifier names used in the calling sequence, must be declared in an appropriate declaration section. For example:

```
VARIABLE
  LIST OF ACTIVITY ac_list;
```

The identifier `ac_list` may be assigned a list of activities and then be included in a statement that generates a Dictionary Report, like this:

```
stm_rpt_dictionary(ac_list, . . . );
```

The report is generated for each item in the list represented by the `ac_list` variable.

A number of arguments are used to define the parameters for each report. Some are called *single character string arguments*. These are used to indicate restricted parameter choices. For example, in the Interface Report statement

```
stm_rpt_interface(elist, 'A', ...);
```

the value of the second argument 'A' indicates that the Interface Report should be of type *activities*; specifying an 'M' for this parameter would indicate that the report should be generated for *modules*.

The single character string arguments can be more than one character, but only the first character of the string is actually passed to the Reports Tool. If a non-valid character is passed to the Reports Tool, the report is not generated, and an error status code is returned.

Some of the arguments are Boolean and are evaluated as true or false to indicate whether or not some parameter is set. For example, in the Dictionary Report statement `stm_rpt_dictionary(elist, true, ...)`; the Boolean constant `true` indicates that the *long description* will be included in the report.



The following is a list of the Statemate reports that can be invoked, together with their calling name, and required parameters.

◆ **Dictionary Report**

```
stm_rpt_dictionary(elist, ldes, attr, attr_title) ;
```

`elist` is a list expression of Statemate elements for which the report is produced.

`ldes` is a Boolean expression indicating whether or not the long description of each element should be included in the report.

`attr` is a Boolean expression indicating whether the attributes of an element are included in the report.

`attr_title` is a string indicating the attribute names whose value will precede the element name in the report.

◆ **Tree Report**

```
stm_rpt_tree(elist, depth) ;
```

`elist` is a list expression of Statemate elements for which the report is produced.

`depth` is an integer argument indicating to what hierarchical level the report should be generated. For all levels, enter the number "99".

◆ **Protocol Report**

```
stm_rpt_protocol(elist, attr_title) ;
```

`elist` is a list expression of Statemate elements for which the report is produced.

`attr_title` is a string indicating the attribute name whose value will precede the element name in the report.

◆ **List Report**

```
stm_rpt_list(elist) ;
```

`elist` is a list expression of Statemate elements for which the report is produced.

◆ **Structure Report**

```
stm_rpt_structure(elist, width) ;
```

`elist` is a list expression of Statemate elements for which the report is produced.

`width` is an integer argument indicating the page width (in characters) to be used for the report.

◆ **Attribute Report**

```
stm_rpt_attribute(elist, attrs, attr_title) ;
```

`elist` is a list expression of Statemate elements for which the report is produced.

`attrs` is a list of strings that contains the specific attribute names for which the report should be generated. If this list is empty, then the report retrieves all the attributes for each element.

`attr_title` is a string indicating that the attribute value will precede its element name in the report.

◆ **Interface Report**

```
stm_rpt_interface(elist, rtype, chart, lact  
lmod, ftype, dis, names) ;
```

`elist` is a list expression, that must be of the type *list of modules*, for which the report is produced.

`rtype` is a single character string argument indicating the report type:

- ‘A’ indicates activity interface report.
- ‘M’ indicates module interface report.
- ‘I’ indicates information interface report.

`chart` is a single character string argument indicating which arrows are taken into account when the report is generated:

- ‘A’ indicates activity-chart arrows.
- ‘M’ indicates the module-chart arrows.

`lact` is an argument of type *list of activities* indicating which activities are taken into account when the report is generated. If `lact` is an empty list, the default is *all* activities implemented by the center module.

**Note:** If `chart` is ‘M’, then this parameter has no function. `lact` must still be supplied here. For simplicity, the null list (`null`) may be used.

`lmod` is an argument of type *list of modules* indicating the *side modules* that interface with the central module for which the report is to be generated. If `lmod` is empty, the default is *all* modules except the center module’s own ancestors and descendants.

`ftype` is a single character string argument indicating the kind of information flow to appear in the report:

- ‘*D*’ indicates data-flows.
- ‘*C*’ indicates control-flows.
- ‘*B*’ indicates both.

`dis` is a single character string argument indicating the kind of information to appear in the report:

- ‘*T*’ indicates flow labels.
- ‘*P*’ indicates parent information items.
- ‘*B*’ indicates basic information items.

`names` is a single character string argument:

- ‘*N*’ indicates that the *name* appears for elements that flow between the boxes.
- ‘*S*’ indicates that the *synonym* appears for elements that flow between the boxes.

◆ **N2 Chart Report**

```
stm_rpt_n2chart(elist, names, level, env, chart,  
dis, ftype)
```

`elist` is a list expression, which must be of the *type list of modules* or *list of activities*, specifying the elements in the diagonal.

`names` is a single character string argument:

- ‘*N*’ indicates that *names* of the elements appear on the diagonal of the matrix.
- ‘*S*’ indicates that *synonyms* of the elements appear on the diagonal of the matrix.

`level` is a single character string argument indicating what appears on the diagonal when both parent box and sub-box are in the list.

- ‘*B*’ indicates that *sub-box* is placed on the diagonal of the matrix.
- ‘*P*’ indicates that the *parent box* is placed on the diagonal of the matrix.

`env` is a Boolean expression; if true then the *environment* is added to the matrix.

`chart` is a single character string argument indicating which arrows are taken into account when the report is generated:

- ‘*A*’ indicates activity-chart arrows.
- ‘*M*’ indicates the module-chart arrows.

`dis` is a single character string argument indicating the kind of information to appear in the report:

- ‘*T*’ indicates flow labels.
- ‘*P*’ indicates information items.
- ‘*B*’ indicates basic information items.

`ftype` is a single character string argument indicating the kind of information flow to appear in the report:

- ‘*D*’ indicates data-flows.
- ‘*C*’ indicates control-flows.
- ‘*B*’ indicates both.

### ◆ Resolution Report

```
stm_rpt_resolution(clist, ttype)
```

`clist` is a list of charts. It determines the scope of the report.

`ttype` is `stmtype` of elements to include in the report. The `ttype` may be one of the following:

`stmtextual`, `stmgraphical`, `stmmixed`, `stmstate`, `stmmodule`, `stmactivity`, or `stmdata_store`.

## Include Plots Statement

### Statement Syntax:

```
stm_plot (plot_parameters) ;
```

This statement is used to include any chart in the document. It generates the specified plot with the indicated parameters (e.g., plot size, output device, etc.). The `plot_parameters` are specified in the order given below.

The output is designated for a particular device (one of the output devices defined in Statemate). The destination of the plot output is specified by one of the parameters. If its destination is left unspecified, it is included as part of the output segment file. This is done by using an empty string for the output file parameter.

The plot function returns a status which can be one of: `stmsuccess`, `stmunknown_plotter`, `stmcan_not_open_file`, `stmid_not_found`, `stmid_out_of_range`, `stmplot_failure`, `stmillegal_parameter`, `stmnot_enough_memory`, `stmempty_chart`, `stmunresolved`.

Whether you are plotting Statecharts, Activity-charts, Module-charts, or Block Diagrams, the parameters are the same:

```
stm_plot (id, file, width, height, with_label, with_name, with_note, device,  
title_position, title, do_rotate, with_file_header, actual_height)
```

- ◆ **id** - is the ID number of a Statemate chart to be plotted.
- ◆ **file** - is a STRING with the name of the file destination to which the plot is written. The operating system pathname conventions are followed. You may specify a full pathname to any directory for which you have write access. If a simple filename is specified, the plot is written to your Workarea. If the parameter is left empty ( ' '), the plot is included as part of the output file.
- ◆ **width** - is a numeric argument of type FLOAT that indicates the maximum possible width of the plot (in inches).
- ◆ **height** - is a numeric argument of type FLOAT that indicates the maximum possible height of the plot (in inches).
- ◆ **with\_label** - is a BOOLEAN parameter which indicates whether arrow labels are (TRUE) or are not (FALSE) printed in the plot.
- ◆ **with\_name** - is a BOOLEAN parameter indicating whether box names are (TRUE) or are not (FALSE) printed in the plot.
- ◆ **with\_note** - is a BOOLEAN parameter indicating whether notes are (TRUE) or are not (FALSE) printed in the plot.

**Note:**

- **device** is a STRING argument that indicates the plotting device. This may indicate a supported formatting language if the plot is to be handled by a formatting processing system that has its own graphics language. To configure a new plotter or printer (for example, a paper type), select **Utilities > Output Devices** from the main Statemate menu.
- Plots created using the Word format in the Output Device dialog are **RTF** files.
- ◆ **title\_position** is a STRING parameter indicating where to place the plot title. This parameter accepts one of the following values:
  - stm\_plt\_none - the title is not included.
  - stm\_plt\_top - the title is placed at the top of the plot.
  - stm\_plt\_bottom - the title is placed at the bottom of the plot.

- ◆ **title** is a STRING argument that specifies what title will be printed with the plot.
- ◆ **do\_rotate** is a BOOLEAN parameter where TRUE indicates landscape and FALSE indicates portrait.
- ◆ **with\_file\_header** is a BOOLEAN parameter where TRUE indicates that a header is to be added at the beginning of the file. (Use this if you do not want the plot as part of the document.)
- ◆ **actual\_height** is a numeric argument of type FLOAT that indicates the actual height (in inches) of the plotted output.

The following is an example of how a plot is generated in DGL. A template contains the following statements:

```
VARIABLE
CHART          ch_id;
INTEGER        status;
FLOAT          real_ht;

ch_id:= stm_r_ch('XL25',status);
stm_plot(ch_id,'/sam/p_xl25',5.0,7.0,true,true,false,
         'POSTSCRIPT',stm_plt_top,'SystemXL25',true,true,real_ht);
```

This produces a plot for the chart XL25 that is limited to a maximum size of 5 inches by 7 inches, prints labels and box names and does not print notes.

Output to the file specified by the path `/sam/p_xl25`.

This file is in Postscript format, as defined for device called 'POSTSCRIPT' in **Utilities > Output Devices** from the main Statemate menu.

The file will have the appropriate header for the output device.

The plot will be printed in landscape orientation.

```
A title "system XL25 is printed at the top of the plot; the actual
height is returned by the variable real_ht.
```

## Include Table Statement

### Statement Syntax:

```
stm_table_simple (title, columns, contents, page_width, page_height, anchor);
```

This statement generates a simple table. You specify the number of columns and their width, and the information to be included in the table. The parameters are as follows:

- ◆ **title** is the title of the table. The title appears centered over the table.
- ◆ **columns** is a list of integers that specify the width of each column in number of characters. For instance: {16} & {16} & {20} specifies a table having three columns, the first two columns being 16 characters wide and the last column being 20 characters wide.
- ◆ **contents** is a list of strings containing the information to be entered into each cell of the table. The information fills the table horizontally, row by row, depending on how many columns were specified. For instance, if you specified three columns in the **columns** parameter, the following contents:

```
{'Project Name'} & {'Date'} & {'Location'} &
{'Alpha'} & {'April 1987'} & {'Boston'}
```

would produce this table:

Project Name	Date	Location
Alpha	April 1987	Boston

- ◆ **page\_width** determines the width of the page in inches. It is only relevant for Interleaf - for other systems, you may specify 0.0 for this parameter.
- ◆ **page\_height** determines the height of the page in inches. It is only relevant for Interleaf - for other systems, you may specify 0.0 for this parameter.
- ◆ **anchor**, relevant only for Interleaf. A character string indicates where to place the table. The options are 'A' - at anchor, 'F' - following anchor. The default is 'F'.

For formatters other than Interleaf, precede the table with your system's *no fill* and *no adjust* formatting commands.

The following is an example of how a table can be generated using function calls. Notice how we repeatedly assign new values to the `List_str` variable to build the table.

Also note that the first statement uses the NROFF commands for *no fill* (`.nf`) and *no adjust* (`.na`). These commands cause the word processor to take the text “as is.” Some word processors refer to this mode as “verbatim” or “literal.” The last statement uses the NROFF commands (`.fi`) and (`.ad`) to return to fill and adjust modes.

```
WRITE ('\n.nf \n.na \n');
BEGIN
List_str:={'ACTIVITY NAME'} & {'ID'} & {'LANGUAGE'};
act_list:=stm_r_ac_logical_desc_of_ac({act_chart},st);
FOR act IN act_list LOOP
  List_str:=List_str & {stm_r_ac_name(act,st)};

  attr_list:=stm_r_ac_attr_val(act,'ID_NUMBER',st);

  IF (st = stm_success) THEN
    attr_val:=stm_list_first_element(attr_list,st);
    List_str:=List_str & {attr_val};
  ELSE
    List_str:=List_str & { 'N/A' };
  END IF;

attr_list:=stm_r_ac_attr_val(act,'LANGUAGE',st);
  IF (st = stm_success) THEN
    attr_val:=stm_list_first_element(attr_list,st);
    List_str:=List_str & {attr_val};
  ELSE
    List_str:=List_str & { 'N/A' };
  END IF;
END LOOP;

WRITE ('\n.nf\n.na\n');
title := 'Table CC1. Simple Table Example';
Col_list := {16} & {10} & {40};
stm_table_simple(title,Col_list,List_str,pg_w,pg_h, 'A');
WRITE ('\n.fi\n.ad\n');
END;
```

The formatted output:

ACTIVITY NAME	ID	LANGUAGE
FUNCTION1	AC1-A1.1	FORTRAN
FUNCTION2	AC1-A1.2	N/A
FUNCTION3	AC1-A1.5	ADA
FUNCTION4	AC1-A1.4	N/A



## Control Flow Statements

Several control flow constructs provide you with options for conditional and iterating statement execution. These resemble constructs in other conventional programming languages.

### IF/THEN/ELSE Statement

#### Statement Syntax:

```

    IF boolean_expression THEN
      statements
    [ ELSE statements ]
    END IF ;

```

The IF/THEN/ELSE construct is used for conditional execution of DGL statements.

In this statement, the statements following the THEN (and before any ELSE) are executed if the `boolean_expression` evaluates to true. If it is evaluated to false, the statements following the ELSE are executed, when present.

Here is an example:

```

IF a >= b THEN
  EXECUTE ('DATE') ;
  INCLUDE ('sample.txt') ;
ELSE
  WRITE ('a is less than b') ;
END IF ;

```

### SELECT/WHEN Statement

The SELECT/WHEN construct is used for conditional execution of DGL statements. This statement is more powerful than the previous IF/THEN/ELSE statement, in that it allows you to systematically list multiple conditions for statement execution.

#### Statement Syntax:

```

SELECT [selection_mode]
  WHEN trigger => statements
[ WHEN trigger => statements ]
  :
[ WHEN ANY => statements ]
  :
[ WHEN trigger => statements ]
  :
[ WHEN ANY => statements ]
  :
[ OTHERWISE => statements ]
END SELECT ;

```

The optional `selection_mode` can be either the keyword `FIRST` or `ANY`. The selection-mode determines the way the statements are checked for possible execution - this will be explained shortly. The default selection-mode is `FIRST`.

Note that `WHEN` statements are composed of two parts: the `trigger` to the left of the arrow, and `statements` on the right side of the arrow. The trigger is any valid Boolean expression. The statements following a trigger are performed only when the trigger is true. Whether or not these statements are actually executed also depends upon the selection-mode, as follows:

- ◆ If the selection-mode is `ANY`, then the statements are executed whenever their corresponding trigger is true.
- ◆ If the mode is `FIRST` (or not given), then only the first true trigger in the entire `SELECT` construct is executed; the rest are ignored, regardless of whether their triggers are true or not.
- ◆ The `WHEN ANY` statements are executed when one or more of the preceding `WHEN` statements have been executed.
- ◆ The `OTHERWISE` statements are executed only if no `WHEN` statement within the `SELECT` construct is triggered.

To demonstrate the execution of the `SELECT/WHEN` construct, consider the following example. `a`, `b`, and `c` are numeric variables.

```
SELECT ANY
  WHEN a = 5 => b := 10 ;
  WHEN a > b => b := 10 ;
  WHEN a = 0 => b := 0 ;
  WHEN ANY => write ('a may influence b') ;
  WHEN c = 5 => b := 5 ;
  WHEN c > b => b := 5 ;
  WHEN c = 0 => b := 0 ;
  WHEN ANY => write ('c may influence b') ;
  OTHERWISE => write ('b has not been changed') ;
END SELECT;
```

The execution is determined by:

1. Each `WHEN` statement is triggered if its corresponding expression is evaluated to true.
2. The first `WHEN ANY` statement is triggered if `a` is equal to 5, greater than `b`, or equal to zero.
3. The second `WHEN ANY` statement is triggered if at least one of these same conditions is true with respect to the variable `c` instead of `a`.
4. The `OTHERWISE` statement is triggered only if the value of `b` has not been changed within the `SELECT` statement's evaluation.

When processing a `WHEN ANY` statement, the Documentor Tool only “looks back” to the previous `WHEN ANY` construct (if one exists). Therefore, in the above example, if `a = 0` and none of the tests of `c` were true, the `WRITE` statement's message `c may influence b` is not issued.

In this example, what would happen if the selection mode was `FIRST` instead of `ANY`, and the conditions `a > b` and `c = 5` were both true ?

In this case the assignment `b: = 10` and the first write message (the corresponding `WHEN ANY` statement) are executed. The assignment of `b` to 5 along with its corresponding `WHEN ANY` statement are not done because `c = 5` is not the *first* true trigger.

The statements following the `=>` symbol in the `WHEN` constructs may be any valid DGL statements. You may even enter a `SELECT` construct at this point. This allows you to nest `SELECT` constructs. There is no limit to the depth of nested `SELECT` blocks.

## FOR/LOOP Statement

### Statement Syntax:

```
FOR identifier IN list LOOP
  statements
END LOOP ;
```

The `FOR/LOOP` construct is used for iterative execution of DGL statements. The statements after the keyword `LOOP` are executed for each element in the specified `list`.

Alternatively, a range of integers can be specified in place of the `list`, as in the following example:

```
FOR i IN {1..100} LOOP
  statement ;
  :
  :
  :
END LOOP ;
```

The `identifier` is a variable whose value is set sequentially to the items in the `list`. The type of the `identifier` must match the type of the `list`. This variable may be used within the body of the loop.

The following example writes the name and synonym for each state found in the list defined by the variable `sub_states`:

```
VARIABLE
  STATE          st_id ;
  LIST OF STATE  sub_states;
  :
  :
  :
  FOR st_id IN sub_states LOOP
    WRITE(stm_r_st_name(st_id,status), ' ',
          stm_r_st_synonym(st_id,status), '\n') ;
  END LOOP ;
```

## WHILE/LOOP Statement

### Statement Syntax:

```
WHILE boolean_expression LOOP
    statements
END LOOP ;
```

The WHILE/LOOP construct is used to iteratively execute DGL statements. The execution of statements is determined by evaluation of the `boolean_expression`.

The statements are executed until the expression evaluates to false. For example,

```
WHILE a > b LOOP
    .
    b := b + k ;
    .
END LOOP ;
```

The statements between the keywords `LOOP` and `END LOOP` are executed as long as `a` is greater than `b`.

Assume that `b` changes its value inside the `loop` and in one of the iterations the expression `a > b` becomes false. In the next iteration, the expression is examined and, since `a > b` is now false, the execution of template statements continues with the first statement after the `END LOOP`.

## EXIT Statement

### Statement Syntax:

```
EXIT ;
```

The `EXIT` statement is relevant only inside `FOR` or `WHILE` loops. It results in an exit from the current loop to the statement after the loop construct or to the construct that contains the current loop.

Typically, a condition would be tested in a loop, and the exit would be based upon the evaluation of that condition.

For example, the following structure would continue the execution of the statements between the `LOOP` and `END LOOP`, depending on the value of the condition `a > b`.

The iterations go on as long as the status `st` is equal to 0. When `st` is not equal to `stm_success`, it causes the iterations to stop. The `IF` statement here is used to force an abnormal `EXIT` from within the `WHILE` loop.

The execution resumes at the next statement after the `END LOOP`.

```
WHILE a > b LOOP
  md_id := stm_r_md (name , st) ;
  IF st <> stm_success THEN
    WRITE ('Illegal Status') ;
    EXIT ;
  END IF ;
  .
  .
END LOOP ;
```

## STOP Statement

### Statement Syntax:

```
STOP ;
```

The `STOP` statement stops execution of the template.

Typically, a specific condition is tested and the template is stopped if this condition has a value for which further processing is meaningless.

For example, in the following statements we check whether the specified `system_name` is proper. If not, i.e., if an error has been detected, a message is issued to the dialog area and the template is stopped.

```
md := stm_r_md (system_name, status) ;
IF (status <> stm_success) THEN
  WRITE (dialog_area, 'Execution Stopped due          to
error') ;
  STOP ;
END IF ;
```



# Documentor Functions

---

When you write reports on system design specifications, it is often necessary to include information from the Statemate database that contains the specifications. The Documentor provides you with this capability through the use of database extraction functions.

## Overview of the Extraction Functions

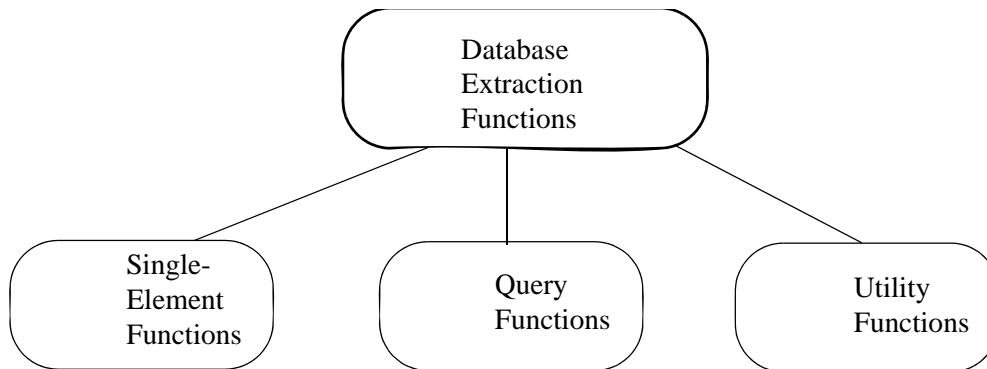
The database extraction functions are a collection of routines that enable you to extract information from the specification database.

There are four types of database extraction functions:

- ◆ **Single-element** - Provide information about a discrete Statemate element in the specification database. For example, you can retrieve the contents of the description field in the form for a particular state. Refer to [Single-Element Functions](#) for detailed information.
- ◆ **Query** - Extract lists of elements from the database that conform to a specific criterion. For example, you can extract a list of activities from the database that are control activities. Each function corresponds directly to a query of the property sheet. Function output consists of a list of Statemate elements. Refer to [Query Functions](#) for detailed information.
- ◆ **Utility** - Perform operations on lists and strings. Most of these functions do not extract information from the database, but enable you to manipulate the information you have already retrieved. Refer to [Utility Functions](#) for detailed information.
- ◆ **Project management** - Extract information about the Rhapsody project, manager, and members.

## Function Structure

The following figure shows the structure of the database extraction functions.



There are scores of database extraction functions. However, it is easy to become proficient in their use because of the systematic structure of the package. The functions are documented in [Single-Element Functions](#).

## Using Database Extraction Functions

Database extraction function calls can appear anywhere in your template where expressions of the same type are valid.

Consider the following call:

```
state_id := stm_r_st ('S1', status);
```

This call retrieves the state whose name is `S1` from the database and assigns it to the variable `state_id`. (In actuality, the call retrieves the state's ID. This ID is a value that Statemate uses to identify each element in the database.)

Function calls are frequently used in sequence. For example, because you have already retrieved the ID for state `S1`, you can now call the following function:

```
sub_st := stm_r_st_physical_sub_of_st  
({state_id}, status);
```

This function call builds a list of substates contained in state `S1` and assigns the list to the variable `sub_st`.



At this point, you can print out a list of all substates of state `s1`. The list is to include the name of the individual state as well as the description appearing in the state's form. To do this, include the following lines in your template:

```
FOR s in sub_st LOOP
WRITE ('\n Name:', stm_r_st_name (s, status));
WRITE ('\n Desc:', stm_r_st_description (s, status));
END LOOP;
```

## Calling Conventions

Database extraction functions provide you with information from the Statemate database. To extract this information, you call the specific function that retrieves the information you want. You specify the particular Statemate elements that interest you as input arguments to the function. The function returns the information and a status code (as an output argument). This status code informs you whether your function call was successful.

### Function Names

Single-element and query functions use the following prefix:

```
stm_r_
```

This prefix designates the function as a Statemate database retrieval function.

Utility functions use the following prefix:

```
stm_
```

### Element Type Abbreviations

Database extraction functions use two-character abbreviations to identify the type of Statemate elements referenced in function calls. The following table lists the element types and their abbreviations.

Element	Abbreviation
A-flow-lines (basic)	ba
A-flow-lines (compound)	af
A-flow-lines (local)	laf
Actions	an
Actors	actor
Activities	ac

<b>Element</b>	<b>Abbreviation</b>
Boundary boxes	bb
Charts	ch
Combinational assignments	ca
Conditions	co
Connectors	cn
Data-items	di
Data-stores	ds
Enumerated value	en
Events	ev
Fields	fd
Information-flows	if
Local data	ld
Messages	msg
Mixed (multiple types)	mx
M-flow-lines (basic)	bm
M-flow-lines (compound)	mf
M-flow-lines (local)	lmf
Modules	md
Module-occurrences	om
Off-page activities	oa
Routers	router
Separators	sep
States	st
Subroutines	sb
Subroutine parameters	sp
Timing constraints	tc
Transitions (basic)	bt
Transitions (compound)	tr
Use cases	uc
User-defined types	dt

For example, `stm_r_ac_name` retrieves the name of an activity, whereas `stm_r_st_name` retrieves the name of a state.

The naming structure for each type of database extraction function is explained in the section that describes each specific type. Note that element type and the information to be extracted are contained in the function name and are *not* passed as arguments.

## Arrow Elements

Arrow elements (transitions, a-flow-lines, and m-flow-lines) can be either basic or compound:

- ◆ A basic arrow connects a box or connector to another box or connector.
- ◆ Compound arrows are the logical connection between boxes and can possibly be composed of several basic arrows, passing through their connectors.

## Function Input Arguments

Database extraction functions require input arguments in order to locate Statemate elements in the database. Input arguments consist of elements or lists of elements for which information is sought.

Some functions require additional input arguments. Each argument must be declared to be of a data type recognized by DGL. This guide includes a complete list of input arguments for each type of database extraction function in the sections that describe the specific function type.

See the function reference appendixes for the lists of arguments relevant for each function.

## Status Codes

Database extraction functions return only one argument—the function status code. This code reports whether the function call was successfully completed. When the function call fails, the status code indicates the problem. You can use the status code to pinpoint run-time errors in your template. For example, assume the following call appears in your template:

```
state_id := stm_r_st ('%', status);
```

The function requires a state name for the first (input) argument. In this case, the function returns a status code of 3, `stm_illegal_name`, because % is not a valid element name.

The status code is an integer value. Therefore, the `status` argument must be a variable declared as `INTEGER`. The Documentor provides predefined constants for the function status codes. This enables you to use the status name attached to each status code in your template.

For example, assume that you want to print out the synonym of the state `s1`. If there is no synonym defined in the state's form, print "missing synonym". Your template should contain the following code:

```
VARIABLE
    INTEGER          status;
                    .
                    .
                    .
state_id := stm_r_st ('S1', status);
synonym  := stm_r_st_synonym (state_id, status);
IF status = stm_missing_synonym THEN
    WRITE ( '\n synonym:      *missing synonym* ');
ELSE
    WRITE ( '\n synonym:      ', synonym);
END IF;
```

Status codes have severity levels that you can check to ensure that your function call was successful. These severity levels, and a complete list of status codes are documented in [Function Status Codes](#).

## Function Return Values

Database extraction functions return values that are of DGL data types (refer to [Data-types](#) for more information). Different functions return different types of values. For example, a function that retrieves the name of a Statemate element returns a value of type `STRING`, whereas a function that retrieves a state's ID returns a value of type `STATE` (or `ELEMENT`).

The return value data type must be valid when the function is used in statements and expressions. For example, a return value that is a `STRING` can appear in places where string expressions are allowed, as shown in the following `WRITE` statement:

```
WRITE (stm_r_st_name (st_id, status));
```

This statement prints out the name of the state whose ID is `st_id`. The return values of each function are listed in the sections that describe the specific functions.

### Return Values of Type `ELEMENT`

There are a number of database extraction functions that retrieve elements or list of elements from the specification database. In the case of elements, the functions return values that belong to DGL data types `STATE`, `EVENT`, `ELEMENT`, and so on. For example, consider the following function call:

```
state_id := stm_r_st ('S1', status);
```

This call extracts the state `S1` from the database. Because the function returns a `STATE`, `state_id` must be declared to be of type `STATE` or `ELEMENT`.

In the case of list of elements, the functions return values that belong to the DGL data types `LIST OF STATE`, `LIST OF ACTION`, `LIST OF ELEMENT`, and so on. All query functions return a list of Statemate elements.

## Return Values of Filename

A number of database extraction functions store extracted information in files, such as a function that retrieves an element's long description. In this case, the function returns the name of the file that contains the requested information. The filename returned is of DGL type `STRING`.

## Return Values of Enumerated Types

DGL does not directly support enumerated data types. Functions that return discrete numerical values are considered to be of type `INTEGER`. The Documentor enables you to reference these numerical values by name. In reality, these names are internally defined as predefined constants in DGL. These names contain the prefix `stm_`.

For example, the function `stm_r_st_type` extracts the type of state specified in the function call. The possible state types are `stm_st_and`, `stm_st_or`, `stm_st_diagram`, and `stm_st_component`. These correspond to the values 0, 1, 2, 3 (respectively).

You can use the value names in your template. For example:

```
IF stm_r_st_type (st_id, status) = stm_st_component
THEN
    ...
```

The possible values that functions return, and their names, are documented in the function reference sections.



# Model Templates

---

This section provides complete examples of templates—two that do not contain commands for a formatter (formatting is accomplished through DGL `WRITE` statements) and two that contain formatting commands for specific formatters.

The templates are as follows:

- ◆ [Properties](#)
- ◆ [Activity Interface Report](#)
- ◆ [Template for nroff](#)
- ◆ [Template for Interleaf](#)

## Properties

This template uses database extraction functions to generate properties. It produces the property report for a list of data-items whose names match the specified pattern.

Each entry includes the name of the data-item, a short description, synonym, type (real, integer, and so on), and attributes and their values. In addition, it includes specific information extracted from each data-item's long description.

The template is as follows:

```
TEMPLATE di_dict;
-- this template produces information for each
-- data-item in a list no text formatter is used.
-- Formatting is done in WRITE statements.

PARAMETER
  STRING di_pattern; -- the pattern that
                    -- determines the data-item list

VARIABLE
  INTEGER st;          -- return status code
  LIST OF DATA_ITEM di_ids; -- list of
                          -- data-items for which the report
                          -- is produced

BEGIN
  di_ids:= stm_r_di_name_of_di (di_pattern, st);
END;
```

```

SEGMENT report;

VARIABLE
  DATA_ITEM      di;
  STRING          di_name, di_synonym;
  STRING          di_sdesc;
  INTEGER         di_type;
  STRING          di_desc_file;
  LIST OF STRING  attr_list, attr_val_list;
  STRING          attr, attr_val;

BEGIN

-- write report heading
WRITE ('\n\n', ' ':20, 'DATA-ITEM PROPERTY');

FOR di IN di_ids LOOP

  di_name := stm_r_di_name (di, st);

  -- write entry heading
  WRITE ('\n\n\n  DATA-ITEM: ', di_name);
  -- get short description and print it out
  di_sdesc:= stm_r_di_description (di, st);
  WRITE ('\n\n', di_sdesc);
  -- get synonym and print it out
  di_synonym := stm_r_di_synonym (di, st);
  WRITE ('\n\n SYNONYM: ', di_synonym, '\n' );
  -- get structure\type and print it out
  di_type    := stm_r_di_data_type (di, st);

  SELECT
    WHEN di_type = stm_di_alias =>
      WRITE ('\n TYPE: ALIAS \n' );
    WHEN di_type = stm_di_constant =>
      WRITE ('\n TYPE: CONSTANT \n' );
    WHEN di_type = stm_di_primitive =>
      WRITE ('\n TYPE: PRIMITIVE \n' );
    WHEN di_type = stm_di_reference =>
      WRITE ('\n TYPE: REFERENCE \n' );
    WHEN di_type = stm_di_compound =>
      WRITE ('\n TYPE: COMPOUND \n' );

  END SELECT;

  --
  -- retrieve the attributes for the
  -- data-item of interest
  --
  attr_list := stm_r_di_attr_name (di, st);
  --
  -- write out the attribute names and values

  FOR attr IN attr_list LOOP
    attr_val_list:= stm_r_di_attr_val (di, attr,st);
    FOR attr_val IN attr_val_list LOOP
      WRITE ('\n', attr + ' ':20, attr_val);
    END LOOP;
  END LOOP;
  --
  --
  -- write out various parts of the data-item's
  -- long description

```



```
--
di_desc_file := stm_r_di_keyword (di, '!PURPOSE',
                                '!END PURPOSE', '', st);
WRITE ('\n\nPURPOSE: \n');

IF st = stm_success
THEN
  INCLUDE (di_desc_file);
ELSE
  IF st = stm_starting_keyword_not_found
  THEN
    WRITE(' not available \n');
  END IF;

END IF;

di_desc_file := stm_r_di_keyword (di, '!TIMING',
                                '!END TIMING', '', st);
WRITE ('\n\nTIMING: \n');

IF st = stm_success
THEN
  INCLUDE (di_desc_file);
ELSE
  IF st = stm_starting_keyword_not_found
  THEN
    WRITE(' not available \n');
  END IF;
END IF;

di_desc_file:=
  stm_r_di_keyword (di, '!REPRESENTATION',
                  '!END REPRESENTATION', '', st);
WRITE ('\n\nREPRESENTATION: \n');

IF st = stm_success
THEN
  INCLUDE (di_desc_file);
ELSE
  IF st = stm_starting_keyword_not_found
  THEN
    WRITE (' not available \n');
  END IF;
END IF;
END LOOP;
END;
```

## Properties Template Structure

As for all templates, this template is divided into two parts:

- ◆ An initiation section that contains global declarations and statements.
- ◆ Template segments, each of which contains local declarations and statements. Recall that each template segment, when executed, generates a document segment.

In this template, there is an initiation section and one segment. The initiation section follows the template identifier line:

```
TEMPLATE di_dict;.
```

The segment section follows the segment identifier line:

```
SEGMENT report;.
```

Both the initiation section and the template segments consist of two parts:

- ◆ **Declaration section** - Where the identifiers' types are declared.
- ◆ **Body** - Contains execution statements. These statements are contained within the keywords `BEGIN` and `END`.

## Properties Initiation Section

The initiation section begins with the mandatory `TEMPLATE` identifier line that specifies the name of the template (in this case, `di_dict`). Following this are several optional comment lines describing the purpose and features of the template. As the comment lines state, this template does not contain formatting commands for any specific formatter.

### Declaration Part

Identifier declarations follow the comment lines. There is one parameter, `di_pattern`, which is declared as a string and represents a string pattern. It will be used in a database extraction function to retrieve data-items whose names match this pattern. Because `di_pattern` is a parameter, you specify its value in a form before executing the template. Changing the string pattern enables you to generate the properties for different lists of data-items.

Next, the template declares two variables:

- ◆ `st` - Returns a status code used with all Statemate database extraction functions. It is declared as type `INTEGER`. Refer to [Documentor Functions](#) for more information on using return status codes
- ◆ `di_ids` - Represents the list of IDs of data-items for which the report is produced. It is declared as a `LIST OF DATA_ITEM`.

The identifiers declared in the template so far belong to the initiation section and thus are global in scope (they can be used throughout the entire template).

## Body

The body of the initiation section is between the `BEGIN` and `END` keywords. In this template, the body consists of only one statement, an assignment statement that assigns a value to the `LIST OF DATA_ITEM` variable, `di_ids`. A database extraction function assigns to `di_ids` a list of elements whose names match the string pattern `di_pattern`. These are the data-items for which the report will be generated. In this example, assume that you specified the string pattern `SIGNAL*`. The data-items for which the properties are generated will all have names that begin with `SIGNAL` (refer to the generated output in [Final Output for Data-item Properties](#)).

Note that the template uses a status code variable, `st`, as an argument in the function. However, the template does not include a statement to check `st`. If `st` does not indicate success, the returned list of data-items assigned to `di_ids` is empty.

## Properties Segment Section

The long segment of the template begins with its mandatory identifier line, `SEGMENT report ;`. This name is used to attach a name to the segment that follows. In templates possessing more than one segment, you can use this name to identify the specific segment. Refer to [Using Documentor](#).

The declaration part follows. First, it declares the variable `di` to represent a data-item. This variable will be used to iterate on the list of data-items.

Next, the template declares variables used in database functions to extract information about each data-item in the properties. The variables—`di_name`, `di_synonym`, and `di_desc`—correspond to the information contained in the name, synonym, and description fields of the data-item form shown previously. All three of these variables are declared to be of type `STRING`

The next variable, `di_type`, represents the information entered in the type field of the data-item form. Because each of the options in this field corresponds to an integer value, this variable is declared to be of type `INTEGER`. Each integer value corresponds to a predefined constant; these constants, instead of the integer values, are used later in the template.

The string variable `di_desc_file` represents the name of a file. This variable will be used for the names of the files that include portions of the data-item's long description.

Next, the template includes variables having to do with information pertaining to the attribute fields. The variable `attr_list` represents the list of attribute names entered in the attribute name field of the data-item form. You will build this list later in the template through the use of a Statemate database extraction function. Similarly, the variable `attr_val_list` represents the list of attribute values. Lastly, the variables `attr` and `attr_val` are used to represent individual attribute names and values, respectively; these variables are used later to iterate on the list of attribute names and values.

### Generating the Report Heading

The first line of the template body produces a report heading for the properties. To produce this heading, use a `WRITE` statement (preceded by a comment):

```
-- write report heading
  WRITE ('\n\n', ' ':20, 'DATA-ITEM PROPERTY');
```

The character `\n` causes a blank line to be inserted in the text. In this case, two blank lines are inserted and the string “DATA-ITEM PROPERTY” is written, indented by blanks. The blank, `' '`, followed by the number 20 that appears in the `WRITE` statement acts as a tab, indenting the line 20 spaces. In actuality, this number determines the minimum length that the printed expression is to reach. If the output falls short of this, blank spaces are added after the last character to complete the number. In this case, a single blank is used as the expression to be printed, causing 19 spaces to be added to the blank before the start of the next word.

### Iteration: Using the FOR/LOOP Statement

The data-item properties will consist of information for more than one data-item—you want information for a *list* of data-items. Each data-item in the list should appear as a separate entry with the corresponding information (synonym, description, attributes, and so on). To generate the same type of information for all the data-items in the list, use a `FOR/LOOP` statement. This structure iteratively performs statements for a given identifier whose value is set sequentially to the items in a list. The program executes all statements following the `LOOP` keyword for the identifier’s current value until it reaches the `END LOOP`; the program then loops back to the first statement in the loop and performs the same sequence of statements for the next item in the list.

In this example, the list is composed of data-items and is represented by the variable `di_ids`. The sequence of statements in the loop is generated for each item of the list, represented by the variable `di`. The loop terminates at the `END LOOP`, which is the second to the last statement in the template.

## Generating the Entry Heading

Each data-item entry in the properties is marked by a heading that writes the words “DATA-ITEM:” followed by the name of the data-item for which the entry is made. The function `stm_r_di_name` extracts the name for the data-item represented by `di`. The name of the data-item entry is represented by the variable `di_name`.

To produce the entry heading, use a `WRITE` statement (preceded by a comment):

```
-- write entry heading
WRITE ('\n\n\n DATA-ITEM: ', di_name);
```

## Extracting and Printing Information from the Data-Item Form

The next part of the template includes statements that extract and write information contained in the data-item form. This includes the data-item short description from the description field, synonym, data-item type, and attribute names and their values.

### Description and Synonym

The first two statements in this part extract the short description of the data-item (the description field of the data-item form) and prints it out. The template uses a database extraction function to assign a value to the variable `di_sdesc` for the data-item represented by `di`. The `WRITE` statement that follows inserts a blank line, then writes the description (`di_sdesc`) of the data-item.

Next, two statements extract the data-item’s synonym and prints it.

### Using the SELECT/WHEN Construct

The next part of the template uses a `SELECT/WHEN` construct to extract and write the data-item type. To see how this works, first consider the fact that there are a finite number of values that can be specified for `type`. Turning to the `type` field in the data-item form, you can see that a data-item type can be one of the following:

- ◆ Record
- ◆ Integer
- ◆ Float
- ◆ String

The data-item type is referred to as an *enumerated type*—each of the values for this field is represented by a unique number. These numbers correspond to predefined constants (`stm_di_record`, `stm_di_list`, and so on). You use a database extraction function to extract this type for each data-item `di` in your properties.

Next, you want the name of the data-item type to be written in the document. To do this, use a `SELECT/WHEN` construct as shown in the template. This construct selects for execution one of the statements contained between the keywords `SELECT` and `END SELECT`. Only one of these statements is executed for each data-item. Each statement contains a trigger, beginning with the keyword `WHEN` that, when satisfied, leads to the execution of the statement following the arrow. In this template, the triggering condition is the value of the data-item type represented by the variable `di_type`. When this value is equal to that of the particular value of the data-item type contained in the `WHEN` statement, a `WRITE` statement is executed that inserts a blank line and writes “TYPE:”, followed by the data-item type.

Note that you cannot write the data-item type directly from the variable `di_type` that represents it. This is because this variable actually represents an integer value; writing the variable’s value would result in an integer being written, instead of the name of the data-item type. You solve this problem by writing an appropriate string name that corresponds to the variable’s value.

### Using Nested FOR Loops to Extract Attribute Names and Values

The next part of the template involves the use of nested `FOR/LOOP` constructs for extracting and writing the data-item’s attribute name and value. A data-item’s attributes and their values are recorded in special fields in the data-item form. The information in these fields is textual and can involve any features of the data-item that you want. The field enables you to record a list of attributes and values.

Recall that `FOR` loops are used to iterate on items of a list. In this template, the list of data-item attributes is represented by a `LIST OF STRING` variable, `attr_list`. The template executes a sequence of statements for each attribute (represented by the string variable `attr`) contained in the list.

These statements involve extracting and writing the corresponding attribute values for each attribute listed in the attribute name field. First, the template uses a database extraction function to assign values to a `LIST OF STRING` variable, `attr_val_list`. This variable represents the list of attribute values recorded in the attribute value field for a particular attribute name, represented by the variable `attr`.

This template assumes that more than one value can be recorded for each attribute; therefore, another `FOR/LOOP` construct is used to write the information for each value (represented by the variable `attr_val`) in the list.

You want to write attribute name and value as a paired set, with the attribute value written next to the attribute name. The template uses the number 20 in the `WRITE` statement to indent the second output field 20 characters. Note that for “:” to appear after the attribute name, the template uses a concatenation of two strings (`attr + ':'`).

## Using Keywords to Write Portions of the Long Description

In the last part of each property entry, you want to include information from the data-item's long description. This information consists of free text that is attached to the element's form. The long description can be divided into portions through the use of keywords. For example, you can set off those portions of the text that deal with data-item purpose by enclosing the text within the keywords `!PURPOSE` and `!END PURPOSE`. You can extract this information using a database extraction function that searches for keywords in the text and returns (to a file) all the text contained between these words. You can then use an `INCLUDE` statement to cause the file containing the text to appear in the document.

For each section of text that you want to appear, you begin by assigning a value to the string variable `di_desc_file`, which represents the name of the file with the desired text. Note that the fourth argument of the database extraction function is an empty string. This argument is reserved for the name of the file in which the desired text is returned. If you want the information in a specific file, you can specify a name for this argument. If you specify an empty string, as shown in the template, the Documentor writes the text in a temporary file. In either case, the function returns the name of the file in which the text is written. For example, in the first section, `di_desc_file` represents the temporary file that contains all the text between the keywords `!PURPOSE` and `!ENDPURPOSE`.

The assignment for `di_desc_file` is followed by an `IF/THEN/ELSE` statement that either includes the text or prints a "not available" message if the starting keyword is not found. This is done by means of the status parameter `st`, which appears in the keyword's extraction function. If the value of `st` is equal to `stm_success`, the text is included in the document. If the value of `st` is equal to `stm_starting_keyword_not_found`, the message "not available" is written. Any other status values will leave the paragraph empty.

The rest of the template extracts long description information in the same manner. The template execution then loops back to the beginning, repeating the instructions for each data-item in the list.

The final output is shown in the following section.

## Final Output for Data-item Properties

### DATA-ITEM PROPERTY

DATA-ITEM: SIGNAL1

Signal specifying the current address

SYNONYM: SIG\_1

TYPE: INTEGER

RANGE: 0 - 2046

RESOLUTION: 2

#### PURPOSE:

This signal designates the destination of the data that is sent to the system. The system will transfer the signal to the component that is responsible to the routing of information.

#### TIMING:

The address signal must be valid when the write signal is sent.

#### REPRESENTATION:

The signal uses the binary representation.

DATA-ITEM: SIGNAL2

Signal specifying the data to write

SYNONYM: SIG\_2

TYPE: INTEGER

RANGE: 0 - 1000



PURPOSE:

This signal is the data that is sent to the system. The system will transfer the data to its destination according to the specified address.

TIMING:

The data must be valid when the write signal is sent and stays valid for at least two ms.

REPRESENTATION:

The data will be using an ASCII representation.

DATA-ITEM: SIGNALS

Signals specifying address and data

SYNONYM: SIGS

TYPE: RECORD

PURPOSE:

not available

TIMING:

not available

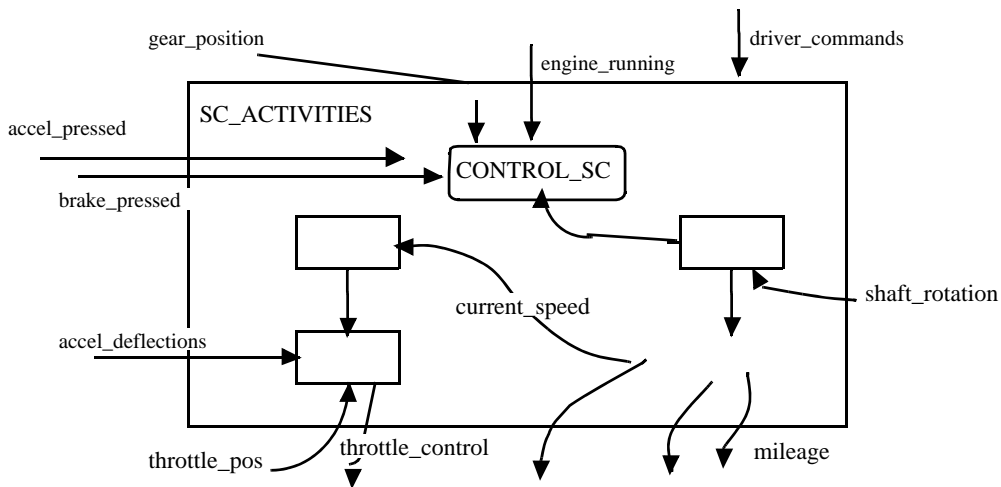
REPRESENTATION:

not available

## Activity Interface Report

An activity interface report presents the input and output for a given activity. This template produces an interface report for a given activity. No text formatter is used—formatting is done using `WRITE` statements.

To understand the report, consider the activity chart in the following figure.



The chart shows an activity named `SC_ACTIVITIES` with its subactivities. Flow-lines indicating the flow of events, conditions, and data-items enter and exit `SC_ACTIVITIES`. Some of these flow-lines lead directly to, or exit from, a particular subactivity, whereas others lead to, or exit from, the parent activity (`SC_ACTIVITIES` itself). The interface report lists the elements flowing through these flow-lines, indicating whether they are events, conditions, or data-items.

For example:

```

    DATA_ITEM  ACCEL_DEFLECTION
    EVENT       ACTIVATE_CRUISE
    EVENT       CLOCK
    DATA_ITEM  MILEAGE
    
```

Note that flow-lines also lead from one subactivity to another. These are not considered as inputs and outputs to the main activity of the chart, and therefore information concerning them does not appear in the interface report.

The template first searches for `a_flow_lines` using a query function. The template then extracts the conditions, events, and a data-items that flow along these lines.

## act\_interface Template

```

TEMPLATE act_interface;

PARAMETER
STRING activity_name;  -- the activity for which
                      -- the report is generated.

VARIABLE
  INTEGER  st;        -- return status code
  ACTIVITY act;      -- the id of subject activity

BEGIN
  act := stm_r_ac (activity_name, st);
END;

PROCEDURE write_elements;

PARAMETER
  LIST OF ELEMENT  element_list;

VARIABLE
  ELEMENT  elm;
  INTEGER  element_type, status;

BEGIN
  -- write elements
  FOR elm IN element_list LOOP
    element_type := stm_r_element_type (elm, status);
    SELECT
      WHEN element_type = stm_event =>
        WRITE('\n EVENT      ',stm_r_ev_name(elm, status));
      WHEN element_type = stm_condition =>
        WRITE('\n CONDITION ',stm_r_co_name(elm, status));
      WHEN element_type = stm_data_item =>
        WRITE('\n DATA_ITEM ',stm_r_di_name(elm, status));
    END SELECT;
  END LOOP;
END;

SEGMENT report;

VARIABLE
  LIST OF A_FLOW_LINE  af_list;
  LIST OF EVENT        ev_list;
  LIST OF CONDITION    co_list;
  LIST OF DATA_ITEM   di_list;
  LIST OF ELEMENT      in_list, out_list;
  ELEMENT              elm;
  INTEGER              elm_type;

BEGIN
  -- write title
  WRITE ('\n\n', ' ':20,'INTERFACE REPORT for ',
  activity_name);

```

```
-- activity's inputs
-- input title
WRITE ('\n\n Input elements:');
WRITE ('\n');-- get inputs

-- get input flow-lines
af_list := stm_r_af_ext_to_target_ac({act},st);

-- get input conditions
co_list := stm_r_co_flowng_through_af(af_list,st);

-- get input events
ev_list := stm_r_ev_flowng_through_af(af_list,st);

-- get input data items
di_list := stm_r_di_flowng_through_af(af_list,st);

in_list:=co_list + ev_list + di_list; -- all inputs

-- order alphabetically
in_list := stm_list_sort_by_name(in_list, st);

-- write elements
write_elements(in_list);

-- activity's outputs
-- output title
WRITE ('\n\n Output elements:');
WRITE ('\n');-- get outputs

-- get output flow-lines
af_list := stm_r_af_ext_from_source_ac ({act}, st);

-- get output conditions
co_list := stm_r_co_flowng_through_af (af_list, st);

-- get output events
ev_list := stm_r_ev_flowng_through_af (af_list, st);

-- get output data items
di_list := stm_r_di_flowng_through_af (af_list, st);

-- all outputs
out_list := co_list + ev_list + di_list ;

-- order alphabetically
out_list := stm_list_sort_by_name (out_list, st);

-- write elements
write_elements (out_list);

END;
```

## Activity Interface Report Initiation Section

Declarations and assignments in the initiation section apply throughout the template. In the initiation section, one parameter is declared, `activity_name`, which can be assigned via a special form before template execution. This is the name of the activity for which the interface report is to be generated. In addition, two variables are declared:

- ◆ `st` - The status code returned by the database extraction function
- ◆ `act` - The ID of the activity for which the report is produced

The body of the initiation section has only one statement between the `BEGIN` and `END` keywords. This is an assignment statement that assigns the activity's ID to the `act` variable. Because elements are identified in database extraction functions by their IDs and not their names, you must first find the ID for the activity before you can extract information about it. To do this, use the activity's name as an argument in a database extraction function that returns the corresponding activity's ID. By assigning this ID to the `act` variable, you can later use it throughout the template whenever you want to extract information about the activity.

## Activity Interface Report Segment Section

The long template segment begins with the segment identifier line, `SEGMENT report, SEGMENT report;`. This line is used to attach a name to the segment that follows. In templates possessing more than one segment, you can use the name to select a segment for exclusive execution. Next, there is a declaration section, followed by the segment body that consists of execution statements enclosed between the `BEGIN` and `END` keywords.

### Declarations

The declaration section consists of variables used to construct the list of element inputs and outputs. The first four variables—`af_list`, `ev_list`, `co_list`, and `di_list`—represent lists of a-flow-line elements and the events, conditions, and data-items that flow along them. The next two variables, `in_list` and `out_list`, represent the union of the list of elements represented by the variables `ev_list`, `co_list`, and `di_list`. The variable `in_list` represents a combined list of input elements. The variable `out_list` represents a combined list of output elements.

The next variable, `elm`, is used to extract elements from lists of elements using a `FOR/LOOP`. The last variable, `elm_type`, is an enumerated type that is declared as an integer. It represents the element type and is used in conjunction with a `SELECT/WHEN` construct to write out the type for each element in the list of inputs and outputs.

### Producing the Headings

The first `WRITE` statement in the body (statements starting after the keyword `BEGIN`) produces the title of the report. This consists of 20 blank spaces (indicated by `' ':20`), followed by the words `INTERFACE REPORT for`, and the name of the activity for which the report is generated (this name is a parameter supplied at the time of template execution).

For a heading introducing the input elements, the template uses a second `WRITE` statement to insert two new lines, then writes the heading `Input Elements:`. This is followed by another new line command.

### Building Element Lists

The next part of the template builds lists of input elements using database extraction functions. First, the template extracts from the database all the a-flow-line elements that have the subject activity (or its subactivities) as their target. This list is assigned to the variable `af_list`. Note that the database extraction function that extracts this list has its parameter, `act`, enclosed in braces. Braces indicate a list of items; because this particular extraction function is a query function, the parameter must consist of a list. In this case, however, the “list” is only one activity ID, represented by `act`.

The template builds three separate lists of elements:

- ◆ A list of input conditions, represented by the variable `co_list`
- ◆ A list of input events, represented by the variable `ev_list`
- ◆ A list of input data-items, represented by the variable `di_list`

Each of these variables is assigned values through a database extraction function that uses the variable `af_list` as a parameter. In other words, once you have extracted all the a-flow-line elements related to the specified activity, you extract the conditions, events, and data-items that flow through them.

### Alphabetizing and Sorting the List

To alphabetize all the elements in the lists, regardless of type, use the operation for union “+” to create a combined list of all the input elements; this list is designated by the variable `in_list`. The template uses a special utility function to sort by name the elements represented by `in_list` and reassign them to `in_list`.

## Writing the Input Elements

The next part of the template segment gives instructions for writing each element's type (event, condition, or data-item), followed by its name. To do this, use a FOR/LOOP construct to both extract an element `elm` out of the alphabetized `in_list` and perform several statements iteratively.

The first statement in the loop extracts the type of the element represented by the variable `elm`. The template then uses a SELECT/WHEN construct to write the element type and name. The triggers of the WHEN statements are constructed so when the variable `elm_type` is equal to a particular element type, the WRITE statement to the right of the arrow is executed.

“Element type” is referred to as an enumerated type; each of the possible values is represented by a unique number. These numbers correspond to predefined constants (`stm_event`, `stm_condition`, and `stm_data_item`). Note that the template does not use the value of the variable `elm_type` to write the element type; doing so would cause the *integer value* to be written. Instead, the template writes the string that corresponds to the integer value.

You must use a database extraction function in the WRITE statements to produce the elements' names because the variable `elm` that represents the individual element (as well as the list of elements represented by the variable `in_list`) consists of *element IDs*, rather than names. To produce the element's names, use the IDs as parameters in a database extraction function that returns element names.

After the WHEN construct is executed and the element's type and name is written, the program loops back to the beginning of the FOR/LOOP and the instructions are carried out for another element. This continues until all input elements are written.

The last part of the template constructs lists of output elements and writes their types and names. This part is identical for that of input elements, except that a different function is used to build the output a-flow-line elements.

The final output is shown in the next section.

## Final Output for Act\_Interface Report

```
INTERFACE REPORT for sc_activities
```

```
Input elements:
```

```
DATA_ITEM  ACCEL_DEFLECTION
EVENT      ACCEL_PRESSED
EVENT      ACTIVATE_CRUISE
EVENT      BRAKE_PRESSED
EVENT      CLOCK
CONDITION  ENGINE_RUNNING
DATA_ITEM  GEAR_POSITION
CONDITION  MEASURE_SIGNAL
DATA_ITEM  MILEAGE
EVENT      RESUME_CRUISE
EVENT      SHAFT_ROTATION
DATA_ITEM  THROTTLE_POS
```

```
Output elements:
```

```
DATA_ITEM  CURRENT_SPEED
DATA_ITEM  MILEAGE
DATA_ITEM  THROTTLE_CONTROL
```

## Template for nroff

Whereas the previous templates in this section produced reports that were formatted by DGL `WRITE` statements, the templates in the following two sections contain commands specific to various formatting languages.

After a template is executed, the generated information, together with the embedded formatting commands for a particular formatting system, is passed to the output document segments. The file can then be further processed by a formatting system so the embedded formatting instructions are executed to produce the final document.

The following sections show how to use embedded formatting commands to produce the final document, which is similar to the one documented in [Overview of Documentor](#). They are different in their chapter and section headers.

This section explains the template logic for nroff. Because Interleaf has a slightly different logic, especially with regards to plot generation, it is explained in a separate section. For both templates, the initiation section is the same.



## Template with nroff Commands

```

TEMPLATE example;

-- Initiation Section
PARAMETER
  STRING act_name := 'SC_ACTIVITIES'; -- the activity
                                     -- for which the report is written.
  STRING plot_dev:= 'HP7475';

VARIABLE
  ACTIVITY act_id;      -- id of 'act_name'.
  INTEGER   st;        -- return status code.
  STRING    title;     -- title of plot.
  FLOAT     acty;      -- actual height of plot.

BEGIN
  act_id := stm_r_ac(act_name, st);
END;

SEGMENT seg1;
BEGIN
  INCLUDE('nroff_glob');
  WRITE('\n.bp');
  WRITE('\n.ce\nDescription of ', act_name);
  WRITE('\n.sp\n.sh 1 Overview');
  INCLUDE('sys_overview'); -- 'sys_overview' is an
                           -- include file in which
                           -- text with formatting
                           -- command is written.

END;

SEGMENT seg2;
VARIABLE
  LIST OF ACTIVITY    ac_list;

BEGIN
/@.bp
.sh 1 "SYSTEM ACTIVITIES"
.sp
.sh 2 "Activity-chart"
.sp
This is the chart that describes the activities
of the system:
.br
@/

```

```
-- leave 40 lines in one page to the plot
WRITE('\n.sp 40\n');
title:= 'Plot of ' + act_name;
-- plot of activity-chart
stm_plt(act_id,'act_plot',6.5,9.0,'F','F',
        true,999,plot_dev,stm_plt_top,title,acty);
-- second section
/@
.bp
.sh 2 "Activities Description"
.sp
Detailed description of each activity in chart:
@/

ac_list := stm_r_ac_physical_sub_of_ac({act_id},st);
-- Property Report
stm_rpt_dictionary(ac_list,true,true,' ');
END;
```

### Initiation Section (nroff)

In the initiation section of the templates, the string parameter `act_name` represents the name of the activity for which the report is generated. The value of `act_name` can be assigned at the time of template execution. Because you can also assign an initial value to the parameter, assume that `act_name` was assigned an initial value of `SC_ACTIVITIES`.

A second parameter, `plot_dev`, represents the plotter output device on which you want to produce the plot. This output device need not be identical to the one used to print the document. For example, you might want to produce the plot and text separately, merging them together later. This parameter is assigned an initial value, which you can change in a special form at the time of template execution.

Next, the template declares four variables:

- ◆ `act_id` - The ID of the activity for which the report is generated. This variable is declared of type `ACTIVITY`.
- ◆ `st` - A return status code for the database extraction functions. It is declared as an `INTEGER`.
- ◆ `title` - The string used to pass the title to the plot tool.
- ◆ `acty` - A float that is an output parameter from the plot tool, which specifies the actual height of the plot.

Lastly, in the body of the initiation section, there is one statement, an assignment for the variable `act_id`. To make the assignment, the template uses a database extraction function that takes `act_name` as a parameter and returns the corresponding ID.

## Segment 1: Heading and Report Overview (nroff)

The first template segment includes global declarations and produces a report heading and overview section. This section explains how these are produced with nroff.

### Including Global Declarations

The first statement of `seg1` passes the file `nroff_glob` to the output segment. This file, which is found in the databank, must be checked out to the workarea to be used by the Documentor. The file's function is to change default settings for nroff. For example, it cancels the special setting for the underscore symbol; this is done because Statemate uses underscores in element names.

### Producing the Heading and Overview (nroff)

The template uses `WRITE` statements to write nroff formatting commands to the output document segments. Note that there is a `\n` before each nroff command so each command written to the output segment starts on a separate line. For example, after executing the template, the first three `WRITE` statements cause the following to appear in the output segment `seg1`:

```
.PAGE
.CENTER; Description of SC_ACTIVITIES
.SKIP
.HL 1 Overview
```

The contents of the overview section are taken from an include file, `sys_overview`. This file resides in your workarea. As written in the comments, this file can contain formatting instructions in addition to the text. The entire file is passed verbatim to the output segment following the overview header.

After the output segment has been generated, it can then be formatted by nroff so the formatting commands are interpreted and the final, formatted document produced. For example, the formatting commands in `seg1` would cause a new page to be started; the words "Description of SC\_ACTIVITIES" would then be centered in the middle of the line, followed by a "skipped" line. The next formatting command would produce a header marked by the number 1. Lastly, the text contained in `sys_overview` is formatted according to the formatting instructions in this file or, if missing, according to the nroff defaults.

## Segment 2: Activity-Chart Plot and Property Report (nrff)

In the second segment of the template, `seg2`, the template produces the activity's plot and generates a property report of all of its subactivities using `plot` and `report` functions. Formatting commands are introduced where appropriate. One variable (`ac_list`) is declared, which represents the list of subactivities from which the property report is constructed.

The segment's body begins with the DGL verbatim symbol, `/@`. This command indicates that everything between it and the following `@/` symbol is to be included verbatim in the output document segment. The comments do not actually appear in the generated output, but explain the purpose of each command when interpreted by `nrff`. The generated output is as follows:

```
.PAGE -- starts a new page.
.HL 1 SYSTEM ACTIVITIES -- produces a header.
.SKIP -- skips a line.
.HL 2 Activity-chart -- produces a subheading.
.SKIP -- skips a line.
This is the chart that describes the activities
of the system:
.BREAK -- causes hard return (new line).
```

### Note

---

`nrff` commands written in a verbatim environment must start at the extreme left of the line (otherwise, the formatter does not correctly interpret them). Do not be misled into indenting these commands in the desire to produce a more readable template.

The next `WRITE` statement passes a `nrff` formatting command to the output segment. This command formats the page so as to provide a place for the plot that follows. A place for the plot must be made because `nrff` has no inherent graphics capability. The statement passes a `SKIP` command that causes the document to skip 40 lines to leave room for the activity chart plot.

Next, the template uses an `Include Plot` statement to generate a plot of the activity for which you are producing the document. Note that the second argument is the name of the file in which the output plot is contained. The file belongs to the file system outside of Statemate. If you do not give the full path name (as in the example), the file will appear in your workarea.

The ninth argument represents the plotter type. This determines the graphical language in which the plot is generated. Because this is a template parameter, you can change its initial value in a form before executing the template.

The example template produces the plot for a pen plotter, and places it in a file called `act_plot`. Following the execution of the template you must explicitly output the plot on the particular plotting device and thereafter manually merge the plot into the blank space set aside for this. For an explanation of the other parameters of the plot, refer to [INCLUDE](#).

Now you can give instructions for the inclusion of the property report for the subactivities. First, you include another section of formatting commands to be passed verbatim to the output segments. These include instructions for a subheading, a blank line, and a sentence introducing the property report. Next you assign a value to the variable `ac_list`, which represents the subactivities of the subject activity. This is followed by an `Include Property Report` statement, which causes the property report to appear in the output segment before formatting with its `nroff` commands embedded in the text. The Documentor embeds `nroff` commands because you attached the template to the `nroff` formatter at the Create Template stage. Upon formatting, these commands are interpreted along with the rest of the formatting commands.

## Template for Interleaf

This section describes a template with formatting commands for Interleaf. The final formatted output is the same as for the templates explained in the previous section.

The template is as follows:

```

TEMPLATE example;

-- Initiation Section
PARAMETER
  STRING act_name;  -- the activity for which the
                    -- report is written.

VARIABLE
  ACTIVITY  act_id;      -- id of 'act_name'.
  INTEGER   st;         -- return status code.

BEGIN
  act_id := stm_r_ac(act_name, st);
END;

SEGMENT seg1;
BEGIN
  INCLUDE ('interleaf_glob');
  INCLUDE ('my_glob');
  WRITE ('\n<doc_title>');
  WRITE ('\nDescription of ',act_name);
  WRITE ('\n<sect>');
  WRITE ('\n<Autonum, List8, 1, first-Yes><TAB>Overview');
  INCLUDE ('sys_overview');  -- 'sys_overview' is an
                             -- include file in which
                             -- text with formatting
                             -- commands is written.

END;

SEGMENT seg2;

VARIABLE
  LIST OF ACTIVITY  ac_list;

  STRING            title;

```

```

                                FLOAT                acty;

BEGIN
/@
<my_new_page>
<sect>
<Autonum, List8, 1><TAB>System Activities
<subsect>
<Autonum, List8, 2><TAB>Activity-chart
This is the chart that describes the activities of the system:
@/

title:= 'Plot of ' + act_name;
-- plot of activity-chart.
stm_plt (act_id, ' ', 6.5, 9.0, 'F', 'F', true, 999,
        'interleaf', stm_plt_top, title, acty);

/@
<my_new_page>
<subsect>
<Autonum, List8, 2><TAB>Activities Description
Detailed description of each activity in the chart:
@/
ac_list := stm_r_ac_physical_sub_of_ac ({act_id}, st);
-- property report
stm_rpt_dictionary (ac_list, true, true, '');
END;
```

### Initiation Section (Interleaf)

The initiation section is similar to that for the nroff template; refer to the explanation of the nroff initiation section.

### Segment 1: Heading and Report Overview (Interleaf)

Because Interleaf commands have definitions that must be specified before the commands are used, the template includes the file `interleaf_glob`, which contains these definitions. Among other definitions, it contains instructions used in producing Statemate predefined reports for Interleaf. This file is found in the databank and must be checked out to your workarea to be used by the Documentor. In addition, the template includes user-defined definitions contained in the file `my_glob`.

In the first segment, you pass both files to the output segment using `INCLUDE` statements. When the segment is formatted, the formatting command definitions included in the files are interpreted.

## Producing the Heading and Overview (Interleaf)

The template uses `WRITE` statements to write the Interleaf formatting commands to the output document segments. Note that there is a `\n` before each Interleaf command so each command written to the output segment starts on a separate line.

The Interleaf commands written to the output segment and used for the heading and overview section are as follows:

```
<doc_title> -- starts a new page and defines the
              title font and centers the title.
              The definition of this command is
              included in "my_glob".
<sect> -- starts a new section. It defines the
          font of a section title, its margins etc.
          The definition of this command is
          included in "my_glob".
<Autonum, List8, 1, first-Yes> -- writes the first
                                section number.
```

The following text appears in the output segment immediately following the contents of the include file:

```
<doc_title>
Description of SC_ACTIVITIES
<sect>
<Autonum, List8, 1, first-Yes><TAB>Overview
```

This is followed by the text of the `sys_overview` file. This is an include file and can contain text and Interleaf formatting commands.

## Segment 2: Activity-Chart Plot and Property Report (Interleaf)

The second segment of the template, `seg2`, produces the activity's plot and generates a property report for all of its subactivities using `plot` and `report` functions. Formatting commands are introduced where appropriate. One variable (`ac_list`) is declared, which represents the list of subactivities from which the property is constructed.

The segment's body begins with the DGL verbatim symbol, `/@`. This command indicates that everything between it and the following `@/` symbol is to be included verbatim in the output document segment. The comments do not actually appear in the generated output, but explain the function of each command when interpreted by Interleaf. The generated output is as follows:

```
<my_new_page> -- starts a new page. The definition
                of this command is included
                in "my_glob".
<sect> -- starts a section. (see Segment 1).
<Autonom, List8, 2><TAB>System Activities -- writes
                -- section header.
<subject> -- starts a subsection.
<Autonom, List8, 2><TAB> Activity-chart -- writes
                -- subsection header.

This is the chart that describes the activities
of the system.
```

### Note

---

Certain Interleaf commands, such as `<sect>` and `<subject>`, that are written in a verbatim environment must start at the extreme left of the line. Otherwise, the formatter does not correctly interpret them). Do not be misled into indenting these commands in the desire to produce a more readable template.

In the case of the activity chart plot, you follow a slightly different formatting logic from that of `nroff` because Interleaf has a graphics capacity. Therefore, you do not have to leave space for the plot—the language automatically prepares the page for this. To include the plot in the output file, you supply an empty string as the second parameter in the `Include Plot` statement. This causes the plot information to be passed directly to the output segment. Note that the ninth parameter is `'interleaf'`, which specifies the output language in which the plot is generated. Upon formatting, Interleaf interprets the plot graphics and draws it into the document.

For an explanation of the other parameters of this plot, refer to the [stm\\_plot](#) function.



Now you are ready to include the property report for the subactivities. First, you include another section of formatting commands to be passed verbatim to the output segments. These include instructions for a new page, a subheading, and a sentence introducing the property report. Next, you assign a value to the variable `ac_list`, which represents the subactivities of the subject activity. This is followed by an `Include Property Report` statement. This statement causes the property report to appear in the output segment before formatting, with its Interleaf commands embedded in the text. The Documentor embeds Interleaf commands because you attached the template to the Interleaf formatter at the Create Template stage. Upon formatting, these commands are interpreted along with the rest of the formatting commands. The final output is very similar to the document in [Overview of Documentor](#) (but there are differences in the appearance of the headings).



# DGL Statement Reference

---

[Document Templates](#) provides general information about the Document Generator Language (DGL), including its overall use, conventions, and a detailed description of the types of statements used in programming document templates. This section provides a comprehensive reference for the DGL statements.

This section presents the statements in alphabetical order by name or keyword. For each DGL statement, the following information is provided:

- ◆ **Name** - The keywords used to write the statement.
- ◆ **Description** - A detailed description of the statement and what it does.
- ◆ **Syntax** - A general syntax description of the statement's structure, including parameters, qualifiers, and options.

The syntax section uses the following conventions:

- ◆ Except for literal strings, DGL statements are *not* case-sensitive, and your input can be in either upper- or lowercase. However, upper- and lowercase letter are used in the syntax section to indicate the following:
  - ◆ Keywords are shown using uppercase letters.
  - ◆ Words written entirely in lowercase indicate variable qualifiers or parameters.
- ◆ Items are optional when written inside square brackets [].
- ◆ The pipeline symbol (|) denotes "or" between items.
- ◆ Ellipses (...) indicate multiple item occurrences. Ellipses can appear horizontally or vertically.

Consider the following examples:

```
ALPHA beta [gamma]
```

ALPHA is a statement that takes a required qualifier `beta`, and might take an optional qualifier, `gamma`.

```
JUNK filename [,filename...]
```

JUNK is a statement that takes a file name as a parameter. You can specify multiple file names, separating them with commas.

### Note

---

Each DGL statement *must* be concluded by a semicolon. This is shown in the statement syntax. Flow control and structure statements can contain a number of statements. Multiple statements can be contained on a single line, or a single statement can span several lines.

- ◆ **Notes** - Special points of interest regarding the use of the statement.
- ◆ **Examples** - Examples that illustrate how to use the statement.
- ◆ **See Also** - This refers you to the section of this manual where the particular statement is explained.

The remainder of this section presents the DGL statements in alphabetical order.

- ◆ [ASSIGNMENT](#)
- ◆ [BEGIN](#)
- ◆ [CLOSE](#)
- ◆ [COMMENT](#)
- ◆ [CONSTANT](#)
- ◆ [END](#)
- ◆ [EXECUTE](#)
- ◆ [EXIT](#)
- ◆ [FOR/LOOP](#)
- ◆ [IF/THEN/ELSE](#)
- ◆ [INCLUDE](#)
- ◆ [OPEN](#)
- ◆ [PARAMETER](#)
- ◆ [PROCEDURE](#)
- ◆ [READ](#)
- ◆ [REPORT](#)
- ◆ [SEGMENT](#)

- 
- ◆ [SELECT/WHEN](#)
  - ◆ [STOP](#)
  - ◆ [TABLE](#)
  - ◆ [TEMPLATE](#)
  - ◆ [VARIABLE](#)
  - ◆ [VERBATIM](#)
  - ◆ [WHILE/LOOP](#)
  - ◆ [WRITE](#)

# ASSIGNMENT

## Description

Assigns the value of an expression to a variable. An assignment statement should be interpreted as follows:

The variable on the left-hand side of the statement is assigned the value of the expression on the right-hand side.

The expression and the variable must be of the same or compatible type. If the expression is of type `STATE`, `ACTIVITY`, and so on, the variable is of the same type, or of type `ELEMENT`.

## Syntax

```
variable := expression;
```

## Notes

The expression and the variable must be of the same or compatible type. If the expression is of type `STATE`, `ACTIVITY`, and so on, the variable is of the same type or of type `ELEMENT`.

## Example

```
VARIABLE
  LIST OF STATE      st_list;
  LIST OF ACTIVITY   act_list;
  LIST OF ELEMENT    el_list;
BEGIN
  .
  el_list := st_list + act_list;
  .
END;
```

## See Also

- ◆ [CONSTANT](#)
- ◆ [PARAMETER](#)
- ◆ [VARIABLE](#)
- ◆ [TEMPLATE Statement](#)

# BEGIN

## Description

Delineates the boundaries of a template section. Any number of other DGL statements can be between the `BEGIN` and `END`. The statements are optional; a section having no statements between the `BEGIN` and the `END` is legal.

## Syntax

```
BEGIN
[statement;]
.
.
.
END;
```

## Parameters

Parameter	Description
statement	A comment, or DGL statement other than structure or declaration type statements

## Notes

`BEGIN` and `END` always appear as a pair. Note that there is no end-of-statement symbol (;) after `BEGIN`.

## Example

```
SEGMENT SEG1;
VARIABLE
  STRING name;
  STATE state-chart;
  INTEGER status;
BEGIN
  name:= stm_r_st_name (state_chart, status);
  IF status = stm_success AND name <> '' THEN
    WRITE (.sh l" , name, "\n");
    WRITE ('\n');
  ELSE
    WRITE ('empty name or error status\n');
  END IF;
END ;
```

**See Also**

- ◆ [COMMENT](#)
- ◆ [END](#)
- ◆ [PROCEDURE](#)
- ◆ [SEGMENT](#)
- ◆ [TEMPLATE](#)
- ◆ [Structure Statements](#)



# CLOSE

## Description

Closes files after they have been opened with the `OPEN` statement. After opening a file and passing information to and from it, use the `CLOSE` statement to prevent further information from being passed to or from it.

## Syntax

```
CLOSE (f1);
```

## Parameters

Parameter	Description
f1	An identifier of type <code>FILE</code> , previously assigned by the <code>OPEN</code> statement

## See Also

- ◆ [OPEN](#)
- ◆ [READ](#)
- ◆ [VARIABLE Statement](#)

# COMMENT

## Description

Prefaces a comment in a template.

Comments are used in templates for documentation purposes. When processing a template, the Documentor compiler ignores these comments. A comment can contain any printable ASCII character. The comment symbol (double-dash) can appear anywhere a space can, except within a literal string.

Comments are preceded by two dashes. The comment symbol can start anywhere in a line, except within a literal string. The Documentor ignores all the characters after the comment symbol until the end of the line.

For example:

```
-- This line contains program comments.
```

## Syntax

```
-- free-text
```

## Parameters

Parameter	Description
free-text	Any printable string

## Example

```
-- This is a valid comment line.  
a:=a+1; -- This is another comment.
```

## See Also

- ◆ [BEGIN](#)
- ◆ [END](#)
- ◆ [PROCEDURE](#)
- ◆ [SEGMENT](#)
- ◆ [TEMPLATE](#)
- ◆ [Structure Statements](#)

---

# CONSTANT

## Description

Declares constants (identifiers whose values do not change throughout the template or during execution).

The keyword `CONSTANT` appears only once in the declaration section, before the data-type assignments for constants. Each data-type statement can be followed by as many identifiers of the same type as you want to define. For example:

```
CONSTANT integer a:=1, b:=2, c:=3;
```

Similarly, as many type statements as you want to define can follow the `CONSTANT` keyword. For example:

```
CONSTANT
  STRING activity_name := 'Print';
  FLOAT a:=3.243;
  INTEGER c:=6;
```

**Note:** The constant type cannot be a `StateMate` element or a `LIST OF` type.

The identifiers in the `CONSTANT` statement must have their values assigned in the statement. The value can be any expression that does not contain variables or parameters.

## Syntax

```
CONSTANT
type identifier := value [, identifier := value, ...];
[ type identifier := value, ...;]
```

## Parameters

Parameter	Description
type	The type of constant (integer, float, string, or Boolean)
identifier	The name of the constant
value	An expression that specifies the value of the constant

### Example

```
CONSTANT
STRING  out_file := '/usr/group/doc12.stm' ;
STRING  in_file  := '/usr/group/insert12.doc';
INTEGER page_length := 66, page_width:=72;
```

### See Also

- ◆ [ASSIGNMENT](#)
- ◆ [PARAMETER](#)
- ◆ [VARIABLE](#)
- ◆ [TEMPLATE Statement](#)

# END

## Description

Used as part of the `BEGIN/END` construct in the delineating of template sections.

See the [BEGIN](#) statement for more information.

## Syntax

```
BEGIN  
[statement;]  
.  
.  
.  
END;
```

## See Also

- ◆ [BEGIN](#)
- ◆ [COMMENT](#)
- ◆ [PROCEDURE](#)
- ◆ [SEGMENT](#)
- ◆ [TEMPLATE](#)
- ◆ [Structure Statements](#)

# EXECUTE

## Description

Invokes a program external to the Statement system. This statement invokes a program that is external to Statemate. The Documentor searches for the program name using the regular system search path, invokes the program, and sends its standard output to the document's output segment file.

The `EXECUTE` statement function operates as a function that returns either `stm_success` or `stm_error`.

## Syntax

```
[status := ] EXECUTE (calling_sequence);
```

## Parameters

Parameter	Description
<code>calling_sequence</code>	<p>A string expression (or string literal) that specifies the name and parameters of an external program.</p> <p>The <code>calling_sequence</code> is written in the same manner as when invoking the program from the shell command line. Therefore, the calling sequence can include arguments to be passed to the program. However, no evaluation of these arguments is done in the template—the calling sequence is passed as a pure string.</p>

## Example

This following statement calls the operating system function `DATE` and writes the date to the output file:

```
EXECUTE ('date');
```

## See Also

- ◆ [INCLUDE](#)
- ◆ [REPORT](#)
- ◆ [TABLE](#)
- ◆ [VERBATIM](#)
- ◆ [WRITE](#)
- ◆ [READ Statement](#)

# EXIT

## Description

Exits from the current loop to the statement after the loop construct or to the construct that contains the current loop.

Typically, a condition would be tested in a loop, and the exit would be based upon the evaluation of that condition.

## Syntax

```
EXIT;
```

## Note

EXIT can be used only inside a FOR and WHILE loop.

## Example

This example continues the execution of the statements between the LOOP and END LOOP, depending on the value of the condition `a > b`.

The iterations go on as long as the status `st` is equal to 0. When `st` is not equal to `stm_success`, it causes the iterations to stop. The IF statement here is used to force an abnormal EXIT from within the WHILE loop.

The execution resumes at the next statement after the END LOOP.

```
WHILE a > b LOOP
  md_id := stm_r_md (name, st);
  IF st <> stm_success THEN
    WRITE ('Illegal Status');
    EXIT;
  END IF;
  .
  .
END LOOP;
```

## See Also

- ◆ [FOR/LOOP](#)
- ◆ [IF/THEN/ELSE](#)
- ◆ [SELECT/WHEN](#)
- ◆ [STOP](#)
- ◆ [WHILE/LOOP](#)
- ◆ [Control Flow Statements](#)

## FOR/LOOP

### Description

Provides iterative execution of template statements.

The `FOR/LOOP` construct is used to execute iterative DGL statements. This statement executes the statements between `LOOP` and `END LOOP` for each item in the specified list. The identifier is a variable whose value is set sequentially to the items in the list. This variable can be used within the body of the loop, but its value cannot be reassigned.

Alternatively, a range of integers can be specified in place of the `list`, as in the following example:

```
FOR i IN {1..100} LOOP
  statement;
  .
  .
END LOOP;
```

### Syntax

```
FOR identifier IN list
LOOP [statement;]
  .
  .
END LOOP;
```

### Parameters

Parameter	Description
identifier	A variable whose value is set sequentially to the items in the list. The type of the <code>identifier</code> must match the type of the <code>list</code> . This variable can be used within the body of the loop.
list	A list expression.
statement	A DGL statement.



## Notes

The type of the variable identifier must match the type of the list. The list can be a range of integers written as follows, where  $x$  is an integer variable:

```
FOR x IN {1..100} LOOP ...
```

## Example

This example writes to the output file the names of all the states in the list `statelist`:

```
FOR id IN statelist LOOP
    WRITE ('\n', stm_r_st_name (id, status));
END LOOP;
```

## See Also

- ◆ [EXIT](#)
- ◆ [IF/THEN/ELSE](#)
- ◆ [SELECT/WHEN](#)
- ◆ [STOP](#)
- ◆ [WHILE/LOOP](#)
- ◆ [Control Flow Statements](#)

## IF/THEN/ELSE

### Description

Provides conditional execution of template statements.

The IF/THEN/ELSE construct is used for conditional execution of DGL statements. The statements following the THEN (and before any ELSE) are executed if `boolean` evaluates to TRUE. If it evaluates to FALSE, the statements following ELSE are executed, when present.

### Syntax

```
IF boolean THEN
  [statement;]
  .
  .
  .
  [ELSE
    [statement;]
    .
    .
    .
  ]
END IF;
```

### Parameters

Parameter	Description
<code>boolean</code>	A Boolean expression
<code>statement</code>	A DGL statement

**Example**

```
IF a >= b THEN
    EXECUTE ('DATE');
    INCLUDE ('sample.txt');
ELSE
    WRITE ('a is less than b');
END IF;
```

**See Also**

- ◆ [EXIT](#)
- ◆ [FOR/LOOP](#)
- ◆ [SELECT/WHEN](#)
- ◆ [STOP](#)
- ◆ [WHILE/LOOP](#)
- ◆ [Control Flow Statements](#)

# INCLUDE

## Description

Copies the text of the specified file to an output segment file. The text is passed to the file verbatim and can contain formatting commands for the format processor used to produce the formatted document.

The file can be an include file within Statemate or a file outside Statemate in the general file system of the host computer.

## Syntax

```
INCLUDE (file_description [, status])
```

## Parameters

Parameter	Description
<code>file_description</code>	<p>A file name or an identifier of type <code>FILE</code>. When it is a file name, it can be any string expression, such as a literal string inside quotes (for example, <code>'ABC'</code>) or an evaluated expression that produces a file name (for example, a variable that contains a file name).</p> <p>The file name can include the directory path name of the file, using the file name conventions of the host operating system. If you do not specify the path name, the Documentor searches the workarea for the file.</p> <p>When the <code>file_description</code> is a file identifier, the file from which you are copying text must first be opened with an <code>OPEN</code> statement in <code>OUTPUT</code> mode (refer to <a href="#">OPEN</a>).</p>
<code>status</code>	<p>The function status code. It returns one of the following values:</p> <ul style="list-style-type: none"><li>• <code>stm_success</code>—Successful execution of the statement.</li><li>• <code>stm_error</code>—Failure.</li></ul>

### Example

In this example, a UNIX file external to Statemate is included in the output segment:

```
INCLUDE('/usr/group/gx_file');
```

### See Also

- ◆ [EXECUTE](#)
- ◆ [REPORT](#)
- ◆ [TABLE](#)
- ◆ [VERBATIM](#)
- ◆ [WRITE](#)
- ◆ [READ Statement](#)

# OPEN

## Description

Opens files or the dialog area so text can be written to or read from them.

This statement is used with `mode=OUTPUT` to open a file or the dialog area so that a subsequent `WRITE` statement can pass text to it, and with `mode=INPUT` for a subsequent `READ` statements.

When writing to the dialog area, the syntax is as follows:

```
OPEN (f1, 'DIALOG', OUTPUT);
```

## Syntax

```
OPEN (f1, file_name, mode [, status]);
```

## Parameters

Parameter	Description
<code>f1</code>	An identifier of type <code>FILE</code> , assigned by the <code>OPEN</code> statement.
<code>file_name</code>	The name of the file to (or from) which to pass the text, or the dialog area. If the file opened for output does not exist, this statement creates a new file. If the file exists, the <code>OPEN</code> statement overwrites the existing contents of the file.  To open the dialog area, use the string <code>'DIALOG'</code> (with apostrophes) for this parameter. Open the dialog area so you can pass run-time messages to it.
<code>mode</code>	If <code>mode=OUTPUT</code> , the <code>OPEN</code> statement opens a file or the dialog area so a subsequent <code>WRITE</code> statement can pass text to it.  If <code>mode=INPUT</code> , the <code>OPEN</code> statement opens a file or the dialog area so a subsequent <code>READ</code> statement can pass text to it.
<code>status</code>	The function status code. It returns one of the following values: <ul style="list-style-type: none"><li><code>stm_success</code>—Successful execution of the statement.</li><li><code>stm_error</code>—Failure.</li></ul>

**Notes**

The name of the file is specified in accordance with the conventions of the operating system. If the directory is not specified as part of the file name, the workarea is searched for the file. If the file opened for output does not exist, this statement creates a new file. If the file exists, it is initialized by the `OPEN` statement (the written information overwrites the existing contents of the file).

**See Also**

- ◆ [CLOSE](#)
- ◆ [READ](#)
- ◆ [VARIABLE Statement](#)

# PARAMETER

## Description

Declares variables and values within a template.

There are *template parameters* for the entire template and *procedure parameters* for procedures. Template parameters are variables whose value can be changed interactively when the template is executed.

You can declare template parameters only in the initiation section. The keyword `PARAMETER` appears only once in the declaration section, before the data-type assignments for parameters. Each data-type statement can be followed by as many identifiers of the same type as you want to define. For example:

```
PARAMETER STRING activity_name, state_name, event_name;
```

As many type statements as desired can follow the `PARAMETER` keyword. For example:

```
PARAMETER
  STRING activity_name;
  FLOAT a:=3.243;
```

Template parameters cannot hold a Statemate element, although it is legal for a procedure parameter to be of this type. Procedure parameters are In/Out parameters.

Value assignments for parameter statements are optional and are allowed only for template parameters. If it is assigned, it represents the default value of the parameter at the first generation of a particular document. The value can only be a literal constant, *not* a constant identifier or an expression.

**Note:** Avoid changing template parameters within the template. In addition to being confusing, there might be inconsistent results when parameters are changed within segments.

## Syntax

```
PARAMETER
type identifier [:= value][, identifier [:= value],...];
[type identifier [:= value],...]
.
.
.
```



## Parameters

Parameter	Description
type	The parameter type (integer, float, string, or Boolean—or a list of these types).
identifier	The name of the parameter. The maximum length of the identifier is 16 characters. If you specify more than 16 characters, the name is truncated.
value	value

## Example

```
PARAMETER
  STRING statechart_name;
  INTEGER page_width := 80;
```

## See Also

- ◆ [ASSIGNMENT](#)
- ◆ [CONSTANT](#)
- ◆ [VARIABLE](#)
- ◆ [TEMPLATE Statement](#)

# PROCEDURE

## Description

Begins a procedure section. It assigns an identifying name to the procedure and defines the return type, if the function returns a value.

The definition of a procedure is as follows:

```
PROC <procname> [return <type>];  
[PARAMETER  
  <type> <parm1>;  
  ....  
  <type> <parmN>;  
]  
[VARIABLE  
  var section;  
]  
BEGIN  
  <statements>;  
  return <expression>;  
END;
```

## Syntax

```
PROCEDURE procedure_name [RETURN type];
```

## Parameters

Parameter	Description
procedure_name	The name of the procedure. The maximum length is 16 characters.
RETURN type	Assigns the value that is returned by the function and returns from the procedure.

## Notes

The scoping rules for procedures are as follows:

- ◆ For simplification reasons, procedures are not lexically nested within other procedures, nor are they nested within segments.
- ◆ Like segments, procedures can refer to template global variables.
- ◆ Procedures can be referenced by any template segment and can be defined anywhere within the template (but not within segments).
- ◆ Parameters are always In/Out.

**Example**

```

TEMPLATE sss;
PARAMETER
  string      plot_dir      := '';
  string      appendix      := '';
VARIABLE
  string      module_name;
  string      key_bg , key_en;

BEGIN
  open (da, 'DIALOG', OUTPUT);
  .....
END;

PROCEDURE get_generic_info;
PARAMETER
  activity  cap;      -- Input
  activity  capdef;
  boolean   is_generic;
  element   chrt;     -- Outputs
VARIABLE
  integer   st, st1, st2, st3;
  activity  acttmp;
BEGIN
  capdef := cap;
  is_generic := false;

  IF (stm_r_mx_generic_instance_mx ({cap}, st) <> {})
  THEN
    chrt:=stm_list_first_element
      (stm_r_ch_defining_ac({cap},st),st2);
    IF (st = stm_success AND st2 = stm_success) THEN
      acttmp := stm_list_first_element
        (stm_r_ac_root_in_ch({chrt},st),st2);
    IF (st = stm_success AND st2 = stm_success) THEN
      acttmp := stm_list_first_element
        (stm_r_ac_internal_ac(
          stm_r_ac_logical_sub_of_ac ({acttmp}, st)
          ,st2),st3);
    IF (st = stm_success AND st2 = stm_success AND
        st3 = st2) THEN capdef := acttmp;
      is_generic := true;
    END IF;
  END IF;
END IF;
END IF;
END IF;
END;

PROCEDURE put_plot_lines;
PARAMETER
  float acty; -- Whatever it means
  string title;
VARIABLE
  integer lines;
BEGIN
  acty := (acty/INCH_PER_LINE) + 0.5;
  lines := 0;
  WHILE (acty > 0) LOOP
    acty := acty-1;
  END LOOP;
END;

```

```
        lines := lines + 1;
    END LOOP;
    write ('\n<HardReturn>');
    write ('\n First line of plot ',title,'
          (Delete between lines)--->>');
    WHILE (lines > 0) LOOP
        write ('\n<HardReturn>\nReserved for plot ---> ',fl);
        lines := lines - 1;
    END LOOP;
    write ('\n<HardReturn>');
    write ('\n Last line of plot ',title,'
          (Delete between lines)--->>');
END;
```

```
PROCEDURE put_bindings ;
PARAMETER
    element cap;
VARIABLE
    element par;
    list of element formals;
    integer st, etype;
    string title, stype, actual;
BEGIN
    -- Specify the bindings
    formals := stm_r_mx_parameter_of_ch
              (stm_r_ch_defining_ac ({cap}, st), st);
    .....
END; -- Put bindings
```

```
PROCEDURE put_mini_spec;
PARAMETER
    activity act;
VARIABLE
    integer st1,st;
    string mini_spec;
BEGIN
    mini_spec := stm_r_ac_mini_spec (act, st1);
    IF (st1 = stm_success) THEN
        write ('\n<Bold>\n');
        write ('Mini-Spec :');
        write ('\n<NoBold>\n');
        write ('\n\n');
        lwrite (mini_spec);
        write ('\n\n');
    END IF;
    IF (st <> stm_success and st1 <> stm_success) THEN
        write ('TBD.');
```

```
    END IF;
END ; -- put_mini_spec

SEGMENT body;

    cap:=....
    put_bindings (cap);
    put_mini_spec (cap);

    .....
END;
```

**See Also**

- ◆ [BEGIN](#)
- ◆ [COMMENT](#)
- ◆ [END](#)
- ◆ [SEGMENT](#)
- ◆ [TABLE](#)
- ◆ [Structure Statements](#)

# READ

## Description

Reads a line of information from an external file into variables in the template. The numeric elements in the input line are separated by blanks or tabs. Reading to a string variable reads the rest of the line.

The `READ` statement can operate as a function that returns either `stm_success`, `stm_cannot_read_file`, or `stm_end_of_file`.

## Syntax

```
READ (f1, variable1, variable2, ...);
```

## Parameters

Parameter	Description
<code>f1</code>	An identifier of type FILE, that points to a file previously opened by the OPEN statement in INPUT mode
<code>variable1</code> , <code>variable2...</code>	Identifiers of type integer, float, or string

## Example

In this example, `i` is an integer and `str` is a string. This statement, when applied to an input line `12 May 2003`, results in `i=12`, `str='May 2003'`.

```
READ(fd, i, str)
```

## See Also

- ◆ [CLOSE](#)
- ◆ [OPEN](#)
- ◆ [VARIABLE Statement](#)

# REPORT

## Description

Invokes the Reports tool to generate a predefined report as part of a document.

For example:

```
stm_rpt_tree(list,5);
```

This call produces a tree report for the items in a list represented by the variable `list` to a depth of 5 in the hierarchy. The report is included in the output file.

The output from the Reports tool contains formatting commands applicable to the format processor attached to the template. If no formatter is specified, the Reports tool cannot be invoked and the Documentor generates an error message.

Each of the predefined reports is invoked for a list of elements. The input parameter that represents this list is denoted by a variable name that must be of type `list` of one of the Statemate element types. This variable, along with all other identifier names used in the calling sequence, must be declared in an appropriate declaration section. For example:

```
VARIABLE  
LIST OF ACTIVITY ac_list;
```

The identifier `ac_list` can be assigned a list of activities and then be included in a statement that generates a property report, as follows:

```
stm_rpt_dictionary (ac_list,...);
```

The report is generated for each item in the list represented by the `ac_list` variable.

A number of arguments are used to define the parameters for each report. Some are called “single-character string arguments,” which are used to indicate restricted parameter choices. Consider the following interface report statement

```
stm_rpt_interface (elist, 'A', ...);
```

The value of the second argument, `A`, indicates that the interface report should be of type `activities`; specifying an `M` for this parameter would indicate that the report should be generated for `modules`.

The single-character string arguments can be more than one character, but only the first character of the string is actually passed to the Reports tool. If a non-valid character is passed to the Reports tool, the report is not generated and an error status code is returned.

Some of the arguments are Boolean and are evaluated as `TRUE` or `FALSE` to indicate whether some parameter is set. For example, consider the following property report statement:

```
stm_rpt_dictionary (elist, true, ...);
```

The Boolean constant `true` indicates that the long description will be included in the report.

### Syntax

```
stm_rpt_<report_name> (report_parameters);
```

### Parameters

Parameter	Description
<code>report_name</code>	The type of report to be generated
<code>report_parameters</code>	The report parameters

### Supported Statement Reports

The supported report types are as follows:

- ◆ [Attribute Report](#)
- ◆ [Property Report](#)
- ◆ [Interface Report](#)
- ◆ [List Report](#)
- ◆ [N2 Chart Report](#)
- ◆ [Protocol Report](#)
- ◆ [Resolution Report](#)
- ◆ [Structure Report](#)
- ◆ [Tree Report](#)



---

## Attribute Report

### Syntax

```
stm_rpt_attribute (elist, attrs, attr_title);
```

### Parameters

Parameter	Description
elist	A list expression of Statemate elements for which the report is produced.
attrs	A list of strings that contains the specific attribute names for which the report should be generated. If this list is empty, the report retrieves all the attributes for each element.
attr_title	A string indicating that the attribute value will precede its element name in the report.

## Property Report

### Syntax

```
stm_rpt_dictionary (elist, ldes, attr, attr_title);
```

### Parameters

Parameter	Description
elist	A list expression of Statemate elements for which the report is produced
ldes	A Boolean expression indicating whether to include the long description of each element in the report
attr	A Boolean expression indicating whether to include the attributes of an element in the report
attr_title	A string indicating the attribute names whose value will precede the element name in the report

## Interface Report

### Syntax

```
stm_rpt_interface (elist, rtype, chart, lact lmod, ftype,
                 dis, names);
```

### Parameters

Parameter	Description
elist	A list expression that must be of the type list of modules, for which the report is produced.
rtype	A single-character string argument indicating the report type, as follows: <ul style="list-style-type: none"> <li>• <b>A</b>—Activity interface report</li> <li>• <b>M</b>—Module interface report</li> <li>• <b>I</b>—Information interface report</li> </ul>
chart	A single-character string argument indicating which arrows are taken into account when the report is generated. The possible values are as follows: <ul style="list-style-type: none"> <li>• <b>A</b>—Activity-chart arrows</li> <li>• <b>M</b>—Module-chart arrows</li> </ul>
lact	An argument of type list of activities indicating which activities are taken into account when the report is generated. If lact is an empty list, the default is all activities implemented by the center module. If chart is 'M', the lact parameter has no use, but still must be specified. For simplicity, use the null list (null).
lmod	An argument of type list of modules indicating the side modules that interface with the central module for which the report is to be generated. If lmod is empty, the default behavior is to use <i>all</i> modules except the center module's own ancestors and descendants.
ftype	A single-character string argument indicating the kind of information flow to appear in the report. The possible values are as follows: <ul style="list-style-type: none"> <li>• <b>D</b>—Data-flows</li> <li>• <b>C</b>—Control-flows</li> <li>• <b>B</b>—Both data-flows and control-flows</li> </ul>
dis	A single-character string argument indicating the kind of information to appear in the report. The possible values are as follows: <ul style="list-style-type: none"> <li>• <b>I</b>—Flow labels</li> <li>• <b>P</b>—Parent information items</li> <li>• <b>B</b>—Basic information items</li> </ul>
names	A single-character string argument specifying how names appear in the report. The possible values are as follows: <ul style="list-style-type: none"> <li>• <b>N</b>—The name appears for elements that flow between the boxes.</li> <li>• <b>S</b>—The synonym appears for elements that flow between the boxes.</li> </ul>

## List Report

### Syntax

```
stm_rpt_list (elist);
```

### Parameters

Parameter	Description
elist	A list expression of Statemate elements for which the report is produced

## N2 Chart Report

### Syntax

```
stm_rpt_n2chart(elist, names, level, env, chart, dis, ftype);
```

### Parameters

Parameter	Description
elist	A list expression that must be of the type list of modules or list of activities, which specifies the elements in the diagonal.
names	A single-character string argument specifying how names appear in the report. The possible values are as follows: <ul style="list-style-type: none"> <li>• <b>N</b>—Names of the elements appear on the diagonal of the matrix.</li> <li>• <b>S</b>—Synonyms of the elements appear on the diagonal of the matrix.</li> </ul>
level	A single-character string argument indicating what appears on the diagonal when both parent box and sub-box are in the list. The possible values are as follows <ul style="list-style-type: none"> <li>• <b>B</b>—The sub-box is placed on the diagonal of the matrix.</li> <li>• <b>P</b>—The parent box is placed on the diagonal of the matrix.</li> </ul>
env	A Boolean expression. If this is TRUE, the environment is added to the matrix.
chart	A single-character string argument indicating which arrows are taken into account when the report is generated. The possible values are as follows: <ul style="list-style-type: none"> <li>• <b>A</b>—Activity-chart arrows</li> <li>• <b>M</b>—Module-chart arrows</li> </ul>

Parameter	Description
<code>dis</code>	A single-character string argument indicating the kind of information to appear in the report. The possible values are as follows: <ul style="list-style-type: none"><li>• <b>I</b>—Flow labels</li><li>• <b>P</b>—Parent information items</li><li>• <b>B</b>—Basic information items</li></ul>
<code>ftype</code>	A single-character string argument indicating the kind of information flow to appear in the report. The possible values are as follows: <ul style="list-style-type: none"><li>• <b>D</b>—Data-flows</li><li>• <b>C</b>—Control-flows</li><li>• <b>B</b>—Both data-flows and control-flows</li></ul>

## Protocol Report

### Syntax

```
stm_rpt_protocol (elist, attr_title);
```

### Parameters

Parameter	Description
<code>elist</code>	A list expression of Statemate elements for which the report is produced.
<code>attr_title</code>	A string indicating the attribute name whose value will precede the element name in the report.

---

## Resolution Report

### Syntax

```
stm_rpt_resolution (clist, type)
```

### Parameters

Parameter	Description
<code>clist</code>	A list of charts. It determines the scope of the report. The <code>stm_type</code> of elements to include in the report. The possible types are as follows: <ul style="list-style-type: none"><li>• <code>stm_textual</code></li><li>• <code>stm_graphical</code></li><li>• <code>stm_mixed</code></li><li>• <code>stm_state</code></li><li>• <code>stm_module</code></li><li>• <code>stm_activity</code></li><li>• <code>stm_data_store</code></li></ul>

## Structure Report

### Syntax

```
stm_rpt_structure (elist, width);
```

### Parameters

Parameter	Description
<code>elist</code>	A list expression of Statemate elements for which the report is produced.
<code>width</code>	An integer argument indicating the page width (in characters) to be used for the report.

## Tree Report

### Syntax

```
stm_rpt_tree (elist, depth);
```

### Parameters

Parameter	Description
elist	A list expression of Statemate elements for which the report is produced.
depth	An integer argument indicating the to what hierarchical level the report should be generated. For all levels, use the value "99".

### See Also

- ◆ [EXECUTE](#)
- ◆ [INCLUDE](#)
- ◆ [TABLE](#)
- ◆ [VERBATIM](#)
- ◆ [WRITE](#)
- ◆ [READ Statement](#)

# SEGMENT

## Description

Starts a new segment section. It is the first statement of a segment and assigns an identifying name to it. The segment name is used by the Documentor in its operation forms to identify the output segments refer to [Creating a Document](#)).

## Syntax

```
SEGMENT segment_name;
```

## Parameters

Parameter	Description
segment_name	A name used to identify the segment. This identifier is limited to a maximum length of 16 characters. Characters in excess of this amount are truncated and do not appear when the segment names are displayed in the operations forms.

## Example

```
SEGMENT chapter_2B;
```

## See Also

- ◆ [BEGIN](#)
- ◆ [COMMENT](#)
- ◆ [END](#)
- ◆ [PROCEDURE](#)
- ◆ [TEMPLATE](#)
- ◆ [Structure Statements](#)

## SELECT/WHEN

### Description

Performs conditional execution of DGL statements. This statement is more powerful than the `IF/THEN/ELSE` statement in that it allows you to systematically list multiple conditions for statement execution.

Note that `WHEN` statements are composed of two parts: the trigger to the left of the arrow, and statements on the right side of the arrow. The trigger is any valid Boolean expression. The statements following a trigger are performed only when the trigger is `TRUE`. Whether or not these statements are actually executed also depends on the value of the `selection_mode`. In addition:

- ◆ The `WHEN ANY` statements are executed when one or more of the preceding `WHEN` statements have been executed.
- ◆ The `OTHERWISE` statements are executed only if no `WHEN` statement within the `SELECT` construct is triggered.

Regardless of the mode, when all of the triggers are false, the tool will execute the `OTHERWISE` clause, if it exists.

### Syntax

```
SELECT [selection_mode]
    WHEN trigger => statements
  [ WHEN trigger => statements ]
    .
  [ WHEN ANY => statements ]
    .
  [ WHEN trigger => statements ]
    .
  [ WHEN ANY => statements ]
    .
  [ OTHERWISE => statements ]
END SELECT ;
```



## Parameters

Parameter	Description
trigger	A Boolean expression.
selection_mode	Specifies the selection mode, which determines the way the statements are checked for possible execution. The possible values are as follows: <ul style="list-style-type: none"> <li>• <b>FIRST</b>—Only the first true trigger in the entire <code>SELECT</code> construct is executed; the rest are ignored, regardless of whether their triggers are <code>TRUE</code>. This is the default value.</li> <li>• <b>ANY</b>—If the selection mode is <code>ANY</code>, the statements are executed whenever their corresponding trigger is <code>TRUE</code>.</li> </ul>
statement	The DGL statements to invoke.

## Notes

Both the `WHEN ANY` and the `OTHERWISE` statements are optional.

The Documentor permits the nesting of `SELECT/WHEN` constructs.

## Example

Consider the following example, where `a`, `b`, and `c` are numeric variables:

```
SELECT ANY
  WHEN a = 5 => b := 10 ;
  WHEN a > b => b := 10 ;
  WHEN a = 0 => b := 0 ;
  WHEN ANY => write ('a may influence b') ;
  WHEN c = 5 => b := 5 ;
  WHEN c > b => b := 5 ;
  WHEN c = 0 => b := 0 ;
  WHEN ANY => write ('c may influence b') ;
  OTHERWISE => write ('b has not been changed') ;
END SELECT;
```

The execution is determined by the following:

- ◆ Each `WHEN` statement is triggered if its corresponding expression is evaluated to `TRUE`.
- ◆ The first `WHEN ANY` statement is triggered if `a` is equal to 5, greater than `b`, or equal to zero. The second `WHEN ANY` statement is triggered if at least one of the previous conditions is true with respect to the variable `c` instead of `a`.
- ◆ The `OTHERWISE` statement is triggered only if the value of `b` has not been changed within the `SELECT` statement's evaluation.

When processing a `WHEN ANY` statement, the Documentor only “looks back” to the previous `WHEN ANY` construct (if one exists). Therefore, in this example, if `a = 0` and

none of the tests of `c` were true, the `WRITE` statement's message "c may influence b" is not issued.

Using the same example, what would happen if the selection mode is `FIRST` instead of `ANY`, and the conditions `a > b` and `c = 5` are both true?

In this case, the assignment `b := 10` and the first write message (the corresponding `WHEN ANY` statement) are executed. The assignment of `b` to 5, along with its corresponding `WHEN ANY` statement are not done because `c = 5` is not the first true trigger.

The statements following the `=>` symbol in the `WHEN` constructs can be any valid DGL statements. You can even enter a `SELECT` construct at this point. This allows you to nest `SELECT` constructs. There is no limit to the depth of nested `SELECT` blocks.

### See Also

- ◆ [EXIT](#)
- ◆ [FOR/LOOP](#)
- ◆ [IF/THEN/ELSE](#)
- ◆ [STOP](#)
- ◆ [WHILE/LOOP](#)
- ◆ [Control Flow Statements](#)

# STOP

## Description

Stops execution of the template.

Typically, a specific condition is tested and the template is stopped if this condition has a value for which further processing is meaningless.

## Syntax

```
stop;
```

## Example

The following example checks whether the specified `system_name` is valid. If not (if an error has been detected), a message is issued to the dialog area and the template is stopped.

```
md := stm_r_md (system_name, status);  
  
IF (status <> stm_success) THEN  
    WRITE (dialog_area, 'Execution Stopped due  
        to error');  
STOP;  
END IF;
```

## See Also

- ◆ [EXIT](#)
- ◆ [FOR/LOOP](#)
- ◆ [IF/THEN/ELSE](#)
- ◆ [SELECT/WHEN](#)
- ◆ [WHILE/LOOP](#)
- ◆ [Control Flow Statements](#)

## TABLE

### Description

Creates a simple table to be included in your document. You specify the number of columns and their widths, and the information to be included in the table in the statement parameters.

### Syntax

```
stm_table_simple (title, columns, contents, page_width, page_height [, anchor]);
```

### Parameters

Parameter	Description						
title	The title of the table. The title is centered over the table.						
columns	A list of integers that specifies the width of each column, in number of characters. For example, {16} & {16} & {20} specifies a table having three columns, the first two columns being 16 characters wide and the last column being 20 characters wide.						
contents	<p>A list of strings that contain the information to be entered into each cell of the table. The Documentor fills the table horizontally, row by row, depending on how many columns were specified.</p> <p>For example, if there are three columns, and you specified the following contents:  {'Project Name'} &amp; {'Date'} &amp; {'Location'} &amp; {'Alpha'} &amp; {'April 2004'} &amp; {'Boston'}</p> <p>The resultant table would be:</p> <table border="1" data-bbox="786 1339 1297 1486"> <thead> <tr> <th>Project Name</th> <th>Date</th> <th>Location</th> </tr> </thead> <tbody> <tr> <td>Alpha</td> <td>April 2008</td> <td>Boston</td> </tr> </tbody> </table>	Project Name	Date	Location	Alpha	April 2008	Boston
Project Name	Date	Location					
Alpha	April 2008	Boston					
page_width	Specifies the width of the page, in inches. This parameter is relevant for Interleaf only—for other systems, specify 0.0 for this parameter.						

page\_height

Specifies the height of the page, in inches. This parameter is relevant for Interleaf only—for other systems, specify 0.0 for this parameter.

anchor

This parameter is relevant for Interleaf only. For formatters other than Interleaf, precede the table with your system’s “no fill” and “no adjust formatting” commands.

For Interleaf, this parameter is a character string that indicates where to place the table. The possible values are as follows:

A—At the anchor.

F—Following the anchor. This is the default value.

### Example 1

This example produces a table named “Table-1”, where the page is 6 inches wide and 9.5 inches long. The column widths are determined by the list held in the integer list variable `col_list`, and the cell contents are held in the string list variable `cell_list`.

```
stm_table_simple ('Table-1', col_list, cell_list,
6.0, 9.5);
```

### Example 2

The following example shows how to generate a table using function calls. Note that in the example, new values are repeatedly assigned to the `List_str` variable to build the table.

The first statement uses the `nroff` commands for no fill (`.nf`) and no adjust (`.na`). These commands cause the word processor to take the text “as-is.” Some word processors refer to this mode as *verbatim* or *literal*. The last statement uses the `nroff` commands `.fi` and `.ad` to return to fill and adjust modes.

```
WRITE ('\n.nf \n.na \n');
BEGIN
List_str:= {'ACTIVITY NAME'} & {'ID'} & {'LANGUAGE'};
act_list:=stm_r_ac_logical_desc_of_ac({act_chart},st);
FOR act IN act_list LOOP
List_str:=List_str & {stm_r_ac_name (act, st)};

attr_list:=stm_r_ac_attr_val (act, 'ID_NUMBER', st);

IF (st = stm_success) THEN
attr_val:=stm_list_first_element (attr_list, st);
List_str:=List_str & {attr_val};
ELSE
List_str:=List_str & { 'N/A'};
END IF;

attr_list:=stm_r_ac_attr_val(act, 'LANGUAGE', st);
IF (st = stm_success) THEN
attr_val:=stm_list_first_element (attr_list, st);
```

```
        List_str:=List_str & {attr_val};
    ELSE
        List_str:=List_str & {'N/A'};
    END IF;
END LOOP;

WRITE ('\n.nf\n.na\n');
title := 'Table CC1. Simple Table Example';
Col_list := {16} & {10} & {40};
stm_table_simple(title, Col_list, List_str, pg_w, pg_h,
'A');
WRITE ('\n.fi\n.ad\n');
END;
```

The formatted output is as follows:

ACTIVITY NAME	ID	LANGUAGE
FUNCTION1	AC1-A1.1	FORTRAN
FUNCTION2	AC1-A1.2	N/A
FUNCTION3	AC1-A1.5	ADA
FUNCTION4	AC1-A1.4	N/A

### See Also

- ◆ [EXECUTE](#)
- ◆ [INCLUDE](#)
- ◆ [REPORT](#)
- ◆ [VERBATIM](#)
- ◆ [WRITE](#)
- ◆ [READ Statement](#)

# TEMPLATE

## Description

Begins a template.

The `TEMPLATE` statement is the first statement in the template. It assigns an identifying name to the template. This name is used for internal documentation purposes only and does not have to correspond to the name you use to designate the template in the Create Template form.

## Syntax

```
TEMPLATE template_name;
```

## Parameters

Parameter	Description
<code>template_name</code>	The name of the template. This name is for internal documentation purposes only.

## Example

```
TEMPLATE document_574B;
```

## See Also

- ◆ [BEGIN](#)
- ◆ [COMMENT](#)
- ◆ [END](#)
- ◆ [PROCEDURE](#)
- ◆ [SEGMENT](#)
- ◆ [Structure Statements](#)

## VARIABLE

### Description

Declares variables, which are identifiers whose values can be changed in other DGL statements.

Variable values can be global when included in the initiation section, or local to a segment.

The keyword `VARIABLE` appears only once in the declaration section, before the data-type assignments for variables. Each data-type statement can be followed by as many identifiers of the same type as you want to define. For example:

```
VARIABLE STRING act_name, act_syn, act_desc;
```

Similarly, as many type statements as you want to define can follow the `VARIABLE` keyword. For example:

```
VARIABLE
  string activity_name;
  float a := 3.243;
  activity act_id;
```

Value assignments are optional. If they are assigned, they represent the default value of the variable at the first generation of a particular document. The value can be any expression that does not contain other variables or parameters.

Variables that are declared as Statemate elements and list of items cannot be assigned initial values.

### Syntax

```
VARIABLE
type identifier [:= value] [, identifier [:= value],...;
[type identifier [:= value],...;]
.
.
.
```

### Parameters

Parameter	Description
type	The parameter type
identifier	The name of the variable
variable	The initial value of the variable



### Notes

The assignment of initial values is optional. An initial value is an expression that cannot contain other variables. Initial values are allowed only for integer, float, and string.

### Example

```
VARIABLE
  INTEGER      status := 0;
  STATE        statechart;
  LIST OF STATE sub_list;
  STATE        sub_state, parent;
  STRING       name;
```

### See Also

- ◆ [ASSIGNMENT](#)
- ◆ [CONSTANT](#)
- ◆ [PARAMETER](#)
- ◆ [TEMPLATE Statement](#)

## VERBATIM

### Description

Passes text literally to an output file.

When the verbatim symbols `/@` and `@/` frame text in the template file, the text is passed literally (without interpretation) to the output segment file. Comments inside the frame, rather than being ignored, are also passed literally. The end-of-statement character, “;”, is not required following the concluding verbatim symbol.

Verbatim statements can be used to pass the following to the output file:

- ◆ Formatting commands applicable to a specific formatter.
- ◆ Short text passages such as titles, opening remarks, and so on. Despite the absence of any length restriction on verbatim text, longer text passages are usually passed using the `INCLUDE` file statement.

### Syntax

```
/@ verbatim_text @/
```

### Parameters

Parameter	Description
<code>verbatim_text</code>	One or more lines of printable text characters

### Example

Consider the following verbatim section:

```
SEGMENT section1;
BEGIN          /@
.title ACTIVITY-SPEC
.skip 2
.center; AN ACTIVITY SPECIFICATION
-- This section will describe the
-- purpose of the activities.
@/
-- this line will not appear in the output
END;
```

When the template is executed, the resulting output is as follows:

```
title ACTIVITY-SPEC
.skip 2
.center; AN ACTIVITY SPECIFICATION
-- This section will describe the
-- purpose of the activities.
```

### See Also

- ◆ [EXECUTE](#)
- ◆ [INCLUDE](#)
- ◆ [REPORT](#)
- ◆ [TABLE](#)
- ◆ [WRITE](#)
- ◆ [READ Statement](#)

## WHILE/LOOP

### Description

Provides iterative execution of DGL statements. The execution of statements is determined by evaluation of the `boolean_expression`.

The statements are executed until the expression evaluates to false. For example:

```
WHILE  a > b  LOOP
      .
      b := b + k;
      .
      .
END LOOP;
```

The statements between the keywords `LOOP` and `END LOOP` are executed as long as `a` is greater than `b`.

Assume that `b` changes its value inside the loop and in one of the iterations the expression `a > b` becomes false. In the next iteration, the expression is examined and, because `a > b` is now `FALSE`, the execution of template statements continues with the first statement after the `END LOOP`.

### Syntax

```
WHILE boolean_expression LOOP
  statements
END LOOP;
```

### Parameters

Parameter	Description
<code>boolean_expression</code>	The Boolean expression used to test the conditions
<code>statements</code>	One or more DGL statements

### Notes

The value of `boolean_expression` must be altered within the loop, or an `EXIT` statement must be executed in order to terminate the loop.

### Example

```
WHILE count < 25 LOOP
  num_control := alpha / 5;
  WRITE (count, num_control, '\n';
  alpha := synch +7;
  count := count + 1;
END LOOP;
```

### See Also

- ◆ [EXIT](#)
- ◆ [FOR/LOOP](#)
- ◆ [IF/THEN/ELSE](#)
- ◆ [SELECT/WHEN](#)
- ◆ [STOP](#)
- ◆ [Control Flow Statements](#)

# WRITE

## Description

Writes expression values to any of the following:

- ◆ The document output segment
- ◆ Another file
- ◆ The dialog area of the tool window

You can write a numeric or string expression that is evaluated in the template, or a literal piece of text. In addition, the `WRITE` statement can be used to write information retrieved from the database, such as element names.

The `WRITE` statement is commonly used to write lines that include text (string literals) together with expression values. For example:

```
WRITE ('NAME:', di_name);
```

- ◆ This results in `KUKU` being written in the output segment file `NAME`, where `KUKU` is a value of `di_name`.

Note the following:

- There can be more than one write expression. When there are multiple expressions, they are separated by commas.
- For lines of pure text, it is better to use the `Verbatim` statement.
- Literal strings can include the formatting characters

<code>\n</code>	New line
<code>\t</code>	Tab

- ◆ For example, the following call writes the value of `alpha` at the beginning of the next line in the output segment file:

```
WRITE ('\n', alpha);
```

- ◆ Optionally, you can specify the minimum number of characters to be written in the output file using the following syntax:

```
expression : num
```

- ◆ In this syntax, `expression` can be either a numeric or a string expression, and `num` is an integer constant or integer expression that represents the minimum number of characters that `expression` will occupy. `expression` and `num` can involve operands, operations, and function calls.

- ◆ For example:

```
WRITE (act_name: 10, ' ', act_synonym);
```

- ◆ This call results in the string value for `act_name` being written in the output file to a length of at least 10 characters; if the name has less than this number, blanks are added to achieve the specified string length. For example:

```
COMP      , SET
```

- ◆ In this example, spaces have been added to “COMP” to give it a length of 10 characters.
- ◆ The use of `num` determines the minimum number of characters to be written in the output file, as follows:
  - ◆ For a string, the length of the string is the minimum number of output characters. When specified, and where `num` is greater than the string length, blanks are padded to the right of the string to achieve a total string length of `num`.
  - ◆ For an integer, when `num` is specified, and where `num` is greater than the number of digits in the integer, blanks are padded to the left of the number to achieve a total output length of `num`.
  - ◆ For a real number, when `num` is *not* specified, the value is output to no more than 8 decimal places. Thereafter, the number is automatically rounded. When specified, and where `num` is greater than the digits output according to the default, blanks are padded to the left of the number to achieve a total output length of `num`. Where `num` is less than the digits output according to the default, the decimal portion of the number might be rounded to arrive at a specified output length of `num`. However, in no case will the integer portion of the real number be truncated.

## Syntax

```
WRITE ([f1,] write_expression,...);
```

## Parameters

Parameter	Description
<code>f1</code>	An identifier of type <code>FILE</code> , previously assigned by the <code>OPEN</code> statement, to which to write the specified string. Using the <code>WRITE</code> statement to write to a file or to the dialog area is particularly useful if you want to write messages (error messages, run-time messages, and so on). When writing to a file or to the dialog area, you must include the <code>f1</code> identifier. In such cases, you must also precede the <code>WRITE</code> statement with an <code>OPEN</code> statement.
<code>write_expression</code>	The expression value to write to the document segment, file, or dialog area.

### Example

```
WRITE ('Name:':8, name, '\n', 'Value:':8, v);
```

Assume that `name` contains “Xfactor” and `v` is an integer that equals 5105. This call writes the following lines to the output file:

```
Name:   Xfactor
Value:  5105
```

**Note:** The `WRITE` command cannot access a `list` variable directly; if you attempt to write a variable that refers to a list, the Documentor displays an error message. To output a list, use a control flow construct such as a loop, writing one list item at a time.

### Using `WRITE` to Produce Messages

You can use a `WRITE` statement to write information to a file or to the dialog area, instead of to the document itself. For this, you must first open a file in `OUTPUT` mode using the `OPEN` statement (refer to [OPEN](#)).

To write a string message to a file, use the following syntax:

```
WRITE (f1, write_expression);
```

In the syntax, `f1` is a pointer to the file to which you want to write the messages.

### See Also

- ◆ [EXECUTE](#)
- ◆ [INCLUDE](#)
- ◆ [REPORT](#)
- ◆ [TABLE](#)
- ◆ [VERBATIM](#)
- ◆ [READ Statement](#)



# Single-Element Functions

---

This section documents the single-element extraction functions. For each function, the following information is provided:

- ◆ Return value type
- ◆ The elements for which it is relevant
- ◆ Description
- ◆ Syntax
- ◆ Arguments
- ◆ Status codes

The two characters *xx* in the function names denote element type abbreviations. Refer to [Element Type Abbreviations](#) for the list of element abbreviations.

Single-element functions provide information about discrete Statemate elements in the database.

Using single-element functions, you can retrieve any information attached to a particular element. This information is usually entered into the database via forms. Data extraction is a multi-stage procedure. Generally, when working with a Statemate element, you know the element's name (path name). You might want to know more about such an element, such as the element's synonym or what attributes are defined in the element's form. Such information can be retrieved using the single-element functions.

The retrieval process is as follows:

1. Specify the element name or synonym. Receive the element ID.
2. Specify the ID and the information requested. Receive the extracted information
3. Use the extracted information.

The element ID is an internal representation that Statemate uses to identify each element. You do not see the ID; you extract it from the database using one function and pass it along to another function that processes your information request.

The vertical ellipses in the diagram indicate that multiple functions can be called in succession for the same element. For example, you can call a function to return an element's synonym, then immediately call a different function to return the contents of the same element's description field.

## Calling Single-Element Functions

As shown in the diagram, extracting information from your database is at least a two-stage process.

- ♦ **Stage 1**—Pass the element name or synonym as a function argument to get the element ID. The function calling sequence is as follows:

```
stm_r_xx (name, status)
```

In this syntax:

- **stm\_r\_**—Designates the function as a Statemate database retrieval function.
- **xx**—The two-character element type abbreviation.
- **name**—The name of the element for which information is requested. The input argument name contains the name (path name) or synonym that uniquely identifies the element of interest. The name can be a variable or a literal string (enclosed by single apostrophe marks).
- **status**—The return function status code.

For example:

```
stm_r_st('S1', status)
```

This function call returns the ID for state `S1`. The value returned by the function is a Statemate element of the type specified by `xx`. In this example, the value returned by the function is of type `STATE`.

- ♦ **Stage 2**—Pass the element ID as a function argument to get the information requested. The function calling sequence is as follows:

```
stm_r_xx_info (inarg, ..., status)
```

or

```
stm_r_info (inarg, ..., status)
```

In this syntax:

- **stm\_r\_**—Designates the function as a Statemate database retrieval function.
- **xx**—The two-character element type abbreviation. Note that in some functions, these two characters are omitted.
- **info**—The type of information to be extracted from the database.
- **inarg**—The required input arguments.
- **status**—The return function status code.

For example:

```
stm_r_ac_description (a, status)
or
stm_r_description (a, status)
```

This function call retrieves the contents of the **Description** field for the activity whose ID is contained in the variable a.

There is one function whose calling sequence differs from that shown above. This function, `stm_r_element_type`, receives an element ID as input and returns the element type. The function returns an enumerated type value of the form “`stm_state`”, “`stm_activity`”, and so on.

## Single-Element Function Input Arguments

The following table lists the input arguments for single-element functions.

Argument	Function	DGL Data Type
<b>name</b>	The name of the StateMate element. It can be an element name, path name, or synonym, including the chart name, (for example, K:L.M).	String
<b>element ID</b>	The value that StateMate uses to identify each element in the database. StateMate assigns a unique ID to every element.	StateMate element (ELEMENT, STATE, EVENT, and so on). For example, if you call a function that operates on events, the element ID must be defined to be of type EVENT (or ELEMENT).
<b>attribute name</b>	Name of an attribute defined in the form for a StateMate element (in the attribute field).	String
<b>begin keyword</b>	The string of text appearing in the long description attached to the specified StateMate element. This string represents the beginning of the portion of the long description that you want to extract from the database.	String
<b>end keyword</b>	The string of text appearing in the long description attached to the specified element. This string represents the end of the portion of the long description that you want to extract from the database.	String
<b>filename</b>	The path name of a system file. Long descriptions (and portions thereof) are copied to the system file specified in this argument.  If this is an empty string ( ' ' - two contiguous apostrophes), the Documentor creates a temporary file where it stores the text. The name of this temporary file is returned by the function.	String

## Examples of Single-Element Function Calls

This section shows how to use single-element function calls to perform common tasks.

### Single-Element Function Example 1

Suppose you want to find the synonym and the short description for a state `S1`, as they appear in the state's form. Assume that the path name `SSS.S1` uniquely identifies the state. Include the following code in your Documentor template:

```
VARIABLE
  STATE      state_id;
  STRING     state_desc, state_syn;
  INTEGER    status;
  .
state_id := stm_r_st ('SSS.S1', status);
state_syn := stm_r_st_synonym (state_id, status);
state_desc := stm_r_st_description (state_id, status);
  .
```

This example uses two consecutive function calls to extract first the synonym, then the short description of the same element. The assigned variable and the function return value must have compatible data types. Therefore, `state_id` is declared as `STATE`. (`state_id` could also be declared as `ELEMENT`.)

### Single-Element Function Example 2

The following example shows functions that return enumerated type values.

```
VARIABLE
  STATE      state_id;
  STRING     state_name, state_type;
  INTEGER    st_type;
  INTEGER    status;
  .
state_id := stm_r_st (state_name, status);
st_type := stm_r_st_type (state_id, status);
  SELECT
    WHEN st_type = stm_st_or =>
      state_type := 'or';
    WHEN st_type = stm_st_and =>
      state_type := 'and';
  .
  END SELECT;
WRITE ('The state ', state_name, ' is of type ',
state_type);
```

This example queries the database to determine the type of the state in `state_name`. When the type is determined, the name and type of the state are printed out.

### Single-Element Function Example 3

The following example shows how to include a portion of the long description of the state `S1` in a document. The section extracted is the text appearing between the strings `'!BEGIN'` and `'!END'`.

```
VARIABLE
  STATE      state_id;
  STRING     descr_file;
  .
state_id := stm_r_st ('S1', status);
descr_file := stm_r_st_keyword (state_id, '!BEGIN',
                              '!END', '', status);
INCLUDE (descr_file);
.
.
.
```

The fourth input parameter (the empty string) of the function `stm_r_st_keyword` determines the name of the file to which the extracted text is written. If the string is empty, as in this case, the Documentor creates a temporary file where it stores the text. The name of this temporary file is returned by the function. Using the `INCLUDE` statement, you write the text to the generated document.

## List of Functions

As previously mentioned, the extraction functions take the form `stm_r_<element_type><task>`. For example, `stm_r_uc_attr_name` returns the names of attributes associated with the specified use case. Because this function can retrieve values for other elements besides use cases, it is denoted as `stm_r_xx_attr_name`. This function would be included in the A section (for `attr_name`).

The functions are as follows:

Function	Description
<a href="#"><u>stm_r_sb_action_lang</u></a>	Returns the action language of the specified subroutine.
<a href="#"><u>stm_r_sb_action_lang_expression</u></a>	Returns the action language expression of the specified subroutine.
<a href="#"><u>stm_r_sb_action_lang_local_data</u></a>	Returns the action language local data associated with the specified subroutine.
<a href="#"><u>stm_r_actual_parameter_exp</u></a>	Returns the actual binding of the formal parameter name in the specified instance chart or component.
<a href="#"><u>stm_r_actual_parameter_type</u></a>	Returns the type of the formal parameter name in the specified instance chart or component.
<a href="#"><u>stm_r_elem_in_ddb_list</u></a>	Searches for the specified element in the list created by the properties browser.
<a href="#"><u>stm_r_sb_ada_user_code</u></a>	Returns the Ada code that was manually written by the user for the specified subroutine.

## Single-Element Functions

---

<a href="#"><u>stm r sb ansi c user code</u></a>	Returns the ANSI C code that was manually written by the user for the specified subroutine.
<a href="#"><u>stm r xx</u></a>	Retrieves the element ID of the specified element.
<a href="#"><u>stm r xx array lindex</u></a>	Returns the left index of an element array.
<a href="#"><u>stm r xx bit array rindex</u></a>	Returns the right index of an element array.
<a href="#"><u>stm r xx attr enforced</u></a>	Returns the enforced attributes specified by <code>attr_name</code> .
<a href="#"><u>stm r xx attr name</u></a>	Returns the names of attributes associated with the specified element.
<a href="#"><u>stm r xx attr val</u></a>	Returns the attribute values associated with a particular attribute name for the specified element.
<a href="#"><u>stm r xx bit array lindex</u></a>	Returns the left index of a bit array.
<a href="#"><u>stm r xx bit array rindex</u></a>	Returns the right index of a bit array.
<a href="#"><u>stm r xx cbk binding</u></a>	Retrieves the callback binding for the specified element.
<a href="#"><u>stm r xx cbk binding enable</u></a>	Retrieves the enabled callback bindings for the specified element.
<a href="#"><u>stm r xx cbk binding expression</u></a>	Retrieves the callback binding expression for the specified element.
<a href="#"><u>stm r xx cbk binding expression hyper</u></a>	Retrieves the callback binding expressions.
<a href="#"><u>stm r tt cell</u></a>	Returns the contents of the specified cell in the given truth table.
<a href="#"><u>stm r tt cell type</u></a>	Returns the data-type of the specified cell in the given truth table.
<a href="#"><u>stm r changes log</u></a>	Documents all the changes made to the specified charts in a log file.
<a href="#"><u>stm r xx chart</u></a>	Returns the chart ID for the specified element.
<a href="#"><u>stm r xx combinational</u></a>	Returns a list of strings. Each element of this list holds one combinational assignment, which is connected to the specified element.
<a href="#"><u>stm r sb connected chart</u></a>	Returns the ID of the procedural statechart connected to the specified subroutine.
<a href="#"><u>stm r xx containing fields</u></a>	Returns a list of union or record elements that contain fields.
<a href="#"><u>stm r ch creation date</u></a>	Returns the date (as a string) on which the specified chart was created.
<a href="#"><u>stm r ch creator</u></a>	Returns the name of the Rhapsody user who created the specified chart.
<a href="#"><u>stm r xx data type</u></a>	Returns the element subtype, including its data type and data structure.
<a href="#"><u>stm r xx definition type</u></a>	Returns the definition type of the specified textual element.
<a href="#"><u>stm r xx desc file</u></a>	Returns the description file for the specified element.
<a href="#"><u>stm r xx description</u></a>	Returns the short description of the specified element.
<a href="#"><u>stm r design attr</u></a>	Retrieves the design attributes of an element in a form of a list of strings.
<a href="#"><u>stm r xx displayed name</u></a>	Returns the name of the chart as it appears in the graphic editor where the specified element is located.
<a href="#"><u>stm r ddb list names</u></a>	Returns the names of the lists created by the properties browser.

<a href="#">stm r element type</a>	Returns the element type of the specified element.
<a href="#">stm r xx expr hyper</a>	Returns the definition expression of the specified element found in the <b>Definition</b> field of the element's form, including hyperlinks to referenced elements.
<a href="#">stm r xx expression</a>	Returns the definition expression of the specified element found in the <b>Definition</b> field of the element's form.
<a href="#">stm r xx ext link</a>	Retrieves the external file link for the specified element.
<a href="#">stm r uc ext point def</a>	Retrieves the extension point definitions for the specified use case diagram.
<a href="#">stm r formal parameter names</a>	Retrieves the list of names of formal parameters that appear in the bindings of instance boxes and components.
<a href="#">stm r global interface report</a>	Creates a global interface report from the specified input list.
<a href="#">stm r sb global data</a>	Returns the global data associated with the specified subroutine.
<a href="#">stm r hyper key</a>	Retrieves the unique key for the specified element.
<a href="#">stm r md implementation</a>	Retrieves the implementation type for the specified module.
<a href="#">stm r included gds</a>	Returns the list of global definition sets contained in the specified chart.
<a href="#">stm r msg included in ord insig</a>	Returns a list of messages that are bounded by an order-insignificant element.
<a href="#">stm r cd info</a>	Retrieves the description of the specified continuous chart.
<a href="#">stm r inherited gds</a>	Retrieves the list of global definition sets that are "inherited" (included indirectly) by the specified chart.
<a href="#">stm r xx instance name</a>	Retrieves the name of an instance as it appears in the chart for the specified hierarchical Rhapsody element.
<a href="#">stm r xx keyword</a>	Returns a portion of the element's long description.
<a href="#">stm r sb kr c user code</a>	Returns the K&R C code that was manually written by the user for the specified subroutine.
<a href="#">stm r xx labels</a>	Returns a list of strings that consists of all the labels of the specified compound transition.
<a href="#">stm r xx labels hyper</a>	Returns a list of strings of message or transition labels, with hyperlinks to referenced elements.
<a href="#">stm r local interface report</a>	Creates a local interface report from the specified input list.
<a href="#">stm r xx longdes</a>	Returns the long description attached to the specified element.
<a href="#">stm r xx max val</a>	Returns the maximum value of the specified element.
<a href="#">stm r xx min val</a>	Returns the minimum value of the specified element.
<a href="#">stm r xx mini spec</a>	Returns a string with mini-spec reactions or actions.
<a href="#">stm r ac mini spec hyper</a>	Returns a string with the mini-spec, including hyperlinks to referenced elements.
<a href="#">stm r xx mode</a>	Returns the parameter or router mode.
<a href="#">stm r ch modification date</a>	Returns the date on which the version of the chart in the workarea was saved to the database.
<a href="#">stm r xx name</a>	Returns the element name.

## Single-Element Functions

---

<a href="#"><u>stm r next msg</u></a>	Returns the message after (in time) the decomposed sequence diagram.
<a href="#"><u>stm r xx note</u></a>	Returns the notes in the specified element.
<a href="#"><u>stm r tt num of col</u></a>	Retrieves the number of columns in the specified truth table.
<a href="#"><u>stm r tt num of in</u></a>	Retrieves the number of input columns in the specified truth table.
<a href="#"><u>stm r tt num of out</u></a>	Retrieves the number of output columns in the specified truth table.
<a href="#"><u>stm r tt num of row</u></a>	Retrieves the number of rows in the specified truth table.
<a href="#"><u>stm r uc num of scen</u></a>	Retrieves the number of scenarios for the specified use case.
<a href="#"><u>stm r xx number of bits</u></a>	Returns the number of bits in the specified element.
<a href="#"><u>stm r xx of enum type</u></a>	Returns the enumeration type ID (a user-defined type) for the specified element.
<a href="#"><u>stm r xx of enum type name type</u></a>	Returns the enumerated name and type for the specified element.
<a href="#"><u>stm r ord insig defined in ch</u></a>	Returns the origin of the specified requirement record.
<a href="#"><u>stm r parameter binding</u></a>	Returns the parameter expression from the generic charts and components.
<a href="#"><u>stm r parameter mode</u></a>	Retrieves the parameter mode, including subroutine parameters and the parameters of generic charts and components.
<a href="#"><u>stm r sb parameters</u></a>	Returns the parameters of the specified subroutine.
<a href="#"><u>stm r en parent</u></a>	Returns the local data of the procedural statechart implemented by the specified subroutine.
<a href="#"><u>stm r previous msg</u></a>	Returns the message previous (in time) to the decomposed sequence diagram.
<a href="#"><u>stm r sb proc sch local data</u></a>	Retrieves the local data of the procedural statechart implemented by the specified subroutine.
<a href="#"><u>stm r md purpose</u></a>	Retrieves the purpose of the module.
<a href="#"><u>stm r xx reactions</u></a>	Returns the static reactions of the specified state.
<a href="#"><u>stm r sb return type</u></a>	Retrieves the subroutine's return type.
<a href="#"><u>stm r sb return user type</u></a>	Returns the user-defined type ID returned by the specified subroutine.
<a href="#"><u>stm r sb return user type name</u></a>	Retrieves the subroutine's return user type and name type.
<a href="#"><u>stm r tt row</u></a>	Retrieves the values in the specified row in the truth table.
<a href="#"><u>stm r uc scen</u></a>	Retrieves the scenario, based on the specified index number.
<a href="#"><u>stm r uc scen attr name</u></a>	Returns the names of attributes associated with the specified use case scenario.
<a href="#"><u>stm r uc scen attr val</u></a>	Retrieves attribute values associated with a particular attribute name for the specified use case scenario.
<a href="#"><u>stm r sd scope</u></a>	Retrieves the scope of the specified sequence diagram.
<a href="#"><u>stm r xx select implementation</u></a>	Returns the implementation type of the specified element.
<a href="#"><u>stm r st static reactions</u></a>	Returns the static reactions defined for the specified state element.



<a href="#"><u>stm r st static reactions hyper</u></a>	Returns a string with the static reactions, including hyperlinks to referenced elements.
<a href="#"><u>stm r xx string length</u></a>	Returns the string length of the specified element.
<a href="#"><u>stm r xx structure type</u></a>	Returns the structure or type of the specified textual element.
<a href="#"><u>stm r ac subroutine bind</u></a>	Returns the subroutine binding connected to the specified activity.
<a href="#"><u>stm r ac subroutine bind enable</u></a>	Determines whether the subroutine bound to the specified activity is enabled or disabled.
<a href="#"><u>stm r ac subroutine bind expr</u></a>	Returns the subroutine binding expression connected to the specified activity.
<a href="#"><u>stm r xx synonym</u></a>	Returns the synonym of the specified element.
<a href="#"><u>stm r ac termination</u></a>	Returns the activity termination type specified in the activity's form.
<a href="#"><u>stm r xx truth table</u></a>	Returns the elements that are implemented as truth tables.
<a href="#"><u>stm r xx truth table expression</u></a>	Returns the truth table expression for all the named elements.
<a href="#"><u>stm r sb truth table local data</u></a>	Returns a list of local data elements defined in truth tables that are related to the input subroutine.
<a href="#"><u>stm r xx type</u></a>	Returns the element subtypes for the specified element.
<a href="#"><u>stm r xx type expression</u></a>	Returns the type expression for the specified element.
<a href="#"><u>stm r xx uniquename</u></a>	Returns the unique path name for the specified element.
<a href="#"><u>stm r ch usage type</u></a>	Returns the usage type for the specified chart.
<a href="#"><u>stm r xx user type</u></a>	Returns the user-defined type ID referenced by the specified element.
<a href="#"><u>stm r xx user type name type</u></a>	Returns the name type of the user-defined type referenced by the specified element.
<a href="#"><u>stm r ch version</u></a>	Returns the version of the specified chart.
<a href="#"><u>stm r gds visibility mode</u></a>	Returns the visibility mode for the specified global definition set (GDS).
<a href="#"><u>stm r msg where tc begins</u></a>	Returns the message where the timing constraint begins.
<a href="#"><u>stm r msg where tc ends</u></a>	Returns the message where the timing constraint ends.
<a href="#"><u>stm r xx chart</u></a>	Returns the chart ID for the specified element.
<a href="#"><u>stm r xx longdes</u></a>	Retrieves the long description attached to the specified element.
<a href="#"><u>stm r xx attr name</u></a>	Returns the names of attributes associated with the specified element. Attributes are associated with elements via element forms.
<a href="#"><u>stm r xx attr val</u></a>	Retrieves attribute values associated with a particular attribute name for the specified element.
<a href="#"><u>stm r sb connected statechart</u></a>	Returns the enforced attributes specified by attr_name.
<a href="#"><u>stm r sb connected statechart</u></a>	Returns the ID of the procedural Statechart connected to the specified subroutine.
<a href="#"><u>stm r sb connected flowchart</u></a>	Returns the ID of the Flowchart connected to the specified subroutine.
<a href="#"><u>stm r sb proc fch local data</u></a>	Retrieves the local data of the Flowchart implemented by the specified subroutine.

## Single-Element Functions

---

<a href="#"><u>stm r xx des attr val</u></a>	Retrieves attribute values associated with a particular attribute name for the specified element.
<a href="#"><u>stm r xx des attr name</u></a>	Returns the names of attributes associated with the specified element. Attributes are associated with elements via element forms.
<a href="#"><u>stm r tt cell hyper</u></a>	Retrieves the contents of the specified cell in the given truth table, including hyperlinks to referenced elements.
<a href="#"><u>stm r xx default val</u></a>	Returns a list of strings that represents a row in the truth table, including hyperlinks to referenced elements. Each string in the list includes the text in the truth table cell. The row's index range is [0..num_of_rows-1]. Row 0 returns the list of table header strings.
<a href="#"><u>stm r xx default val</u></a>	Returns the default value associated with the specified element.
<a href="#"><u>stm r component param binding</u></a>	Returns the value bound to the formal parameter of the specified component's instance
<a href="#"><u>stm r component param mode</u></a>	Returns the mode of the formal parameter of the specified component's instance.
<a href="#"><u>stm r stubs names</u></a>	Returns a list of stub names of the specified component instance.
<a href="#"><u>stm r information stub names</u></a>	Returns a list of stub names flowing through an info-flow of the specified component instance.

**stm\_r\_xx****Function type:** Statemate element**For elements:**

action	an
activity	ac
actor	actor
boundary box	bb
chart	ch
condition	co
data-item	di
data-store	ds
enumerated value	en
event	ev
field	fd
function	fn
information-flow	if
lifeline	ll
module	md
router	router
state	st
subroutine	sb
subroutine parameter	sp
use case	uc
user_defined_type	dt

**Description**

Retrieves the element ID of the specified element. This ID is an internal representation that Statemate uses to identify each element in the database. Because Statemate requires the ID to locate elements, this function is very often the first one called when using database extraction functions.

**Syntax**

```
stm_r_xx (name, status)
```

### Arguments

Argument	Input/Output	Type	Description
name	In	STRING	A StateMate element name or synonym. Note the following: <ul style="list-style-type: none"><li>• This can be an element name (path name) or synonym. Hierarchical elements must be identified uniquely by specifying a unique path name.</li><li>• The name can include the chart name (for example, A:B).</li><li>• The name is not case-sensitive.</li></ul>
status	Out	INTEGER	The function status code. The data type of the return value must be declared to be either a Rhapsody element (STATE, EVENT, and so on) or ELEMENT.

### Status Codes

- ◆ stm\_success
- ◆ stm\_illegal\_address
- ◆ stm\_illegal\_name
- ◆ stm\_name\_not\_found
- ◆ stm\_name\_not\_unique

### Example

Identify the ID of an event EV1. Once the ID has been determined, you can use it to retrieve information about EV1 from the database, as follows:

```
VARIABLE
    EVENT      ev_id;
    INTEGER    status;
    STRING     synonym;
    .
    .
    .
ev_id := stm_r_ev ('EV1', status);
IF status = stm_success
    THEN
        synonym:= stm_r_ev_synonym (ev_id, status);
    .
    .
    .
```

The ID for EV1 is assigned to the variable `ev_id`. Note that `ev_id` is declared to be of type `EVENT`.

## stm\_r\_sb\_action\_lang

**Function type:** LIST OF STRING

**For elements:**

subroutine	sb
------------	----

### Description

Retrieves the action language of the specified subroutine.

### Syntax

```
stm_r_sb_action_lang (sb_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
sb_id	In	Stemate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_id\_not\_found
- ◆ stm\_missing\_stemate\_action\_lang
- ◆ stm\_unresolved

## stm\_r\_sb\_action\_lang\_expression

**Function type:** `STRING`

**For elements:**

subroutine	sb
------------	----

### Description

Retrieves the action language expression of the specified subroutine.

### Syntax

```
stm_r_sb_action_lang_expression (sb_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
sb_id	In	Statestate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_missing_statestate_action_lang`
- ◆ `stm_unresolved`

## stm\_r\_sb\_action\_lang\_local\_data

**Function type:** LIST OF STATEMATE ELEMENTS

**For elements:**

subroutine	sb
------------	----

### Description

Retrieves the action language local data associated with the specified subroutine.

### Syntax

```
stm_r_sb_action_lang_local_data (sb_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
sb_id	In	Statemate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_missing\_local\_data
- ◆ stm\_id\_not\_found
- ◆ stm\_id\_out\_of\_range

## stm\_r\_actual\_parameter\_exp

**Function type:** INTEGER

**For elements:**

activity	ac
condition	co
data-item	di
event	ev

### Description

Returns the actual binding of the formal parameter name in the specified instance chart or component.

### Syntax

```
stm_r_actual_parameter_exp (xx_inst_boxid, formal_param_name, status)
```

### Arguments

Argument	Input/Output	Type	Description
inst_boxid	In	Statestate element	The element ID.
formal_param_name	In	STRING	The formal parameter name. If this is a data-element (from the information stub matrix in the DDE), the function returns the corresponding data-element. If this argument is the stub's name, the function returns the information flowing on the arrow connected to that stub.
status	Out	int	The function status code.

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_name\_not\_found



## stm\_r\_actual\_parameter\_type

**Function type:** `STRING`

**For elements:**

activity	ac
condition	co
data-item	di
event	ev

### Description

Returns the type of the formal parameter name in the specified instance chart or component.

**Note:** If there is an information-flow stub, the function returns `stm_information_flow`.

### Syntax

```
stm_r_actual_parameter_type (inst_boxid, formal_param_name, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>inst_boxid</code>	In	Statestate element	The element ID
<code>formal_param_name</code>	In	STRING	The formal parameter name
<code>status</code>	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_name_not_found`

## stm\_r\_elem\_in\_ddb\_list

**Function type:** LIST OF ELEMENT `stm_list`

### Description

Searches for the specified element in the list created by the properties browser.

### Syntax

```
stm_r_elem_in_ddb_list (list_name, &status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>list_name</code>	In	<code>char * STRING</code>	The input list of elements
<code>status</code>	Out	<code>int INTEGER</code>	Function status code

### Status Codes

- ◆ `stm_no_such_list`
- ◆ `stm_success`

## stm\_r\_sb\_ada\_user\_code

**Function Type:** LIST OF STRING

**For elements:**

subroutine	sb
------------	----

### Description

Returns the Ada code that was manually written for the specified subroutine.

### Syntax

```
stm_r_ada_user_code (sb_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
sb_id	In	Stemate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_missing\_user\_code

## stm\_r\_sb\_ansi\_c\_user\_code

**Function type:** LIST OF STRING

**For elements:**

subroutine	sb
------------	----

### Description

Returns the ANSI C code that was manually written for the specified subroutine.

### Syntax

```
stm_r_sb_ansi_c_user_code (sb_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
sb_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_missing\_user\_code

## stm\_r\_st\_combinationals

**Function type:** LIST OF STRING

**For elements:**

activity	ac
chart	ch

### Description

Returns a list of strings. Each element of the list holds one combinational assignment, which is connected to the specified element.

You can call this function without indicating the specific element type, as follows:

```
stm_r_combinationals (id, status)
```

### Syntax

```
stm_r_xx_combinationals (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	Stateate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_error_in_file`
- ◆ `stm_missing_field`
- ◆ `stm_missing_label`
- ◆ `stm_missing_name`
- ◆ `stm_file_not_found`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_illegal_parameter`

## stm\_r\_xx\_array\_index

**Function type:** `STRING`

**For elements:**

condition	co
data-item	di
event	ev
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

### Description

Returns the left index of an element array.

You can call this function without indicating the specific element type, as follows:

```
stm_r_array_index (id, status)
```

### Syntax

```
stm_r_xx_array_index (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>xx_id</code>	In	StateMate element	The element ID
<code>status</code>	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`

## stm\_r\_xx\_array\_rindex

**Function type:** `STRING`

**For elements:**

condition	co
data-item	di
event	ev
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

### Description

Returns the right index of an element array.

You can call this function without indicating the specific element type, as follows:

```
stm_r_array_rindex (id, status)
```

### Syntax

```
stm_r_xx_array_rindex (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>xx_id</code>	In	StateMate element	The element ID
<code>status</code>	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`

## stm\_r\_xx\_attr\_enforced

**Function type:** BOOLEAN

**For elements:**

action	an
activity	ac
actor	actor
boundary box	bb
chart	ch
condition	co
data-item	di
data-store	ds
event	ev
field	fd
information-flow	if
lifeline	ll
module	md
router	router
state	st
subroutine	sb
use case	uc
user-defined type	dt

### Description

Returns the enforced attributes specified by `attr_name`.

You can call this function without indicating the specific type, as follows:

```
stm_r_attr_enforced (id, attr_name, attr_val, status)
```

### Syntax

```
stm_r_xx_attr_enforced (xx_id, attr_name, attr_val, status)
```



### Arguments

Argument	Input/Output	Type	Description
xx_id	In	State element	The element ID.
attr_name	In	STRING	The attribute name.
attr_val	In	STRING	The attribute value.
status	Out	INTEGER	The function status code. If no attributes exist for the specified element, status receives the value <code>stm_attribute_name_not_found</code> .

### Status Codes

- ◆ `stm_success`
- ◆ `stm_attribute_name_not_found`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_illegal_name`
- ◆ `stm_unresolved`

### stm\_r\_xx\_attr\_name

**Function type:** LIST OF STRING

**For elements:**

action	an
activity	ac
actor	actor
boundary box	bb
chart	ch
condition	co
data-item	di
data-store	ds
event	ev
field	fd
information-flow	if
lifeline	ll
module	md
router	router
state	st
subroutine	sb
use case	uc
user-defined type	dt

#### Description

Returns the names of attributes associated with the specified element. Attributes are associated with elements via element forms.

You can call this function without indicating the specific element type, as follows:

```
stm_r_attr_name (id, status)
```

#### Syntax

```
stm_r_xx_attr_name (xx_id, status)
```

## Arguments

Argument	Input/Output	Type	Description
xx_id	In	State element	The element ID.
status	Out	INTEGER	The function status code. If no attributes exist for the specified element, status receives the value <code>stm_attribute_name_not_found</code> .

## Status Codes

- ◆ `stm_success`
- ◆ `stm_attribute_name_not_found`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_unresolved`

## Example

To perform operations on the attributes of the state `WAIT`, retrieve a list of `WAIT`'s attribute names. The template contains the following statements:

```
VARIABLE
  STATE          st_id;
  LIST OF STRING attr_list;
  STRING         attrib;
  INTEGER        status;
.
.
.
st_id := stm_r_st ('WAIT', status);
attr_list := stm_r_st_attr_name (st_id, status);
FOR attrib IN attr_list LOOP
.
.
.
```

`attr_list` contains a list of attribute names for `WAIT`. In the `FOR` loop, perform the operations on each item in the list of attributes (such as retrieving and printing the corresponding values).

### stm\_r\_xx\_attr\_val

**Function type:** LIST OF STRING

**For elements:**

action	an
activity	ac
actor	actor
boundary box	bb
chart	ch
condition	co
data-item	di
data-store	ds
event	ev
field	fd
information-flow	if
lifeline	ll
module	md
router	router
state	st
subroutine	sb
use case	uc
user-defined type	dt

#### Description

Retrieves attribute values associated with a particular attribute name for the specified element.

You can call this function without indicating the specific element type, as follows:

```
stm_r_attr_val (id, attr_name, status)
```

#### Syntax

```
stm_r_xx_attr_val (xx_id, attr_name, status)
```

## Arguments

Argument	Input/Output	Type	Description
<code>xx_id</code>	In	Statestate element	The element ID.
<code>attr_name</code>	In	STRING	The attribute name. The attribute name is not case-sensitive.
<code>status</code>	Out	INTEGER	The function status code. If <code>attr_name</code> does not exist for the specified element, <code>status</code> receives the value <code>stm_attribute_name_not_found</code> .

Note the following:

- ◆ Attribute values might exist for attributes with no name. Therefore, if you supply contiguous apostrophes ( ' ' ) for `attr_name`, you retrieve all values for unnamed attributes.
- ◆ In most cases, attributes have only one value. However, there are some cases where more than one attribute value is simultaneously meaningful. For example, a module has an attribute `implementation`. The attributes `software` and `hardware` might both be meaningful for some modules. Therefore, Statestate provides the capability of assigning multiple values to attributes, and the function returns a list of these values. When there is a single value, the list consists of one component.

## Status Codes

- ◆ `stm_success`
- ◆ `stm_attribute_name_not_found`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_illegal_name`
- ◆ `stm_unresolved`

### Example

The following code prints the attribute values of all the attributes defined for the state `WAIT`:

```
VARIABLE
STATE          st_id;
LIST OF STRING attr_list;
STRING         attrib, value;
INTEGER        status;

.
.
.
st_id := stm_r_st ('WAIT', status);
attr_list :=stm_r_st_attr_name (st_id, status);
FOR attrib IN attr_list LOOP
  FOR value IN stm_r_st_attr_val (st_id, attrib, status)
  LOOP
    WRITE ('\n', attrib,' is ',value);
  END LOOP;
END LOOP;

.
.
.
```

`attr_list` contains a list of attributes for `WAIT`. Write the attribute values for each item in this list to the document.

## stm\_r\_xx\_bit\_array\_index

**Function type:** `STRING`

**For elements:**

data-item	di
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

### Description

Returns the left index of a bit array.

You can call this function without indicating the specific element type, as follows:

```
stm_r_bit_array_lindex (id, status)
```

### Syntax

```
stm_r_xx_bit_array_lindex (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>xx_id</code>	In	Stamete element	The element ID
<code>status</code>	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`

## stm\_r\_xx\_bit\_array\_rindex

**Function type:** `STRING`

**For elements:**

data-item	di
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

### Description

Returns the right index of a bit array.

You can call this function without indicating the specific element type, as follows:

```
stm_r_bit_array_rindex (id, status)
```

### Syntax

```
stm_r_xx_bit_array_rindex (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>xx_id</code>	In	StateMate element	The element ID
<code>status</code>	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`



## stm\_r\_xx\_cbk\_binding

**Function type:** LIST OF STRING

**For elements:**

activity	ac
condition	co
data-item	di
event	ev
state	st

### Description

Retrieves the callback binding for specified elements.

You can call this function without indicating the specific element type, as follows:

```
stm_r_cbk_binding (id, status)
```

### Syntax

```
stm_r_xx_cbk_binding (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	Statestate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_not\_found
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_missing\_cbk\_binding
- ◆ stm\_unresolved

## stm\_r\_xx\_cbk\_binding\_enable

**Function type:** LIST OF STATEMATE ELEMENTS (predefined constant)

**For elements:**

activity	ac
condition	co
data-item	di
event	ev
state	st

### Description

Retrieves the enabled callback bindings.

You can call this function without indicating the specific element type, as follows:

```
stm_r_cbk_binding_enable (id, status)
```

### Syntax

```
stm_r_xx_cbk_binding_enable (id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	Stemate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_missing_cbk_binding`

### Return Values

Although the return value of this function is of type `INTEGER`, the Documentor enables you to reference this value by name. The name is defined internally as a predefined constant in DGL.

<b>Element</b>	<b>Element Subtype</b>
activity	<code>stm_ac_cbk_enable</code>
	<code>stm_ac_cbk_disable</code>
	<code>stm_ac_cbk_bind_missing</code>
condition	<code>stm_co_cbk_enable</code>
	<code>stm_co_cbk_disable</code>
	<code>stm_ac_cbk_bind_missing</code>
data-item	<code>stm_di_cbk_enable</code>
	<code>stm_di_cbk_disable</code>
	<code>stm_di_cbk_bind_missing</code>
event	<code>stm_ev_cbk_enable</code>
	<code>stm_ev_cbk_disable</code>
	<code>stm_ev_cbk_bind_missing</code>
state	<code>stm_st_cbk_enable</code>
	<code>stm_st_cbk_disable</code>
	<code>stm_st_cbk_bind_missing</code>

## stm\_r\_xx\_cbk\_binding\_expression

**Function type:** STRING

**For elements:**

activity	ac
condition	co
data-item	di
event	ev
state	st

### Description

Retrieves the callback binding expressions.

You can call this function without indicating the specific element type, as follows:

```
stm_r_cbk_binding_expression (id, status)
```

### Syntax

```
stm_r_xx_cbk_binding_expression (id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_not\_found
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_missing\_cbk\_binding
- ◆ stm\_unresolved

## stm\_r\_xx\_cbk\_binding\_expression\_hyper

**Function type:** `STRING`

**For elements:**

activity	ac
condition	co
data-item	di
event	ev
state	st

### Description

Retrieves the callback binding expressions.

You can call this function without indicating the specific element type, as follows:

```
stm_r_cbk_binding_expression_hyper (id, status)
```

### Syntax

```
stm_r_xx_cbk_binding_expression_hyper (id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	Statestate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_missing_cbk_binding`
- ◆ `stm_unresolved`

### stm\_r\_tt\_cell

**Function type:** STRING

**For elements:**

action	an
activity	ac
subroutine	sb

#### Description

Retrieves the contents of the specified cell in the given truth table.

#### Syntax

```
stm_r_tt_cell (el, row_num, col_num, status)
```

#### Arguments

Argument	Input/Output	Type	Description
el	In	Statement element	The element ID
row_num	In	INTEGER	The row number of the cell
col_num	In	INTEGER	The column number of the cell
status	Out	INTEGER	The function status code

#### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_id\_not\_found
- ◆ stm\_unresolved
- ◆ stm\_missing\_truth\_table
- ◆ stm\_truth\_table\_invalid\_row
- ◆ stm\_truth\_table\_invalid\_column

## stm\_r\_tt\_cell\_type

**Function type:** INTEGER

**For elements:**

action	an
activity	ac
subroutine	sb

### Description

Retrieves the data-type of the specified cell in the given truth table.

### Syntax

```
stm_r_tt_cell_type (el, row_num, col_num, status)
```

### Arguments

Argument	Input/Output	Type	Description
el	In	StateMate element	The element ID
row_num	In	INTEGER	The row number of the cell
col_num		INTEGER	The column number of the cell
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_id\_not\_found
- ◆ stm\_unresolved
- ◆ stm\_missing\_truth\_table
- ◆ stm\_truth\_table\_invalid\_row
- ◆ stm\_truth\_table\_invalid\_column

### Return Values

Although the return value of this function is of type `INTEGER`, the Documentor enables you to reference this value by name. The name is defined internally as a predefined constant in DGL. The possible values are as follows:

- ◆ `stm_tt_cell_type_missing`
- ◆ `stm_tt_cell_rpn_same_as_down`
- ◆ `stm_tt_cell_rpn`
- ◆ `stm_tt_cell_dont_care`
- ◆ `stm_tt_is_generate_ev`
- ◆ `stm_tt_is_not_generate_ev`
- ◆ `stm_tt_cell_empty_same_as_up`
- ◆ `stm_tt_cell_empty_same_as_up_and_down`
- ◆ `stm_tt_is_empty_cell`



## stm\_r\_changes\_log

**Function type:** LIST OF STRING

**For elements:**

chart	ch
-------	----

### Syntax

```
stm_r_changes_log (ch_lst, ascending, per_date, dont_format, status)
```

### Arguments

Argument	Input/Output	Type	Description
ch_lst	In	LIST OF CHART	The list of charts to track.
ascending	In	BOOLEAN	Determines whether the changes are listed in ascending order (TRUE).
per_date	In	BOOLEAN	Determines whether the changes are listed chronologically (TRUE).
dont_format	In	BOOLEAN	Determines whether the log file is formatted. If this is TRUE, each log entry is inserted into a returned list element. If it is FALSE, each field of the log entry is inserted into a returned list element.
status	Out	INTEGER	The function status code.

### Status Codes

- ◆ stm\_id\_out\_of\_range
- ◆ stm\_not\_chart\_id
- ◆ stm\_id\_not\_found
- ◆ stm\_success

## stm\_r\_xx\_chart

**Function type:** CHART

**For elements:**

a-flow-line (basic)	ba
a-flow-line (compound)	af
action	an
activity	ac
actor	actor
boundary box	bb
condition	co
connector	cn
data-item	di
data-store	ds
event	ev
field	fd
function	fn
information-flow	if
lifeline	ll
local data	ld
m-flow-line (compound)	mf
module	md
router	router
state	st
subroutine	sb
subroutine parameter	sp
transition (basic)	bt
transition (compound)	tr
use case	uc
user-defined type	dt

## Description

Returns the chart ID for the specified element.

Note the following:

- ◆ You can call this function without indicating the specific element type, as follows:
 

```
stm_r_chart (id, status)
```
- ◆ For compound arrows, this function retrieves the chart only when all the arrow segments are in the same element. Otherwise, it returns the value 0.

## Syntax

```
stm_r_xx_chart (xx_id, status)
```

## Arguments

Argument	Input/Output	Type	Description
xx_id	In	State element	The element ID
status	Out	INTEGER	The function status code

## Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_id\_not\_found

## Example

To write the name of the chart in which state s1 is found, the template would contain the following statements:

```
VARIABLE
  STATE      state_id;
  INTEGER    status1, status2, status3;
  .
  .
state_id := stm_r_st ('S1', status1);
WRITE ('\n Chart name is:', stm_r_ch_name(
  stm_r_st_chart(state_id, status2), status3);
  .
  .
```

## stm\_r\_xx\_combinationals

**Function type:** LIST OF STRING

**For elements:**

activity	ac
chart	ch

### Description

Returns a list of strings. Each element of the list holds one combinational assignment, which is connected to the specified element.

You can call this function without indicating the specific element type, as follows:

```
stm_r_combinationals (id, status)
```

### Syntax

```
stm_r_xx_combinationals (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	Stateate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_error_in_file`
- ◆ `stm_missing_field`
- ◆ `stm_missing_label`
- ◆ `stm_missing_name`
- ◆ `stm_file_not_found`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_illegal_parameter`

## stm\_r\_sb\_connected\_chart

**Function type:** STATEMATE ELEMENT

**For elements:**

subroutine	sb
------------	----

### Description

Returns the ID of the procedural statechart connected to the specified subroutine.

### Syntax

```
stm_r_sb_connected_chart (sb_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
sb_id	In	Statemate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_not\_found
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_no\_connected\_chart

## stm\_r\_xx\_containing\_fields

**Function type:** LIST OF FIELDS

**For elements:**

data-item	di
user-defined type	dt

### Description

Returns the list of union or record elements that contain fields.

You can call this function without indicating the specific element type, as follows:

```
stm_r_containing_fields (id, status)
```

### Syntax

```
stm_r_xx_containing_fields (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	State element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_missing_field`
- ◆ `stm_unresolved`

## stm\_r\_ch\_creation\_date

**Function type:** `STRING`

**For elements:**

chart	ch
-------	----

### Description

Returns the date (as a string) on which the specified chart was created.

**Note:** This function is relevant only for charts that were explicitly defined using one of the graphic editors.

### Syntax

```
stm_r_ch_creation_date (ch_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
ch_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_unresolved`

### Example

To retrieve the chart date, use the following statements:

```
VARIABLE
  CHART      chart_id;
  INTEGER    status;
  .
  .
  .
chart_id := stm_r_ch ('TOP', status);
WRITE ('\n Chart created on:',
  stm_r_ch_creation_date (chart_id, status));
  .
  .
  .
```

The template writes to the document the date on which the chart named TOP was created.



## stm\_r\_ch\_creator

**Function type:** `STRING`

**For elements:**

chart	ch
-------	----

### Description

Returns the name of the StateMate user who created the specified chart.

**Note:** This function is relevant only for charts that were explicitly created using one of the graphic editors.

### Syntax

```
stm_r_ch_creator (ch_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
ch_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_unresolved`

### Example

To retrieve the name of the user who created the chart, use the following statements:

```
VARIABLE
  CHART      chart_id;
  INTEGER    status;
  .
  .
  .
chart_id := stm_r_ch ('TOP', status);
WRITE ('\n Chart created by:',
  stm_r_ch_creator (chart_id, status));
  .
  .
  .
```

The name written to the document is the name of the user who created the chart named TOP.

## stm\_r\_xx\_data\_type

**Function type:** INTEGER

**For elements:**

data-item	di
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

### Description

Returns the element subtype, including its data type and data structure. For example:

```
stm_xx_union_array, stm_xx_integer, stm_xx_real_queue
```

You can call this function without indicating the specific element type, as follows:

```
stm_r_data_type (id, status)
```

### Syntax

```
stm_r_xx_data_type (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_error\_in\_file
- ◆ stm\_file\_not\_found
- ◆ stm\_illegal\_parameter
- ◆ stm\_missing\_field

## stm\_r\_rt\_date

**Function type:** `STRING` `stm_date`

**For elements:**

requirement	rt
-------------	----

### Description

Returns the date field of the specified requirement record.

### Syntax

```
stm_r_rt_date (rt_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
rt_id	In	State element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_error_in_file`
- ◆ `stm_file_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_illegal_parameter`
- ◆ `stm_missing_field`

## stm\_r\_xx\_definition\_type

**Function type:** INTEGER

**For elements:**

action	an
condition	co
data-item	di
enumerated value	en
event	ev
function	fn
information-flow	if
local data	ld
subroutine	sb
subroutine parameter	sp

### Description

Returns the definition type of the specified textual element.

Note the following:

- ◆ You can call this function without indicating the specific element type, as follows:  
`stm_r_definition_type (id, status)`
- ◆ The enumerated type that reflects whether the textual element has a form. The nature of the definition field in the form is `stm_definition_type`, whose values are as follows:
  - `stm_reference`—The element has no form.
  - `stm_primitive`—The definition field is empty.
  - `stm_compound`—The definition field contains a compound expression.
  - `stm_constant`—The definition field contains a constant.
  - `stm_alias`—The definition field contains an identifier, a bit array, a component, or a slice (relevant for `di` only).
  - `stm_explicit`—The `info_flow` has a form.
  - `stm_predefined`—Predefined function.
- ◆ Note that these types are not explicitly specified, but derived from the specification.

### Syntax

```
stm_r_xx_definition_type (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_not_found`

### Return Values

Although the return value of this function is of type `INTEGER`, the Documentor enables you to reference this value by name. The name is defined internally as a predefined constant in DGL. The following table lists the possible values for each StateMate element sub type:

Element	Abbreviation	Element Sub-Type
action	an	<code>stm_an_reference</code>
		<code>stm_an_primitive</code>
		<code>stm_an_compound</code>
condition	co	<code>stm_co_reference</code>
		<code>stm_co_primitive</code>
		<code>stm_co_compound</code>
		<code>stm_co_constant</code>
data-item	di	<code>stm_di_reference</code>
		<code>stm_di_primitive</code>
		<code>stm_di_compound</code>
		<code>stm_di_constant</code>
		<code>stm_di_alias</code>
event	ev	<code>stm_ev_reference</code>
		<code>stm_ev_primitive</code>
		<code>stm_ev_compound</code>
field	fd	<code>stm_fd_primitive</code>

---

information-flow	if	stm_if_reference
		stm_if_explicit
local data	ld	stm_sp_defined
subroutine	sb	stm_sb_reference
		stm_sb_predefined
		stm_sb_function
		stm_sb_procedure
		stm_sb_task
subroutine parameter	sp	stm_sp_defined
user-defined type	dt	stm_dt_reference
		stm_dt_primitive

### stm\_r\_xx\_desc\_file

**Function type:** `STRING`

**For elements:**

actor	actor
boundary box	bb
use case	uc

#### Description

Returns the description file for the specified element.

You can call this function without specifying the element, as follows:

```
stm_r_desc_file (el, status)
```

#### Syntax

```
stm_r_xx_desc_file (elem, status)
```

#### Arguments

Argument	Input/Output	Type	Description
<code>elem</code>	In	State element	The element
<code>status</code>	Out	INTEGER	The function status code

#### Status Codes

- ◆ `stm_id_out_of_range`
- ◆ `stm_missing_description_file`
- ◆ `stm_not_use_case`
- ◆ `stm_not_actor`
- ◆ `stm_not_boundary_box`
- ◆ `stm_success`



## stm\_r\_xx\_description

**Function type:** `STRING`

**For elements:**

action	an
activity	ac
actor	actor
boundary box	bb
chart	ch
condition	co
data-item	di
data-store	ds
event	ev
field	fd
information-flow	if
lifeline	ll
local data	ld
module	md
router	router
state	st
subroutine	sb
subroutine parameter	sp
user-defined type	dt

### Description

Returns the short description of the specified element. The short description is defined in the element's form.

You can call this function without indicating the specific element type, as follows:

```
stm_r_description (id, status)
```

### Syntax

```
stm_r_xx_description (xx_id, status)
```

### Arguments

Argument	Input/ Output	Type	Description
xx_id	In	State element	The element ID.
status	Out	INTEGER	The function status code. If no description exists in the element's form, status receives the value <code>stm_missing_short_description</code> .

### Status Codes

- ◆ `stm_success`
- ◆ `stm_unresolved`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_missing_short_description`

### Example

To retrieve the contents of the short description field in the form of state `SSS.S1`, use the following statements:

```
VARIABLE
STATE          state_id;
STRING         state_desc;
INTEGER        status;
.
.
state_id := stm_r_st ('SSS.S1', status);
state_desc:= stm_r_st_description (state_id, status);
.
.
```

`state_desc` contains the short description for the state `SSS.S1` (whose ID is `state_id`).

## stm\_r\_design\_attr

**Function type:** LIST OF STRING

**For elements:**

element ID	
------------	--

### Description

Retrieves the design attributes of an element in a form of a list of strings. Each element in the list includes a key-value pair.

### Syntax

```
stm_r_design_attr (el, status)
```

### Arguments

Argument	Input/Output	Type	Description
el	In	Statestate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`

## stm\_r\_xx\_displayed\_name

**Function type:** `STRING`

**For elements:**

activity	ac
data-store	ds
module	md
module-occurrence	om
off-page activity	oa
router	router
state	st

### Description

Returns the name of a chart, as it appears in the graphic editor where the specified element is located.

You can call this function without indicating the specific element type, as follows:

```
stm_r_displayed_name (id, status)
```

### Syntax

```
stm_r_xx_displayed_name (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>xx_id</code>	In	StateMate element	The element ID
<code>status</code>	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_error_in_file`
- ◆ `stm_illegal_parameter`
- ◆ `stm_file_not_found`
- ◆ `stm_missing_name`
- ◆ `stm_missing_field`

### Example

To retrieve the name of the chart as it appears in the title bar of the graphic editor (for example, `FUNC PROC EXAMPLE`) and write it to the document, use the following statements:

```
VARIABLE
  STATE      state_id;
  INTEGER    status;
            .
            .
            .
state_id := stm_r_st ('S1.S3', status);
WRITE (stm_r_st_displayed_name (state_id, status));
            .
            .
            .
```

If this function does not meet your needs, there are two other `name` functions that return different values:

- ◆ `stm_r_xx_name`
- ◆ `stm_r_xx_uniquename`

## stm\_r\_ddb\_list\_names

**Function type:**

**For element:**

### Description

Returns the names of the lists created by the properties browser.

### Syntax

```
stm_r_ddb_list_names (&status)
```

### Arguments

Argument	Input/Output	Type	Description
status	Out	int INTEGER	The function status code

### Status Codes

- ◆ `stm_success`

## stm\_r\_element\_type

**Function type:** INTEGER (predefined constant)

**For elements:** All types

### Description

Returns the element type of the specified element.

### Syntax

```
stm_r_element_type (id, status)
```

### Arguments

Argument	Input/Output	Type	Description
id	In	State element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`

### Return Values

Although the return value of this function is of type `INTEGER`, the Documentor enables you to reference this value by name. The name is defined internally as a predefined constant in DGL. Each element type has an associated predefined value, as shown in the following table:

Element Type	Value
a-flow-line (basic)	<code>stm_a_flow_line</code>
a-flow-line (compound)	<code>stm_compound_a_flow_line</code>
action	<code>stm_action</code>
activity	<code>stm_activity</code>
chart	<code>stm_chart</code>
condition	<code>stm_condition</code>
connector in activity-chart	<code>stm_a_connector</code>
connector in module-chart	<code>stm_m_connector</code>
connector in statechart	<code>stm_s_connector</code>
data-item	<code>stm_data_item</code>
data-store	<code>stm_data_store</code>
decomposed sequence diagram	<code>stm_decomposed_sd</code>
event	<code>stm_event</code>
external lifeline	<code>stm_external_lifeline</code>
flow label	<code>stm_flow_label</code>
information-flow	<code>stm_information_flow</code>
lifeline	<code>stm_lifeline</code>
m-flow-line (basic)	<code>stm_label</code>
m-flow-line (compound)	<code>stm_m_flow_line</code>
message	<code>stm_message</code>
module	<code>stm_compound_m_flow_line</code>
module occurrence	<code>stm_module</code>
order insignificant	<code>stm_order_insignificant</code>
separator	<code>stm_separator</code>
subroutine	<code>stm_subroutine</code>
state	<code>stm_module_occurrence</code>
timing constraint	<code>stm_timing_constraint</code>
transition (basic)	<code>stm_state</code>
transition (compound)	<code>stm_transition</code>
transition label	<code>stm_compound_transition</code>



**Example**

The sample template performs the following tasks:

- ◆ Generates a list of elements (of type MIXED) using `stm_r_mx_in_definition_of_co`. Elements in this list are all elements (not necessarily conditions) appearing in the definition field of the condition `C1`.
- ◆ Searches this list for conditions. If any are found, it prints them in the document.

```
VARIABLE
  CONDITION          cond_id;
  LIST OF ELEMENT    elmnt_list;
  ELEMENT            el;
  INTEGER            status, el_type;
.
.
.
cond_id := stm_r_co ('C1', status);
elmnt_list := stm_r_mx_in_definition_of_co
({cond_id}, status);
FOR el IN elmnt_list LOOP
  el_type := stm_r_element_type (el, status);
  IF el_type = stm_condition THEN
    WRITE ('\n Condition Name:', stm_r_co_name(
      el, status));
  END IF;
END LOOP;
.
.
.
```

## stm\_r\_xx\_expr\_hyper

**Function type:** `STRING`

**For elements:**

a-flow-lines (basic)	ba
action	an
condition	co
data-item	di
event	ev
message	msg
m-flow-line (basic)	bm
subroutine action language	sb_action_lang
transitions (basic)	bt
user-defined type	dt

### Description

Returns the definition expression of the specified element found in the **Definition** field of the element's form, including hyperlinks to referenced elements.

### Syntax

```
stm_r_xx_expr_hyper (elem, format, status)
```

### Arguments

Argument	Input/Output	Type	Description
elem	In	Stateate element	The element ID
format	In	STRING	Either FRAMEMAKER or WORD
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_unresolved`
- ◆ `stm_primitive_element`

## stm\_r\_xx\_expression

**Function type:** `STRING`

**For elements:**

a-flow-line (basic)	ba
action	an
condition	co
data-item	di
event	ev
message	msg
m-flow-line (basic)	bm
transition (basic)	bt
user-defined type	dt

### Description

Returns the definition expression of the specified element found in the **Definition** field of the element's form. For arrows, this function returns the label attached to the arrow. The function is performed for basic arrows (arrow segments that connect boxes and connectors).

Note the following:

- ◆ You can call this function without indicating the specific element type, as follows:  
`stm_r_expression (id, status)`
- ◆ This function is valid for compound textual elements, which are defined as an expression using the **Definition** field of its form.

### Syntax

```
stm_r_xx_expression (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	Stateate element	The element ID.
status	Out	INTEGER	The function status code. If xx_id belongs to a primitive (not a compound) element, status receives the value <code>stm_primitive_element</code> .

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_unresolved`
- ◆ `stm_primitive_element`

### Example

To retrieve the definition of `c1` from the database for a system that contains a condition `c1` (where `c1` is defined as `c2` or `c3` in the form of `c1`), use the following function calls:

```
VARIABLE
CONDITION      cond_id;
STRING         cond_def;
INTEGER        status;
.
.
.
cond_id := stm_r_co ('C1', status);
cond_def := stm_r_co_expression (cond_id, status);
.
.
.
```

`cond_def` is assigned the string value "C2 or C3".

## stm\_r\_xx\_ext\_link

**Function type:** `STRING`

**For elements:**

action	an
activity	ac
actor	actor
boundary box	bb
chart	ch
condition	co
data-item	di
data-store	ds
event	ev
field	fd
information-flow	if
lifeline	ll
module	md
router	router
state	st
subroutine	sb
use case	uc
user-defined type	dt

### Description

Retrieves the external file link for the specified element. You specify the external link by selecting **Edit > Link** in the properties for the element.

You can call this function without specifying an element type, as follows:

```
stm_r_ext_link (elem, status)
```

### Syntax

```
stm_r_xx_ext_link (elem, status)
```

### Arguments

Argument	Input/Output	Type	Description
elem	In		The Rhapsody element
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_name_not_found`
- ◆ `stm_auto_defined`
- ◆ `stm_missing_external_link`
- ◆ `stm_success`

## stm\_r\_uc\_ext\_point\_def

**Function type:** `STRING`

**For elements:**

use case	uc
----------	----

### Description

Retrieves the extension point definitions for the specified use case.

### Syntax

```
stm_r_uc_ext_point_def (uc, status)
```

### Arguments

Argument	Input/Output	Type	Description
uc	In	USE_CASE	The use case
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_id_out_of_range`
- ◆ `stm_not_use_case`
- ◆ `stm_missing_extension_point_definition`
- ◆ `stm_success`

## stm\_r\_formal\_parameter\_names

**Function type:** LIST OF STRING

**For elements:**

action	an
condition	co
data-item	di
event	ev

### Description

Returns a list of names of formal parameters that appear in bindings of instance boxes and components.

### Syntax

```
stm_r_formal_parameter_names (inst_box_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	State element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`



## stm\_r\_sb\_global\_data

**Function type:** LIST OF ELEMENT

**For elements:**

subroutine	sb
------------	----

### Description

Returns the global data associated with the specified subroutine.

### Syntax

```
stm_r_sb_global_data (sb_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	Stateate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_missing\_global\_data
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_id\_not\_found

## stm\_r\_global\_interface\_report

**Function type:** LIST OF STRING

**For elements:**

activity	ac
external router	ext_router
router	router

### Description

Creates a global interface report from the specified input list.

### Syntax

```
stm_r_global_interface_report (elm_lst, sort_by_elm, status)
```

### Arguments

Argument	Input/Output	Type	Description
elm_list	In	LIST OF ELEMENT	The list of elements to include in the report
sort_by_elm	In	BOOLEAN	Specifies whether to sort the elements in the report by element name (TRUE) or activity name (FALSE)
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_id\_out\_of\_range
- ◆ stm\_success

## stm\_r\_xx\_cbk\_binding\_expression\_hyper

### Function Type

stm\_expression

### For Elements

activity	ac
condition	co
data-item	di
event	ev
state	st

### Description

Retrieves the callback binding expressions, with hyperlinks to referenced elements.

You can call this function without indicating the specific element type, as follows:

```
stm_r_cbk_binding_expression_hyper (id, &status)
```

### Syntax

```
stm_r_xx_cbk_binding_expression_hyper (id, char* formator int*status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	stm_id	The element ID.
format	In	STRING	Either FrameMaker or Word.
status	Out	int	The function status code.

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_not\_found
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_missing\_cbk\_binding
- ◆ stm\_unresolved

## stm\_r\_xx\_graphic

**Function type:** `STRING`

**For elements:**

activity	ac
basic a-flow-line	ba
basic m-flow-line	bm
basic transition	bt
combinational assignment	ca
connector	cn
data-store	ds
module	md
module-occurrence	om
note	nt
off-page activity	oa
state	st

### Description

Returns the graphical information associated with the specified element.

Note the following:

- ◆ You can call this function without indicating the specific element type, as follows:  
`stm_r_graphic(id, status)`
- ◆ Each environment module can have several occurrences with the same name in a chart. Call the query function `stm_r_om_of_md` to get the graphical information of its occurrences, then use the function `stm_r_om_graphic` for each occurrence.

### Syntax

```
stm_r_xx_graphic (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	Statestate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_unresolved`
- ◆ `stm_id_not_found`
- ◆ `stm_missing_graphic_data`

### Note

---

When `stm_unresolved` is returned, no record is received.

## stm\_r\_hyper\_key

**Function type:** `STRING`

**For elements:**

element ID	ac
------------	----

### Description

Retrieves the unique key for the specified element.

### Syntax

```
stm_r_hyper_key (el, status)
```

### Arguments

Argument	Input/Output	Type	Description
el	In	ELEMENT	The element ID whose key you want
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_id_out_of_range`
- ◆ `stm_success`

## stm\_r\_md\_implementation

**Function type:** `STRING`

**For elements:**

module	md
--------	----

### Description

Retrieves the implementation type for the specified module.

### Syntax

```
stm_r_md_implementation (md_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
md_id	In	State element	The module ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_unresolved`
- ◆ `stm_not_instance`

## stm\_r\_included\_gds

**Function type:** LIST OF CHART

**For elements:**

chart	ch
-------	----

### Description

Returns the list of global definition sets contained in the specified chart.

### Syntax

```
stm_r_included_gds (ch_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
ch_id	In	CHART	The chart
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_id\_out\_of\_range
- ◆ stm\_use\_all\_public\_gds
- ◆ stm\_success



## stm\_r\_msg\_included\_in\_ord\_insig

**Function type:** LIST OF MESSAGE

**For elements:**

Last of order insignificance	ord_insig_list
------------------------------	----------------

### Description

Returns a list of messages that are bounded by an order-insignificant element.

### Syntax

```
stm_r_msg_included_in_ord_insig (ord_insig_list, status)
```

### Arguments

Argument	Input/ Output	Type	Description
ord_insig_list	In	LIST OF ORDER_INSIGNIFICANT	A list of elements
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_not_order_insignificant`

## stm\_r\_cd\_info

**Function type:** `STRING`

**For elements:**

chart	ch
-------	----

### Description

Retrieves the description of the specified continuous chart.

### Syntax

```
stm_r_cd_info (ch, status)
```

### Arguments

Argument	Input/Output	Type	Description
cd	In	ChART	The chart
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_id_out_of_range`
- ◆ `stm_null_string`
- ◆ `stm_success`

## stm\_r\_inherited\_gds

**Function type:** LIST OF CHART

**For elements:**

chart	ch
-------	----

### Description

Retrieves the list of global definition sets that are “inherited” (included indirectly) by the specified chart.

### Syntax

```
stm_r_inherited_gds (ch_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
ch_id	In	CHART	The chart
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_id_out_of_range`
- ◆ `stm_use_all_public_gds`
- ◆ `stm_success`

## stm\_r\_xx\_instance\_name

**Function type:** `STRING`

**For elements:**

activity	ac
module	md
state	st

### Description

Returns the name of the instance as it appears in the chart for a specific hierarchical Rhapsody element.

Note the following:

- ◆ You can call this function without indicating the specific element type, as follows:  
`stm_r_instance_name (id, status)`
- ◆ This function is relevant only for states, internal modules, and regular or control activities, because only these elements can have instances.

### Syntax

```
stm_r_xx_instance_name (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>xx_id</code>	In	State/element	The element ID
<code>status</code>	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_unresolved`
- ◆ `stm_not_instance`

**Example**

To retrieve the name of an instance for state named `S1@S1_def`, use the following statements:

```
VARIABLE
  STATE          state_id;
  INTEGER        status;
  .
  .
  .
state_id := stm_r_st ('S1', status);
WRITE ('\n Instance Name:',
  stm_r_st_instance_name (state_id, status));
  .
  .
  .
```

The name written to the document is `S1@S1_DEF`.

## stm\_r\_xx\_keyword

**Function type:** `STRING` (a file name)

**For elements:**

action	an
activity	ac
actor	actor
boundary box	bb
chart	ch
condition	co
data-item	di
data-store	ds
event	ev
field	fd
information-flow	if
lifeline	ll
module	md
router	router
state	st
subroutine	sb
use case	uc
user-defined type	dt

### Description

Retrieves a portion of the element's long description. An element's long description is attached to its form.

You can call this function without indicating the specific element type, as follows:

```
stm_r_keyword (id, begin_keyword, end_keyword, filename, status)
```

### Syntax

```
stm_r_xx_keyword (xx_id, begin_keyword, end_keyword, filename, status)
```

## Arguments

Argument	Input/Output	Type	Description
<code>xx_id</code>	In	<code>stm_id</code>	The element ID
<code>begin_&lt;keyword</code>	In	<code>char *</code>	The beginning of the portion of the string in the long description to extract
<code>end_keyword</code>	In	<code>char *</code>	The end of the portion of the string in the long description to extract
<code>filename</code>	In	<code>stm_filename</code>	The name of the file to contain the long description
<code>status</code>	Out	<code>int</code>	The function status code

Note the following:

- ◆ The arguments `begin_keyword` and `end_keyword` are strings of text appearing in the element's long description. The portion extracted from the database begins with the line following `begin_keyword` and extends to the line preceding `end_keyword`.
- ◆ If the value of `begin_keyword` does not appear in the long description, the function creates an empty file; `status` then receives the value `stm_starting_keyword_not_found`.
- ◆ If the value of `end_keyword` does not appear in the long description, the entire long description (from the line following the value of `begin_keyword`) is retrieved; `status` receives the value `stm_ending_keyword_not_found`.
- ◆ The values of `begin_keyword` and `end_keyword` must appear at the beginning of a line in the long description.
- ◆ `filename` follows the conventions of the operating system. It returns the value of the argument `filename` (when one is specified). If an empty string '' (two contiguous quotation marks) is specified for `filename`, Rhapsody creates a temporary file where it stores the text. The name of this temporary file is returned by this function.
- ◆ If no long description exists for the element, `status` receives the value `stm_missing_long_description`.

### Status Codes

- ◆ `stm_success`
- ◆ `stm_unresolved`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_can_not_open_file`
- ◆ `stm_name_not_found`
- ◆ `stm_missing_long_description`
- ◆ `stm_starting_keyword_not_found`
- ◆ `stm_ending_keyword_not_found`

### Example

The long description for the state `WAIT` contains the following section:

```
!BHV_DESCR
When the assembly process reaches the critical stage where all parts
must be carefully selected, mounted and assembled, we wait for the
interrupt signal to tell us that all the required parts are in place
before continuing. This state acts as a synchronization point in the
assembly process.
!END_DESCR
```

To extract the portion of the long description beginning with “When the...” and ending with “... assembly process,” use the following function call:

```
VARIABLE
STATE          state_id;
STRING         descr_file;
INTEGER        status;
.
.
.
state_id:=stm_r_st ('WAIT', status);
descr_file:=stm_r_st_keyword (state_id,'!BHV_DESCR',
                             '!END_DESCR','',status);
INCLUDE (descr_file);
.
.
.
```

The portion of the long description is written to a temporary file and written to your document.



## stm\_r\_sb\_kr\_c\_user\_code

**Function type:** LIST OF STRING

**For elements:**

subroutine	sb
------------	----

### Description

Returns the K&R C code that was manually written by the user for the specified subroutine.

### Syntax

```
stm_r_sb_kr_c_user_code (sb_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
sb_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_missing_user_code`

## stm\_r\_xx\_labels

**Function type:** LIST OF STRING

**For elements:**

message	msg
transition	tr

### Description

Returns a list of strings that consists of all the labels of the specified compound transition or message. The labels appear on the transition segments that comprise the specified compound transition, or on the message. The syntax of these labels is `trigger/action`.

### Note

To divide the labels into their trigger and action parts, use the utility routines `stm_trigger_of_reaction` and `stm_action_of_reaction`.

### Syntax

```
stm_r_xx_labels (tr_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	TRANSITION ELEMENT OR MESSAGE	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_unresolved`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_missing_label`

**Example**

To extract all the labels of transitions exiting from state s1, the template should contain the following statements:

```
VARIABLE
  STATE          state_id
  INTEGER        status;
  LIST OF STRING labels;
  LIST OF TRANSITION trans;
  TRANSITION     tr;
  STRING         lab;
                .
                .
state_id := stm_r_st ('S1',status);
trans := stm_r_tr_from_source_st ({state_id}, status);
FOR tr IN trans LOOP
  labels:=stm_r_tr_labels(tr, status);
  IF status = stm_success THEN
    FOR lab IN labels LOOP
      .
      .
      .
    END FOR;
  END IF;
END FOR;
```

## stm\_r\_xx\_labels\_hyper

**Function type:** LIST OF STRING

**For elements:**

message	msg
transition	tr

### Description

Returns a list of strings of message or transition labels, with hyperlinks to referenced elements.

### Syntax

```
stm_r_xx_labels_hyper (message, format, status)
```

### Arguments

Argument	Input/Output	Type	Description
elem_id	In	State element	The element ID
format	In	STRING	Either FrameMaker or Word
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_unresolved`

## stm\_r\_local\_interface\_report

**Function type:** LIST OF STRING

**For elements:**

activity	ac
external router	ext_router
router	router

### Description

Creates a local interface report from the specified input list.

### Syntax

```
stm_r_local_interface_report (elm_lst, status)
```

### Arguments

Argument	Input/Output	Type	Description
elm_list	In	LIST OF ELEMENT	The list of elements to include in the report
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_id_out_of_range`
- ◆ `stm_success`

## stm\_r\_xx\_longdes

**Function type:** `STRING` (file name)

**For elements:**

action	an
activity	ac
actor	actor
boundary box	bb
chart	ch
condition	co
data-item	di
data-store	ds
event	ev
field	fd
information-flow	if
lifeline	ll
module	md
router	router
state	st
subroutine	sb
use case	uc
user-defined type	dt

### Description

Retrieves the long description attached to the specified element.

You can call this function without indicating the specific element type, as follows:

```
stm_r_longdes (id, filename, status)
```

### Syntax

```
stm_r_xx_longdes (xx_id, filename, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>xx_id</code>	In	State element	The element ID
<code>filename</code>	In	<code>stm_filename</code> STRING	The name of the file that will contain the long description
<code>status</code>	Out	INTEGER	The function status code

Note the following:

- ◆ The `filename` follows the conventions of the host operating system.
- ◆ This function returns the value of the argument `filename` when one is specified. If an empty string `''` (two contiguous quotation marks) is specified, Statemate creates a temporary file where it stores the text. The name of this temporary file is returned by the function.
- ◆ If no long description exists for the element, `status` receives the value `stm_missing_long_description`.

### Status Codes

- ◆ `stm_unresolved`
- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_can_not_open_file`
- ◆ `stm_missing_long_description`

### Example

To retrieve the long description for the activity A1, use the following statements:

```
VARIABLE
  ACTIVITY      act_id;
  STRING        long_des_file;
  INTEGER       status;
.
act_id := stm_r_ac ('A1',status);
long_des_file:=
  stm_r_ac_longdes(act_id,'text.txt',status);
.
```

The long description for activity A1 is written to the file `text.txt`. This file resides in the current working directory. The variable `long_des_file` contains the string `'text.txt'`.



## stm\_r\_lookup\_table\_header

**Function type:** LIST OF STRING

**For elements:**

lookup table	lookup table
--------------	--------------

### Description

Retrieves the header for the lookup table.

### Syntax

```
stm_r_lookup_table_header (el, status)
```

### Arguments

Argument	Input/Output	Type	Description
el	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`

## stm\_r\_xx\_max\_val

**Function type:** `STRING`

**For elements:**

data-item	di
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

### Description

Returns the maximum value of the specified element.

You can call this function without indicating the specific element type:

```
stm_r_max_val (id, status)
```

### Syntax

```
stm_r_xx_max_val (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`

## stm\_r\_xx\_min\_val

**Function type:** `STRING`

**For elements:**

data-item	di
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

### Description

Returns the minimum value of the specified element.

You can call this function without indicating the specific element type, as follows:

```
stm_r_min_val (id, status)
```

### Syntax

```
stm_r_xx_min_val (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`

## stm\_r\_xx\_mini\_spec

**Function type:** `STRING`

**For elements:**

activity	ac
----------	----

### Description

Returns a string with mini-spec reactions or actions.

You can call this function without indicating the specific element type, as follows:

```
stm_r_mini_spec (id, status)
```

### Syntax

```
stm_r_xx_mini_spec (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>xx_id</code>	In	State element	The element ID
<code>status</code>	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_unresolved`
- ◆ `stm_missing_label`

## stm\_r\_ac\_mini\_spec\_hyper

**Function type:** `STRING`

**For elements:**

activity	ac
----------	----

### Description

Returns a string with the mini-spec, including hyperlinks to referenced elements.

### Syntax

```
stm_r_ac_mini_spec_hyper (elem, format, status)
```

### Arguments

Argument	Input/Output	Type	Description
elem	In	StateMate element	The element ID
format	In	STRING	Either FrameMaker or Word
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_unresolved`

## stm\_r\_xx\_mode

**Function type:** INTEGER

**For elements:**

parameter	parameter
router	router

### Description

Returns the parameter or router mode.

### Syntax

```
stm_r_xx_mode (elem_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
elem_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_unresolved`

### Retrun Values

Although the return value of this function is of type `INTEGER`, the Documentor enables you to reference this value by name. The name is defined internally as a predefined constant in DGL. The possible values are as follows:

- ◆ `stm_transparent_router`
- ◆ `stm_non_transparent_router`

## stm\_r\_ch\_modification\_date

**Function type:** `STRING`

**For elements:**

chart	ch
-------	----

### Description

Returns the date in which the version of the chart in the workarea was saved in the databank.

### Note

This function is relevant only for charts that were explicitly defined using one of the graphics editors.

### Syntax

```
stm_r_ch_modification_date (ch_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
ch_id	In	State element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_unresolved`

### Example

To retrieve the date of the last modification for a chart, use the following statements:

```
VARIABLE
  CHART          chart_id;
  INTEGER        status;
                .
                .
                .
chart_id := stm_r_ch ('TOP', status);
WRITE ('\n Chart modified on:',
      stm_r_ch_modification_date (chart_id, status));
                .
                .
                .
```

The template writes to the document the date on which the chart named `TOP` was last modified.



**stm\_r\_xx\_name****Function type:** `STRING`**For elements:**

action	an
activity	ac
actor	actor
boundary box	bb
chart	ch
condition	co
data-item	di
data-store	ds
enumerated value	en
event	ev
field	fd
function	fn
information-flow	if
lifeline	ll
local data	ld
module	md
router	router
state	st
subroutine	sb
subroutine parameter	sp
use case	uc
user-defined type	dt

### Description

Returns the element name. For hierarchical elements, the function returns the name associated with the box. Because hierarchical elements can share the same name, the return value does not necessarily uniquely identify an element. To return a unique name, use the function `stm_r_xx_uniquename`.

Note the following:

- ◆ This function returns a pointer to a static area of memory. Subsequent calls to this procedure will overwrite the old string. If the name needs to be preserved, use the `strdup()` function from the string library.
- ◆ You can call this function without indicating the specific element type, as follows:  

```
stm_r_name (id, status)
```
- ◆ For boxes that have no names, this function returns the definition chart name. For example, for box `@ABC`, this function returns `ABC`.

### Syntax

```
stm_r_xx_name (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>xx_id</code>	In	Stateate element	The element ID
<code>status</code>	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_error_in_file`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_missing_name`
- ◆ `stm_missing_field`
- ◆ `stm_illegal_parameter`
- ◆ `stm_file_not_found`

**Example**

To retrieve and print the name of a state in a statechart, use the following statements:

```
VARIABLE
  STATE      state_id;
  INTEGER    status;
  .
state_id := stm_r_st ('S1.S3',status);
WRITE (stm_r_st_name (state_id, status));
  .
```

In this example, the state name is provided and this value is used to retrieve the same state name from the database. The purpose of this example is to demonstrate the value returned by this function, in contrast to the value returned by the function `stm_r_xx_uniquename`.

### stm\_r\_next\_msg

**Function type:** MESSAGE

**For elements:**

decomposed SD	dec_sd
---------------	--------

#### Description

Returns the message after (in time) the decomposed sequence diagram.

#### Syntax

```
stm_r_next_msg (dec_sd_id, status)
```

#### Arguments

Argument	Input/Output	Type	Description
dec_sd_id	In	REFERENCED_SD	The element ID
status	Out	INTEGER	The function status code

#### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_not\_decomposed\_sd
- ◆ stm\_message\_not\_found

## stm\_r\_xx\_note

**Function type:** LIST OF STRING

**For elements:**

separator	sep
order insignificant line	ord_insig
message	msg
timing constraint	tc

### Description

Returns the note for the specified element in the sequence diagram.

### Syntax

```
stm_r_xx_note (tr_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
tr_id	In	Statestate element	The transition ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_id\_out\_of\_range
- ◆ stm\_missing\_note
- ◆ stm\_success

## stm\_r\_xx\_notes

**Function type:** LIST OF STRING

**For elements:**

chart	ch
transistion	tr

### Description

Returns the note for the specified input transition.

### Syntax

```
stm_r_xx_notes (tr_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
tr_id	In	Stateate element	The transition ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_id_out_of_range`
- ◆ `stm_missing_note`
- ◆ `stm_success`

## stm\_r\_tt\_num\_of\_col

**Function type:** INTEGER

**For elements:**

truth table	tt
-------------	----

### Description

Retrieves the number of columns (including blank ones) in the specified truth table, as viewed in the truth table editor.

### Syntax

```
function stm_r_tt_num_of_col (el, status)
```

### Arguments

Argument	Input/Output	Type	Description
el	In	ELEMENT	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_missing_truth_table`
- ◆ `stm_success`

## stm\_r\_tt\_num\_of\_in

**Function type:** INTEGER

**For elements:**

truth table	tt
-------------	----

### Description

Retrieves the number of input columns in the specified truth table.

### Syntax

```
stm_r_tt_num_of_in (el_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
el_id	In	ELEMENT	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_missing_truth_table`
- ◆ `stm_success`



## stm\_r\_tt\_num\_of\_out

**Function type:** INTEGER

**For elements:**

truth table	tt
-------------	----

### Description

Retrieves the number of output columns in the specified truth table.

### Syntax

```
stm_r_tt_num_of_out (el_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
el_id	In	ELEMENT	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_missing_truth_table`
- ◆ `stm_success`

## stm\_r\_tt\_num\_of\_row

**Function type:** INTEGER

**For elements:**

truth table	tt
-------------	----

### Description

Retrieves the number of rows (including blank ones) in the specified truth table, as viewed in the truth table editor.

### Syntax

```
stm_r_tt_num_of_row (el_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
el_id	In	ELEMENT	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_missing_truth_table`
- ◆ `stm_success`

## stm\_r\_uc\_num\_of\_scen

**Function type:** INTEGER

**For elements:**

use case	uc
----------	----

### Description

Retrieves the number of scenarios for the specified use case.

### Syntax

```
function stm_r_uc_num_of_scen (uc, status)
```

### Arguments

Argument	Input/Output	Type	Description
uc	In	USE_CASE	The use case
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`

## stm\_r\_xx\_number\_of\_bits

**Function type:** `STRING`

**For elements:**

data-item	di
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

### Description

Returns the number of bits in the element.

You can call this function without indicating the specific element type, as follows:

```
stm_r_number_of_bits (id, status)
```

### Syntax

```
stm_r_xx_number_of_bits (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	State element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`

## stm\_r\_xx\_of\_enum\_type

**Function type:** DATA\_TYPE

**For elements:**

data-item	di
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

### Description

Retrieves the enumerated type ID (a user-defined type) for the specified element.

You can call this function without indicating the specific element type, as follows:

```
stm_r_of_enum_type (id, status)
```

### Syntax

```
stm_r_xx_of_enum_type (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	Stamete element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_id\_not\_found
- ◆ stm\_missing\_of\_enum\_type

## stm\_r\_xx\_of\_enum\_type\_name\_type

**Function type:** INTEGER (predefined constant)

**For elements:**

data-item	di
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

### Description

Retrieves the enumerated name type for the specified elements.

You can call this function without indicating the specific element type, as follows:

```
stm_r_of_enum_type_name_type (id, status)
```

### Syntax

```
stm_r_xx_of_enum_type_name_type (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	State element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_missing_of_enum_type`

### Return Types

Although the return value of this function is of type `INTEGER`, the Documentor enables you to reference this value by name. The name is defined internally as a predefined constant in DGL. The possible values are as follows:

- ◆ `stm_ntc_name`
- ◆ `stm_ntc_synonym`
- ◆ `stm_ntc_unknown`

## stm\_r\_ord\_insig\_defined\_in\_ch

**Function type:** LIST OF ORDER\_INSIGNIFICANT

**For elements:**

chart	ch
-------	----

### Description

Returns the list of order-insignificant elements for the specified charts.

### Syntax

```
stm_r_ord_insig_defined_in_ch (ch_lst, status)
```

### Arguments

Argument	Input/Output	Type	Description
ch_lst	In	LIST OF CHART	The list of charts
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_not_order_insignificant`



## stm\_r\_parameter\_binding

**Function type:** `STRING`

### Description

Returns the parameter expression from generic charts and components.

### Syntax

```
stm_r_parameter_binding (xx_paramid_in_gen, inst_boxid, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>xx_paramid_in_gen</code>	In	StateMate element	The element ID
<code>inst_boxid</code>	In	StateMate element	The element ID
<code>status</code>	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_param_not_compatible`
- ◆ `stm_name_not_found`
- ◆ `stm_not_a_parameter`

## stm\_r\_parameter\_mode

**Function type:** `STRING`

**For elements:**

chart	ch
subroutine parameter	sp

### Description

Retrieves the parameter mode, including subroutine parameters and the parameters of generic charts and components.

You can call this function without indicating the specific element type, as follows:

```
stm_r_parameter_mode (xx_id, status)
```

### Syntax

```
stm_r_xx_parameter_mode (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>xx_id</code>	In	State element	The element ID
<code>status</code>	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_not_a_parameter`

### Return Values

Although the return value of this function is of type `INTEGER`, the Documnetor enables you to reference this value by name. The name is defined internally as a predefined constant in DGL. The possible values are as follows:

- ◆ `stm_in_parameter`
- ◆ `stm_out_parameter`
- ◆ `stm_inout_parameter`
- ◆ `stm_constant_parameter`

## stm\_r\_sb\_parameters

**Function type:** LIST OF SUBROUTINE PARAMETERS

**For elements:**

subroutine	sb
------------	----

### Description

Retrieves the parameters of the subroutine.

### Syntax

```
stm_r_sb_paramaters (sb_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
sb_id	In	Stateate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_not_a_parameter`
- ◆ `stm_missing_subroutine_params`

## stm\_r\_en\_parent

**Function type:** DATA\_TYPE

**For elements:**

enumerated value	en
------------------	----

### Description

Returns the parent type of the specified enumerated value.

### Syntax

```
stm_r_en_parent (en, status)
```

### Arguments

Argument	Input/Output	Type	Description
en	In	• ELEMENT	The enumerated value
status	Out	• INTEGER	The function status code

### Status Codes

- ◆ `stm_success`

## stm\_r\_previous\_msg

**Function type:** MESSAGE

**For elements:**

decomposed SD	dec_sd
---------------	--------

### Description

Returns the message previous (in time) to the decomposed sequence diagram.

### Syntax

```
stm_r_previous_msg (dec_sd_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
dec_sd_id	In	REFERENCED_SD	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_not_decomposed_sd`
- ◆ `stm_message_not_found`

## stm\_r\_sb\_proc\_sch\_local\_data

**Function type:** LIST OF LOCAL\_DATA

**For elements:**

subroutine	sb
------------	----

### Description

Retrieves the local data of the procedural statechart implemented by the specified subroutine.

### Syntax

```
stm_r_sb_proc_sch_local_data (sb_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
sb_id	In	Statechart element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_missing\_local\_data
- ◆ stm\_no\_connected\_chart

## stm\_r\_md\_purpose

**Function type:** INTEGER

**For elements:**

module	md
--------	----

### Description

Returns the purpose of the module.

### Syntax

```
stm_r_md_purpose (id, status)
```

### Arguments

Argument	Input/Output	Type	Description
id	In	State element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_unresolved`
- ◆ `stm_not_instance`



## stm\_r\_xx\_reactions

**Function type:** LIST OF STRING

**For elements:**

activity	ac
state	st

### Description

Returns the static reactions of the specified state. The syntax of these reactions is `trigger/action`.

Note the following:

- ◆ To divide the static reactions into their trigger and action parts, use the utility routines `stm_trigger_of_reaction` and `stm_action_of_reaction`.
- ◆ You can call this function without indicating the specific element type, as follows:

```
stm_r_reactions (st_id, status)
```

### Syntax

```
stm_r_xx_reactions (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_st_id	In	Statestate element	The state ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_unresolved`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_missing_label`

### Example

To extract all static reactions of state `s1`, use the following statements:

```
VARIABLE
  STATE          state_id
  INTEGER        status;
  LIST OF STRING reactions;
  STRING         react;
  .
  .
state_id := stm_r_st ('S1', status);
reactions := stm_r_st_reactions (state_id, status);
IF status = stm_success THEN
  FOR react IN reactions LOOP;
  .
  .
  .
```

## stm\_r\_param\_binding\_hyper

**Function type:** `STRING`

### Description

Retrieves the generic instance actual parameter.

### Syntax

```
stm_r_param_binding_hyper (prm_id, ins, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>prm_id</code>	In	StateMate element	The element ID
<code>ins</code>	In	StateMate element	The element ID.
<code>status</code>	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_unresolved`

## stm\_r\_param\_binding\_id

**Function type:** `STRING`

### Description

Retrieves the Generic Instance actual parameter ID.

### Syntax

```
stm_r_param_binding_id (prm_id, ins, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>prm_id</code>	In	StateMate element	The element ID
<code>ins</code>	In	StateMate element	The element ID.
<code>status</code>	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_unresolved`

## stm\_r\_sb\_return\_type

**Function type:** INTEGER (predefined constant)

**For elements:**

subroutine	sb
------------	----

### Description

Retrieves the subroutine's return type.

### Syntax

```
stm_r_sb_return_type (sb_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
sb_id	In	State element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_id\_not\_found

### Return Values

Although the return value of this function is of type `INTEGER`, the Documentor enables you to reference this value by name. The name is defined internally as a predefined constant in DGL.

## stm\_r\_sb\_return\_user\_type

**Function type:** DATA\_TYPE

**For elements:**

subroutine	sb
------------	----

### Syntax

```
stm_r_sb_return_user_type (sb_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
sb_id	In	State element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_missing_user_type`

## stm\_r\_sb\_return\_user\_type\_name

**Function type:** INTEGER (predefined constant)

**For elements:**

subroutine	sb
------------	----

### Description

Retrieves the subroutine's return user type and name type.

### Syntax

```
stm_r_sb_return_user_type_name_type (sb_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
sb_id	In	Stateate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_id\_not\_found
- ◆ stm\_missing\_user\_type

### Return Values

Although the return value of this function is of type `INTEGER`, the Documentor enables you to reference this value by name. The name is defined internally as a predefined constant in DGL. The possible values are as follows:

- ◆ stm\_ntc\_name
- ◆ stm\_ntc\_synonym
- ◆ stm\_ntc\_unknown

### stm\_r\_tt\_row

**Function type:** LIST OF STRING

**For elements:**

truth table	tt
-------------	----

#### Description

Returns a list of strings that represents a row in the truth table. Each string in the list includes the text in the truth table cell. The row's index range is [0..num\_of\_rows-1]. Row 0 returns the list of table header strings.

#### Syntax

```
stm_r_tt_row (el, row_num, status)
```

#### Arguments

Argument	Input/Output	Type	Description
el	In	StateMate element	The element ID
row_num	In	INTEGER	The row number to retrieve
status	Out	INTEGER	The function status code

#### Status Codes

- ◆ stm\_truth\_table\_invalid\_row
- ◆ stm\_missing\_truth\_table
- ◆ stm\_success



## stm\_r\_uc\_scen

**Function type:** LIST OF STRING

**For elements:**

use case	uc
----------	----

### Description

Retrieves the scenario, based on the specified index number.

### Syntax

```
stm_r_uc_scen (uc, scen_index, status)
```

### Arguments

Argument	Input/Output	Type	Description
uc	In	USER_CASE	The use case diagram
scen_index	In	INTEGER	The index for the scenario
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`

**stm\_r\_uc\_scen\_attr\_name****Function type:** LIST OF STRING**For elements:**

use case	uc
----------	----

**Description**

Returns the names of attributes associated with the specified use case scenario. Attributes are associated with elements via element forms.

**Syntax**

```
stm_r_uc_scen_attr_name (uc, scen_index, status)
```

**Arguments**

Argument	Input/Output	Type	Description
uc	In	USE_CASE	The use case.
scen_index	In	INTEGER	The index for the scenario.
status	Out	INTEGER	The function status code. If no attributes exist for the specified element, status receives the value <code>stm_attribute_name_not_found</code> .

**Status Codes**

- ◆ `stm_success`
- ◆ `stm_attribute_name_not_found`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_unresolved`

## stm\_r\_uc\_scen\_attr\_val

**Function type:** LIST OF STRING

**For elements:**

use case	uc
----------	----

### Description

Retrieves attribute values associated with a particular attribute name for the specified use case scenario.

### Syntax

```
stm_r_uc_scen_attr_val (uc, scen_index, attr_name, status)
```

### Arguments

Argument	Input/Output	Type	Description
uc	In	USE_CASE	The use case.
scen_index	In	INTEGER	The index of the scenario.
attr_name	In	STRING	The attribute name. The attribute name is not case-sensitive.
status	Out	INTEGER	The function status code. If <code>attr_name</code> does not exist for the specified element, <code>status</code> receives the value <code>stm_attribute_name_not_found</code> .

Note the following:

- ◆ Attribute values might exist for attributes with no name. Therefore, if you supply contiguous apostrophes ( ' ' ) for `attr_name`, you retrieve all values for unnamed attributes.
- ◆ In most cases, attributes have only one value. However, there are some cases where more than one attribute value is simultaneously meaningful. For example, a module has an attribute `implementation`. The attributes `software` and `hardware` might both be meaningful for some modules. Therefore, Statemate provides the capability of assigning multiple values to attributes, and the function returns a list of these values. When there is a single value, the list consists of one component.

### Status Codes

- ◆ `stm_success`
- ◆ `stm_attribute_name_not_found`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_illegal_name`
- ◆ `stm_unresolved`

## stm\_r\_sd\_scope

**Function type:** CHART

**For elements:**

sequence diagram	sd
------------------	----

### Description

Retrieves the scope of the specified sequence diagram.

### Syntax

```
stm_r_sd_scope (sd, status)
```

### Arguments

Argument	Input/Output	Type	Description
sd	In	CHART	The sequence diagram
status	Out	INTEGER	The function status code

## stm\_r\_xx\_select\_implementation

**Function type:** INTEGER

**For elements:**

action	an
activity	ac
subroutine	sb

### Description

Retrieves the implementation type of the specified element.

### Syntax

```
stm_r_xx_select_implementation (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_out\_of\_range
- ◆ stm\_id\_not\_found

### Return Values

Although the return value of this function is of type `INTEGER`, the Documentor enables you to reference this value by name. The name is defined internally as a predefined constant in DGL. The possible values are as follows:

- ◆ `stm_sb_action_lang`
- ◆ `stm_sb_procedural_sch`
- ◆ `stm_sb_kr_c_code`
- ◆ `stm_sb_ansi_c_code`
- ◆ `stm_sb_ada_code`
- ◆ `stm_sb_vhdl_code`
- ◆ `stm_sb_verilog_code`
- ◆ `stm_sb_truth_table_code`
- ◆ `stm_sb_best_match`
- ◆ `stm_sb_none`

## stm\_r\_st\_static\_reactions

**Function type:** `STRING`

**For elements:**

state	st
-------	----

### Description

Returns the static reactions defined for the specified state element.

### Syntax

```
stm_r_st_static_reactions (st_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>st_id</code>	In	State element	The element ID
<code>status</code>	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_unresolved`
- ◆ `stm_missing_label`



## stm\_r\_st\_static\_reactions\_hyper

**Function type:** `STRING`

**For elements:**

state	st
-------	----

### Description

Returns a string with the static reactions, including hyperlinks to referenced elements.

### Syntax

```
stm_r_st_static_reactions_hyper (elem, format, status)
```

### Arguments

Argument	Input/Output	Type	Description
elem	In	StateMate element	The element ID
format	In	• <code>STRING</code>	Either FrameMaker or Word
status	Out	• <code>INTEGER</code>	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_unresolved`

## stm\_r\_xx\_string\_length

**Function type:** `STRING`

**For elements:**

data-item	di
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

### Description

Retrieves the string length of the specified element.

You can call this function without indicating the specific element type, as follows:

```
stm_r_string_length (id, status)
```

### Syntax

```
stm_r_xx_string_length (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_missing_field`
- ◆ `stm_illegal_parameter`
- ◆ `stm_file_not_found`
- ◆ `stm_error_in_file`

## stm\_r\_xx\_structure\_type

**Function type:** INTEGER

**For elements:**

condition	co
data-item	di
event	ev
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

### Description

Returns the structure or type of the specified textual element. The structure or type can be single, array, or queue.

You can call this function without specifying an element type, as follows:

```
stm_r_structure_type (id, status)
```

### Syntax

```
stm_r_xx_structure_type (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_id\_not\_found

### Return Values

Although the return value of this function is of type `INTEGER`, the Documentor enables you to reference this value by name. The name is defined internally as a predefined constant in DGL. The following are all possible values allowed for each Statemate element subtype:

Element Type	Element Subtype
condition	<code>stm_co_array</code>
	<code>stm_co_missing</code>
	<code>stm_co_single</code>
data-item	<code>stm_di_array</code>
	<code>stm_di_queue</code>
	<code>stm_di_single</code>
event	<code>stm_ev_array</code>
	<code>stm_ev_missing</code>
	<code>stm_ev_single</code>
field	<code>stm_fd_array</code>
	<code>stm_fd_queue</code>
	<code>stm_fd_single</code>
local data	<code>stm_ld_array</code>
	<code>stm_ld_queue</code>
	<code>stm_ld_single</code>
subroutine parameter	<code>stm_sp_array</code>
	<code>stm_sp_queue</code>
	<code>stm_sp_single</code>
user-defined type	<code>stm_dt_array</code>
	<code>stm_dt_queue</code>
	<code>stm_dt_single</code>

## stm\_r\_ac\_subroutine\_bind

**Function type:** LIST OF STRING

**For elements:**

activity	ac
----------	----

### Description

Returns the subroutine binding connected to the specified activity.

### Syntax

```
stm_r_ac_subroutine_bind (ac_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
ac_id	In	Stateate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_unresolved
- ◆ stm\_id\_not\_found
- ◆ stm\_missing\_subroutine\_binding

## stm\_r\_ac\_subroutine\_bind\_enable

**Function type:** INTEGER

**For elements:**

activity	ac
----------	----

### Description

Determines whether the subroutine bound to the specified activity is enabled or disabled.

### Syntax

```
stm_r_ac_subroutine_bind_enable (ac_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
ac_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_unresolved`
- ◆ `stm_id_not_found`

### Return Values

Although the return value of this function is of type `INTEGER`, the Documentor enables you to reference this value by name. The name is defined internally as a predefined constant in DGL. The possible values are as follows:

- ◆ `stm_ac_cbk_enable`
- ◆ `stm_ac_cbk_disable`
- ◆ `stm_ac_cbk_bind_missing`

## stm\_r\_ac\_subroutine\_bind\_expr

**Function type:** `STRING`

**For elements:**

activity	ac
----------	----

### Description

Returns the subroutine binding expression that is connected to the specified activity.

### Syntax

```
stm_r_ac_subroutine_bind_expr (ac_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
ac_id	In	State element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_unresolved`
- ◆ `stm_missing_subroutine_binding`

## stm\_r\_xx\_synonym

**Function type:** `STRING`

**For elements:**

action	an
activity	ac
actor	actor
boundary box	bb
chart	ch
condition	co
data-item	di
data-store	ds
event	ev
field	fd
information-flow	if
lifeline	ll
module	md
router	router
state	st
subroutine	sb
use case	uc
user-defined type	dt

### Description

Retrieves the synonym of the specified element. The synonym is defined in the element's form.

You can call this function without indicating the specific element type, as follows:

```
stm_r_synonym (id, status)
```

### Syntax

```
stm_r_xx_synonym (xx_id, status)
```



## Arguments

Argument	Input/Output	Type	Description
xx_id	In	Stateate element	The element ID.
status	Out	INTEGER	The function status code. If no synonym is defined in the element's form, <code>status</code> receives the value <code>stm_missing_synonym</code> .

## Status Codes

- ◆ `stm_success`
- ◆ `stm_unresolved`
- ◆ `stm_missing_subroutine_params`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_missing_synonym`

## Example

To write out the synonym of activity A1, use the following statements:

```
VARIABLE
  ACTIVITY          act_id;
  INTEGER           status;
.
.
.
act_id := stm_r_ac('A1', status);
WRITE ('\\n Synonym:', stm_r_ac_synonym
      (act_id, status));
.
.
.
```

## stm\_r\_ac\_termination

**Function type:** INTEGER (predefined constant)

**For elements:**

activity	ac
----------	----

### Description

Returns the activity termination type specified in the activity form.

### Syntax

```
stm_r_ac_termination (act_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
act_id	In	ACTIVITY or <b>ELEMENT</b>	The activity whose termination type you want to retrieve
status	Out	INTEGER	Function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_unresolved`

### Return Values

Although the return value of this function is of type `INTEGER`, the Documentor enables you to reference this value by name. The name is defined internally as a predefined constant in DGL. The possible values are:

- ◆ `stm_ac_self_termination`
- ◆ `stm_ac_controlled_termination`
- ◆ `stm_ac_missing`

**Example**

To determine the termination type of the activity A1 and if the activity is self-terminated write the activity's name, use the following statements:

```
VARIABLE
  ACTIVITY      act_id;
  INTEGER       act_term_type;
  INTEGER       status;
  .
  .
  .
act_id := stm_r_ac ('A1', status);
act_term_type:=stm_r_ac_termination (act_id, status);
IF act_term_type = stm_ac_self_termination THEN
  WRITE ('\n Self-terminated activity:', 'A1');
END IF;
  .
  .
  .
```

## stm\_r\_xx\_truth\_table

**Function type:** LIST OF STRING

**For elements:**

action	an
activity	ac
subroutine	sb

### Description

Returns the elements that are implemented as truth tables.

### Syntax

```
stm_r_xx_truth_table (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	Stategate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_error\_in\_file
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_illegal\_parameter
- ◆ stm\_id\_not\_found
- ◆ stm\_file\_not\_found
- ◆ stm\_missing\_name
- ◆ stm\_missing\_field

## stm\_r\_xx\_truth\_table\_expression

**Function type:** `STRING`

**For elements:**

action	an
activity	ac
subroutine	sb

### Purpose

Returns the truth table expression for all named elements.

### Syntax

```
stm_r_xx_truth_table_expression (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	Statebate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_error_in_file`
- ◆ `stm_id_out_of_range`
- ◆ `stm_illegal_parameter`
- ◆ `stm_id_not_found`
- ◆ `stm_file_not_found`
- ◆ `stm_missing_name`
- ◆ `stm_missing_field`

## stm\_r\_sb\_truth\_table\_local\_data

**Function type:** LIST OF LOCAL\_DATA

**For elements:**

action	an
activity	ac
subroutine	sb

### Description

Returns the list of local data elements defined in the truth table related to the input subroutine.

### Syntax

```
stm_r_xx_truth_table_local_data (sb_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
sb_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_error\_in\_file
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_illegal\_parameter
- ◆ stm\_id\_not\_found
- ◆ stm\_file\_not\_found
- ◆ stm\_missing\_name
- ◆ stm\_missing\_field

## stm\_r\_xx\_type

**Function type:** INTEGER (predefined constant)

**For elements:**

a-flow-line (basic)	ba
a-flow-line (compound)	af
action	an
activity	ac
actor	actor
boundary box	bb
chart	ch
condition	co
connector	cn
data-item	di
data-store	ds
event	ev
field	fd
function	fn
information-flow	if
lifeline	ll
module	md
module-occurrence	om
router	router
state	st
use case	uc
user-defined type	dt

### Description

Retrieves element subtypes for the specified element. Most Statemate elements are divided into classes, referred to as *subtypes*. For example, a state might belong to one of a number of subtypes, such as *and*, *or*, *basic*, *diagram*, *instance*, or *reference*.

You can call this function without indicating the specific element type, as follows:

```
stm_r_type (id, status)
```

### Syntax

```
stm_r_xx_type (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_id\_not\_found

### Return Values

Although the return value of this function is of type `INTEGER`, the Documentor enables you to reference this value by name. The name is internally defined as a predefined constant in DGL. The following table lists the possible values:

Element Type	Function Type	Element Subtype
a-flow-line	stm_a_flow_line_type	stm_af_control
		stm_af_data
action	stm_action_type	stm_an_compound
		stm_an_reference
activity	stm_activity_type	stm_ac_control
		stm_ac_control_instance
		stm_ac_diagram
		stm_ac_external
		stm_ac_instance
		stm_ac_internal
		stm_ac_reference



Element Type	Function Type	Element Subtype
chart	stm_chart_type	stm_ch_activity
		stm_ch_module
		stm_ch_reference_activity
		stm_ch_reference_module
		stm_ch_reference_state
		stm_ch_state
condition	stm_condition_type	stm_co_compound
		stm_co_primitive
		stm_co_reference
connector	stm_connector_type	stm_cn_composition
		stm_cn_condition
		stm_cn_control
		stm_cn_deep_history
		stm_cn_default
		stm_cn_diagram
		stm_cn_history
		stm_cn_joint
		stm_cn_junction
		stm_cn_selection
		stm_cn_termination
data-item	stm_data_item_type	stm_di_compound
		stm_di_alias
		stm_di_constant
		stm_di_primitive
		stm_di_reference
data-store	stm_data_store_type	stm_ds_internal
		stm_ds_reference
event	stm_event_type	stm_ev_compound
		stm_ev_primitive
		stm_ev_reference
field	stm_field_type	stm_fd_primitive
information-flow	stm_information_flow_type	stm_if_explicit
		stm_if_reference

Element Type	Function Type	Element Subtype
module	stm_module_type	stm_md_diagram
		stm_md_subsystem
		stm_md_environment
		stm_md_reference
		stm_md_instance
		stm_md_storage_module
router	stm_router_type	stm_router_external
		stm_router_internal
state	stm_state_type	stm_st_diagram
		stm_st_and
		stm_st_or
		stm_st_instance
		stm_st_reference
		stm_st_basic
subroutine	stm_subroutine_type	stm_sb_reference
user-defined type		stm_dt_primitive
		stm_dt_reference

### Example

To retrieve the type of state `READY` and execute some statements if the state is an `or` state, use the following statements:

```
VARIABLE
    STATE          st_id;
    INTEGER        st_type;
    INTEGER        status;
    .
    .
    .
    st_id := stm_r_st ('READY', status);
    st_type := stm_st_type (st_id, status);
    IF st_type = stm_st_or THEN
    .
    .
    .
```

If `READY` is an or-state, the statements following `THEN` are executed.

## stm\_r\_xx\_type\_expression

**Function type:** `STRING`

**For elements:**

condition	co
data-item	di
event	ev
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

### Description

Returns the type expression for the specified element. The expression is the same as used in the properties, reports, and Info.

You can call this function without indicating the specific type, as follows:

```
stm_r_type_expression (id, status)
```

### Syntax

```
stm_r_xx_type_expression (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_unresolved`

## stm\_r\_xx\_uniquename

**Function type:** `STRING`

**For elements:**

action	an
activity	ac
actor	actor
boundary box	bb
condition	co
data-item	di
data-store	ds
event	ev
field	fd
function	fn
information-flow	if
lifeline	ll
local data	ld
module	md
router	router
state	st
subroutine	sb
subroutine parameter	sp
use case	uc
user-defined type	dt

### Description

Returns the unique path name for the specified element. The name returned by the function contains the minimum number of levels necessary to uniquely identify an element in its chart. It is especially relevant to boxes.

You can call this function without indicating the specific element type, as follows:

```
stm_r_uniquename (id, status)
```

### Syntax

```
stm_r_xx_uniquename (xx_id, status)
```

### Arguments

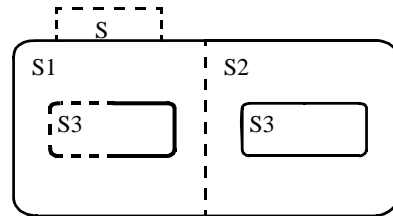
Argument	Input/Output	Type	Description
xx_id	In	Statestate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_id\_not\_found
- ◆ stm\_missing\_synonym

### Example

Consider the following statechart:



To retrieve the unique name of the highlighted element, use the following statements:

```
VARIABLE
    STATE          state_id;
    INTEGER        status;
    .
    :
    .
state_id := stm_r_st ('S1.S3', status);
WRITE ('\n Unique Name:', (stm_r_st_uniquename (
    state_id, status)));
    .
    :
```

The state name printed is `s1.s3` (not `s.s1.s3` or `s3`). In this example, a unique state name is provided, and this value is used to retrieve the same unique state name from the database. This example demonstrates the value returned by this function, in contrast to the value returned by the function `stm_r_xx_name`.

## stm\_r\_ch\_usage\_type

**Function type:** INTEGER (predefined constant)

**For elements:**

chart	ch
-------	----

### Description

Returns the usage type for a chart.

### Syntax

```
stm_r_ch_usage_type (ch_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
ch_id	In	State element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_id\_not\_found

### Return Value

Although the return value of this function is of type `INTEGER`, the Documentor enables you to reference this value by name. The name is defined internally as a predefined constant in DGL. The possible values are as follows:

- ◆ stm\_ch\_usage\_generic
- ◆ stm\_ch\_usage\_normal
- ◆ stm\_ch\_usage\_ref\_generic
- ◆ stm\_ch\_usage\_ref\_offpage
- ◆ stm\_ch\_usage\_ref\_describing

## stm\_r\_xx\_user\_type

**Function type:** DATA\_TYPE

**For elements:**

data-item	di
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

### Description

Returns the user-defined type ID referenced by the element.

You can call this function without indicating the specific element type, as follows:

```
stm_r_user_type (id, status)
```

### Syntax

```
stm_r_xx_user_type (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`
- ◆ `stm_missing_user_type`



## stm\_r\_xx\_user\_type\_name\_type

**Function type:** INTEGER

**For elements:**

data-item	di
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

### Description

Returns the name type of the user-defined type referenced by the element.

You can call this function without indicating the specific element type, as follows:

```
stm_r_user_type_name_type (id, status)
```

### Syntax

```
stm_r_xx_user_type_name_type (xx_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
xx_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_ntc\_name
- ◆ stm\_ntc\_synonym
- ◆ stm\_ntc\_unknown

## stm\_r\_ch\_version

**Function type:** `STRING`

**For elements:**

chart	ch
-------	----

Returns the version of the specified chart.

### Syntax

```
stm_r_ch_version (ch, status)
```

### Arguments

Argument	Input/Output	Type	Description
ch	In	CHART	The chart whose version you want to retrieve
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_unresolved`

## stm\_r\_gds\_visibility\_mode

**Function type:** INTEGER

**For elements:**

element ID	
------------	--

### Description

Returns the visibility mode for the specified global definition set (GDS).

### Syntax

```
stm_r_gds_visibility_mode (gds_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
gds_id	In	CHART	The GDS whose visibility you want to retrieve
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_id\_out\_of\_range
- ◆ stm\_success

### Return Values

Although the return value of this function is of type INTEGER, the Documentor enables you to reference this value by name. The name is internally defined as a predefined constant in DGL. The possible values are as follows:

- ◆ stm\_explicit\_usage
- ◆ stm\_public\_usage

## stm\_r\_msg\_where\_tc\_begins

**Function type:** MESSAGE

**For elements:**

message	msg
---------	-----

### Description

Returns the message where the timing constraint begins.

### Syntax

```
stm_r_msg_where_tc_begins (tc_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
tc_id	In	TIMING CONSTRAINT	The timing constraint
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_not_timing_constraint`

## stm\_r\_msg\_where\_tc\_ends

**Function type:** MESSAGE

**For elements:**

chart	ch
-------	----

### Description

Returns the message where the timing constraint ends.

### Syntax

```
stm_r_msg_where_tc_ends (tc_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
tc_id	In	TIMING CONSTRAINT	The timing constraint
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_not_timing_constraint`

## stm\_r\_sb\_connected\_statechart

**Function type:** `stm_list`

**For elements:**

subroutine	sb
------------	----

### Description

Returns the ID of the procedural Statechart connected to the specified subroutine.

### Syntax

```
STM_R_SB_CONNECTED_STATECHART(IN sb: SUBROUTINE, OUT st: INTEGER): CHART;
```

### Arguments

Argument	Input/Output	Type	Description
<code>sb_id</code>	In	<code>stm_id</code>	The element ID.
<code>status</code>	Out	<code>int</code>	The function status code.

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_no_connected_chart`

## stm\_r\_sb\_connected\_flowchart

### Function type:

stm\_list

### For elements:

subroutine	sb
------------	----

### Description

Returns the ID of the Flowchart connected to the specified subroutine.

### Syntax

```
STM_R_SB_CONNECTED_FLOWCHART(IN sb: SUBROUTINE, OUT st: INTEGER): CHART;
```

### Arguments

Argument	Input/Output	Type	Description
sb_id	In	stm_id	The element ID.
status	Out	int	The function status code.

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_not\_found
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_no\_connected\_chart

## stm\_r\_sb\_proc\_fch\_local\_data

**Function type:** LIST OF LOCAL\_DATA

**For elements:**

subroutine	sb
------------	----

### Description

Retrieves the local data of the procedural flowchart implemented by the specified subroutine.

### Syntax

```
stm_r_sb_proc_fch_local_data (sb_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
sb_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_missing\_local\_data
- ◆ stm\_no\_connected\_chart



## stm\_r\_xx\_des\_attr\_val

**Function type:** LIST OF STRING

**For elements:**

activity	ac
action	an
chart	ch
condition	co
data-item	di
data-type	dt
event	ev
field	fd
information-flow	if
local data	ld
state	st
subroutine	sb
subroutine parameter	sp
transitions	tr

### Description

Retrieves design-attribute values associated with a particular design-attribute name for the specified element.

You can call this function without indicating the specific element type, as follows:

```
stm_r_des_attr_val (id, attr_name, status)
```

### Syntax

```
stm_r_xx_des_attr_val(xx_id, des_attr_name, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>xx_id</code>	In	State element	The element ID.
<code>attr_name</code>	In	STRING	The design-attribute name. The design-attribute name is not case-sensitive.
<code>status</code>	Out	INTEGER	The function status code. If <code>attr_name</code> does not exist for the specified element, <code>status</code> receives the value <code>stm_attribute_name_not_found</code> .

### Status Codes

- ◆ `stm_success`
- ◆ `stm_des_attribute_name_not_found`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_illegal_name`
- ◆ `stm_unresolved`

## stm\_r\_xx\_des\_attr\_name

**Function type:** LIST OF STRING

**For elements:**

activity	ac
action	an
chart	ch
condition	co
data-item	di
data-type	dt
event	ev
field	fd
information-flow	if
local data	ld
state	st
subroutine	sb
subroutine parameter	sp
transitions	tr

### Description

Returns the names of design-attributes associated with the specified element. Design-attributes are associated with elements via element forms.

You can call this function without indicating the specific type, as follows:

```
stm_r_des_attr_name(id, status)
```

### Syntax

```
stm_r_xx_des_attr_name(xx_id, status)
```

### Arguments

Argument	Input/ Output	Type	Description
<code>xx_id</code>	In	Statestate element	The element ID.
<code>attr_name</code>	In	STRING	The design-attribute name. The design-attribute name is not case-sensitive.
<code>status</code>	Out	INTEGER	The function status code. If <code>attr_name</code> does not exist for the specified element, <code>status</code> receives the value <code>stm_attribute_name_not_found</code> .

### Status Codes

- ◆ `stm_success`
- ◆ `stm_des_attribute_name_not_found`
- ◆ `stm_id_not_found`
- ◆ `stm_id_out_of_range`
- ◆ `stm_illegal_name`
- ◆ `stm_unresolved`

## stm\_r\_tt\_cell\_hyper

**Function type:** STRING

**For elements:**

action	an
activity	ac
subroutine	sb

### Description

Retrieves the contents of the specified cell in the given truth table, including hyperlinks to referenced elements.

### Syntax

```
STM_R_TT_CELL_HYPER(IN el: ELEMENT ,IN row_num: INTEGER,IN col_num:
INTEGER ,IN format: STRING, OUT status:INTEGER): STRING;
```

### Arguments

Argument	Input/Output	Type	Description
el	In	StateMate element	The element ID
row_num	In	INTEGER	The row number of the cell
col_num		INTEGER	The column number of the cell
format	In	STRING	Either FrameMaker or Word
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_id\_not\_found
- ◆ stm\_unresolved
- ◆ stm\_missing\_truth\_table
- ◆ stm\_truth\_table\_invalid\_row
- ◆ stm\_truth\_table\_invalid\_column

## stm\_r\_tt\_row\_hyper

**Function type:** LIST OF STRING

**For elements:**

action	an
activity	ac
subroutine	sb

### Description

Returns a list of strings that represents a row in the truth table, including hyperlinks to referenced elements. Each string in the list includes the text in the truth table cell. The row's index range is [0..num\_of\_rows-1]. Row 0 returns the list of table header strings.

### Syntax

```
STM_R_TT_ROW_HYPER(IN el: ELEMENT ,IN row_num: INTEGER ,IN format: STRING,  
OUT status: INTEGER): LIST OF STRING;
```

### Arguments

Argument	Input/Output	Type	Description
el	In	Stamemate element	The element ID
row_num	In	INTEGER	The row number of the cell
formatat	In	STRING	Either FrameMaker or Word
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_id\_not\_found
- ◆ stm\_unresolved
- ◆ stm\_missing\_truth\_table
- ◆ stm\_truth\_table\_invalid\_row
- ◆ stm\_truth\_table\_invalid\_column

## stm\_r\_xx\_default\_val

**Function type:** `STRING`

**For elements:**

condition	co
data-item	di
data-type	dt
local data	ld
field	fd

### Description

Returns the default-value associated with the specified element.

### Syntax

```
stm_r_xx_default_val(id, status)
```

### Arguments

Argument	Input/Output	Type	Description
id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_id_not_found`

## stm\_r\_component\_param\_binding

**Function type:** `STRING`

**For elements:**

activity	ac
condition	co
data-item	di
event	ev

### Description

Returns the value bound to the formal parameter of the specified component's instance.

### Syntax

```
STM_R_COMPONENT_PARAM_BINDING(IN ins: ELEMENT, IN formal:STRING, OUT status:
INTEGER):STRING;
```

### Arguments

Argument	Input/Output	Type	Description
inst_boxid	In	Stemate element	The element ID.
formal_param_name	In	STRING	The formal parameter name. If this is a data-element (from the information stub matrix in the DDE), the function returns the corresponding data-element. If this argument is the stub's name, the function returns the information flowing on the arrow connected to that stub.
status	Out	int	The function status code.

### Status Codes

- ◆ `stm_success`
- ◆ `stm_id_out_of_range`
- ◆ `stm_name_not_found`



## stm\_r\_component\_param\_mode

**Function type:** INTEGER

**For elements:**

activity	ac
condition	co
data-item	di
event	ev

### Description

Returns the mode of the formal parameter of the specified component's instance.

### Syntax

```
STM_R_COMPONENT_PARAM_MODE(IN ins: ELEMENT, IN formal:STRING, OUT status:
INTEGER):STRING;
```

### Arguments

Argument	Input/Output	Type	Description
inst_boxid	In	Stemate element	The element ID.
formal_param_name	In	STRING	The formal parameter name. If this is a data-element (from the information stub matrix in the DDE), the function returns the corresponding data-element. If this argument is the stub's name, the function returns the information flowing on the arrow connected to that stub.
status	Out	int	The function status code.

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_name\_not\_found

## stm\_r\_stubs\_names

**Function type:** LIST OF STRING

**For elements:**

activity	ac
----------	----

### Description

Returns a list of stub names of the specified component instance.

### Syntax

```
STM_R_STUBS_NAMES(IN ins: ELEMENT, OUT status: INTEGER):LIST OF STRING;
```

### Arguments

Argument	Input/Output	Type	Description
inst_boxid	In	Stateate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_name\_not\_found

## stm\_r\_information\_stub\_names

**Function type:** LIST OF STRING

**For elements:**

activity	ac
----------	----

### Description

Returns a list of stub names flowing through an info-flow of the specified component instance.

### Syntax

```
STM_R_INFORMATION_STUB_NAMES((IN ins: ELEMENT, IN formal: STRING, OUT status:
INTEGER):LIST OF STRING;
```

### Arguments

Argument	Input/Output	Type	Description
inst_boxid	In	State element	The element ID
formal_param_name	In	STRING	The formal parameter name
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_name\_not\_found

## stm\_r\_sb\_connected\_statechart

**Function type:** CHART

**For elements:**

subroutine	sb
------------	----

### Description

Returns the ID of the procedural statechart connected to the specified subroutine.

### Syntax

```
stm_r_sb_connected_statechart (sb_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
sb_id	In	Statechart element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_not\_found
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_no\_connected\_chart

## stm\_r\_sb\_connected\_flowchart

**Function type:** CHART

**For elements:**

subroutine	sb
------------	----

### Description:

Returns the ID of the procedural flowchart connected to the specified subroutine.

### Syntax

```
stm_r_sb_connected_flowchart (sb_id, status)
```

### Arguments

Argument	Input/Output	Type	Description
sb_id	In	StateMate element	The element ID
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_id\_not\_found
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_no\_connected\_chart



# Query Functions

---

This section documents each query function, its purpose, and other notes. The functions are organized into sections by “element types returned by functions.” Within each section, functions are organized by “type of element in the input list.” Each function description includes the query from the Query Model tool of Rhapsody that matches it, if applicable.

The topics are as follows:

- ◆ [Calling Query Functions](#)
- ◆ [Query Function Input Arguments](#)
- ◆ [Examples of Query Functions](#)

Query functions extract lists of elements from the database that conform to a specific criterion.

The search enables you to query the Statemate database. This tool uses a comprehensive set of predefined queries to obtain information. All these queries operate on a list of Statemate elements, called the *input list*. Each query generates an output list of elements that meet a criterion designated by the specific query. Generally, elements in the output list are related to elements in the input list in one of two ways:

- ◆ The output list is a subset of input list elements that have a specific characteristic. For example, the output list consists of all And-states in the input list.
- ◆ Elements in the output list fulfill a specific relationship to elements in the input list. For example, the output list consists of all states that are descendants of states in the input list.

Most query functions correspond to queries from the search. These functions give you the same information that the corresponding queries do. Most functions require you to provide an input list as an input argument. This input list generally consists of elements of a particular type. The function returns a list of elements of the same or different type (as the input list).

The retrieval process is as follows:

1. Generate the input list.
2. Specify the query and input list. Receive the input list. Note that other procedures may be performed before you use the retrieved information.
3. Use the output list.

## Calling Query Functions

Most of the query functions use the following calling sequence:

```
stm_r_yy_relation_xx (xx_list, status)
```

In this syntax:

- ◆ **stm\_r\_**—Designates the function as a Statemate database retrieval function.
- ◆ **yy**—The two-character type abbreviation for elements in the output list.
- ◆ **relation**—The relationship between the input and output lists (describes the query to be applied to the input list).
- ◆ **xx**—The two-character type abbreviation for elements in the input list.
- ◆ **xx\_list**—The input list to the function.
- ◆ **status**—The return function status code. There are three possible status codes: `stm_success`, `stm_nil_list`, and `stm_missing_element_in_list`.

For example:

```
stm_r_st_and_st (state_list, status)
```

This function returns the states from the input list `state_list` that are and-states.

The following function returns the activities performed throughout the states in `state_list`:

```
stm_r_st_ac_throughout_st (state_list, status)
```

The following sections document the query functions that use a different calling sequence.

### By Attributes

The `by_attributes` function returns all elements in the input list that have an attribute `attr_name`, whose value is `attr_val`.

The syntax is as follows:

```
stm_r_xx_by_attributes_xx (xx_list, attr_name, attr_val, status)
```

In this syntax:

- ◆ **stm\_r\_**—Designates the function as a Statemate database retrieval function.
- ◆ **xx**—The two-character type abbreviation for elements in the input and output lists.
- ◆ **by\_attributes**—The criterion to be met by elements in the input list.
- ◆ **xx\_list**—The input list to the function.
- ◆ **attr\_name**—A pattern for the attribute name.



- ◆ **attr\_val**—A pattern for the attribute value to be matched.
- ◆ **status**—The return function status code. There are three possible status codes: `stm_success`, `stm_nil_list`, and `stm_missing_element_in_list`.

For example:

```
stm_r_md_by_attributes_md (module_list, 'LANGUAGE',
    'PASCAL', status)
```

This function returns all modules in `module_list` that have an attribute `LANGUAGE`, whose value is `PASCAL`.

## By Structure Type

The `by_structure_type` function returns all elements in the input list that have a structure type `xx_structure_type`.

The syntax of the `by_structure_type` function is as follows:

```
stm_r_xx_by_structure_type_xx (xx_list, xx_structure_type, status)
```

In this syntax:

- ◆ **stm\_r\_**—Designates the function as a Statemate database retrieval function
- ◆ **xx**—The two-character type abbreviation for elements in the input and output list
- ◆ **by\_structure\_type**—The structure type referenced
- ◆ **xx\_list**—The input list to the function
- ◆ **xx\_structure\_type**—The structure type referenced (array, single, or queue in the element's form)
- ◆ **status**—The return function status code

For example:

```
stm_r_di_by_structure_type_di (di_list, stm_di_array, status)
```

This function returns all data items in `di_list` that have an array structure type.

## Name and Synonym Patterns

The `name_of` and `synonym_of` functions search the entire database for elements whose name (or synonym) matches the pattern specified in the argument `pattern`.

The syntax is as follows:

```
stm_r_xx_name_of_xx (pattern, status)
stm_r_xx_synonym_of_xx (pattern, status)
```

In this syntax:

- ◆ **stm\_r\_**—Designates the function as a Statemate database retrieval function
- ◆ **xx**—The two-character type abbreviation of elements in the output list
- ◆ **name\_of** or **synonym\_of**—The criterion to be met (specifies that the query “Element whose name matches a pattern” or “Element whose synonym matches a pattern” is to be applied)
- ◆ **pattern**—A character string you supply as an input argument
- ◆ **status**—The return function status code

For example:

```
stm_r_ev_name_of_ev ('EV*', status)
```

This function returns all events from the database whose names begin with the string `EV`.

## Query Function Input Arguments

Most query functions operate on an input list that you specify. The following table lists the arguments for query functions.

Argument	Description	DGL Data Type
xx_list	The input list of elements upon which to perform the query.	List of Statemate element type (for example, LIST OF STATE, LIST OF ELEMENT, and so on).
attribute name	The pattern for the name of an attribute defined in the attribute field of a Statemate element form. Two special characters can be used as wildcards; the question mark (?) and the asterisk (*). A question mark indicates that any single character can occupy that position. An asterisk indicates that any number of characters (including 0) can occupy that position.	String
attribute value	The pattern for the value of an attribute defined in the attribute field of a Statemate element form. The pattern can include wildcards (? and *).	String
pattern	An alphanumeric string to match a Statemate element name (or synonym). The pattern can include wildcards (? and *).	String

## Examples of Query Functions

This section shows how to use query function calls to perform common tasks.

### Query Function Example 1

The following example shows how to build an input list for the query functions:

```
VARIABLE
ACTIVITY          act_id;
LIST OF ACTIVITY  act_list;
INTEGER           status;
.
.
act_id := stm_r_ac ('A1', status);
act_list := stm_r_ac_physical_sub_of_ac ({act_id},
status);
.
.
```

The variable `act_id` contains the ID of the activity A1. You built the input list by enclosing `act_id` in curly braces (`{}`). In this case, the input list consists of one element. If you want to build an input list of multiple elements, enclose them all in curly braces, separated by commas.

#### Note

---

The input list is built from element IDs, *not* element names.

### Query Function Example 2

The following example shows how you use query functions in succession. Assume that you want to know all the basic states that are descendants of the state `s1`.

```
VARIABLE
STATE             st_id;
INTEGER           status;
LIST OF STATE     descen_states, basic_states;
.
.
st_id := stm_r_st ('S1', status);
descen_states := stm_r_st_physical_desc_of_st (
{st_id}, status);
basic_states := stm_r_st_basic_st (descen_states,
status);
.
.
```

Note that `descen_states` is *not* enclosed in braces because its value comprises a list of states.

## Query Function Example 3

The following example shows how to use query function calls in a place where lists can appear (for example, in FOR statements):

```
VARIABLES
  EVENT      e;
  INTEGER    status;
  .
  .
FOR e IN stm_r_ev_name_of_ev ('EV*', status)
  LOOP
    WRITE ('\n', stm_r_ev_name (e, status));
  END LOOP;
```

This example writes out the name of all the events in the database whose name begins with the string EV.

## List of Query Functions

The query functions are grouped alphabetically first by output list type, then by input list type. The output types are as follows:

- ◆ [Activities \(ac\)](#)
- ◆ [A-Flow-Lines \(af, ba, laf\)](#)
- ◆ [Actions \(an\)](#)
- ◆ [Charts \(ch\)](#)
- ◆ [Connectors \(cn\)](#)
- ◆ [Conditions \(co\)](#)
- ◆ [Data-Items \(di\)](#)
- ◆ [Data-Stores \(ds\)](#)
- ◆ [User-Defined Types \(dt\)](#)
- ◆ [Events \(ev\)](#)
- ◆ [Fields \(fd\)](#)
- ◆ [Functions \(fn\)](#)
- ◆ [Information-Flows \(if\)](#)
- ◆ [M-Flow-Lines \(bf, bm, lmf, mf\)](#)
- ◆ [Modules \(md\)](#)
- ◆ [Mixed \(mx\)](#)
- ◆ [Routers \(router\)](#)

- ◆ [Subroutines \(sb\)](#)
- ◆ [States \(st\)](#)
- ◆ [Timing Constraint \(tc\)](#)
- ◆ [Transitions \(tr\)](#)

## Activities (ac)

This section documents the query functions that return a list of activities.

### Input List Type: ac

<b>stm_r_ac_associates_uc</b>	<p><b>Query:</b> Activities by use cases</p> <p><b>Purpose:</b> This API returns the activities that associate with use-cases in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_ac_associates_uc(IN uc_lst: LIST OF USE_CASE, OUT status: INTEGER)</pre>
<b>stm_r_ac_basic_ac</b>	<p><b>Query:</b> Basic activities</p> <p><b>Purpose:</b> Returns the activities in the input list that have no descendants</p> <p><b>Syntax:</b></p> <pre>stm_r_ac_basic_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</pre>
<b>stm_r_ac_by_attributes_ac</b>	<p><b>Query:</b> Activities by attributes</p> <p><b>Purpose:</b> Returns the activities in the input list that match the specified attribute name and value</p> <p><b>Syntax:</b></p> <pre>stm_r_ac_by_attributes_ac (IN ac_list: LIST OF ACTIVITY, IN attr_name: STRING, IN attr_value: STRING, OUT status: INTEGER)</pre>
<b>stm_r_ac_callback_binding_ac</b>	<p><b>Query:</b> Activities with callback bindings</p> <p><b>Purpose:</b> Returns the activities in the input list that have callback bindings</p> <p><b>Syntax:</b></p> <pre>stm_r_ac_callback_binding_ac (IN el_list: LIST OF ACTIVITY, OUT status: INTEGER)</pre>

<b>stm_r_ac_component_instance_ac</b>	<p><b>Query:</b> Activities that are instances of components</p> <p><b>Purpose:</b> Returns the activities in the input list that have instances of components</p> <p><b>Syntax:</b> stm_r_ac_component_instance_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_ac_continuous_instance_ac</b>	<p><b>Query:</b> Activities with continuous instances</p> <p><b>Purpose:</b> Returns the activities in the input list that have continuous instances</p> <p><b>Syntax:</b> stm_r_ac_continuous_instance_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_ac_control_ac</b>	<p><b>Query:</b> Control activities</p> <p><b>Purpose:</b> Returns the activities in the input list that are control activities</p> <p><b>Syntax:</b> stm_r_ac_control_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_ac_control_terminated_ac</b>	<p><b>Query:</b> Controlled-terminated activities</p> <p><b>Purpose:</b> Returns the activities in the input list that are control-terminated activities</p> <p><b>Syntax:</b> stm_r_ac_control_terminated_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_ac_data_store_ac</b>	<p><b>Query:</b> Data-stores</p> <p><b>Purpose:</b> Returns the activities in the input list that are data-stores</p> <p><b>Syntax:</b> stm_r_ac_data_store_ac (IN el_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_ac_def_of_instance_ac</b>	<p><b>Query:</b> Definition activities of a given activity</p> <p><b>Purpose:</b> Returns the definition activities (top-level in the definition chart) for instances in the input list</p> <p><b>Syntax:</b> stm_r_ac_def_of_instance_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_ac_defined_environment_ac</b>	<p><b>Query:</b> Environment activities</p> <p><b>Purpose:</b> Returns the activities in the input list that were defined as environment activities</p> <p><b>Syntax:</b> stm_r_ac_defined_environment_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>

<b>stm_r_ac_explicit_defined_ac</b>	<p><b>Query:</b> Activities explicitly defined</p> <p><b>Purpose:</b> Returns from the input list those activities that were explicitly defined</p> <p><b>Syntax:</b>  <code>stm_r_ac_explicit_defined_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</code></p>
<b>stm_r_ac_external_ac</b>	<p><b>Query:</b> External activities</p> <p><b>Purpose:</b> Returns the activities in the input list that are external</p> <p><b>Syntax:</b>  <code>stm_r_ac_external_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</code></p>
<b>stm_r_ac_generic_instance_ac</b>	<p><b>Query:</b> Generic instance activities</p> <p><b>Purpose:</b> Returns the activities in the input list that are instances of generic charts</p> <p><b>Syntax:</b>  <code>stm_r_ac_generic_instance_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</code></p>
<b>stm_r_ac_imp_best_match_ac</b>	<p><b>Query:</b> Activities whose selected implementation is <b>Best Match</b></p> <p><b>Purpose:</b> Returns the activities in the input list that are implemented as the <b>Best Match</b> using <b>Select Implementation</b> in the properties</p> <p><b>Syntax:</b>  <code>stm_r_ac_imp_best_match_ac (IN el_list: LIST OF ACTIVITY, OUT status: INTEGER)</code></p>
<b>stm_r_ac_imp_mini_spec_ac</b>	<p><b>Query:</b> Activities implemented in a mini-spec</p> <p><b>Purpose:</b> Returns the activities in the input list that are implemented in a mini-spec</p> <p><b>Syntax:</b>  <code>stm_r_ac_imp_mini_spec_ac (IN el_list: LIST OF ACTIVITY, OUT status: INTEGER)</code></p>
<b>stm_r_ac_imp_none_ac</b>	<p><b>Query:</b> Activities whose selected implementation is <b>None</b></p> <p><b>Purpose:</b> Returns the activities in the input list that are not implemented</p> <p><b>Syntax:</b>  <code>stm_r_ac_imp_none_ac (IN el_list: LIST OF ACTIVITY, OUT status: INTEGER)</code></p>
<b>stm_r_ac_imp_sb_bind_ac</b>	<p><b>Query:</b> Activities implemented with subroutine bindings</p> <p><b>Purpose:</b> Returns the activities in the input list that are implemented as <b>Subroutine Binding</b> using <b>Select Implementation</b> in the properties</p> <p><b>Syntax:</b>  <code>stm_r_ac_imp_sb_bind_ac (IN el_list: LIST OF ACTIVITY, OUT status: INTEGER)</code></p>



<b>stm_r_ac_imp_truth_table_ac</b>	<p><b>Query:</b> Activities implemented in a truth table</p> <p><b>Purpose:</b> Returns the activities in the input list that were implemented in a truth table</p> <p><b>Syntax:</b> stm_r_ac_imp_truth_table_ac (IN el_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_ac_instance_ac</b>	<p><b>Query:</b> Instance activities</p> <p><b>Purpose:</b> Returns those activities in the input list that are instances</p> <p><b>Syntax:</b> stm_r_ac_instance_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_ac_instance_of_def_ac</b>	<p><b>Query:</b> Instance activities of a given definition activity</p> <p><b>Purpose:</b> Returns the instance activities for definition activities (top-level activities in a definition chart) in the input list</p> <p><b>Syntax:</b> stm_r_ac_instance_of_def_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_ac_internal_ac</b>	<p><b>Query:</b> Internal activities</p> <p><b>Purpose:</b> Returns the activities in the input list that are internal activities (not external or control)</p> <p><b>Syntax:</b> stm_r_ac_internal_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_ac_is_occurrence_of_ac</b>	<p><b>Query:</b> Activity occurrences of a given activity</p> <p><b>Purpose:</b> Returns the activities for which the activities in the input list appear in the <b>Is activity</b> field of their form</p> <p><b>Syntax:</b> stm_r_ac_is_occurrence_of_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_ac_is_principal_of_ac</b>	<p><b>Query:</b> Principal activities of a given activity</p> <p><b>Purpose:</b> Returns the activities appearing in the <b>Is activity</b> field of the activities in the input list</p> <p><b>Syntax:</b> stm_r_ac_is_principal_of_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_ac_logical_desc_of_ac</b>	<p><b>Query:</b> Logical descendants of a given activity</p> <p><b>Purpose:</b> Returns the logical descendants of the activities in the input list, taking into account the translation of instances to their definition charts</p> <p><b>Syntax:</b> stm_r_ac_logical_desc_of_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>

<p><b>stm_r_ac_logical_parent_of_ac</b></p>	<p><b>Query:</b> Logical parent activities of a given activity  <b>Purpose:</b> Returns the logical parent activities of the activities in the input list, taking into account the translation of instances to their definition charts  <b>Syntax:</b>  stm_r_ac_logical_parent_of_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<p><b>stm_r_ac_logical_sub_of_ac</b></p>	<p><b>Query:</b> Logical subactivities of a given activity  <b>Purpose:</b> Returns the logical subactivities of the activities in the input list, taking into account the translation of instances to their definition charts  <b>Syntax:</b>  stm_r_ac_logical_sub_of_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<p><b>stm_r_ac_mini_spec_ac</b></p>	<p><b>Query:</b> Activities having mini-specs  <b>Purpose:</b> Returns the activities in the input list that have a mini-spec  <b>Syntax:</b>  stm_r_ac_mini_spec_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<p><b>stm_r_ac_name_of_ac</b></p>	<p><b>Query:</b> Activities whose names match a given pattern  <b>Purpose:</b> Returns all the activities whose names match a given pattern  <b>Syntax:</b>  stm_r_ac_name_of_ac (IN pattern: STRING, OUT status: INTEGER)</p>
<p><b>stm_r_ac_offpage_instance_ac</b></p>	<p><b>Query:</b> Offpage instance activities  <b>Purpose:</b> Returns the activities in the input list that are instances of offpage charts  <b>Syntax:</b>  stm_r_ac_offpage_instance_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<p><b>stm_r_ac_physical_desc_of_ac</b></p>	<p><b>Query:</b> Physical descendants of a given activity  <b>Purpose:</b> Returns the physical descendants (those within the same chart) for the activities in the input list  <b>Syntax:</b>  stm_r_ac_physical_desc_of_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<p><b>stm_r_ac_physical_parent_of_ac</b></p>	<p><b>Query:</b> Physical parent activities of a given activity  <b>Purpose:</b> Returns the physical parent activities (those within the same chart) for the activities in the input list  <b>Syntax:</b>  stm_r_ac_physical_parent_of_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>

<b>stm_r_ac_physical_sub_of_ac</b>	<p><b>Query:</b> Physical subactivities of a given activity</p> <p><b>Purpose:</b> Returns the physical subactivities (those within the same chart) for the activities in the input list</p> <p><b>Syntax:</b> stm_r_ac_physical_sub_of_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_ac_procedure_like_ac</b>	<p><b>Query:</b> Procedure-like activities</p> <p><b>Purpose:</b> Returns the activities in the input list that are procedure-like activities</p> <p><b>Syntax:</b> stm_r_ac_procedure_like_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_ac_resolved_to_ext_ac</b>	<p><b>Query:</b> Activities resolved to a given external activity</p> <p><b>Purpose:</b> Returns the activities (internal, external, or environment) to which the external activities in the input list are resolved</p> <p><b>Syntax:</b> stm_r_ac_resolved_to_ext_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_ac_self_terminated_ac</b>	<p><b>Query:</b> Self-terminated activities</p> <p><b>Purpose:</b> Returns the activities in the input list that are self-terminated</p> <p><b>Syntax:</b> stm_r_ac_self_terminated_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_ac_subroutine_binding_ac</b>	<p><b>Query:</b> Activities with subroutine bindings</p> <p><b>Purpose:</b> Returns the activities in the input list that have subroutine bindings (regardless of the implementation setting in the properties)</p> <p><b>Syntax:</b> stm_r_ac_subroutine_binding_ac (IN el_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_ac_synonym_of_ac</b>	<p><b>Query:</b> Activities whose synonyms match a given pattern</p> <p><b>Purpose:</b> Returns all the activities whose synonyms match the specified pattern</p> <p><b>Syntax:</b> stm_r_ac_synonym_of_ac (IN pattern: STRING, OUT status: integer)</p>
<b>stm_r_ac_unresolved_ac</b>	<p><b>Query:</b> Unresolved activities</p> <p><b>Purpose:</b> Returns the unresolved activities in the input list</p> <p><b>Syntax:</b> stm_r_ac_unresolved_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>

## Query Functions

---

<b>stm_r_ac_ext_ll_ac</b>	<b>Query:</b> External life-line activities <b>Purpose:</b> Returns the External life-line activities in the input list <b>Syntax:</b> stm_r_ac_ext_ll_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)
<b>stm_r_ac_lifeline_ac</b>	<b>Query:</b> Life-line activities <b>Purpose:</b> Returns the life-line activities in the input list <b>Syntax:</b> stm_r_ac_lifeline_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)
<b>stm_r_ac_boundary_box_ac</b>	<b>Query:</b> Boundary-box activities <b>Purpose:</b> Returns the Boundary-box activities in the input list <b>Syntax:</b> stm_r_ac_boundary_box_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)
<b>stm_r_ac_use_case_ac</b>	<b>Query:</b> Use-case activities <b>Purpose:</b> Returns the Use-case activities in the input list <b>Syntax:</b> stm_r_ac_use_case_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)
<b>stm_r_ac_actor_ac</b>	<b>Query:</b> Actor activities <b>Purpose:</b> Returns the Actor activities in the input list <b>Syntax:</b> stm_r_ac_actor_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)
<b>stm_r_ac_router_ac</b>	<b>Query:</b> Router activities <b>Purpose:</b> Returns the Router activities in the input list <b>Syntax:</b> stm_r_ac_router_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)
<b>stm_r_ac_external_router_ac</b>	<b>Query:</b> External router activities <b>Purpose:</b> Returns the Router activities in the input list <b>Syntax:</b> stm_r_ac_external_router_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)

## Input List Type: af

<b>stm_r_ba_defined_in_ch</b>	<p><b>Query:</b> Activities that are sources for a given a-flow-line</p> <p><b>Purpose:</b> Returns the A-Flow-lines defined in the input list of charts.</p> <p><b>Syntax:</b>  STM_R_BA_DEFINED_IN_CH(IN ch_list: LIST OF CHART,  OUT  status: INTEGER):LIST OF A_FLOW_LINE;</p>
<b>stm_r_ba_enter_ds</b>	<p><b>Query:</b> Activities that are sources for a given a-flow-line</p> <p><b>Purpose:</b> Returns the A-Flow-lines entering the Data-Stores in the input list.</p> <p><b>Syntax:</b>  STM_R_BA_ENTER_DS(IN in_list: LIST OF DATA_STORE,  OUT status: INTEGER):LIST OF A_FLOW_LINE;</p>
<b>stm_r_ba_exit_from_ds</b>	<p><b>Query:</b> Activities that are sources for a given a-flow-line</p> <p><b>Purpose:</b> Returns the activities that are sources of a-flow-lines in the input list</p> <p><b>Syntax:</b>  STM_R_BA_EXIT_FROM_DS (IN in_list: LIST OF  DATA_STORE, OUT status: INTEGER):LIST OF  A_FLOW_LINE;</p>
<b>stm_r_ba_enter_an</b>	<p><b>Query:</b> Activities that are sources for a given a-flow-line</p> <p><b>Purpose:</b> Returns the activities that are sources of a-flow-lines in the input list</p> <p><b>Syntax:</b>  STM_R_BA_ENTER_AN(IN in_list: LIST OF ACTIVITY, OUT  status: INTEGER):LIST OF A_FLOW_LINE;</p>
<b>stm_r_ba_exit_from_ac</b>	<p><b>Query:</b> Activities that are sources for a given a-flow-line</p> <p><b>Purpose:</b> Returns the activities that are sources of a-flow-lines in the input list</p> <p><b>Syntax:</b>  STM_R_BA_EXIT_FROM_AC(IN in_list: LIST OF ACTIVITY,  OUT status: INTEGER):LIST OF A_FLOW_LINE;</p>
<b>stm_r_ba_enter_cn</b>	<p><b>Query:</b> Activities that are sources for a given a-flow-line</p> <p><b>Purpose:</b> Returns the activities that are sources of a-flow-lines in the input list</p> <p><b>Syntax:</b>  STM_R_BA_ENTER_CN(IN in_list: LIST OF CONNECTOR,  OUT status: INTEGER):LIST OF A_FLOW_LINE;</p>

<p><b>stm_r_ba_exit_from_cn</b></p>	<p><b>Query:</b> Activities that are sources for a given a-flow-line  <b>Purpose:</b> Returns the activities that are sources of a-flow-lines in the input list  <b>Syntax:</b>  STM_R_BA_EXIT_FROM_CN(IN in_list: LIST OF CONNECTOR, OUT status: INTEGER):LIST OF A_FLOW_LINE;</p>
<p><b>stm_r_laf_containing_ba</b></p>	<p><b>Query:</b> Activities that are sources for a given a-flow-line  <b>Purpose:</b> Returns the activities that are sources of a-flow-lines in the input list  <b>Syntax:</b>  STM_R_LAF_CONTAINING_BA(IN ba_list: LIST OF A_FLOW_LINE, OUT status: INTEGER):LIST OF A_FLOW_LINE</p>
<p><b>stm_r_ba_contained_in_af</b></p>	<p><b>Query:</b> Activities that are sources for a given a-flow-line  <b>Purpose:</b> Returns the activities that are sources of a-flow-lines in the input list  <b>Syntax:</b>  STM_R_BA_CONTAINED_IN_AF(IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER):LIST OF A_FLOW_LINES</p>
<p><b>stm_r_ac_source_of_af</b></p>	<p><b>Query:</b> Activities that are sources for a given a-flow-line  <b>Purpose:</b> Returns the activities that are sources of a-flow-lines in the input list  <b>Syntax:</b>  stm_r_ac_source_of_af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)</p>
<p><b>stm_r_ac_target_of_af</b></p>	<p><b>Query:</b> Activities that are targets of a given a-flow-line  <b>Purpose:</b> Returns the activities that are targets for a-flow-lines in the input list  <b>Syntax:</b>  stm_r_ac_target_of_af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)</p>

## Input List Type: ch

<b>stm_r_ac_def_or_unres_in_ch</b>	<p><b>Query:</b> Activities defined or unresolved in a given chart</p> <p><b>Purpose:</b> Returns activities that are explicitly defined or unresolved in the charts of the input list</p> <p><b>Syntax:</b> stm_r_ac_def_or_unres_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</p>
<b>stm_r_ac_defined_in_ch</b>	<p><b>Query:</b> Activities defined in a given chart</p> <p><b>Purpose:</b> Returns the activities that are explicitly defined in the charts of the input list</p> <p><b>Syntax:</b> stm_r_ac_defined_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</p>
<b>stm_r_ac_described_by_ch</b>	<p><b>Query:</b> Control activities described by a given statechart</p> <p><b>Purpose:</b> Returns the control activities described by statecharts in the input list</p> <p><b>Syntax:</b> stm_r_ac_described_by_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</p>
<b>stm_r_ac_instance_of_ch</b>	<p><b>Query:</b> Activities instance of a given chart</p> <p><b>Purpose:</b> Returns the instance activities defined by the charts in the input list</p> <p><b>Syntax:</b> stm_r_ac_instance_of_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</p>
<b>stm_r_ac_root_in_ch</b>	<p><b>Query:</b> Root activities of a given chart</p> <p><b>Purpose:</b> Returns the internally defined activities (of type diagram) attached to the charts in the input list</p> <p><b>Syntax:</b> stm_r_ac_root_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</p>
<b>stm_r_ac_top_level_in_ch</b>	<p><b>Query:</b> Top-level activities of a given chart</p> <p><b>Purpose:</b> Returns the top-level activities (not contained in any box) of the charts in the input list</p> <p><b>Syntax:</b> stm_r_ac_top_level_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</p>

<b>stm_r_ac_unresolved_in_ch</b>	<p><b>Query:</b> Activities unresolved in a given chart</p> <p><b>Purpose:</b> Returns activities that are unresolved in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_ac_unresolved_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</pre>
----------------------------------	---

### Input List Type: ds

<b>stm_r_ac_parent_of_ds</b>	<p><b>Query:</b> Parent activities of a given data-store</p> <p><b>Purpose:</b> Returns the activities that encapsulate the specified data-stores from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_ac_parent_of_ds (IN ds_list: LIST OF DATA_STORE, OUT status: INTEGER)</pre>
------------------------------	--

### Input List Type: md

<b>stm_r_ac_carried_out_by_md</b>	<p><b>Query:</b> Activities carried out by a given module.</p> <p><b>Purpose:</b> Returns the activities carried out by modules in the input list. The module appears in the <b>Implemented by Module</b> field of the activity's form.</p> <p><b>Syntax:</b></p> <pre>stm_r_ac_carried_out_by_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>
-----------------------------------	---



## Input List Type: mx

<b>stm_r_ac_affecting_mx</b>	<p><b>Query:</b> Activities in which a given element is affected.</p> <p><b>Purpose:</b> Returns the activities that affect (modify, generate, or activate) the elements (for example, events, data-items, or activities) in the input list.</p> <p><b>Syntax:</b> stm_r_ac_affecting_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</p>
<b>stm_r_ac_meaningly_affecting_mx</b>	<p><b>Query:</b> Activities in which a given element is affected.</p> <p><b>Purpose:</b> Identical to stm_r_ac_affecting_mx, but when the input list includes an ID of a record/union, stm_r_ac_meaningly_affecting_mx will also return elements that affect a field of the record/union, and not necessarily the whole record/union element.</p> <p><b>Syntax:</b> stm_r_ac_meaningly_affecting_mx (stm_list in_list, int *status);</p>
<b>stm_r_ac_using_mx</b>	<p><b>Query:</b> Activities in which a given element is used.</p> <p><b>Purpose:</b> Returns the activities that use (evaluate) the elements (basic events, conditions, data-items, states, and activities) in the input list.</p> <p><b>Syntax:</b> stm_r_ac_using_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</p>
<b>stm_r_ac_meaningly_using_mx</b>	<p><b>Query:</b> Activities in which a given element is used.</p> <p><b>Purpose:</b> Identical to stm_r_ac_using_mx, but when the input list includes an ID of a record/union, stm_r_ac_meaningly_using_mx will also return elements that use a field of the record/union, and not necessarily the whole record/union element</p> <p><b>Syntax:</b> stm_r_ac_meaningly_using_mx (stm_list in_list, int *status);</p>

## Input List Type: router

<b>stm_r_ac_parent_of_router</b>	<p><b>Query:</b> Parent activities of a given router</p> <p><b>Purpose:</b> Returns the activities that encapsulate the specified routers from the input list</p> <p><b>Syntax:</b> stm_r_ac_parent_of_router (IN router_list: LIST OF ROUTER, OUT status: INTEGER)</p>
----------------------------------	---

**Input List Type: st**

<p><b>stm_r_ac_thoroughout_st</b></p>	<p><b>Query:</b> Activities performed throughout a given state</p> <p><b>Purpose:</b> Returns the activities performed throughout states in the input list (as defined in the <b>Activities Within/Throughout</b> field of the state's form)</p> <p><b>Syntax:</b></p> <pre>stm_r_ac_thoroughout_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</pre>
<p><b>stm_r_ac_within_st</b></p>	<p><b>Query:</b> Activities performed within a given state</p> <p><b>Purpose:</b> Returns the activities performed within states in the input list (as defined in the <b>Activities Within/Throughout</b> field of the state's form)</p> <p><b>Syntax:</b></p> <pre>stm_r_ac_within_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</pre>

## A-Flow-Lines (af, ba, laf)

This section lists the query functions that return a list of a-flow-lines.

Two abbreviations are used in these functions:

- ◆ af—Global (compound) a-flow-lines
- ◆ ba—Basic a-flow-lines
- ◆ laf—Local a-flow-lines

### Output List Type: af

#### Input List Type: ac

<b>stm_r_af_from_source_ac</b>	<p><b>Query:</b> A-flow-lines whose source is a given activity</p> <p><b>Purpose:</b> Returns global compound a-flow-lines that originate at activities in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_af_from_source_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</pre>
<b>stm_r_af_input_to_ac</b>	<p><b>Query:</b> A-flow-lines input to a given activity within chart</p> <p><b>Purpose:</b> Returns all local compound a-flow-lines that originate outside and terminate at (or inside) activities in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_af_input_to_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</pre>
<b>stm_r_af_output_from_ac</b>	<p><b>Query:</b> A-flow-lines output from a given activity</p> <p><b>Purpose:</b> Returns all global compound a-flow-lines that originate at (or inside) and terminate outside activities in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_af_output_from_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</pre>
<b>stm_r_af_to_target_ac</b>	<p><b>Query:</b> A-flow-lines whose target is a given activity</p> <p><b>Purpose:</b> Returns global a-flow-lines that terminate at activities in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_af_to_target_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</pre>

### Input List Type: ba

<b>stm_r_af_containing_ba</b>	<p><b>Query:</b> A-flow lines from an input list.</p> <p><b>Purpose:</b> Returns the a-flow-lines that contain the basic a-flow-lines in the input list.</p> <p><b>Syntax:</b>  <code>STM_R_AF_CONTAINING_BA(IN ba_list: LIST OF A_FLOW_LINE, OUT status:INTEGER):LIST OF A_FLOW_LINE</code></p>
-------------------------------	--

### Input List Type: co

<b>stm_r_af_within_flows_co</b>	<p><b>Query:</b> A-flow-lines through which a given condition flows</p> <p><b>Purpose:</b> Returns the a-flow-lines through which conditions in the input list actually flow</p> <p><b>Syntax:</b>  <code>stm_r_af_within_flows_co (IN co_list: LIST OF CONDITION, OUT status: INTEGER)</code></p>
<b>stm_r_af_within_labels_co</b>	<p><b>Query:</b> A-flow-lines labeled with a given condition</p> <p><b>Purpose:</b> Returns the a-flow-lines labeled with conditions in the input list</p> <p><b>Syntax:</b>  <code>stm_r_af_within_labels_co (IN co_list: LIST OF CONDITION, OUT status: INTEGER)</code></p>

### Input List Type: di

<b>stm_r_af_within_flows_di</b>	<p><b>Query:</b> A-flow-lines through which a given data-item flows</p> <p><b>Purpose:</b> Returns the a-flow-lines through which data-items in the input list actually flow</p> <p><b>Syntax:</b>  <code>stm_r_af_within_flows_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</code></p>
<b>stm_r_af_within_labels_di</b>	<p><b>Query:</b> A-flow-lines labeled by a given data-item</p> <p><b>Purpose:</b> Returns the a-flow-lines labeled with data-items in the input list</p> <p><b>Syntax:</b>  <code>stm_r_af_within_labels_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</code></p>

## Input List Type: ds

<b>stm_r_af_from_source_ds</b>	<p><b>Query:</b> A-flow-lines whose source is a given data-store</p> <p><b>Purpose:</b> Returns global compound a-flow-lines that originate at data-stores in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_af_from_source_ds (IN ds_list: LIST OF DATA_STORE, OUT status: INTEGER)</pre>
<b>stm_r_af_to_target_ds</b>	<p><b>Query:</b> A-flow-lines whose target is a given data-store</p> <p><b>Purpose:</b> Returns global compound a-flow-lines that terminate at data-stores in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_af_to_target_ds (IN ds_list: LIST OF DATA_STORE, OUT status: INTEGER)</pre>

## Input List Type: ev

<b>stm_r_af_within_flows_ev</b>	<p><b>Query:</b> A-flow-lines through which a given event flows</p> <p><b>Purpose:</b> Returns the a-flow-lines through which events in the input list actually flow</p> <p><b>Syntax:</b></p> <pre>stm_r_af_within_flows_ev (IN ev_list: LIST OF EVENT, OUT status: INTEGER)</pre>
<b>stm_r_af_within_labels_ev</b>	<p><b>Query:</b> A-flow-lines through which a given event flows</p> <p><b>Purpose:</b> Returns the a-flow-lines labeled with events in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_af_within_labels_ev (IN ev_list: LIST OF EVENT, OUT status: INTEGER)</pre>

### Input List Type: if

<p><b>stm_r_af_within_flows_if</b></p>	<p><b>Query:</b> A-flow-lines through which a given information-flow flows</p> <p><b>Purpose:</b> Returns the a-flow-lines through which information-flows in the input list actually flow</p> <p><b>Syntax:</b></p> <pre>stm_r_af_within_flows_if (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</pre>
<p><b>stm_r_af_within_labels_if</b></p>	<p><b>Query:</b> A-flow-lines labeled with a given information-flow</p> <p><b>Purpose:</b> Returns the a-flow-lines labeled with information-flows in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_af_within_labels_if (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</pre>

### Input List Type: laf

<p><b>stm_r_af_containing_laf</b></p>	<p><b>Query:</b> None</p> <p><b>Purpose:</b> Returns the global a-flow-lines (which might spread over several charts) that contain the local a-flow-lines (those within charts) in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_af_containing_laf (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)</pre>
---------------------------------------	---

## Input List Type: mx

<b>stm_r_af_from_source_mx</b>	<p><b>Query:</b> A-flow-lines whose source is a given element</p> <p><b>Purpose:</b> Returns global compound a-flow-lines whose source is an element from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_af_from_source_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_af_to_target_mx</b>	<p><b>Query:</b> A-flow-lines whose target is given element</p> <p><b>Purpose:</b> Returns global compound a-flow-lines whose target is an element from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_af_to_target_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_af_within_flows_mx</b>	<p><b>Query:</b> A-flow-lines through which a given element flows</p> <p><b>Purpose:</b> Returns the a-flow-lines through which elements in the input list actually flow</p> <p><b>Syntax:</b></p> <pre>stm_r_af_within_flows_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_af_within_labels_mx</b>	<p><b>Query:</b> A-flow-lines labeled by a given element</p> <p><b>Purpose:</b> Returns the a-flow-lines labeled with elements in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_af_within_labels_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>

## Input List Type: router

<b>stm_r_af_from_source_router</b>	<p><b>Query:</b> A-flow-lines whose source is a given router</p> <p><b>Purpose:</b> Returns global compound a-flow-lines whose source is a router from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_af_from_source_router (IN router_list: LIST OF ROUTER, OUT status: INTEGER)</pre>
<b>stm_r_af_to_target_router</b>	<p><b>Query:</b> A-flow-lines whose target is given router</p> <p><b>Purpose:</b> Returns global compound a-flow-lines whose target is a router from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_af_to_target_router (IN router_list: LIST OF ROUTER, OUT status: INTEGER)</pre>

## Output List Type: laf

### Input List Type: ac

<p><b>stm_r_laf_from_source_ac</b></p>	<p><b>Query:</b> A-flow-lines whose source is a given activity  <b>Purpose:</b> Returns local compound a-flow-lines that originate at activities in the input list  <b>Syntax:</b>          stm_r_laf_from_source_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<p><b>stm_r_laf_input_to_ac</b></p>	<p><b>Query:</b> A-flow-lines input to a given activity  <b>Purpose:</b> Returns all the local a-flow-lines  <b>Syntax:</b>          stm_r_laf_input_to_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<p><b>stm_r_laf_output_from_ac</b></p>	<p><b>Query:</b> A-flow-lines output from a given activity within chart  <b>Purpose:</b> Returns all local compound a-flow-lines that originate at (or inside) and terminate outside activities in the input list  <b>Syntax:</b>          stm_r_laf_output_from_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<p><b>stm_r_laf_to_target_ac</b></p>	<p><b>Query:</b> A-flow-lines whose target is a given activity within chart  <b>Purpose:</b> Returns local a-flow-lines (those within charts) that terminate at activities in the input list  <b>Syntax:</b>          stm_r_laf_to_target_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>

### Input List Type: af

<p><b>stm_r_laf_contained_in_af</b></p>	<p><b>Query:</b> None  <b>Purpose:</b> Returns the local a-flow-lines that contain the global a-flow-lines in the input list  <b>Syntax:</b>          stm_r_laf_contained_in_af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)</p>
---	--



## Input List Type: ds

<b>stm_r_laf_from_source_ds</b>	<p><b>Query:</b> A-flow-lines whose source is a given data-store within chart</p> <p><b>Purpose:</b> Returns local compound a-flow-lines that originate at data-stores in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_laf_from_source_ds (IN ds_list: LIST OF DATA_STORE, OUT status: INTEGER)</pre>
<b>stm_r_laf_to_target_ds</b>	<p><b>Query:</b> A-flow-lines whose target is a given data-store within chart</p> <p><b>Purpose:</b> Returns local compound a-flow-lines that terminate at data-stores in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_laf_to_target_ds (IN ds_list: LIST OF DATA_STORE, OUT status: INTEGER)</pre>

## Input List Type: mx

<b>stm_r_laf_from_source_mx</b>	<p><b>Query:</b> A-flow-lines whose source is a given element within chart</p> <p><b>Purpose:</b> Returns local compound a-flow-lines whose source is an element from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_laf_from_source_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_laf_to_target_mx</b>	<p><b>Query:</b> A-flow-lines whose target is given element within chart</p> <p><b>Purpose:</b> Returns local compound a-flow-lines whose target is an element from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_laf_to_target_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>

### Input List Type: router

<b>stm_r_laf_from_source_router</b>	<p><b>Query:</b> A-flow-lines whose source is a given router within chart</p> <p><b>Purpose:</b> Returns local compound a-flow-lines whose source is a router from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_laf_from_source_router (IN router_list: LIST OF ROUTER, OUT status: INTEGER)</pre>
<b>stm_r_laf_to_target_router</b>	<p><b>Query:</b> A-flow-lines whose target is given router within chart</p> <p><b>Purpose:</b> Returns local compound a-flow-lines whose target is a router from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_laf_to_target_router (IN router_list: LIST OF ROUTER, OUT status: INTEGER)</pre>

## Actions (an)

This section documents the query functions that return a list of actions.

### Input List Type: an

<b>stm_r_an_by_attributes_an</b>	<p><b>Query:</b> Actions by attribute</p> <p><b>Purpose:</b> Returns the actions in the input list that match a given attribute and value</p> <p><b>Syntax:</b></p> <pre>stm_r_an_by_attributes_an (IN an_list: LIST OF ACTION, IN attr_name : STRING, IN attr_value : STRING, OUT status: INTEGER)</pre>
<b>stm_r_an_explicit_defined_an</b>	<p><b>Query:</b> Actions explicitly defined</p> <p><b>Purpose:</b> Returns the actions of the input list that were explicitly defined</p> <p><b>Syntax:</b></p> <pre>stm_r_an_explicit_defined_an (IN an_list: LIST OF ACTION, OUT status: INTEGER)</pre>
<b>stm_r_an_imp_best_match_an</b>	<p><b>Query:</b> Actions whose selected implementation is <b>Best Match</b></p> <p><b>Purpose:</b> Returns the actions in the input list implemented as the <b>Best Match</b> using <b>Select Implementation</b> in the properties</p> <p><b>Syntax:</b></p> <pre>stm_r_an_imp_best_match_an (IN el_list: LIST OF ACTION, OUT status: INTEGER)</pre>

<b>stm_r_an_imp_definition_an</b>	<p><b>Query:</b> Actions with a defined implementation</p> <p><b>Purpose:</b> Returns the actions in the input list that have a defined implementation in the properties</p> <p><b>Syntax:</b> stm_r_an_imp_definition_an (IN el_list: LIST OF ACTION, OUT status: INTEGER)</p>
<b>stm_r_an_imp_none_an</b>	<p><b>Query:</b> Actions whose selected implementation is <b>None</b></p> <p><b>Purpose:</b> Returns the actions in the input list that are not implemented using Select Implementation</p> <p><b>Syntax:</b> stm_r_an_imp_none_an (IN el_list: LIST OF ACTION, OUT status: INTEGER)</p>
<b>stm_r_an_imp_truth_table_an</b>	<p><b>Query:</b> Actions implemented in a truth table</p> <p><b>Purpose:</b> Returns the actions in the input list that are implemented with a truth table in the properties</p> <p><b>Syntax:</b> stm_r_an_imp_truth_table_an (IN el_list: LIST OF ACTION, OUT status: INTEGER)</p>
<b>stm_r_an_name_of_an</b>	<p><b>Query:</b> Actions whose names match a given pattern</p> <p><b>Purpose:</b> Returns all the actions whose names match a specified pattern</p> <p><b>Syntax:</b> stm_r_an_name_of_an (IN pattern: STRING, OUT status: INTEGER)</p>
<b>stm_r_an_synonym_of_an</b>	<p><b>Query:</b> Actions whose synonyms match a given pattern</p> <p><b>Purpose:</b> Returns all the actions whose synonyms match a specified pattern</p> <p><b>Syntax:</b> stm_r_an_synonym_of_an (IN pattern: STRING, OUT status: INTEGER)</p>
<b>stm_r_an_unresolved_an</b>	<p><b>Query:</b> Unresolved actions</p> <p><b>Purpose:</b> Returns the unresolved actions in the input list</p> <p><b>Syntax:</b> stm_r_an_unresolved_an (IN an_list: LIST OF ACTION, OUT status: INTEGER)</p>

### Input List Type: ch

<p><b>stm_r_an_def_or_unres_in_ch</b></p>	<p><b>Query:</b> Actions defined or unresolved in a given chart</p> <p><b>Purpose:</b> Returns the actions that are explicitly defined or unresolved in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_an_def_or_unres_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</pre>
<p><b>stm_r_an_defined_in_ch</b></p>	<p><b>Query:</b> Actions defined in a given chart</p> <p><b>Purpose:</b> Returns the actions that are explicitly defined in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_an_defined_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</pre>
<p><b>stm_r_an_unresolved_in_ch</b></p>	<p><b>Query:</b> Actions unresolved in a given chart</p> <p><b>Purpose:</b> Returns the actions that are unresolved in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_an_unresolved_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</pre>

## Charts (ch)

This section documents the query functions that return a list of charts.

### Input List Type: ac

<b>stm_r_ch_define_ac</b>	<p><b>Query:</b> Charts in which a given activity is defined</p> <p><b>Purpose:</b> Returns the charts in which the activities in the input list are explicitly defined or unresolved</p> <p><b>Syntax:</b>  <code>stm_r_ch_define_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</code></p>
<b>stm_r_ch_defining_ac</b>	<p><b>Query:</b> Activity-charts defining a given activity</p> <p><b>Purpose:</b> Returns the activity-charts that define the instance activities in the input list</p> <p><b>Syntax:</b>  <code>stm_r_ch_defining_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</code></p>
<b>stm_r_ch_describing_ac</b>	<p><b>Query:</b> Statecharts describing a given control activity</p> <p><b>Purpose:</b> Returns the statecharts that describe the control activities in the input list</p> <p><b>Syntax:</b>  <code>stm_r_ch_describing_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</code></p>

### Input List Type: an

<b>stm_r_ch_define_an</b>	<p><b>Query:</b> Charts in which a given action is defined</p> <p><b>Purpose:</b> Returns the charts in which the actions in the input list are explicitly defined or unresolved</p> <p><b>Syntax:</b>  <code>stm_r_ch_define_an (IN an_list: LIST OF ACTION, OUT status: INTEGER)</code></p>
---------------------------	---

## Input List Type: ch

<b>stm_r_ch_activitychart_ch</b>	<p><b>Query:</b> Activity-charts</p> <p><b>Purpose:</b> Returns the activity-charts in the input list</p> <p><b>Syntax:</b>  <code>stm_r_ch_activitychart_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>
<b>stm_r_ch_ancestors_of_ch</b>	<p><b>Query:</b> Ancestors of a given chart</p> <p><b>Purpose:</b> Returns the ancestors (in the static structure) of the charts in the input list</p> <p><b>Syntax:</b>  <code>stm_r_ch_ancestors_of_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>
<b>stm_r_ch_by_attributes_ch</b>	<p><b>Query:</b> Chart by attribute</p> <p><b>Purpose:</b> Returns the charts in the input list that match the specified attribute name and value</p> <p><b>Syntax:</b>  <code>stm_r_ch_by_attributes_ch (IN ch_list: LIST OF CHART, IN attr_name : STRING, IN attr_value : STRING, OUT status: INTEGER)</code></p>
<b>stm_r_ch_descendants_of_ch</b>	<p><b>Query:</b> Descendants of a given chart</p> <p><b>Purpose:</b> Returns the descendants (in the static structure) of the charts in the input list</p> <p><b>Syntax:</b>  <code>stm_r_ch_descendants_of_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>
<b>stm_r_ch_dictionary_ch</b>	<p><b>Query:</b> Global definition sets (GDSs)</p> <p><b>Purpose:</b> Returns the GDSs in the input list</p> <p><b>Syntax:</b>  <code>stm_r_ch_dictionary_ch (IN el_list: LIST OF CHART, OUT status: INTEGER)</code></p>
<b>stm_r_ch_explicit_defined_ch</b>	<p><b>Query:</b> Charts explicitly defined</p> <p><b>Purpose:</b> Returns the charts of the input list that were explicitly defined</p> <p><b>Syntax:</b>  <code>stm_r_ch_explicit_defined_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>
<b>stm_r_ch_generic_ch</b>	<p><b>Query:</b> Generic charts</p> <p><b>Purpose:</b> Returns the generic charts in the input list</p> <p><b>Syntax:</b>  <code>stm_r_ch_generic_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>

<b>stm_r_ch_modulechart_ch</b>	<p><b>Query:</b> Module-charts</p> <p><b>Purpose:</b> Returns the charts in the input list that are module-charts</p> <p><b>Syntax:</b> stm_r_ch_modulechart_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</p>
<b>stm_r_ch_name_of_ch</b>	<p><b>Query:</b> Charts whose names match a given pattern</p> <p><b>Purpose:</b> Returns all the charts whose names match the specified pattern</p> <p><b>Syntax:</b> stm_r_ch_name_of_ch (IN pattern: STRING, OUT status: INTEGER)</p>
<b>stm_r_ch_offpage_ch</b>	<p><b>Query:</b> Offpage charts</p> <p><b>Purpose:</b> Returns the offpage charts in the input list</p> <p><b>Syntax:</b> stm_r_ch_offpage_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</p>
<b>stm_r_ch_parent_ch</b>	<p><b>Query:</b> Returns the parent charts of a given chart</p> <p><b>Purpose:</b> Returns the parents (in the static structure) of the charts in the input list</p> <p><b>Syntax:</b> stm_r_ch_parent_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</p>
<b>stm_r_ch_procedural_sch_ch</b>	<p><b>Query:</b> Procedural statecharts</p> <p><b>Purpose:</b> Returns the charts in the input list that are procedural statecharts</p> <p><b>Syntax:</b> stm_r_ch_procedural_sch_ch (IN el_list : LIST OF CHART, OUT status: INTEGER)</p>
<b>stm_r_ch_referenced_all_by_ch</b>	<p><b>Query:</b> Charts referenced in all levels by a given chart</p> <p><b>Purpose:</b> Returns all charts referenced (instantiated) by all levels of charts in the input list</p> <p><b>Syntax:</b> stm_r_ch_referenced_all_by_ch (IN el_list: LIST OF CHART, OUT status: INTEGER)</p>
<b>stm_r_ch_referenced_by_ch</b>	<p><b>Query:</b> Charts referenced by a given chart</p> <p><b>Purpose:</b> Returns all charts referenced (instantiated) by the charts in the input list</p> <p><b>Syntax:</b> stm_r_ch_referenced_by_ch (IN el_list: LIST OF CHART, OUT status: INTEGER)</p>

<b>stm_r_ch_root_ch</b>	<p><b>Query:</b> Root charts</p> <p><b>Purpose:</b> Returns the root-level charts (that have no parent) in the input list</p> <p><b>Syntax:</b>  <code>stm_r_ch_root_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>
<b>stm_r_ch_statechart_ch</b>	<p><b>Query:</b> Statecharts</p> <p><b>Purpose:</b> Returns the charts in the input list that are statecharts</p> <p><b>Syntax:</b>  <code>stm_r_ch_statechart_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>
<b>stm_r_ch_subchart_ch</b>	<p><b>Query:</b> Subchart of the specified chart</p> <p><b>Purpose:</b> Returns the subcharts (in the static structure) of the charts in the input list</p> <p><b>Syntax:</b>  <code>stm_r_ch_subchart_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>
<b>stm_r_ch_unresolved_ch</b>	<p><b>Query:</b> Unresolved charts</p> <p><b>Purpose:</b> Returns the unresolved charts (used but not defined) in the input list</p> <p><b>Syntax:</b>  <code>stm_r_ch_unresolved_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>

### Input List Type: co

<b>stm_r_ch_define_co</b>	<p><b>Query:</b> Charts in which a given condition is defined</p> <p><b>Purpose:</b> Returns the charts in which the conditions in the input list are explicitly defined or unresolved</p> <p><b>Syntax:</b>  <code>stm_r_ch_define_co (IN co_list: LIST OF CONDITION, OUT status: INTEGER)</code></p>
---------------------------	--



**Input List Type: di**

<b>stm_r_ch_define_di</b>	<p><b>Query:</b> Charts in which a given data-item is defined</p> <p><b>Purpose:</b> Returns the charts in which the data-items in the input list are explicitly defined or unresolved</p> <p><b>Syntax:</b></p> <pre>stm_r_ch_define_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
---------------------------	---

**Input List Type: ds**

<b>stm_r_ch_define_ds</b>	<p><b>Query:</b> Charts in which a given data-store is defined</p> <p><b>Purpose:</b> Returns the charts in which the data-stores in the input list are explicitly defined or unresolved</p> <p><b>Syntax:</b></p> <pre>stm_r_ch_define_ds (IN ds_list: LIST OF DATA_STORE, OUT status: INTEGER)</pre>
---------------------------	--

**Input List Type: dt**

<b>stm_r_ch_define_dt</b>	<p><b>Query:</b> Charts and GDSs in which a given user-defined type is defined</p> <p><b>Purpose:</b> Returns the charts in which the user-defined types in the input list are explicitly defined or unresolved</p> <p><b>Syntax:</b></p> <pre>stm_r_ch_define_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
---------------------------	--

### Input List Type: ev

<b>stm_r_ch_define_ev</b>	<p><b>Query:</b> Charts in which a given event is defined</p> <p><b>Purpose:</b> Returns the charts in which the events in the input list are explicitly defined or unresolved</p> <p><b>Syntax:</b></p> <pre>stm_r_ch_define_ev (IN ev_list: LIST OF EVENT, OUT status: INTEGER)</pre>
---------------------------	---

### Input List Type: fd

<b>stm_r_ch_define_fd</b>	<p><b>Query:</b> Charts and GDSs in which a given field is defined</p> <p><b>Purpose:</b> Returns the charts in which the fields in the input list are defined (in a structured data-item or user-defined type)</p> <p><b>Syntax:</b></p> <pre>stm_r_ch_define_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</pre>
---------------------------	--

### Input List Type: if

<b>stm_r_ch_define_if</b>	<p><b>Query:</b> Charts in which a given information-flow is defined</p> <p><b>Purpose:</b> Returns the charts in which the information-flows in the input list are explicitly defined or unresolved</p> <p><b>Syntax:</b></p> <pre>stm_r_ch_define_if (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</pre>
---------------------------	--

---

**Input List Type: md**

<b>stm_r_ch_define_md</b>	<p><b>Query:</b> Charts in which a given module is defined</p> <p><b>Purpose:</b> Returns charts in which the modules in the input list are explicitly defined or unresolved</p> <p><b>Syntax:</b></p> <pre>stm_r_ch_define_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>
<b>stm_r_ch_defining_md</b>	<p><b>Query:</b> Module-charts defining a given module</p> <p><b>Purpose:</b> Returns the module-charts that define the instance modules in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_ch_defining_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>
<b>stm_r_ch_describing_md</b>	<p><b>Query:</b> Activity-charts describing a given module</p> <p><b>Purpose:</b> Returns the activity-charts that describe the modules in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_ch_describing_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>

### Input List Type: mx

<b>stm_r_ch_define_mx</b>	<p><b>Query:</b> Charts in which a given element is defined</p> <p><b>Purpose:</b> Returns the charts in which the elements in the input list are explicitly defined</p> <p><b>Syntax:</b>  <code>stm_r_ch_define_mx (IN mx_list: LIST OF ELEMENT, OUT status: INTEGER)</code></p>
<b>stm_r_ch_defining_mx</b>	<p><b>Query:</b> Charts defining a given element</p> <p><b>Purpose:</b> Returns the charts that define the elements in the input list</p> <p><b>Syntax:</b>  <code>stm_r_ch_defining_mx (IN mx_list: LIST OF ELEMENT, OUT status: INTEGER)</code></p>
<b>stm_r_ch_describing_mx</b>	<p><b>Query:</b> Statecharts describing a given control activity</p> <p><b>Purpose:</b> Returns the charts that describe the elements in the input list</p> <p><b>Syntax:</b>  <code>stm_r_ch_describing_mx (IN el_list: LIST OF ELEMENT, OUT status: INTEGER)</code></p>

### Input List Type: router

<b>stm_r_ch_define_router</b>	<p><b>Query:</b> Charts in which a given router is defined</p> <p><b>Purpose:</b> Returns the charts in which the routers in the input list are explicitly defined or unresolved</p> <p><b>Syntax:</b>  <code>stm_r_ch_define_router (IN router_list: LIST OF ROUTER, OUT status: INTEGER)</code></p>
-------------------------------	---

**Input List Type: sb**

<b>stm_r_ch_connected_to_sb</b>	<p><b>Query:</b> Charts connected to a given subroutine</p> <p><b>Purpose:</b> Returns the procedural Statecharts that are connected to the subroutines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_ch_connected_to_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>
<b>stm_r_ch_define_sb</b>	<p><b>Query:</b> Charts in which a given subroutine is defined</p> <p><b>Purpose:</b> Returns the charts in which the subroutines in the input list are explicitly defined or unresolved</p> <p><b>Syntax:</b></p> <pre>stm_r_ch_define_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>

**Input List Type: st**

<b>stm_r_ch_define_st</b>	<p><b>Query:</b> Charts in which a given state is defined</p> <p><b>Purpose:</b> Returns the charts in which the states in the input list are explicitly defined or unresolved</p> <p><b>Syntax:</b></p> <pre>stm_r_ch_define_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</pre>
<b>stm_r_ch_defining_st</b>	<p><b>Query:</b> Statecharts defining a given state</p> <p><b>Purpose:</b> Returns the statecharts that define the instance states in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_ch_defining_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</pre>

## Connectors (cn)

This section documents the queries that return a list of connectors.

### Input List Type: cn

<b>stm_r_cn_deep_history_cn</b>	<p><b>Query:</b> Deep history connectors</p> <p><b>Purpose:</b> Returns all the deep history connectors in the input list</p> <p><b>Syntax:</b>  <code>stm_r_cn_deep_history_cn (IN cn_list: LIST OF CONNECTOR,OUT status: INTEGER)</code></p>
<b>stm_r_cn_history_cn</b>	<p><b>Query:</b> History connectors</p> <p><b>Purpose:</b> Returns all the history connectors in the input list</p> <p><b>Syntax:</b>  <code>stm_r_cn_history_cn (IN cn_list: LIST OF CONNECTOR,OUT status: INTEGER)</code></p>
<b>stm_r_cn_termination_cn</b>	<p><b>Query:</b> Termination connectors</p> <p><b>Purpose:</b> Returns all the termination connectors in the input list</p> <p><b>Syntax:</b>  <code>stm_r_cn_termination_cn (IN cn_list: LIST OF CONNECTOR,OUT status: INTEGER)</code></p>
<b>stm_r_cn_source_of_bm</b>	<p><b>Query:</b> Source connectors</p> <p><b>Purpose:</b> Returns all the source connectors in the input list</p> <p><b>Syntax:</b>  <code>stm_list stm_r_cn_source_of_bm(stm_list in_list, int *status);</code></p>
<b>stm_r_cn_target_of_bm</b>	<p><b>Query:</b> Target connectors</p> <p><b>Purpose:</b> Returns all the target connectors in the input list</p> <p><b>Syntax:</b>  <code>stm_list stm_r_cn_target_of_bm(stm_list in_list, int *status);</code></p>
<b>stm_r_cn_source_of_bt</b>	<p><b>Query:</b> Source connectors</p> <p><b>Purpose:</b> Returns all the source connectors in the input list</p> <p><b>Syntax:</b>  <code>STM_R_CN_SOURCE_OF_BT(IN bt_list: LIST OF TRANSITION,OUT status: INTEGER): LIST OF CONNECTOR;</code></p>

<b>stm_r_cn_target_of_bt</b>	<p><b>Query:</b> Target connectors</p> <p><b>Purpose:</b> Returns all the target connectors in the input list</p> <p><b>Syntax:</b></p> <pre>STM_R_CN_TARGET_OF_BT(IN bt_list: LIST OF TRANSITION,OUT status: INTEGER): LIST OF CONNECTOR;</pre>
<b>stm_r_cn_source_of_ba</b>	<p><b>Query:</b> source connectors</p> <p><b>Purpose:</b> Returns all the source connectors in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_cn_termination_cn (IN cn_list: LIST OF CONNECTOR,OUT status: INTEGER)</pre>
<b>stm_r_cn_target_of_ba</b>	<p><b>Query:</b> Termination connectors</p> <p><b>Purpose:</b> Returns all the history connectors in the input list</p> <p><b>Syntax:</b></p> <pre>STM_R_CN_TARGET_OF_BA(IN ba_list: LIST OF A_FLOW_LINE,OUT status: INTEGER): LIST OF CONNECTOR;</pre>

### Input List Type: st

<b>stm_r_cn_history_or_term_in_st</b>	<p><b>Query:</b> Termination or history connectors in a given state</p> <p><b>Purpose:</b> Returns the termination and history connectors contained in the states in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_cn_history_or_term_in_st (IN st_list: LIST OF STATE,OUT status: INTEGER)</pre>
---------------------------------------	---

### Input List Type:bm

<b>stm_r_mf_containing_bm</b>	<p><b>Query:</b> M-Flow-lines</p> <p><b>Purpose:</b> Returns the M-flow-lines containing the basic m-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>STM_R_MF_CONTAINING_BM(IN bm_list: LIST OF M_FLOW_LINE, OUT status: INTEGER):LIST OF M_FLOW_LINE</pre>
-------------------------------	---

### Input List Type: tr

<b>stm_r_cn_source_of_tr</b>	<p><b>Query:</b> History connectors sources of a given transition</p> <p><b>Purpose:</b> Returns the history connectors that are sources of transitions in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_cn_source_of_tr (stm_list in_list, int *status);</pre>
<b>stm_r_cn_target_of_tr</b>	<p><b>Query:</b> Termination or history connectors targets of a given transition</p> <p><b>Purpose:</b> Returns the termination and history connectors that are targets of transitions in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_cn_target_of_tr (IN tr_list: LIST OF TRANSITION,OUT status: INTEGER)</pre>

## Conditions (co)

This section documents the query functions that return a list of conditions.

### Input List Type: af

<b>stm_r_co_flowng_through_af</b>	<p><b>Query:</b> Conditions flowing through a given a-flow-line</p> <p><b>Purpose:</b> Returns the conditions actually flowing through a-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_co_flowng_through_ af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)</pre>
<b>stm_r_co_labeling_af</b>	<p><b>Query:</b> Conditions labeling a given a-flow-line</p> <p><b>Purpose:</b> Returns the conditions which label the a-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_co_labeling_af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)</pre>



### Input List Type: ch

<b>stm_r_co_def_or_unres_in_ch</b>	<p><b>Query:</b> Conditions defined or unresolved in a given chart</p> <p><b>Purpose:</b> Returns conditions that are explicitly defined or unresolved in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_co_def_or_unres_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</pre>
<b>stm_r_co_defined_in_ch</b>	<p><b>Query:</b> Conditions defined in a given chart</p> <p><b>Purpose:</b> Returns the conditions that are explicitly defined in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_co_defined_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</pre>
<b>stm_r_co_unresolved_in_ch</b>	<p><b>Query:</b> Conditions unresolved in a given chart</p> <p><b>Purpose:</b> Returns conditions that are unresolved in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_co_unresolved_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</pre>

### Input List Type: co

<b>stm_r_co_array_co</b>	<p><b>Query:</b> Conditions by subtype</p> <p><b>Purpose:</b> Returns the conditions in the input list that are defined as array</p> <p><b>Syntax:</b></p> <pre>stm_r_co_array_co (IN co_list: LIST OF CONDITION, OUT status INTEGER)</pre>
<b>stm_r_co_by_attributes_co</b>	<p><b>Query:</b> Conditions by attributes</p> <p><b>Purpose:</b> Returns the conditions in the input list that match the specified attribute name and value</p> <p><b>Syntax:</b></p> <pre>stm_r_co_by_attributes_co (IN co_list: LIST OF CONDITION, IN attr_name : STRING, IN attr_value : STRING, OUT status: INTEGER)</pre>

<p><b>stm_r_co_by_structure_type_co</b></p>	<p><b>Query:</b> None</p> <p><b>Purpose:</b> Returns the conditions in the input list that have the specified structure type (for example, single or array)</p> <p><b>Syntax:</b></p> <pre>stm_r_co_by_structure_type_co (IN co_list: LIST OF CONDITION, IN dtype: INTEGER, OUT status: INTEGER)</pre>
<p><b>stm_r_co_callback_binding_co</b></p>	<p><b>Query:</b> Conditions with callback bindings</p> <p><b>Purpose:</b> Returns the conditions of the input list that have callback bindings</p> <p><b>Syntax:</b></p> <pre>stm_r_co_callback_binding_co (IN el_list: LIST OF CONDITION, OUT status: INTEGER)</pre>
<p><b>stm_r_co_explicit_defined_co</b></p>	<p><b>Query:</b> Conditions explicitly defined</p> <p><b>Purpose:</b> Returns the conditions of the input list that were explicitly defined</p> <p><b>Syntax:</b></p> <pre>stm_r_co_explicit_defined_co (IN co_list: LIST OF CONDITION, OUT status: INTEGER)</pre>
<p><b>stm_r_co_name_of_co</b></p>	<p><b>Query:</b> Conditions whose names match a given pattern</p> <p><b>Purpose:</b> Returns all the conditions whose names match the specified pattern</p> <p><b>Syntax:</b></p> <pre>stm_r_co_name_of_co (IN pattern: STRING, OUT status: INTEGER)</pre>
<p><b>stm_r_co_single_co</b></p>	<p><b>Query:</b> Conditions by subtype</p> <p><b>Purpose:</b> Returns the conditions in the input list that are defined as single</p> <p><b>Syntax:</b></p> <pre>stm_r_co_single_co (IN co_list: LIST OF CONDITION, OUT status: INTEGER)</pre>
<p><b>stm_r_co_synonym_of_co</b></p>	<p><b>Query:</b> Conditions whose synonyms match a given pattern</p> <p><b>Purpose:</b> Returns all the conditions whose synonyms match the specified pattern</p> <p><b>Syntax:</b></p> <pre>stm_r_co_synonym_of_co (IN pattern: STRING, OUT status: INTEGER)</pre>
<p><b>stm_r_co_unresolved_co</b></p>	<p><b>Query:</b> Unresolved conditions</p> <p><b>Purpose:</b> Returns the unresolved conditions in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_co_unresolved_co (IN co_list: LIST OF CONDITION, OUT status: INTEGER)</pre>

**Input List Type: di**

<b>stm_r_co_contained_in_di</b>	<p><b>Query:</b> Conditions contained in a given data-item</p> <p><b>Purpose:</b> Returns the conditions contained in data-items from the input list (conditions appearing in the <b>Consists of</b> field of a data-item)</p> <p><b>Syntax:</b></p> <pre>stm_r_co_contained_in_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
---------------------------------	---

**Input List Type: if**

<b>stm_r_co_contained_in_if</b>	<p><b>Query:</b> Conditions contained in a given information-flow</p> <p><b>Purpose:</b> Returns the conditions contained in information-flows from the input list (conditions appearing in the <b>Consists of</b> field of an information-flow)</p> <p><b>Syntax:</b></p> <pre>stm_r_co_contained_in_if (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</pre>
---------------------------------	--

**Input List Type: mf**

<b>stm_r_co_flowng_through_mf</b>	<p><b>Query:</b> Conditions flowing through a given m-flow-line</p> <p><b>Purpose:</b> Returns the conditions actually flowing through m-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_co_flowng_through_ mf (IN mf_list: LIST OF M_FLOW_LINE, OUT status: INTEGER)</pre>
<b>stm_r_co_labeling_mf</b>	<p><b>Query:</b> Conditions labeling a given m-flow-line</p> <p><b>Purpose:</b> Returns the conditions that label the m-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_co_labeling_mf (IN mf_list: LIST OF M_FLOW_LINE, OUT status: INTEGER)</pre>

## Data-Items (di)

This section documents the query functions that return a list of data-items.

### Input List Type: af

<b>stm_r_di_flowng_through_af</b>	<b>Query:</b> Data-items flowing through a given a-flow-line <b>Purpose:</b> Returns the data-items actually flowing through a-flow-lines in the input list <b>Syntax:</b> stm_r_di_flowng_through_ af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)
<b>stm_r_di_labeling_af</b>	<b>Query:</b> Data-items labeling a given a-flow-line <b>Purpose:</b> Returns the data-items which label the a-flow-lines in the input list <b>Syntax:</b> stm_r_di_labeling_af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)

### Input List Type: ch

<b>stm_r_di_def_or_unres_in_ch</b>	<b>Query:</b> Data-items defined or unresolved in a given chart <b>Purpose:</b> Returns the data-items explicitly defined or unresolved in the charts of the input list <b>Syntax:</b> stm_r_di_def_or_unres_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)
<b>stm_r_di_defined_in_ch</b>	<b>Query:</b> Data-items defined in a given chart <b>Purpose:</b> Returns the data-items explicitly defined in the charts of the input list <b>Syntax:</b> stm_r_di_defined_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)
<b>stm_r_di_unresolved_in_ch</b>	<b>Query:</b> Data-items unresolved in a given chart <b>Purpose:</b> Returns the data-items that are unresolved in the charts of the input list <b>Syntax:</b> stm_r_di_unresolved_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)

### Input List Type: co

<b>stm_r_di_containing_co</b>	<p><b>Query:</b> Data-item containing a given condition</p> <p><b>Purpose:</b> Returns the data-items containing the conditions in the input list (as defined in the <b>Consists of</b> field of the data-item's form)</p> <p><b>Syntax:</b></p> <pre>stm_r_di_containing_co (IN co_list: LIST OF CONDITION, OUT status: INTEGER)</pre>
-------------------------------	---

### Input List Type: di

<b>stm_r_di_array_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as array</p> <p><b>Syntax:</b></p> <pre>stm_r_di_array_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_di_array_missing_di</b>	<p><b>Query:</b> Array of data-items by subtype</p> <p><b>Purpose:</b> Returns the arrays of data-items in the input list for which no type is defined</p> <p><b>Syntax:</b></p> <pre>stm_r_di_array_missing_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_di_basic_di</b>	<p><b>Query:</b> Basic data-items</p> <p><b>Purpose:</b> Returns the data-items in the input list that are basic (not defined using other data-items)</p> <p><b>Syntax:</b></p> <pre>stm_r_di_basic_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_di_bit_di</b>	<p><b>Query:</b> Basic data-items</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as Bit in the <b>Structure/Type</b> field of the data-item form</p> <p><b>Syntax:</b></p> <pre>stm_r_di_bit_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>

<b>stm_r_di_bit_queue_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as queue of bits</p> <p><b>Syntax:</b></p> <pre>stm_r_di_bit_queue_di (IN el_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_di_bits_array_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as array of bit array</p> <p><b>Syntax:</b></p> <pre>stm_r_di_bits_array_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_di_bits_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as bit-array in the <b>Structure/Type</b> field of the data-item form</p> <p><b>Syntax:</b></p> <pre>stm_r_di_bits_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_di_bits_queue_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as queue of bit array</p> <p><b>Syntax:</b></p> <pre>stm_r_di_bits_queue_di (IN el_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_di_by_attributes_di</b>	<p><b>Query:</b> Data-items by attributes</p> <p><b>Purpose:</b> Returns the data-items in the input list that match the specified attribute name and value</p> <p><b>Syntax:</b></p> <pre>stm_r_di_by_attributes_di (IN di_list: LIST OF DATA_ITEM, IN attr_name : STRING, IN attr_value : STRING, OUT status: INTEGER)</pre>
<b>stm_r_di_by_structure_type_di</b>	<p><b>Query:</b> None</p> <p><b>Purpose:</b> Returns the data-items in the input list that have a particular structure type (for example, single, array, or queue)</p> <p><b>Syntax:</b></p> <pre>stm_r_di_by_structure_type_di (IN di_list: LIST OF DATA_ITEM, IN dtype: INTEGER, OUT status: INTEGER)</pre>
<b>stm_r_di_callback_binding_di</b>	<p><b>Query:</b> Data-items with callback bindings</p> <p><b>Purpose:</b> Returns the data-items of the input list that have callback bindings</p> <p><b>Syntax:</b></p> <pre>stm_r_di_callback_binding_di (IN el_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>

<b>stm_r_di_explicit_defined_di</b>	<p><b>Query:</b> Data-items explicitly defined</p> <p><b>Purpose:</b> Returns the data-items of the input list that were explicitly defined</p> <p><b>Syntax:</b></p> <pre>stm_r_di_explicit_defined_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_di_integer_di</b>	<p><b>Query:</b> Integer subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as integer in the <b>Structure/Type</b> field of the data-item's form</p> <p><b>Syntax:</b></p> <pre>stm_r_di_integer_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_di_integer_array_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as array of integer</p> <p><b>Syntax:</b></p> <pre>stm_r_di_integer_array_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_di_integer_queue_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as queue of integer</p> <p><b>Syntax:</b></p> <pre>stm_r_di_integer_queue_di (IN el_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_di_missing_di</b>	<p><b>Query:</b> Data-item by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list for which no type is defined</p> <p><b>Syntax:</b></p> <pre>stm_r_di_missing_di (IN el_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_di_name_of_di</b>	<p><b>Query:</b> Data-items whose names match a given pattern</p> <p><b>Purpose:</b> Returns all the data-items whose names match the specified pattern</p> <p><b>Syntax:</b></p> <pre>stm_r_di_name_of_di (IN pattern: STRING, OUT status: INTEGER)</pre>
<b>stm_r_di_parent_of_di</b>	<p><b>Query:</b> Parent data-items of a given data-item</p> <p><b>Purpose:</b> Returns the data-items containing the data-items from the input list (as defined in the <b>Consists of</b> field of the data-item's form)</p> <p><b>Syntax:</b></p> <pre>stm_r_di_parent_of_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>

<b>stm_r_di_queue_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as queue</p> <p><b>Syntax:</b> stm_r_di_queue_di (IN el_list: LIST OF DATA_ITEM, OUT status: INTEGER)</p>
<b>stm_r_di_queue_missing_di</b>	<p><b>Query:</b> Queues of data-items by subtype</p> <p><b>Purpose:</b> Returns the queues of data-items in the input list for which no type is defined</p> <p><b>Syntax:</b> stm_r_di_queue_missing_di (IN el_list: LIST OF DATA_ITEM, OUT status: INTEGER)</p>
<b>stm_r_di_real_di</b>	<p><b>Query:</b> Real subtype</p> <p><b>Purpose:</b> Returns the data-items from the input list that are defined as Real (Float) in the <b>Structure/Type</b> field of the data-item's form</p> <p><b>Syntax:</b> stm_r_di_real_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</p>
<b>stm_r_di_real_array_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as real</p> <p><b>Syntax:</b> stm_r_di_real_array_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</p>
<b>stm_r_di_real_queue_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as queue of real</p> <p><b>Syntax:</b> stm_r_di_real_queue_di (IN el_list: LIST OF DATA_ITEM, OUT status: INTEGER)</p>
<b>stm_r_di_record_array_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as array of record</p> <p><b>Syntax:</b> stm_r_di_record_array_di (IN el_list: LIST OF DATA_ITEM, OUT status: INTEGER)</p>
<b>stm_r_di_record_di</b>	<p><b>Query:</b> Record subtype</p> <p><b>Purpose:</b> Returns the data-items from the input list that are defined as Record in the <b>Structure/Type</b> field of the data-item's form</p> <p><b>Syntax:</b> stm_r_di_record_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</p>



<b>stm_r_di_single_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as single</p> <p><b>Syntax:</b></p> <pre>stm_r_di_single_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_di_string_array_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as array of string</p> <p><b>Syntax:</b></p> <pre>stm_r_di_string_array_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_di_string_di</b>	<p><b>Query:</b> String subtype</p> <p><b>Purpose:</b> Returns the data-items from the input list that are defined as String in the <b>Structure/Type</b> field of the data-item's form</p> <p><b>Syntax:</b></p> <pre>stm_r_di_string_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_di_string_queue_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as queue of string</p> <p><b>Syntax:</b></p> <pre>stm_r_di_string_queue_di (IN el_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_di_subdata_item_of_di</b>	<p><b>Query:</b> Subdata-item of a given data-item</p> <p><b>Purpose:</b> Returns the data-items that are components of data-items in the input list (as defined in the <b>Consists of</b> field of the data-item's form)</p> <p><b>Syntax:</b></p> <pre>stm_r_di_subdata_item_of_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_di_synonym_of_di</b>	<p><b>Query:</b> Data-items whose synonyms match a given pattern</p> <p><b>Purpose:</b> Returns all the data-items whose synonyms match the specified pattern</p> <p><b>Syntax:</b></p> <pre>stm_r_di_synonym_of_di (IN pattern: STRING, OUT status: INTEGER)</pre>
<b>stm_r_di_union_array_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as array</p> <p><b>Syntax:</b></p> <pre>stm_r_di_union_array_di (IN el_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>

<b>stm_r_di_union_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as union</p> <p><b>Syntax:</b>  <code>stm_r_di_union_di (IN el_list: LIST OF DATA_ITEM, OUT status: INTEGER)</code></p>
<b>stm_r_di_unresolved_di</b>	<p><b>Query:</b> Unresolved data-items</p> <p><b>Purpose:</b> Returns the unresolved data-items in the input list</p> <p><b>Syntax:</b>  <code>stm_r_di_unresolved_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</code></p>
<b>stm_r_di_user_type_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as user-defined type</p> <p><b>Syntax:</b>  <code>stm_r_di_user_type_di (IN el_list: LIST OF DATA_ITEM, OUT status: INTEGER)</code></p>
<b>stm_r_di_user_type_array_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as array of user-defined type</p> <p><b>Syntax:</b>  <code>stm_r_di_user_type_array_di (IN el_list: LIST OF DATA_ITEM, OUT status: INTEGER)</code></p>
<b>stm_r_di_user_type_queue_di</b>	<p><b>Query:</b> Data-items by subtype</p> <p><b>Purpose:</b> Returns the data-items in the input list that are defined as queue of user-defined type</p> <p><b>Syntax:</b>  <code>stm_r_di_user_type_queue_di (IN el_list: LIST OF DATA_ITEM, OUT status: INTEGER)</code></p>

### Input List Type: fd

<b>stm_r_di_containing_fd</b>	<p><b>Query:</b> Data-items containing a given field</p> <p><b>Purpose:</b> Returns the data-items (records or unions) in which the fields in the input list are defined</p> <p><b>Syntax:</b>  <code>stm_r_di_containing_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</code></p>
-------------------------------	--

### Input List Type: if

<b>stm_r_di_contained_in_if</b>	<p><b>Query:</b> Data-items contained in a given information-flow</p> <p><b>Purpose:</b> Returns the data-items contained in information-flow from the input list (as defined in the <b>Consists of</b> field of the information-flow's form)</p> <p><b>Syntax:</b>  <b>stm_r_di_contained_in_if</b> (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</p>
---------------------------------	--

### Input List Type: mf

<b>stm_r_di_flowning_through_mf</b>	<p><b>Query:</b> Data-items flowing through a given m-flow-line</p> <p><b>Purpose:</b> Returns the data-items actually flowing through m-flow-lines in the input list</p> <p><b>Syntax:</b>  <b>stm_r_di_flowning_through_mf</b> (IN mf_list: LIST OF M_FLOW_LINE, OUT status: INTEGER)</p>
<b>stm_r_di_labeling_mf</b>	<p><b>Query:</b> Data-items labeling a given m-flow-line</p> <p><b>Purpose:</b> Returns the data-items which label the m-flow-lines in the input list</p> <p><b>Syntax:</b>  <b>stm_r_di_labeling_mf</b> (IN mf_list: LIST OF M_FLOW_LINE, OUT status: INTEGER)</p>

## Data-Stores (ds)

This section documents the query functions that return a list of data-stores.

### Input List Type: ac

<b>stm_r_ds_contained_in_ac</b>	<b>Query:</b> Data-stores contained in a given activity <b>Purpose:</b> Returns the data-stores contained directly in activities from the input list <b>Syntax:</b> <code>stm_r_ds_contained_in_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</code>
<b>stm_r_ds_in_ac</b>	<b>Query:</b> Data-stores in a given activity <b>Purpose:</b> Returns the data-stores contained in the activities from the input list <b>Syntax:</b> <code>stm_r_ds_in_ac (IN el_list: LIST OF DATA_STORE, OUT status: INTEGER)</code>

### Input List Type: af

<b>stm_r_ds_source_of_af</b>	<b>Query:</b> Data-stores that are sources of a given a-flow-line <b>Purpose:</b> Returns the data-stores that are sources of a-flow-lines in the input list <b>Syntax:</b> <code>stm_r_ds_source_of_af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)</code>
<b>stm_r_ds_target_of_af</b>	<b>Query:</b> Data-stores that are targets of a given a-flow-line <b>Purpose:</b> Returns the data-stores that are targets of a-flow-lines in the input list <b>Syntax:</b> <code>stm_r_ds_target_of_af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)</code>

### Input List Type: ch

<b>stm_r_ds_def_or_unres_in_ch</b>	<p><b>Query:</b> Data-stores defined or unresolved in a given chart</p> <p><b>Purpose:</b> Returns the data-stores that are explicitly defined or unresolved in the charts of the input list</p> <p><b>Syntax:</b>  <code>stm_r_ds_def_or_unres_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>
<b>stm_r_ds_defined_in_ch</b>	<p><b>Query:</b> Data-stores defined in a given chart</p> <p><b>Purpose:</b> Returns the data-stores that are explicitly defined in the charts of the input list</p> <p><b>Syntax:</b>  <code>stm_r_ds_defined_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>
<b>stm_r_ds_unresolved_in_ch</b>	<p><b>Query:</b> Data-stores unresolved in a given chart</p> <p><b>Purpose:</b> Returns the data-stores that are unresolved in the charts of the input list</p> <p><b>Syntax:</b>  <code>stm_r_ds_unresolved_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>

### Input List Type: ds

<b>stm_r_ds_by_attributes_ds</b>	<p><b>Query:</b> Data-stores by attributes</p> <p><b>Purpose:</b> Returns the data-stores in the input list that match a given attribute name and value</p> <p><b>Syntax:</b>  <code>stm_r_ds_by_attributes_ds (IN ds_list: LIST OF DATA_STORE, IN attr_name: STRING, IN attr_value: STRING, OUT status: INTEGER)</code></p>
<b>stm_r_ds_explicit_defined_ds</b>	<p><b>Query:</b> Data-stores explicitly defined</p> <p><b>Purpose:</b> Returns the data-stores of the input list that were explicitly defined</p> <p><b>Syntax:</b>  <code>stm_r_ds_explicit_defined_ds (IN ds_list: LIST OF DATA_STORE, OUT status: INTEGER)</code></p>

<p><b>stm_r_ds_is_occurrence_of_ds</b></p>	<p><b>Query:</b> Data-store occurrences of a given data-store</p> <p><b>Purpose:</b> Returns the data-stores for which the data-stores in the input list appear in the <b>Is Data-store</b> field of their form</p> <p><b>Syntax:</b>  <code>stm_r_ds_is_occurrence_of_ds (IN ds_list: LIST OF DATA_STORE, OUT status: INTEGER)</code></p>
<p><b>stm_r_ds_is_principal_of_ds</b></p>	<p><b>Query:</b> Principal data-stores of a given data-store</p> <p><b>Purpose:</b> Returns the data-stores for which the data-stores in the input list appear in the <b>Is Data-store</b> field of their form</p> <p><b>Syntax:</b>  <code>stm_r_ds_is_principal_of_ds (IN ds_list: LIST OF DATA_STORE, OUT status: INTEGER)</code></p>
<p><b>stm_r_ds_name_of_ds</b></p>	<p><b>Query:</b> Data-store whose names match a given pattern</p> <p><b>Purpose:</b> Returns all the data-stores whose names match the specified pattern</p> <p><b>Syntax:</b>  <code>stm_r_ds_name_of_ds (IN pattern: STRING, OUT status: INTEGER)</code></p>
<p><b>stm_r_ds_synonym_of_ds</b></p>	<p><b>Query:</b> Data-store whose synonyms match a given pattern</p> <p><b>Purpose:</b> Returns all the data-stores whose synonyms match the specified pattern</p> <p><b>Syntax:</b>  <code>stm_r_ds_synonym_of_ds (IN pattern: STRING, OUT status: INTEGER)</code></p>
<p><b>stm_r_ds_unresolved_ds</b></p>	<p><b>Query:</b> Unresolved data-stores</p> <p><b>Purpose:</b> Returns the unresolved data-stores in the input list</p> <p><b>Syntax:</b>  <code>stm_r_ds_unresolved_ds (IN ds_list: LIST OF DATA_STORE, OUT status: INTEGER)</code></p>

### Input List Type: md

<p><b>stm_r_ds_resides_in_md</b></p>	<p><b>Query:</b> Data-stores residing in a given module.</p> <p><b>Purpose:</b> Returns the data-stores residing in modules from the input list. The module appears in the <b>Resides in Module</b> field of the data-store's form.</p> <p><b>Syntax:</b>  <code>stm_r_ds_resides_in_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</code></p>
--------------------------------------	--

## User-Defined Types (dt)

This section documents the query functions that return a list of data-types.

### Input List Type: ch

<b>stm_r_dt_def_or_unres_in_ch</b>	<p><b>Query:</b> User-defined types defined or unresolved in a given chart</p> <p><b>Purpose:</b> Returns the user-defined types that are explicitly defined or unresolved in the charts in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_def_or_unres_in_ch (IN el_list: LIST OF CHART, OUT status: INTEGER)</pre>
<b>stm_r_dt_defined_in_ch</b>	<p><b>Query:</b> User-defined types defined in a given chart</p> <p><b>Purpose:</b> Returns the user-defined types that are explicitly defined in the charts in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_defined_in_ch (IN el_list: LIST OF CHART, OUT status: INTEGER)</pre>
<b>stm_r_dt_unresolved_in_ch</b>	<p><b>Query:</b> User-defined types unresolved in a given chart</p> <p><b>Purpose:</b> Returns the user-defined types that are unresolved in the charts in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_unresolved_in_ch (IN el_list: LIST OF CHART, OUT status: INTEGER)</pre>

### Input List Type: dt

<b>stm_r_dt_array_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as array</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_array_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_dt_array_missing_dt</b>	<p><b>Query:</b> Arrays of user-defined type by subtype</p> <p><b>Purpose:</b> Returns the arrays of user-defined types in the input list for which no type is defined</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_array_missing_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_dt_bit_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as bit</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_bit_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_dt_bit_queue_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as queue of bit</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_bit_queue_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_dt_bits_array_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as array of bit array</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_bits_array_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_dt_bits_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as bit array</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_bits_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>



<b>stm_r_dt_bits_queue_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as queue of bit array</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_bits_queue_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_dt_by_attributes_dt</b>	<p><b>Query:</b> User-defined types by attribute</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that match a given attribute and value</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_by_attributes_dt (IN dt_list: LIST OF DATA_TYPE, IN attr_name: STRING, IN attr_value: STRING, OUT status: INTEGER)</pre>
<b>stm_r_dt_by_structure_type_dt</b>	<p><b>Query:</b> None</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that have a given structure type (for example, single, array or queue)</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_by_structure_type_dt (IN el_list: LIST OF DATA_TYPE, IN dtype: INTEGER, OUT status: INTEGER)</pre>
<b>stm_r_dt_condition_array_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as array of condition</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_condition_array_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_dt_condition_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defend types in the input list that are defined as condition</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_condition_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_dt_condition_queue_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as queue of condition</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_condition_queue_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_dt_enums_dt</b>	<p><b>Query:</b> User-defined types defined as enumerated types</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as enumerated types</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_enums_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>

<b>stm_r_dt_explicit_defined_dt</b>	<p><b>Query:</b> User-defined types explicitly defined</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are explicitly defined</p> <p><b>Syntax:</b>  <code>stm_r_dt_explicit_defined_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</code></p>
<b>stm_r_dt_integer_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as integer</p> <p><b>Syntax:</b>  <code>stm_r_dt_integer_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</code></p>
<b>stm_r_dt_integer_array_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as array of integer</p> <p><b>Syntax:</b>  <code>stm_r_dt_integer_array_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</code></p>
<b>stm_r_dt_integer_queue_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as queue of integer</p> <p><b>Syntax:</b>  <code>stm_r_dt_integer_queue_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</code></p>
<b>stm_r_dt_missing_dt</b>	<p><b>Query:</b> User-defined type by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list for which no type is defined</p> <p><b>Syntax:</b>  <code>stm_r_dt_missing_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</code></p>
<b>stm_r_dt_name_of_dt</b>	<p><b>Query:</b> User-defined types whose names match a given pattern</p> <p><b>Purpose:</b> Returns all user-defined types whose names match the specified pattern</p> <p><b>Syntax:</b>  <code>stm_r_dt_name_of_dt (IN pattern: STRING, OUT status: INTEGER)</code></p>
<b>stm_r_dt_queue_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as queue</p> <p><b>Syntax:</b>  <code>stm_r_dt_queue_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</code></p>

<b>stm_r_dt_queue_missing_dt</b>	<p><b>Query:</b> Queues of user-defined type by subtype</p> <p><b>Purpose:</b> Returns the queues of user-defined types in the input list for which no type is defined</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_queue_missing_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_dt_real_array_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as real</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_real_array_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_dt_real_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as real</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_real_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_dt_real_queue_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as queue of real</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_real_queue_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_dt_record_array_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as array of record</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_record_array_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_dt_record_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as record</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_record_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_dt_single_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as single</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_single_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>

<b>stm_r_dt_string_array_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as array of string</p> <p><b>Syntax:</b>  <code>stm_r_dt_string_array_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</code></p>
<b>stm_r_dt_string_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as string</p> <p><b>Syntax:</b>  <code>stm_r_dt_string_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</code></p>
<b>stm_r_dt_string_queue_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as queue of string</p> <p><b>Syntax:</b>  <code>stm_r_dt_string_queue_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</code></p>
<b>stm_r_dt_synonym_of_dt</b>	<p><b>Query:</b> User-defined types whose synonyms match a given pattern</p> <p><b>Purpose:</b> Returns all user-defined types whose synonyms match the specified pattern</p> <p><b>Syntax:</b> <code>stm_r_dt_synonym_of_dt (IN pattern: STRING, OUT status: INTEGER)</code></p>
<b>stm_r_dt_union_dt</b>	<p><b>Query:</b> User-defined type by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as union</p> <p><b>Syntax:</b>  <code>stm_r_dt_union_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</code></p>
<b>stm_r_dt_union_array_dt</b>	<p><b>Query:</b> User-defined type by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as array of union</p> <p><b>Syntax:</b>  <code>stm_r_dt_union_array_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</code></p>
<b>stm_r_dt_unresolved_dt</b>	<p><b>Query:</b> Unresolved user-defined types</p> <p><b>Purpose:</b> Returns the unresolved user-defined types in the input list</p> <p><b>Syntax:</b>  <code>stm_r_dt_unresolved_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</code></p>

<b>stm_r_dt_user_type_array_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as array of another user-defined type</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_user_type_array_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_dt_user_type_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as other user-defined type</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_user_type_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_dt_user_type_queue_dt</b>	<p><b>Query:</b> User-defined types by subtype</p> <p><b>Purpose:</b> Returns the user-defined types in the input list that are defined as queue of another user-defined type</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_user_type_queue_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>

### Input List Type: fd

<b>stm_r_dt_containing_fd</b>	<p><b>Query:</b> User-defined types containing a given field</p> <p><b>Purpose:</b> Returns the user-defined types (records or unions), in which the fields in the input list are defined</p> <p><b>Syntax:</b></p> <pre>stm_r_dt_containing_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</pre>
-------------------------------	--

## Events (ev)

This section documents the query functions that return a list of events.

### Input List Type: af

<b>stm_r_ev_flowng_through_af</b>	<p><b>Query:</b> Events flowing through the specified a-flow-line</p> <p><b>Purpose:</b> Returns the events actually flowing through a-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_ev_flowng_through_af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)</pre>
<b>stm_r_ev_labeling_af</b>	<p><b>Query:</b> Events labeling a given a-flow-line</p> <p><b>Purpose:</b> Returns the events that label the a-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_ev_labeling_af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)</pre>

### Input List Type: ch

<b>stm_r_ev_def_or_unres_in_ch</b>	<p><b>Query:</b> Events defined or unresolved in a given chart</p> <p><b>Purpose:</b> Returns the events that are explicitly defined or unresolved in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_ev_def_or_unres_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</pre>
<b>stm_r_ev_defined_in_ch</b>	<p><b>Query:</b> Events defined in a given chart</p> <p><b>Purpose:</b> Returns the events that are explicitly defined in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_ev_defined_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</pre>
<b>stm_r_ev_unresolved_in_ch</b>	<p><b>Query:</b> Events unresolved in a given chart</p> <p><b>Purpose:</b> Returns the events that are unresolved in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_ev_unresolved_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</pre>

## Input List Type: ev

<b>stm_r_ev_array_ev</b>	<p><b>Query:</b> Events by subtype</p> <p><b>Purpose:</b> Returns the events in the input list that are defined as array</p> <p><b>Syntax:</b></p> <pre>stm_r_ev_array_ev (IN ev_list: LIST OF EVENT, OUT status: INTEGER)</pre>
<b>stm_r_ev_by_attributes_ev</b>	<p><b>Query:</b> Events by attributes</p> <p><b>Purpose:</b> Returns the events in the input list that match the specified attribute name and value</p> <p><b>Syntax:</b></p> <pre>stm_r_ev_by_attributes_ev (IN ev_list: LIST OF EVENT, IN attr_name: STRING, IN attr_value: STRING, OUT status: INTEGER)</pre>
<b>stm_r_ev_by_structure_type_ev</b>	<p><b>Query:</b> None</p> <p><b>Purpose:</b> Returns the events in the input list that have the specified structure type (for example, single or array)</p> <p><b>Syntax:</b></p> <pre>stm_r_ev_by_structure_type_ev (IN ev_list: LIST OF EVENT, IN dtype: INTEGER, OUT status: INTEGER)</pre>
<b>stm_r_ev_callback_binding_ev</b>	<p><b>Query:</b> Events with callback bindings</p> <p><b>Purpose:</b> Returns the events in the input list that have callback bindings</p> <p><b>Syntax:</b></p> <pre>stm_r_ev_callback_binding_ev (IN el_list: LIST OF EVENT, OUT status: INTEGER)</pre>
<b>stm_r_ev_explicit_defined_ev</b>	<p><b>Query:</b> Events explicitly defined</p> <p><b>Purpose:</b> Returns the events of the input list that were explicitly defined</p> <p><b>Syntax:</b></p> <pre>stm_r_ev_explicit_defined_ev (IN ev_list: LIST OF EVENT, OUT status: INTEGER)</pre>
<b>stm_r_ev_name_of_ev</b>	<p><b>Query:</b> Events whose names match a given pattern</p> <p><b>Purpose:</b> Returns all the events whose names match the specified pattern</p> <p><b>Syntax:</b></p> <pre>stm_r_ev_name_of_ev (IN pattern: STRING, OUT status: INTEGER)</pre>

<b>stm_r_ev_single_ev</b>	<p><b>Query:</b> Events by subtype</p> <p><b>Purpose:</b> Returns the events in the input list that are defined as single</p> <p><b>Syntax:</b>  <code>stm_r_ev_single_ev (IN ev_list: LIST OF EVENT, OUT status: INTEGER)</code></p>
<b>stm_r_ev_synonym_of_ev</b>	<p><b>Query:</b> Events whose synonyms match a given pattern</p> <p><b>Purpose:</b> Returns all the events whose synonyms match the specified pattern</p> <p><b>Syntax:</b>  <code>stm_r_ev_synonym_of_ev (IN pattern: STRING, OUT status: INTEGER)</code></p>
<b>stm_r_ev_unresolved_ev</b>	<p><b>Query:</b> Unresolved events</p> <p><b>Purpose:</b> Returns the unresolved events in the input list</p> <p><b>Syntax:</b>  <code>stm_r_ev_unresolved_ev (IN ev_list: LIST OF EVENT, OUT status: INTEGER)</code></p>

### Input List Type: if

<b>stm_r_ev_contained_in_if</b>	<p><b>Query:</b> Events contained in a given information-flow</p> <p><b>Purpose:</b> Returns the events contained in information-flows from the input list (events used in the <b>Consists of</b> field of the information-flow's form)</p> <p><b>Syntax:</b>  <code>stm_r_ev_contained_in_if (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</code></p>
---------------------------------	--



**Input List Type: mf**

<b>stm_r_ev_floating_through_mf</b>	<p><b>Query:</b> Events flowing through a given m-flow-line</p> <p><b>Purpose:</b> Returns the events actually flowing through m-flow-lines from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_ev_floating_through_mf (IN mf_list: LIST OF M_FLOW_LINE, OUT status: INTEGER)</pre>
<b>stm_r_ev_labeling_mf</b>	<p><b>Query:</b> Events labeling a given m-flow-line</p> <p><b>Purpose:</b> Returns the events that label the m-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_ev_labeling_mf (IN mf_list: LIST OF M_FLOW_LINE, OUT status: INTEGER)</pre>

**Fields (fd)**

This section documents the queries that return a list of fields.

**Input List Type: ch**

<b>stm_r_fd_defined_in_ch</b>	<p><b>Query:</b> Fields defined in a given chart</p> <p><b>Purpose:</b> Returns the fields that are part of the structured data-items in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_fd_defined_in_ch (IN el_list: LIST OF CHART, OUT status: INTEGER)</pre>
-------------------------------	--

**Input List Type: di**

<b>stm_r_fd_contained_in_di</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as array</p> <p><b>Syntax:</b></p> <pre>stm_r_fd_contained_in_di (IN el_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
---------------------------------	---

### Input List Type: dt

<b>stm_r_fd_contained_in_dt</b>	<p><b>Query:</b> Fields contained in user-defined type (UDT)</p> <p><b>Purpose:</b> Returns the fields that are part of the structured UDTs in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_fd_contained_in_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
---------------------------------	--

### Input List Type: fd

<b>stm_r_fd_array_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as array</p> <p><b>Syntax:</b></p> <pre>stm_r_fd_array_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</pre>
<b>stm_r_fd_array_missing_fd</b>	<p><b>Query:</b> Array of fields by subtype</p> <p><b>Purpose:</b> Returns the array of fields in the input list for which no type is defined</p> <p><b>Syntax:</b></p> <pre>stm_r_fd_array_missing_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</pre>
<b>stm_r_fd_bit_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as bit</p> <p><b>Syntax:</b></p> <pre>stm_r_fd_bit_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</pre>
<b>stm_r_fd_bit_queue_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as queue of bit</p> <p><b>Syntax:</b></p> <pre>stm_r_fd_bit_queue_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</pre>
<b>stm_r_fd_bits_array_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as bit array</p> <p><b>Syntax:</b></p> <pre>stm_r_fd_bits_array_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</pre>

<b>stm_r_fd_bits_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as bit array</p> <p><b>Syntax:</b> stm_r_fd_bits_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</p>
<b>stm_r_fd_bits_queue_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as queue of bit array</p> <p><b>Syntax:</b> stm_r_fd_bits_queue_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</p>
<b>stm_r_fd_by_attributes_fd</b>	<p><b>Query:</b> Fields by attribute</p> <p><b>Purpose:</b> Returns the fields in the input list that match the specified attribute and value</p> <p><b>Syntax:</b> stm_r_fd_by_attributes_fd (IN fd_list: LIST OF FIELD, IN attr_name: STRING, IN attr_value: STRING, OUT status: INTEGER)</p>
<b>stm_r_fd_by_structure_type_fd</b>	<p><b>Query:</b> None</p> <p><b>Purpose:</b> Returns the fields in the input list that have the specified structure type (for example, single or array)</p> <p><b>Syntax:</b> stm_r_fd_by_structure_type_fd (IN el_list: LIST OF FIELD, IN dtype: INTEGER, OUT status: INTEGER)</p>
<b>stm_r_fd_condition_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as condition</p> <p><b>Syntax:</b> stm_r_fd_condition_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</p>
<b>stm_r_fd_condition_array_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as array of condition</p> <p><b>Syntax:</b> stm_r_fd_condition_array_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</p>
<b>stm_r_fd_condition_queue_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as queue of condition</p> <p><b>Syntax:</b> stm_r_fd_condition_queue_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</p>

<b>stm_r_fd_explicit_defined_fd</b>	<p><b>Query:</b> Fields explicitly defined</p> <p><b>Purpose:</b> Returns the fields in the input list that are explicitly defined</p> <p><b>Syntax:</b>  <code>stm_r_fd_explicit_defined_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</code></p>
<b>stm_r_fd_integer_array_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as array of integer</p> <p><b>Syntax:</b>  <code>stm_r_fd_integer_array_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</code></p>
<b>stm_r_fd_integer_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as integer</p> <p><b>Syntax:</b>  <code>stm_r_fd_integer_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</code></p>
<b>stm_r_fd_integer_queue_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as queue of integer</p> <p><b>Syntax:</b>  <code>stm_r_fd_integer_queue_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</code></p>
<b>stm_r_fd_missing_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list for which no type is defined</p> <p><b>Syntax:</b>  <code>stm_r_fd_missing_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</code></p>
<b>stm_r_fd_name_of_fd</b>	<p><b>Query:</b> Fields whose names match a given pattern</p> <p><b>Purpose:</b> Returns all fields whose name matches the specified pattern</p> <p><b>Syntax:</b>  <code>stm_r_fd_name_of_fd (IN pattern: STRING, OUT status: INTEGER)</code></p>
<b>stm_r_fd_queue_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as queue</p> <p><b>Syntax:</b>  <code>stm_r_fd_queue_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</code></p>

<b>stm_r_fd_queue_missing_fd</b>	<p><b>Query:</b> Queues of field by subtype</p> <p><b>Purpose:</b> Returns the queues of fields in the input list for which no type is defined</p> <p><b>Syntax:</b> stm_r_fd_queue_missing_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</p>
<b>stm_r_fd_real_array_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as array of real</p> <p><b>Syntax:</b> stm_r_fd_real_array_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</p>
<b>stm_r_fd_real_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as real</p> <p><b>Syntax:</b> stm_r_fd_real_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</p>
<b>stm_r_fd_real_queue_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as queue of real</p> <p><b>Syntax:</b> stm_r_fd_real_queue_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</p>
<b>stm_r_fd_single_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as single</p> <p><b>Syntax:</b> stm_r_fd_single_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</p>
<b>stm_r_fd_string_array_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as array of string</p> <p><b>Syntax:</b> stm_r_fd_string_array_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</p>
<b>stm_r_fd_string_fd</b>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as string</p> <p><b>Syntax:</b> stm_r_fd_string_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</p>

<p><b>stm_r_fd_string_queue_fd</b></p>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as queue of string</p> <p><b>Syntax:</b></p> <pre>stm_r_fd_string_queue_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</pre>
<p><b>stm_r_fd_user_type_array_fd</b></p>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as array of user-defined type</p> <p><b>Syntax:</b></p> <pre>stm_r_fd_user_type_array_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</pre>
<p><b>stm_r_fd_user_type_fd</b></p>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as user-defined type</p> <p><b>Syntax:</b></p> <pre>stm_r_fd_user_type_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</pre>
<p><b>stm_r_fd_user_type_queue_fd</b></p>	<p><b>Query:</b> Fields by subtype</p> <p><b>Purpose:</b> Returns the fields in the input list that are defined as queue of user-defined type</p> <p><b>Syntax:</b></p> <pre>stm_r_fd_user_type_queue_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</pre>

### Input List Type: mx

<p><b>stm_r_fd_contained_in_mx</b></p>	<p><b>Query:</b> Fields contained in a given element</p> <p><b>Purpose:</b> Returns the fields that are part of the structured elements (data-items and user-defined types) in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_fd_contained_in_mx (IN el_list: LIST OF ELEMENT, OUT status: INTEGER)</pre>
--	--

## Functions (fn)

This section documents the queries that return a list of functions.

### Input List Type: ch

<b>stm_r_fn_name_of_fn</b>	<b>Query:</b> Function names that match a given pattern <b>Purpose:</b> Returns all the functions whose names match the specified pattern <b>Syntax:</b> stm_r_fn_name_of_fn (IN pattern: STRING, OUT status: INTEGER)
<b>stm_r_fn_unresolved_in_ch</b>	<b>Query:</b> Functions unresolved in a given chart <b>Purpose:</b> Returns the functions that are unresolved in the charts of the input list <b>Syntax:</b> stm_r_fn_unresolved_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)

## Information-Flows (if)

This section documents the queries that return a list of information-flows.

### Input List Type: af

<b>stm_r_if_basic_flowng_af</b>	<p><b>Query:</b> Basic information-flows flowing through a given a-flow-line.</p> <p><b>Purpose:</b> Returns information-flows that are not decomposed to other information items, and are flowing through a-flow-lines in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_if_basic_flowng_af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)</pre>
<b>stm_r_if_flowng_through_af</b>	<p><b>Query:</b> Information-flows flowing through a given a-flow-line.</p> <p><b>Purpose:</b> Returns the information-flows flowing through a-flow-lines in the input list.</p> <p><b>Note:</b> This function returns the highest information-flows, as opposed to <code>stm_r_if_basic_flowng_af</code>, which returns the lowest level.</p> <p><b>Syntax:</b></p> <pre>stm_r_if_flowng_through_af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)</pre>
<b>stm_r_if_labeling_af</b>	<p><b>Query:</b> Information-flows labeling a given a-flow-line.</p> <p><b>Purpose:</b> Returns the information-flows that label a-flow-lines in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_if_labeling_af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)</pre>



### Input List Type: ch

<b>stm_r_if_def_or_unres_in_ch</b>	<p><b>Query:</b> Information-flows defined or unresolved in a given chart</p> <p><b>Purpose:</b> Returns the information-flows that are explicitly defined or unresolved in the charts of the input list</p> <p><b>Syntax:</b>  <code>stm_r_if_def_or_unres_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</code></p>
<b>stm_r_if_defined_in_ch</b>	<p><b>Query:</b> Information-flows defined in a given chart</p> <p><b>Purpose:</b> Returns the information-flows that are explicitly defined in the charts of the input list</p> <p><b>Syntax:</b>  <code>stm_r_if_defined_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>
<b>stm_r_if_unresolved_in_ch</b>	<p><b>Query:</b> Information-flows unresolved in a given chart</p> <p><b>Purpose:</b> Returns the information-flows that are unresolved in the charts of the input list</p> <p><b>Syntax:</b>  <code>stm_r_if_unresolved_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</code></p>

### Input List Type: co

<b>stm_r_if_containing_co</b>	<p><b>Query:</b> Information-flows containing a given condition</p> <p><b>Purpose:</b> Returns the information-flows containing conditions from the input list (conditions appearing in the <b>Consists of</b> field of the information-flow's form)</p> <p><b>Syntax:</b>  <code>stm_r_if_containing_co (IN co_list: LIST OF CONDITION, OUT status: INTEGER)</code></p>
-------------------------------	--

### Input List Type: di

<b>stm_r_if_containing_di</b>	<p><b>Query:</b> Information-flows containing a given data-item</p> <p><b>Purpose:</b> Returns the information-flows containing data-items from the input list (data-items appearing in the <b>Consists of</b> field of the information-flow's form)</p> <p><b>Syntax:</b></p> <pre>stm_r_if_containing_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
-------------------------------	---

### Input List Type: ev

<b>stm_r_if_containing_ev</b>	<p><b>Query:</b> Information-flows containing a given event</p> <p><b>Purpose:</b> Returns the information-flows containing events from the input list (events appearing in the <b>Consists of</b> field of the information-flow's form)</p> <p><b>Syntax:</b></p> <pre>stm_r_if_containing_ev (IN ev_list: LIST OF EVENT, OUT status: INTEGER)</pre>
-------------------------------	---

## Input List Type: if

<b>stm_r_if_basic_if</b>	<p><b>Query:</b> Basic information-flows</p> <p><b>Purpose:</b> Returns the information-flows in the input list that are basic (those not defined using other information-flows)</p> <p><b>Syntax:</b></p> <pre>stm_r_if_basic_if (IN el_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</pre>
<b>stm_r_if_by_attributes_if</b>	<p><b>Query:</b> Information-flows by attributes</p> <p><b>Purpose:</b> Returns the information-flows in the input list that match a particular attribute name and value</p> <p><b>Syntax:</b></p> <pre>stm_r_if_by_attributes_if (IN if_list: LIST OF INFORMATION_FLOW, IN attr_name: STRING, IN attr_value: STRING, OUT status: INTEGER)</pre>
<b>stm_r_if_contained_in_if</b>	<p><b>Query:</b> Information-flows contained in a given information-flow</p> <p><b>Purpose:</b> Returns the information-flows that are contained in information-flows from the input list (as defined in the <b>Consists of</b> field)</p> <p><b>Syntax:</b></p> <pre>stm_r_if_contained_in_if (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</pre>
<b>stm_r_if_containing_if</b>	<p><b>Query:</b> Information-flows containing a given information-flow</p> <p><b>Purpose:</b> Returns the information-flows that contain information-flows from the input list (as defined in the <b>Consists of</b> field)</p> <p><b>Syntax:</b></p> <pre>stm_r_if_containing_if (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</pre>
<b>stm_r_if_explicit_defined_if</b>	<p><b>Query:</b> Information-flows explicitly defined</p> <p><b>Purpose:</b> Returns the information-flows of the input list that were explicitly defined</p> <p><b>Syntax:</b></p> <pre>stm_r_if_explicit_defined_if (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</pre>
<b>stm_r_if_name_of_if</b>	<p><b>Query:</b> Information-flow names that match a given pattern</p> <p><b>Purpose:</b> Returns all the information-flows whose names match the specified pattern</p> <p><b>Syntax:</b> <code>stm_r_if_name_of_if (IN pattern: STRING, OUT status: INTEGER)</code></p>

<b>stm_r_if_synonym_of_if</b>	<p><b>Query:</b> Information-flow synonyms that match a given pattern</p> <p><b>Purpose:</b> Returns all the information-flows whose synonyms match the specified pattern</p> <p><b>Syntax:</b></p> <pre>stm_r_if_synonym_of_if (IN pattern: STRING, OUT status: INTEGER)</pre>
<b>stm_r_if_unresolved_if</b>	<p><b>Query:</b> Unresolved information-flows</p> <p><b>Purpose:</b> Returns the unresolved information-flows in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_if_unresolved_if (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</pre>

### Input List Type: mf

<b>stm_r_if_basic_flowng_mf</b>	<p><b>Query:</b> Basic information-flows flowing through a given m-flow-line.</p> <p><b>Purpose:</b> Returns the information-flows that are not decomposed to other information items and are flowing through m-flow-lines in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_if_basic_flowng_mf (IN mf_list: LIST OF M_FLOW_LINE, OUT status: INTEGER)</pre>
<b>stm_r_if_flowng_through_mf</b>	<p><b>Query:</b> Information-flows flowing through a given m-flow-line.</p> <p><b>Purpose:</b> Returns the information-flow flowing through m-flow-lines in the input list.</p> <p><b>Note:</b> This function returns the highest information-flows as opposed to <code>stm_r_if_basic_flowng_mf</code>, which returns the lowest level.</p> <p><b>Syntax:</b></p> <pre>stm_r_if_flowng_through_mf (IN mf_list: LIST OF M_FLOW_LINE, OUT status: INTEGER)</pre>
<b>stm_r_if_labeling_mf</b>	<p><b>Query:</b> Information-flows labeling a given m-flow-line.</p> <p><b>Purpose:</b> Returns the information-flow labeling m-flow-lines in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_if_labeling_mf (IN mf_list: LIST OF M_FLOW_LINE, OUT status: INTEGER)</pre>

## M-Flow-Lines (bf, bm, lmf, mf)

This section documents the queries that return a list of m-flow-lines. The types are as follows:

- ◆ bf—Basic m-flow-lines
- ◆ lmf—Local m-flow-lines
- ◆ mf—Global (compound) m-flow-lines

### Output List Type: bf

#### Input List Type: co

<b>stm_r_bf_within_flows_co</b>	<p><b>Query:</b> A-flow-lines through which a given condition flows</p> <p><b>Purpose:</b> Returns the a-flow-lines through which conditions from the input list actually flow</p> <p><b>Syntax:</b></p> <pre>stm_r_bf_within_flows_co (IN co_list: LIST OF CONDITION, OUT status: INTEGER)</pre>
<b>stm_r_bf_within_labels_co</b>	<p><b>Query:</b> A-flow-lines labeled by a given condition</p> <p><b>Purpose:</b> Returns the a-flow-lines labeled with conditions in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_bf_within_labels_co (IN co_list: LIST OF CONDITION, OUT status: INTEGER)</pre>

#### Input List Type: di

<b>stm_r_bf_within_flows_di</b>	<p><b>Query:</b> A-flow-lines through which a given data-item flows</p> <p><b>Purpose:</b> Returns the a-flow-lines through which data-items from the input list actually flow</p> <p><b>Syntax:</b></p> <pre>stm_r_bf_within_flows_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_bf_within_labels_di</b>	<p><b>Query:</b> A-flow-lines labeled by a given data-item</p> <p><b>Purpose:</b> Returns the a-flow-lines labeled with data-items in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_bf_within_labels_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>

### Input List Type: ev

<b>stm_r_bf_within_flows_ev</b>	<p><b>Query:</b> A-flow-lines through which a given event flows</p> <p><b>Purpose:</b> Returns the a-flow-lines through which events from the input list actually flow</p> <p><b>Syntax:</b>  <code>stm_r_bf_within_flows_ev (IN ev_list: LIST OF EVENT, OUT status: INTEGER)</code></p>
<b>stm_r_bf_within_labels_ev</b>	<p><b>Query:</b> A-flow-lines labeled by a given event</p> <p><b>Purpose:</b> Returns the a-flow-lines labeled with events in the input list</p> <p><b>Syntax:</b>  <code>stm_r_bf_within_labels_ev (IN ev_list: LIST OF EVENT, OUT status: INTEGER)</code></p>

### Input List Type: if

<b>stm_r_bf_within_flows_if</b>	<p><b>Query:</b> A-flow-lines through which a given information-flow flows</p> <p><b>Purpose:</b> Returns the a-flow-lines through which information-flows from the input list actually flow</p> <p><b>Syntax:</b>  <code>stm_r_bf_within_flows_if (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</code></p>
<b>stm_r_bf_within_labels_if</b>	<p><b>Query:</b> A-flow-lines labeled by a given information-flow</p> <p><b>Purpose:</b> Returns the a-flow-lines labeled with information-flows in the input list</p> <p><b>Syntax:</b>  <code>stm_r_bf_within_labels_if (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</code></p>

## Input List Type: mx

<b>stm_r_bf_from_source_mx</b>	<p><b>Query:</b> A-flow-lines whose source is a given element</p> <p><b>Purpose:</b> Returns basic a-flow-lines that originate at elements in the input list</p> <p><b>Syntax:</b> stm_r_bf_from_source_mx (IN bl_list: LIST OF ELEMENT, OUT status: INTEGER)</p>
<b>stm_r_bf_to_target_mx</b>	<p><b>Query:</b> A-flow-lines whose target is a given element</p> <p><b>Purpose:</b> Returns the basic a-flow-lines whose target is an element from the input list</p> <p><b>Syntax:</b> stm_r_bf_to_target_mx (IN bl_list: LIST OF ELEMENT, OUT status: INTEGER)</p>
<b>stm_r_bf_within_flows_mx</b>	<p><b>Query:</b> A-flow-lines through which a given element flows</p> <p><b>Purpose:</b> Returns the a-flow-lines through which elements from the input list actually flow</p> <p><b>Syntax:</b> stm_r_bf_within_flows_mx (IN if_list: LIST OF ELEMENT, OUT status: INTEGER)</p>
<b>stm_r_bf_within_labels_mx</b>	<p><b>Query:</b> A-flow-lines labeled by a given elements</p> <p><b>Purpose:</b> Returns the a-flow-lines labeled with elements in the input list</p> <p><b>Syntax:</b> stm_r_bf_within_labels_mx (IN di_list: LIST OF ELEMENT, OUT status: INTEGER)</p>

**Output List Type: bm**

**Input List Type: mf**

<p><b>stm_r_bm_contained_in_mf</b></p>	<p><b>Query:</b> M-flow-lines labeled by given elements.  <b>Purpose:</b> Returns the basic m-flow-lines contained in the m-flow-lines in the input list.  <b>Syntax:</b>          STM_R_BM_CONTAINED_IN_MF(IN mf_list: LIST OF M_FLOW_LINE, OUT status: INTEGER):LIST OF M_FLOW_LINE</p>
<p><b>stm_r_bm_enter_om</b></p>	<p><b>Query:</b> M-flow-lines labeled by given elements.  <b>Purpose:</b> Returns the basic m-flow-lines that enter the module-occurrences in the input list.  <b>Syntax:</b>          STM_R_BM_ENTER_OM(IN om_list: LIST OF ELEMENT, OUT status: INTEGER):LIST OF M_FLOW_LINE</p>



**Output List Type: lmf****Input List Type: bm**

<b>stm_r_lmf_containing_bm</b>	<p><b>Query:</b> M-flow-lines labeled by a given elements</p> <p><b>Purpose:</b> Returns the a-flow-lines labeled with elements in the input list</p> <p><b>Syntax:</b></p> <pre>STM_R_MF_CONTAINING_BM(IN bm_list: LIST OF M_FLOW_LINE, OUT status: INTEGER):LIST OF M_FLOW_LINE</pre>
--------------------------------	---

**Input List Type: md**

<b>stm_r_lmf_from_source_md</b>	<p><b>Query:</b> M-flow-lines whose source is a given module within chart</p> <p><b>Purpose:</b> Returns the local compound m-flow-lines (those within charts) whose source is a module from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_lmf_from_source_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>
<b>stm_r_lmf_input_to_md</b>	<p><b>Query:</b> M-flow-lines input to a given module within the chart</p> <p><b>Purpose:</b> Returns all the local compound m-flow-lines that originate outside and terminate at (or inside) modules in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_lmf_input_to_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>
<b>stm_r_lmf_output_from_md</b>	<p><b>Query:</b> M-flow-lines output from a given module within that chart</p> <p><b>Purpose:</b> Returns all the local compound m-flow-lines that originate at (or inside) and terminate outside modules in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_lmf_output_from_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>
<b>stm_r_lmf_to_target_md</b>	<p><b>Query:</b> M-flow-lines whose target is a given module</p> <p><b>Purpose:</b> Returns the local m-flow-lines whose target is a module from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_lmf_to_target_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>

### Input List Type: mf

<b>stm_r_lmf_contained_in_mf</b>	<p><b>Query:</b> None</p> <p><b>Purpose:</b> Returns the local m-flow-lines that contain the global m-flow-lines in the input list</p> <p><b>Syntax:</b>  <code>stm_r_lmf_contained_in_mf (IN af_list: LIST OF M_FLOW_LINE, OUT status: INTEGER)</code></p>
----------------------------------	---

### Output List Type: mf

#### Input List Type: co

<b>stm_r_mf_within_flows_co</b>	<p><b>Query:</b> M-flow-lines through which a given condition flows</p> <p><b>Purpose:</b> Returns the m-flow-lines through which conditions from the input list actually flow</p> <p><b>Syntax:</b>  <code>stm_r_mf_within_flows_co (IN co_list: LIST OF CONDITION, OUT status: INTEGER)</code></p>
<b>stm_r_mf_within_labels_co</b>	<p><b>Query:</b> M-flow-lines labeled by a given condition</p> <p><b>Purpose:</b> Returns the m-flow-lines that are labeled by the conditions in the input list</p> <p><b>Syntax:</b>  <code>stm_r_mf_within_labels_co (IN co_list: LIST OF CONDITION, OUT status: INTEGER)</code></p>

## Input List Type: di

<b>stm_r_mf_within_flows_di</b>	<p><b>Query:</b> M-flow-lines through which a given data-item flows</p> <p><b>Purpose:</b> Returns the m-flow-lines through which data-items from the input list actually flow</p> <p><b>Syntax:</b></p> <pre>stm_r_mf_within_flows_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<b>stm_r_mf_within_labels_di</b>	<p><b>Query:</b> M-flow-lines labeled by a given data-item</p> <p><b>Purpose:</b> Returns the m-flow-lines that are labeled by the data-items in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mf_within_labels_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>

## Input List Type: ev

<b>stm_r_mf_within_flows_ev</b>	<p><b>Query:</b> M-flow-lines through which a given event flows</p> <p><b>Purpose:</b> Returns the m-flow-lines through which events from the input list actually flow</p> <p><b>Syntax:</b></p> <pre>stm_r_mf_within_flows_ev (IN ev_list: LIST OF EVENT, OUT status: INTEGER)</pre>
<b>stm_r_mf_within_labels_ev</b>	<p><b>Query:</b> M-flow-lines labeled by a given event</p> <p><b>Purpose:</b> Returns the m-flow-lines that are labeled by the events in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mf_within_labels_ev (IN ev_list: LIST OF EVENT, OUT status: INTEGER)</pre>

### Input List Type: if

<p><b>stm_r_mf_within_flows_if</b></p>	<p><b>Query:</b> M-flow-lines through which a given information-flow flows</p> <p><b>Purpose:</b> Returns the m-flow-lines through which information-flows from the input list actually flow</p> <p><b>Syntax:</b>  <code>stm_r_mf_within_flows_if (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</code></p>
<p><b>stm_r_mf_within_labels_if</b></p>	<p><b>Query:</b> M-flow-lines labeled with a given information-flow</p> <p><b>Purpose:</b> Returns the m-flow-lines that are labeled with information-flows in the input list</p> <p><b>Syntax:</b>  <code>stm_r_mf_within_labels_if (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</code></p>

### Input List Type: lmf

<p><b>stm_r_mf_containing_lmf</b></p>	<p><b>Query:</b> None</p> <p><b>Purpose:</b> Returns the global m-flow-lines (which might spread over several charts) that contain the local m-flow-lines (those within charts) in the input list</p> <p><b>Syntax:</b>  <code>stm_r_mf_containing_lmf (IN af_list: LIST OF M_FLOW_LINE, OUT status: INTEGER)</code></p>
---------------------------------------	--

## Input List Type: md

<b>stm_r_mf_from_source_md</b>	<p><b>Query:</b> M-flow-lines whose source is a given module</p> <p><b>Purpose:</b> Returns the global compound m-flow-lines (those that might spread over several charts) whose source is a module from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mf_from_source_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>
<b>stm_r_mf_input_to_md</b>	<p><b>Query:</b> M-flow-lines input to a given module</p> <p><b>Purpose:</b> Returns all the global compound m-flow-lines that originate outside and terminate at (or inside) modules in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mf_input_to_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>
<b>stm_r_mf_output_from_md</b>	<p><b>Query:</b> M-flow-lines output from a given module</p> <p><b>Purpose:</b> Returns all the global compound m-flow-lines that originate at (or inside) and terminate outside modules in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mf_output_from_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>
<b>stm_r_mf_to_target_md</b>	<p><b>Query:</b> M-flow-lines whose target is a given module</p> <p><b>Purpose:</b> Returns the global compound m-flow-lines whose target is a module from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mf_to_target_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>
<b>stm_r_lmf_to_target_md</b>	<p><b>Query:</b> M-flow-lines whose target is a given module</p> <p><b>Purpose:</b> Returns the local compound m-flow-lines whose target is a module from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_lmf_to_target_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>

**Input List Type: mx**

<b>stm_r_mf_within_flows_mx</b>	<p><b>Query:</b> M-flow-lines through which a given element flows</p> <p><b>Purpose:</b> Returns the m-flow-lines through which elements from the input list actually flow</p> <p><b>Syntax:</b></p> <pre>stm_r_mf_within_flows_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_mf_within_labels_mx</b>	<p><b>Query:</b> M-flow-lines that are labeled by a given information-flow</p> <p><b>Purpose:</b> Returns the m-flow-lines that are labeled with elements in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mf_within_labels_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>

## Modules (md)

This section documents the queries that return a list of modules.

### Input List Type: ac

<b>stm_r_md_carrying_out_ac</b>	<p><b>Query:</b> Modules carrying out a given activity.</p> <p><b>Purpose:</b> Returns the modules carrying out activities in the input list. The modules appear in the <b>Implemented by Module</b> field of an activity's form.</p> <p><b>Syntax:</b></p> <pre>stm_r_md_carrying_out_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</pre>
---------------------------------	---

### Input List Type: ch

<b>stm_r_md_def_or_unres_in_ch</b>	<p><b>Query:</b> Modules defined or unresolved in a given chart</p> <p><b>Purpose:</b> Returns the modules that are explicitly defined or unresolved in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_md_def_or_unres_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</pre>
<b>stm_r_md_defined_in_ch</b>	<p><b>Query:</b> Modules defined in a given chart</p> <p><b>Purpose:</b> Returns the modules that are explicitly defined in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_md_defined_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</pre>
<b>stm_r_md_described_by_ch</b>	<p><b>Query:</b> Modules described by a given activity-chart</p> <p><b>Purpose:</b> Returns the modules described by activity-charts in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_md_described_by_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</pre>

<b>stm_r_md_instance_of_ch</b>	<p><b>Query:</b> Modules instance of a given chart</p> <p><b>Purpose:</b> Returns the instance modules defined by the charts in the input list</p> <p><b>Syntax:</b>  <code>stm_r_md_instance_of_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>
<b>stm_r_md_root_in_ch</b>	<p><b>Query:</b> Root modules of a given chart</p> <p><b>Purpose:</b> Returns the internally defined modules (of type diagram) attached to the charts in the input list</p> <p><b>Syntax:</b>  <code>stm_r_md_root_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>
<b>stm_r_md_top_level_in_ch</b>	<p><b>Query:</b> Top-level modules of a given chart</p> <p><b>Purpose:</b> Returns the top level modules (not contained in any box) of the charts in the input list</p> <p><b>Syntax:</b>  <code>stm_r_md_top_level_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>
<b>stm_r_md_unresolved_in_ch</b>	<p><b>Query:</b> Modules unresolved in a given chart</p> <p><b>Purpose:</b> Returns the modules that are unresolved in the charts of the input list</p> <p><b>Syntax:</b>  <code>stm_r_md_unresolved_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>

### Input List Type: ds

<b>stm_r_md_contains_ds</b>	<p><b>Query:</b> Modules in which a given data-store resides.</p> <p><b>Purpose:</b> Returns the modules in which data-stores from the input list resides. The modules appear in the <b>Resides in Module</b> field of a data-store's form.</p> <p><b>Syntax:</b>  <code>stm_r_md_contains_ds (IN ds_list: LIST OF DATA_STORE, OUT status: INTEGER)</code></p>
-----------------------------	--



## Input List Type: md

<b>stm_r_md_basic_md</b>	<p><b>Query:</b> Basic modules</p> <p><b>Purpose:</b> Returns the modules in the input list that are basic modules (those that have no descendants)</p> <p><b>Syntax:</b></p> <pre>stm_r_md_basic_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>
<b>stm_r_md_bus_md</b>	<p><b>Query:</b> Bus modules</p> <p><b>Purpose:</b> Returns the modules in the input list that are bus modules</p> <p><b>Syntax:</b></p> <pre>stm_r_md_bus_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>
<b>stm_r_md_by_attributes_md</b>	<p><b>Query:</b> Modules by attributes</p> <p><b>Purpose:</b> Returns the modules in the input list that match a particular attribute name and value</p> <p><b>Syntax:</b></p> <pre>stm_r_md_by_attributes_md (IN md_list: LIST OF MODULE, IN attr_name: STRING, IN attr_value: STRING, OUT status: INTEGER)</pre>
<b>stm_r_md_control_md</b>	<p><b>Query:</b> Control modules</p> <p><b>Purpose:</b> Returns the modules in the input list that are control modules</p> <p><b>Syntax:</b></p> <pre>stm_r_md_control_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>
<b>stm_r_md_def_of_instance_md</b>	<p><b>Query:</b> Definition modules of a given module</p> <p><b>Purpose:</b> Returns the definition modules (top level modules in a definition chart) for instances in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_md_def_of_instance_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>
<b>stm_r_md_defined_environment_md</b>	<p><b>Query:</b> Environment modules</p> <p><b>Purpose:</b> Returns the modules from the input list that were defined as environment modules</p> <p><b>Syntax:</b></p> <pre>stm_r_md_defined_environment_md (IN md_list: LIST OF MODULE,OUT status: INTEGER)</pre>

<b>stm_r_md_environment_md</b>	<p><b>Query:</b> Environment modules</p> <p><b>Purpose:</b> Returns the modules in the input list that are environment modules</p> <p><b>Syntax:</b>  <code>stm_r_md_environment_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</code></p>
<b>stm_r_md_explicit_defined_md</b>	<p><b>Query:</b> Modules explicitly defined</p> <p><b>Purpose:</b> Returns the modules of the input list that were explicitly defined</p> <p><b>Syntax:</b>  <code>stm_r_md_explicit_defined_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</code></p>
<b>stm_r_md_external_md</b>	<p><b>Query:</b> External modules</p> <p><b>Purpose:</b> Returns the modules in the input list that are external</p> <p><b>Syntax:</b>  <code>stm_r_md_external_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</code></p>
<b>stm_r_md_generic_instance_md</b>	<p><b>Query:</b> Generic instance modules</p> <p><b>Purpose:</b> Returns the modules in the input list that are instances of generic charts</p> <p><b>Syntax:</b>  <code>stm_r_md_generic_instance_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</code></p>
<b>stm_r_md_instance_md</b>	<p><b>Query:</b> Instance modules</p> <p><b>Purpose:</b> Returns the instance modules from the modules in the input list</p> <p><b>Syntax:</b>  <code>stm_r_md_instance_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</code></p>
<b>stm_r_md_instance_of_def_m</b>	<p><b>Query:</b> Instance modules of a given definition module</p> <p><b>Purpose:</b> Returns the instance modules for definition modules (top-level modules in a definition chart) in the input list</p> <p><b>Syntax:</b>  <code>stm_r_md_instance_of_def_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</code></p>
<b>stm_r_md_library_md</b>	<p><b>Query:</b> Library modules</p> <p><b>Purpose:</b> Returns the modules from the input list that are library modules</p> <p><b>Syntax:</b>  <code>stm_r_md_library_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</code></p>

<b>stm_r_md_logical_desc_of_md</b>	<p><b>Query:</b> Logical descendants of a given module</p> <p><b>Purpose:</b> Returns the logical descendants of the modules in the input list, taking into account the translation of instances to their definition charts</p> <p><b>Syntax:</b> stm_r_md_logical_desc_of_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</p>
<b>stm_r_md_logical_parent_of_md</b>	<p><b>Query:</b> Logical parent modules of a given module</p> <p><b>Purpose:</b> Returns the logical parent modules of the modules in the input list, taking into account the translation of instances to their definition charts</p> <p><b>Syntax:</b> stm_r_md_logical_parent_of_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</p>
<b>stm_r_md_logical_sub_of_md</b>	<p><b>Query:</b> Logical submodules of a given module</p> <p><b>Purpose:</b> Returns the logical submodules of the modules in the input list, taking into account the translation of instances to their definition charts</p> <p><b>Syntax:</b> stm_r_md_logical_sub_of_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</p>
<b>stm_r_md_name_of_md</b>	<p><b>Query:</b> Modules whose names match a given pattern</p> <p><b>Purpose:</b> Returns all the modules whose names match the specified pattern</p> <p><b>Syntax:</b> stm_r_md_name_of_md (IN pattern: STRING, OUT status: INTEGER)</p>
<b>stm_r_md_offpage_instance_md</b>	<p><b>Query:</b> Offpage instance modules</p> <p><b>Purpose:</b> Returns the modules in the input list that are instances of offpage charts</p> <p><b>Syntax:</b> stm_r_md_offpage_instance_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</p>
<b>stm_r_md_physical_desc_of_md</b>	<p><b>Query:</b> Physical descendants of a given module</p> <p><b>Purpose:</b> Returns the physical descendants (those within the same chart) for the modules in the input list</p> <p><b>Syntax:</b> stm_r_md_physical_desc_of_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</p>

<b>stm_r_md_physical_parent_of_md</b>	<p><b>Query:</b> Physical parent modules of a given module</p> <p><b>Purpose:</b> Returns the physical parent modules (those within the same chart) for the modules in the input list</p> <p><b>Syntax:</b>  <code>stm_r_md_physical_parent_of_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</code></p>
<b>stm_r_md_physical_sub_of_md</b>	<p><b>Query:</b> Physical submodules of a given module</p> <p><b>Purpose:</b> Returns the physical submodules (those within the same chart) for the modules in the input list</p> <p><b>Syntax:</b>  <code>stm_r_md_physical_sub_of_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</code></p>
<b>stm_r_md_regular_md</b>	<p><b>Query:</b> Regular modules</p> <p><b>Purpose:</b> Returns the modules from the input list that are regular modules (not environment or storage)</p> <p><b>Syntax:</b>  <code>stm_r_md_regular_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</code></p>
<b>stm_r_md_resolved_to_ext_md</b>	<p><b>Query:</b> Modules resolved to a given external module</p> <p><b>Purpose:</b> Returns the modules (internal, external, or environment) to which the external modules in the input list are resolved</p> <p><b>Syntax:</b>  <code>stm_r_md_resolved_to_ext_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</code></p>
<b>stm_r_md_storage_md</b>	<p><b>Query:</b> Storage modules</p> <p><b>Purpose:</b> Returns the modules from the input list that are storage modules</p> <p><b>Syntax:</b>  <code>stm_r_md_storage_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</code></p>
<b>stm_r_md_synonym_of_md</b>	<p><b>Query:</b> Modules whose synonyms match a given pattern</p> <p><b>Purpose:</b> Returns all the modules whose synonyms match the specified pattern</p> <p><b>Syntax:</b>  <code>stm_r_md_synonym_of_md (IN pattern: STRING, OUT status: INTEGER)</code></p>
<b>stm_r_md_unresolved_md</b>	<p><b>Query:</b> Unresolved modules</p> <p><b>Purpose:</b> Returns the unresolved modules in the input list</p> <p><b>Syntax:</b>  <code>stm_r_md_unresolved_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</code></p>

**Input List Type: mf**

<b>stm_r_md_source_of_mf</b>	<p><b>Query:</b> Modules that are sources of a given m-flow-line</p> <p><b>Purpose:</b> Returns the modules that are sources of m-flow-lines from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_md_source_of_mf (IN mf_list: LIST OF M_FLOW_LINE, OUT status: INTEGER)</pre>
<b>stm_r_md_target_of_mf</b>	<p><b>Query:</b> Modules that are targets of a given m-flow-line</p> <p><b>Purpose:</b> Returns the modules that are targets of m-flow-lines from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_md_target_of_mf (IN mf_list: LIST OF M_FLOW_LINE, OUT status: INTEGER)</pre>

**Input List Type: router**

<b>stm_r_md_contains_router</b>	<p><b>Query:</b> Modules in which a given router resides.</p> <p><b>Purpose:</b> Returns the modules in which routers from the input list resides. The modules appear in the <b>Resides in Module</b> field of a router's form.</p> <p><b>Syntax:</b></p> <pre>stm_r_md_contains_router (IN router_list: LIST OF ROUTER, OUT status: INTEGER)</pre>
---------------------------------	---

## Mixed (mx)

This section documents the queries that return a list of elements.

### Input List Type: af

<b>stm_r_mx_flowng_through_af</b>	<p><b>Query:</b> Elements flowing through a given a-flow-line</p> <p><b>Purpose:</b> Returns the information elements (conditions, events, data-items, and basic information-flows) that actually flow through the a-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_flowng_through_af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)</pre>
<b>stm_r_mx_labeling_af</b>	<p><b>Query:</b> Elements labeling a given a-flow-line</p> <p><b>Purpose:</b> Returns the elements that label a-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_labeling_af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)</pre>
<b>stm_r_mx_source_of_af</b>	<p><b>Query:</b> Elements that are sources of a given a-flow-line</p> <p><b>Purpose:</b> Returns the elements (activities and data-stores) that are sources of a-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_source_of_af (IN af_list: LIST OF A_FLOW_LINE,OUT status: INTEGER)</pre>
<b>stm_r_mx_target_of_af</b>	<p><b>Query:</b> Elements that are targets of a given a-flow-line</p> <p><b>Purpose:</b> Returns the elements (activities and data-stores) that are sources of a-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_target_of_af (IN af_list: LIST OF A_FLOW_LINE,OUT status: INTEGER)</pre>
<b>stm_r_mx_source_of_ba</b>	<p><b>Query:</b> Elements affected by a given activity.</p> <p><b>Purpose:</b> Returns the elements that are sources of basic a-flow-lines in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_source_of_ ba (stm_list in_list, int *status);</pre>
<b>stm_r_mx_target_of_ba</b>	<p><b>Query:</b> Elements affected by a given activity.</p> <p><b>Purpose:</b> Returns the elements that are targets of basic a-flow-lines in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_target_of_ ba (stm_list in_list, int *status);</pre>

<b>stm_r_mx_target_of_ba</b>	<p><b>Query:</b> Elements that are targets of a given flow line.</p> <p><b>Purpose:</b> Returns the elements that are targets of basic a-flow-lines in the input list</p> <p><b>Syntax:</b>  STM_R_MX_TARGET_OF_BA(IN ba_list: LIST OF A_FLOW_LINE, OUT status: INTEGER): LIST OF ELEMENT;</p>
------------------------------	--

### Input List Type: ac

<b>stm_r_mx_affected_by_ac</b>	<p><b>Query:</b> Elements affected by a given activity.</p> <p><b>Purpose:</b> Returns the elements (data-items, conditions, and events) affected (modified or generated) by activities, in mini-specs, and combinational assignments in the input list.</p> <p><b>Syntax:</b>  stm_r_mx_affected_by_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_mx_influence_ac</b>	<p><b>Query:</b> Elements that are referenced by or influence a given activity.</p> <p><b>Purpose:</b> Returns the elements used in all levels by the activities in the input list.  This includes all logical descendant activities, a-flow-lines that enter or exit these activities, elements that appear in the various fields of these activities, and in the labels of the flow-lines and their components.</p> <p><b>Syntax:</b>  stm_r_mx_influence_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_mx_influenced_by_ac</b>	<p><b>Query:</b> Elements that refer to, or are influenced by, a given activity.</p> <p><b>Purpose:</b> Returns the elements that directly or indirectly use the activities in the input list.  This query identifies all the elements, in all levels, that refer to or affect the input activities.</p> <p><b>Syntax:</b>  stm_r_mx_influenced_by_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<b>stm_r_mx_refer_to_ac</b>	<p><b>Query:</b> Elements that refer to a given activity.</p> <p><b>Purpose:</b> Returns the elements that directly refer to activities in the input list.  This query identifies where input activities are used.</p> <p><b>Syntax:</b>  stm_r_mx_refer_to_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>

<p><b>stm_r_mx_referenced_by_ac</b></p>	<p><b>Query:</b> Elements that are referenced by a given activity.</p> <p><b>Purpose:</b> Returns the elements that appear in the activities of the input list. This includes all physical descendant activities, a-flow-lines that enter or exit these activities, elements that appear in the various fields of these activities, and in the labels of the flow-lines.</p> <p><b>Syntax:</b> stm_r_mx_referenced_by_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<p><b>stm_r_mx_resolved_to_ext_ac</b></p>	<p><b>Query:</b> Elements resolved to a given external activity.</p> <p><b>Purpose:</b> Returns the activities and modules (internal, external, or environment) to which the external activities in the input list are resolved.</p> <p><b>Syntax:</b> stm_r_mx_resolved_to_ext_ac (IN ac_list: LIST OF ACTIVITY,OUT status: INTEGER)</p>
<p><b>stm_r_mx_used_by_ac</b></p>	<p><b>Query:</b> Elements used by a given activity.</p> <p><b>Purpose:</b> Returns the elements (data-items, conditions, and events) used (evaluated) by activities in mini-specs and combinational assignments in the input list.</p> <p><b>Syntax:</b> stm_r_mx_used_by_ac (IN ac_list: LIST OF ACTIVITY,OUT status: INTEGER)</p>

### Input List Type: actor

<p><b>stm_r_actor_explicit_defined_actor</b></p>	<p><b>Query:</b></p> <p><b>Purpose:</b> Extracts a list of elements from the input list that are explicitly defined elements of the requested type</p> <p><b>Syntax:</b> stm_r_actor_explicit_defined_actor (stm_list actor_list, int *status);</p>
--	---



## Input List Type: an

<b>stm_r_mx_in_definition_of_an</b>	<p><b>Query:</b> Elements appearing in the definition of a given action.</p> <p><b>Purpose:</b> Returns the elements that appear in the <b>Definition</b> field of actions (in the action's form) in the input list.</p> <p>This query identifies the elements that are used directly by the actions in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_in_definition_of_an (IN an_list: LIST OF ACTION, OUT status: INTEGER)</pre>
<b>stm_r_mx_influence_value_of_an</b>	<p><b>Query:</b> Elements that influence the value of a given action.</p> <p><b>Purpose:</b> Returns the elements that appear in the <b>Definition</b> field of actions in the input list, and those that appear in the definitions of these elements (for all levels).</p> <p>This query identifies the elements that are used directly or indirectly by the actions in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influence_value_of_an (IN an_list: LIST OF ACTION, OUT status: INTEGER)</pre>
<b>stm_r_mx_influenced_by_an</b>	<p><b>Query:</b> Elements that refer to, or are influenced by, a given action.</p> <p><b>Purpose:</b> Returns the elements that directly or indirectly use the actions in the input list.</p> <p>This query identifies all the elements, in all levels, that use the input actions.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influenced_by_an (IN an_list: LIST OF ACTION, OUT status: INTEGER)</pre>
<b>stm_r_mx_refer_to_an</b>	<p><b>Query:</b> Elements that refer to a given action.</p> <p><b>Purpose:</b> Returns the elements that directly use the actions in the input list.</p> <p>This query identifies where the input actions appear.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_refer_to_an (IN an_list: LIST OF ACTION, OUT status: INTEGER)</pre>

### Input List Type: bb

<b>stm_r_bb_explicit_defined_bb</b>	<p><b>Query:</b></p> <p><b>Purpose:</b> Extracts a list of elements from the input list that are explicitly defined elements of the requested type</p> <p><b>Syntax:</b></p> <pre>stm_r_bb_explicit_defined_bb (stm_list bb_list, int *status);</pre>
-------------------------------------	---

### Input List Type: bt

<b>stm_r_mx_source_of_bt</b>	<p><b>Query:</b> Elements affected by a given activity.</p> <p><b>Purpose:</b> Returns the elements that are sources of basic transitions in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_source_of_bt(stm_list in_list, int *status);</pre>
<b>stm_r_mx_target_of_bt</b>	<p><b>Query:</b> Transitions containing the given basic transitions</p> <p><b>Purpose:</b> Returns the compound transitions that contain the basic transitions in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_tr_containing_bt (IN bt_list: LIST OF TRANSITION, OUT status: INTEGER)</pre>
<b>stm_r_tr_containing_bt</b>	<p><b>Query:</b> Transitions containing the given basic transitions</p> <p><b>Purpose:</b> Returns the compound transitions that contain the basic transitions in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_tr_containing_bt (IN bt_list: LIST OF TRANSITION, OUT status: INTEGER)</pre>
<b>stm_r_bt_exit_from_st</b>	<p><b>Query:</b> Basic transitions whose source is the specified state</p> <p><b>Purpose:</b> Returns the basic transitions whose source is a state appearing in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_bt_exit_from_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</pre>

<b>stm_r_bt_exit_from_cn</b>	<p><b>Query:</b> Basic transitions whose source is the specified connector.</p> <p><b>Purpose:</b> Returns the basic transitions whose source is a connector appearing in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_bt_exit_from_cn (IN cn_list: LIST OF CONNECTOR, OUT status: INTEGER)</pre>
<b>stm_r_bt_enter_st</b>	<p><b>Query:</b> Basic transitions whose target is the specified state</p> <p><b>Purpose:</b> Returns the basic transitions whose target is a state appearing in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_bt_enter_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</pre>
<b>stm_r_bt_enter_cn</b>	<p><b>Query:</b> Basic transitions whose target is the specified connector</p> <p><b>Purpose:</b> Returns the basic transitions whose target is a connector appearing in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_bt_enter_cn (IN st_list: LIST OF CONNECTOR, OUT status: INTEGER)</pre>
<b>stm_r_bt_defined_in_ch</b>	<p><b>Query:</b> Basic transitions</p> <p><b>Purpose:</b> Returns the basic transitions defined in the input list of charts</p> <p><b>Syntax:</b></p> <pre>stm_r_bt_defined_in_ch(IN ch_list: LIST OF CHART, OUT status: INTEGER);</pre>
<b>stm_r_af_containing_ba</b>	<p><b>Query:</b> A-flow-lines</p> <p><b>Purpose:</b> Returns the a-flow-lines that contain the basic a-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_af_containing_ba(IN ba_list: LIST OF A_FLOW_LINE, OUT status: INTEGER);</pre>

## Input List Type: bm

<b>stm_r_mx_source_of_bm</b>	<p><b>Query:</b> Elements affected by a given activity.</p> <p><b>Purpose:</b> Returns the elements that are sources of basic m-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_source_of_bm (stm_list in_list, int *status);</pre>
<b>stm_r_mx_target_of_bm</b>	<p><b>Query:</b> Elements affected by a given activity.</p> <p><b>Purpose:</b> Returns the elements that are targets of basic m-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_target_of_bm (stm_list in_list, int *status);</pre>
<b>stm_r_bm_exit_from_md</b>	<p><b>Query:</b> Basic m-flow-lines whose source is a given module</p> <p><b>Purpose:</b> Returns the basic m-flow-lines whose source is a module from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_bm_exit_from_md (IN md_list: LIST OF MODULE, OUT status: INTEGER);</pre>
<b>stm_r_bm_exit_from_cn</b>	<p><b>Query:</b> Basic m-flow-lines whose source is a given connector</p> <p><b>Purpose:</b> Returns the basic m-flow-lines whose source is a connector from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_bm_exit_from_cn (IN cn_list: LIST OF CONNECTOR, OUT status: INTEGER);</pre>
<b>stm_r_bm_enter_md</b>	<p><b>Query:</b> Basic m-flow-lines whose target is a given module</p> <p><b>Purpose:</b> Returns the basic m-flow-lines whose target is a module from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_bm_enter_md (IN md_list: LIST OF MODULE, OUT status: INTEGER);</pre>
<b>stm_r_bm_enter_cn</b>	<p><b>Query:</b> Basic m-flow-lines whose target is a given connector</p> <p><b>Purpose:</b> Returns the basic m-flow-lines whose target is a connector from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_bm_enter_cn(IN md_list: LIST OF CONNECTOR, OUT status: INTEGER);</pre>

<b>stm_r_bm_exit_from_om</b>	<p><b>Query:</b> Basic m-flow-lines whose source is a given module-occurrence</p> <p><b>Purpose:</b> Returns the basic m-flow-lines whose source is a module-occurrence from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_bm_exit_from_om (IN md_list: LIST OF ELEMENT, OUT status: INTEGER);</pre>
------------------------------	--

### Input List Type: ch

<b>stm_r_mx_constant_parameter_ch</b>	<p><b>Query:</b> Constant parameters in a given chart</p> <p><b>Purpose:</b> Returns the constant formal parameters of generic charts and components in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_constant_parameter_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</pre>
<b>stm_r_mx_def_or_unres_in_ch</b>	<p><b>Query:</b> Elements that are defined or unresolved in a given chart</p> <p><b>Purpose:</b> Returns the elements that are explicitly defined or unresolved in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_def_or_unres_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</pre>
<b>stm_r_mx_defined_in_ch</b>	<p><b>Query:</b> Elements that are defined in a given chart</p> <p><b>Purpose:</b> Returns the elements that are explicitly defined in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_defined_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</pre>
<b>stm_r_mx_in_parameter_ch</b>	<p><b>Query:</b> In parameters in a given chart</p> <p><b>Purpose:</b> Returns the formal in parameters of generic charts and components in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_in_parameter_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</pre>
<b>stm_r_mx_influence_value_of_ch</b>	<p><b>Query:</b> Elements referenced or influenced by a given chart</p> <p><b>Purpose:</b> Returns the elements that are used directly or indirectly (referenced or affected) by the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influence_value_of_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</pre>

<p><b>stm_r_mx_inout_parameter_ch</b></p>	<p><b>Query:</b> Inout parameters in a given chart</p> <p><b>Purpose:</b> Returns the formal inout parameters of generic charts and components in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_inout_parameter_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</pre>
<p><b>stm_r_mx_instance_of_ch</b></p>	<p><b>Query:</b> Element instance of a given chart</p> <p><b>Purpose:</b> Returns the element instances defined by the charts in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_instance_of_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</pre>
<p><b>stm_r_mx_out_parameter_ch</b></p>	<p><b>Query:</b> Out parameters in a given chart</p> <p><b>Purpose:</b> Returns the formal out parameters of generic charts and components in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_out_parameter_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</pre>
<p><b>stm_r_mx_parameter_of_ch</b></p>	<p><b>Query:</b> Parameters in a given chart</p> <p><b>Purpose:</b> Returns the formal parameters of generic charts and components in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_parameter_of_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</pre>
<p><b>stm_r_mx_referenced_by_ch</b></p>	<p><b>Query:</b> Elements referenced by a given chart</p> <p><b>Purpose:</b> Returns the elements that appear in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_referenced_by_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</pre>
<p><b>stm_r_mx_root_in_ch</b></p>	<p><b>Query:</b> Root elements of a given chart</p> <p><b>Purpose:</b> Returns the internally defined elements (of type diagram) attached to the charts in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_root_in_ch (IN el_list: LIST OF CHART, OUT status: INTEGER)</pre>
<p><b>stm_r_mx_text_def_unres_in_ch</b></p>	<p><b>Query:</b> Textual elements defined or unresolved in a given chart</p> <p><b>Purpose:</b> Returns the textual elements that are explicitly defined or unresolved in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_text_def_unres_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</pre>

<b>stm_r_mx_text_unresolved_in_ch</b>	<p><b>Query:</b> Textual elements unresolved in a given chart</p> <p><b>Purpose:</b> Returns the textual elements that are unresolved in the charts of the input list</p> <p><b>Syntax:</b> stm_r_mx_text_unresolved_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</p>
<b>stm_r_mx_textual_defined_in_ch</b>	<p><b>Query:</b> Textual elements that are defined in a given chart</p> <p><b>Purpose:</b> Returns the textual elements that are explicitly defined in the charts of the input list</p> <p><b>Syntax:</b> stm_r_mx_textual_defined_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</p>
<b>stm_r_mx_unresolved_in_ch</b>	<p><b>Query:</b> Elements unresolved in a given chart</p> <p><b>Purpose:</b> Returns elements that are unresolved in the charts of the input list</p> <p><b>Syntax:</b> stm_r_mx_unresolved_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</p>

### Input List Type: co

<b>stm_r_mx_in_definition_of_co</b>	<p><b>Query:</b> Elements appearing in the definition of a given condition.</p> <p><b>Purpose:</b> Returns the elements that appear in the <b>Definition</b> field of conditions (in the condition's form) in the input list. This query identifies the elements that are used directly by the conditions in the input list.</p> <p><b>Syntax:</b> stm_r_mx_in_definition_of_co (IN co_list: LIST OF CONDITION, OUT status: INTEGER)</p>
<b>stm_r_mx_influence_value_of_co</b>	<p><b>Query:</b> Elements that influence the value of a given condition.</p> <p><b>Purpose:</b> Returns the elements that appear in the <b>Definition</b> field of the conditions in the input list, and those that appear in the definitions of these elements (for all levels). This query identifies the elements that directly or indirectly influence the conditions in the input list.</p> <p><b>Syntax:</b> stm_r_mx_influence_value_of_co (IN co_list: LIST OF CONDITION, OUT status: INTEGER)</p>

<p><b>stm_r_mx_influenced_by_co</b></p>	<p><b>Query:</b> Elements that refer to, or are influenced by, a given condition.</p> <p><b>Purpose:</b> Returns the elements that directly or indirectly use the conditions in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influenced_by_co (IN co_list: LIST OF CONDITION, OUT status: INTEGER)</pre>
<p><b>stm_r_mx_refer_to_co</b></p>	<p><b>Query:</b> Elements that refer to a given condition.</p> <p><b>Purpose:</b> Returns the elements that directly use the conditions in the input list. This query identifies where the input conditions appear.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_refer_to_co (IN co_list: LIST OF CONDITION, OUT status: INTEGER)</pre>

### Input List Type: di

<p><b>stm_r_mx_in_definition_of_di</b></p>	<p><b>Query:</b> Elements appearing in the definition of a given data-item.</p> <p><b>Purpose:</b> Returns the elements that appear in the <b>Definition</b> and the <b>Consists of</b> fields of data-items (in the data-item's form) in the input list. This query identifies the elements that are directly used by the data-items in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_in_definition_of_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<p><b>stm_r_mx_influence_value_of_di</b></p>	<p><b>Query:</b> Elements that influence the value of a given data-item.</p> <p><b>Purpose:</b> Returns the elements that appear in the <b>Definition</b> and <b>Consists of</b> fields of data-items in the input list, and those that appear in the fields of these elements (for all levels). This query identifies the elements that directly or indirectly influence the data-items in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influence_value_of_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
<p><b>stm_r_mx_influenced_by_di</b></p>	<p><b>Query:</b> Elements that refer to, or are influenced by, a given data-item.</p> <p><b>Purpose:</b> Returns the elements that directly or indirectly use the data-items in the input list. This query identifies all the elements, in all levels, that refer to or affect the input data-items.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influenced_by_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>



<b>stm_r_mx_refer_to_di</b>	<p><b>Query:</b> Elements that refer to the specified data-item.</p> <p><b>Purpose:</b> Returns the elements that directly use the data-items in the input list. This query identifies where the input data-items appear.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_refer_to_di (IN di_list: LIST OF DATA_ITEM, OUT status: INTEGER)</pre>
-----------------------------	--

### Input List Type: ds

<b>stm_r_mx_refer_to_ds</b>	<p><b>Query:</b> Elements that refer to a given data-store</p> <p><b>Purpose:</b> Returns the elements directly affected by data-stores in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_refer_to_ds (IN ds_list: LIST OF DATA_STORE, OUT status: INTEGER)</pre>
-----------------------------	---

### Input List Type: dt

<b>stm_r_mx_in_definition_of_dt</b>	<p><b>Query:</b> Elements that appear in the definition of a given user-defined type</p> <p><b>Purpose:</b> Returns the elements that appear in the definition form of the user-defined types in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_in_definition_of_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>
<b>stm_r_mx_influence_value_of_dt</b>	<p><b>Query:</b> Elements that influence the definition of a given user-defined type</p> <p><b>Purpose:</b> Returns the elements, data-items and user-defined types, that appear in the definition form of the user-defined types in the input list, and those that appear in the definition form of these elements—in all levels</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influence_value_of_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</pre>

<b>stm_r_mx_influenced_by_dt</b>	<p><b>Query:</b> Elements that refer to or influenced by a given user-defined type</p> <p><b>Purpose:</b> Returns the elements that directly or indirectly use in their definition the user-defined types in the input list.</p> <p><b>Syntax:</b>  <code>stm_r_mx_influenced_by_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</code></p>
<b>stm_r_mx_refer_to_dt</b>	<p><b>Query:</b> Elements that refer to a given user-defined type</p> <p><b>Purpose:</b> Returns the elements that use in their definition form the user-defined types in the input list</p> <p><b>Syntax:</b>  <code>stm_r_mx_refer_to_dt (IN el_list: LIST OF DATA_TYPE, OUT status: INTEGER)</code></p>

### Input List Type: ev

<b>stm_r_mx_in_definition_of_ev</b>	<p><b>Query:</b> Elements appearing in the definition of a given event.</p> <p><b>Purpose:</b> Returns the elements that appear in the <b>Definition</b> field of events (in the event's form) in the input list.  This query identifies the elements that are directly used by the events in the input list.</p> <p><b>Syntax:</b>  <code>stm_r_mx_in_definition_of_ev (IN ev_list: LIST OF EVENT, OUT status: INTEGER)</code></p>
<b>stm_r_mx_influence_value_of_ev</b>	<p><b>Query:</b> Elements that influence the value of a given event.</p> <p><b>Purpose:</b> Returns the elements that appear in the <b>Definition</b> field of events in the input list, and those that appear in the definitions of these elements (for all levels).  This query identifies the elements that are used directly or indirectly by the events in the input list.</p> <p><b>Syntax:</b>  <code>stm_r_mx_influence_value_of_ev (IN ev_list: LIST OF EVENT, OUT status: INTEGER)</code></p>
<b>stm_r_mx_influenced_by_ev</b>	<p><b>Query:</b> Elements that refer to, or are influenced by, a given event.</p> <p><b>Purpose:</b> Returns the elements that directly or indirectly use the events in the input list.  This query identifies all the elements, in all levels, that refer to or affect the input events.</p> <p><b>Syntax:</b>  <code>stm_r_mx_influenced_by_ev (IN ev_list: LIST OF EVENT, OUT status: INTEGER)</code></p>

<b>stm_r_mx_refer_to_ev</b>	<p><b>Query:</b> Elements that refer to a given event.</p> <p><b>Purpose:</b> Returns the elements that directly use the events in the input list. This query identifies where the input events appear.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_refer_to_ev (IN ev_list: LIST OF EVENT, OUT status: INTEGER)</pre>
-----------------------------	--

### Input List Type: fd

<b>stm_r_mx_containing_fd</b>	<p><b>Query:</b> Elements containing a given field.</p> <p><b>Purpose:</b> Returns the data-items and user-defined types in which the fields in the input list are defined.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_containing_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</pre>
<b>stm_r_mx_in_definition_of_fd</b>	<p><b>Query:</b> Elements that appear in the definition of a given field.</p> <p><b>Purpose:</b> Returns the elements that appear in the type definition of the fields in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_in_definition_of_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</pre>
<b>stm_r_mx_influence_value_of_f</b>	<p><b>Query:</b> Elements that influence the definition of a given field.</p> <p><b>Purpose:</b> Returns the elements, data-items, and user-defined types that appear in the type definition of the fields in the input list, and those that appear in the definition form of these elements (in all levels).</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influence_value_of_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</pre>
<b>stm_r_mx_influenced_by_fd</b>	<p><b>Query:</b> Elements that refer to, or are influenced by, a given field.</p> <p><b>Purpose:</b> Returns the elements that directly refer to the fields in the input list. This query identifies where the fields in the input list are used.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influenced_by_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</pre>
<b>stm_r_mx_refer_to_fd</b>	<p><b>Query:</b> Elements that refer to a given field.</p> <p><b>Purpose:</b> Returns the elements that directly refer to the fields in the input list. This query identifies where the fields in the input list are used.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_refer_to_fd (IN el_list: LIST OF FIELD, OUT status: INTEGER)</pre>

### Input List Type: fn

<p><b>stm_r_mx_influenced_by_fn</b></p>	<p><b>Query:</b> Elements that refer to, or are influenced by, a given function</p> <p><b>Purpose:</b> Returns the elements that indirectly or directly use the functions in the input list.</p> <p>This query identifies all the elements, in all levels, that refer to the input functions.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influenced_by_fn (IN fn_list: LIST OF FUNCTION, OUT status: INTEGER)</pre>
<p><b>stm_r_mx_refer_to_fn</b></p>	<p><b>Query:</b> Elements that refer to a given function.</p> <p><b>Purpose:</b> Returns the elements that directly use the functions in the input list.</p> <p>This query identifies where the input functions appear.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_refer_to_fn (IN fn_list: LIST OF FUNCTION, OUT status: INTEGER)</pre>

### Input List Type: if

<p><b>stm_r_mx_in_definition_of_if</b></p>	<p><b>Query:</b> Elements appearing in the definition of a given a information-flow.</p> <p><b>Purpose:</b> Returns the elements listed in the <b>Consists of</b> field (in the information-flow's forms) for information-flows in the input list.</p> <p>This query identifies the elements that are used directly by the information-flows in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_in_definition_of_if (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</pre>
<p><b>stm_r_mx_influence_value_of_if</b></p>	<p><b>Query:</b> Elements that influence the value of a given information-flow.</p> <p><b>Purpose:</b> Returns the elements contained in the information-flows in the input list (as listed in the <b>Consists of</b> field), for all levels of decomposition.</p> <p>This query identifies the elements that are directly or indirectly contained in the information-flows of the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influence_value_of_if (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</pre>

<b>stm_r_mx_influenced_by_if</b>	<p><b>Query:</b> Elements that refer to, or are influenced by, a given information-flow</p> <p><b>Purpose:</b> Returns the elements that directly or indirectly use the information-flows in the input list.</p> <p>This query identifies all the elements, in all levels, that refer to the input information-flows.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influenced_by_if (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</pre>
<b>stm_r_mx_refer_to_if</b>	<p><b>Query:</b> Elements that refer to a given information-flow.</p> <p><b>Purpose:</b> Returns the elements that directly use the information-flows in the input list.</p> <p>This query identifies where the input information-flows appear.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_refer_to_if (IN if_list: LIST OF INFORMATION_FLOW, OUT status: INTEGER)</pre>

### Input List Type: md

<b>stm_r_mx_influence_md</b>	<p><b>Query:</b> Elements that are referenced by or influence a given module.</p> <p><b>Purpose:</b> Returns the elements that are used in all levels by the modules in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influence_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>
<b>stm_r_mx_influenced_by_md</b>	<p><b>Query:</b> Elements that refer to, or are influenced by, a given module</p> <p><b>Purpose:</b> Returns the elements that directly or indirectly use the modules in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influenced_by_md (IN el_list: LIST OF MODULE, OUT status: INTEGER)</pre>
<b>stm_r_mx_refer_to_md</b>	<p><b>Query:</b> Elements that refer to a given module.</p> <p><b>Purpose:</b> Returns the elements that directly refer to modules in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_refer_to_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>

<b>stm_r_mx_referenced_by_md</b>	<p><b>Query:</b> Elements that are referenced by a given module.</p> <p><b>Purpose:</b> Returns the elements that appear in the modules of the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_referenced_by_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>
<b>stm_r_mx_resolved_to_ext_md</b>	<p><b>Query:</b> Elements resolved to a given external module.</p> <p><b>Purpose:</b> Returns the elements to which the external modules in the input list are resolved.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_resolved_to_ext_md (IN md_list: LIST OF MODULE,OUT status: INTEGER)</pre>

### Input List Type: mf

<b>stm_r_mx_flowng_through_mf</b>	<p><b>Query:</b> Elements flowing through a given m-flow-line</p> <p><b>Purpose:</b> Returns the information elements (conditions, events, data-items and basic information-flows) that actually flow through the m-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_flowng_through_mf (IN mf_list: LIST OF M_FLOW_LINE, OUT status: INTEGER)</pre>
<b>stm_r_mx_labeling_mf</b>	<p><b>Query:</b> Elements labeling a given m-flow-line</p> <p><b>Purpose:</b> Returns the elements that label m-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_labeling_mf (IN mf_list: LIST OF M_FLOW_LINE, OUT status: INTEGER)</pre>

### Input List Type: msg

<b>stm_r_mx_labeling_msg</b>	<p><b>Query:</b> Elements labeling a given message</p> <p><b>Purpose:</b> Returns those elements that appear in labels of messages in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_labeling_msg (IN mess_list: LIST OF MESSAGE, OUT status: INTEGER)</pre>
------------------------------	--

## Input List Type: mx

<b>stm_r_mx_affected_by_mx</b>	<p><b>Query:</b> Elements affected by a given element.</p> <p><b>Purpose:</b> Returns the elements (primitive data-items, conditions, events, and activities) that are affected (modified, generated, started, stopped, and so on) by elements (states in static reactions or transitions in labels) in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_affected_by_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_mx_affecting_mx</b>	<p><b>Query:</b> Elements in which a given element is affected.</p> <p><b>Purpose:</b> Returns the elements (states and transitions) that affect (modify, generate, or activate) the elements (for example, events, data-items, or activities) in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_affecting_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_mx_meaningly_affecting_mx</b>	<p><b>Query:</b> Activities in which a given element is affected.</p> <p><b>Purpose:</b> Identical to <code>stm_r_mx_affecting_mx</code>, but when the input list includes an ID of a record/union, <code>stm_r_mx_meaningly_affecting_mx</code> will also return elements that affect a field of the record/union, and not necessarily the whole record/union element.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_meaningly_affecting_mx (stm_list in_list, int *status);</pre>
<b>stm_r_mx_by_attributes_mx</b>	<p><b>Query:</b> Elements by attributes</p> <p><b>Purpose:</b> Returns the elements in the input list that match a particular attribute name and value</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_by_attributes_mx (IN mx_list: LIST OF ELEMENT, IN attr_name: STRING, IN attr_value: STRING, OUT status: INTEGER)</pre>
<b>stm_r_mx_callback_binding_mx</b>	<p><b>Query:</b> Elements with callback bindings.</p> <p><b>Purpose:</b> Returns the elements in the input list that have callback bindings.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_callback_binding_mx (IN el_list: LIST OF ELEMENT, OUT status: INTEGER)</pre>

<b>stm_r_mx_comb_elements_mx</b>	<p><b>Query:</b> None.</p> <p><b>Purpose:</b> Returns the elements (data-items and conditions) in the input list that are combinational elements.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_comb_elements_mx (IN mx_list: LIST OF ELEMENT, OUT status: INTEGER)</pre>
<b>stm_r_mx_def_of_instance_mx</b>	<p><b>Query:</b> Definition elements of a given element.</p> <p><b>Purpose:</b> Returns the definition elements (top-level) for instances in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_def_of_instance_mx (IN el_list: LIST OF ELEMENT, OUT status: INTEGER)</pre>
<b>stm_r_mx_explicit_defined_mx</b>	<p><b>Query:</b> Elements explicitly defined.</p> <p><b>Purpose:</b> Returns the elements of the input list that were explicitly defined.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_explicit_defined_mx (IN _list: LIST OF ELEMENT, OUT status: INTEGER)</pre>
<b>stm_r_mx_generic_instance_mx</b>	<p><b>Query:</b> None.</p> <p><b>Purpose:</b> Returns the boxes (states, activities, and modules) in the input list that are instances of generic charts.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_generic_instance_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_mx_in_definition_of_mx</b>	<p><b>Query:</b> Elements that appear in the definition of a given element.</p> <p><b>Purpose:</b> Returns the elements that appear in the various fields of the element's form, or in labels of elements in the input list. This query identifies the elements that are used directly by the elements in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_in_definition_of_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_mx_influence_value_of_mx</b>	<p><b>Query:</b> Elements that influence the value of a given element.</p> <p><b>Purpose:</b> Returns the elements that appear in various form's fields or labels of elements in the input list, and those that appear in the fields of these elements (for all levels). This query identifies the elements that directly or indirectly influence the elements in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influence_value_of_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>



<b>stm_r_mx_influenced_by_mx</b>	<p><b>Query:</b> Elements that refer to or influenced by a given element.</p> <p><b>Purpose:</b> Returns the elements that directly or indirectly use the elements in the input list.</p> <p>This query identifies all the elements, in all levels, that refer to or affect the input elements.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influenced_by_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_mx_instance_mx</b>	<p><b>Query:</b> Element instance of a given element</p> <p><b>Purpose:</b> Returns the instance elements defined by the elements in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_instance_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_mx_instance_of_def_mx</b>	<p><b>Query:</b> Instance elements</p> <p><b>Purpose:</b> Returns the instance elements for definition elements (top-level) in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_instance_of_def_mx (IN el_list: LIST OF ELEMENT, OUT status: INTEGER)</pre>
<b>stm_r_mx_logical_desc_of_mx</b>	<p><b>Query:</b> Logical descendants of a given element</p> <p><b>Purpose:</b> Returns the logical descendants of the elements in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_logical_desc_of_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_mx_logical_parent_of_mx</b>	<p><b>Query:</b> Logical parent elements of a given element.</p> <p><b>Purpose:</b> Returns the logical parent elements of the elements in the input list, taking into account the translation of instances to their definition charts.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_logical_parent_of_mx (IN el_list: LIST OF ELEMENT, OUT status: INTEGER)</pre>
<b>stm_r_mx_logical_sub_of_mx</b>	<p><b>Query:</b> Logical subelements of a given element</p> <p><b>Purpose:</b> Returns the logical subelements of the elements in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_logical_sub_of_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>

<b>stm_r_mx_name_of_mx</b>	<p><b>Query:</b> Element whose names match a given pattern</p> <p><b>Purpose:</b> Returns all the elements whose names match the specified pattern</p> <p><b>Syntax:</b> stm_r_mx_name_of_mx (IN pattern: STRING, OUT status: INTEGER)</p>
<b>stm_r_mx_offpage_instance_mx</b>	<p><b>Query:</b> None.</p> <p><b>Purpose:</b> Returns the boxes (states, activities, and modules) in the input list that are instances of offpage charts.</p> <p><b>Syntax:</b> stm_r_mx_offpage_instance_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</p>
<b>stm_r_mx_parameter_mx</b>	<p><b>Query:</b> Elements that are parameters.</p> <p><b>Purpose:</b> Returns the elements in the input list that are declared as formal parameters of a generic chart.</p> <p><b>Syntax:</b> stm_r_mx_parameter_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</p>
<b>stm_r_mx_physical_desc_of_mx</b>	<p><b>Query:</b> Physical descendants of a given element</p> <p><b>Purpose:</b> Returns the physical descendants (those within the same chart) for the elements in the input list</p> <p><b>Syntax:</b> stm_r_mx_physical_desc_of_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</p>
<b>stm_r_mx_physical_parent_of_mx</b>	<p><b>Query:</b> Physical parent elements of a given element</p> <p><b>Purpose:</b> Returns the physical parent elements (those within the same chart) for the elements in the input list</p> <p><b>Syntax:</b> stm_r_mx_physical_parent_of_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</p>
<b>stm_r_mx_physical_sub_of_mx</b>	<p><b>Query:</b> Physical subelements of a given element</p> <p><b>Purpose:</b> Returns the physical subelements (those within the same chart) for the elements in the input list</p> <p><b>Syntax:</b> stm_r_mx_physical_sub_of_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</p>
<b>stm_r_mx_refer_to_mx</b>	<p><b>Query:</b> Elements that refer to a given element.</p> <p><b>Purpose:</b> Returns the elements that directly refer to elements in the input list. This query identifies where the input elements are used.</p> <p><b>Syntax:</b> stm_r_mx_refer_to_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</p>

<b>stm_r_mx_resolved_to_ext_mx</b>	<p><b>Query:</b> Elements resolved to a given external box.</p> <p><b>Purpose:</b> Returns the activities and modules (internal, external, or environment) to which the external activities and modules in the input list are resolved.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_resolved_to_ext_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_mx_synonym_of_mx</b>	<p><b>Query:</b> Elements whose synonyms match a given pattern</p> <p><b>Purpose:</b> Returns all the elements whose synonyms match the specified pattern</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_synonym_of_mx (IN pattern: STRING, OUT status: INTEGER)</pre>
<b>stm_r_mx_unresolved_mx</b>	<p><b>Query:</b> Unresolved elements.</p> <p><b>Purpose:</b> Returns the unresolved elements in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_unresolved_mx (IN mx_list: LIST OF ELEMENT, OUT status: INTEGER)</pre>
<b>stm_r_mx_used_by_mx</b>	<p><b>Query:</b> Elements used by a given element.</p> <p><b>Purpose:</b> Returns the elements (primitive events, conditions, data-items, states, and activities) that are used (evaluated by the elements, such as states in static reactions and transitions in labels) in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_used_by_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_mx_using_mx</b>	<p><b>Query:</b> Elements in which a given element is used.</p> <p><b>Purpose:</b> Returns the elements (states in static reactions and transitions in labels) that use (evaluate) the elements (basic events, conditions, data-items, states, and activities) in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_using_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_mx_meaningfully_using_mx</b>	<p><b>Query:</b> Activities in which a given element is used.</p> <p><b>Purpose:</b> Identical to <code>stm_r_mx_using_mx</code>, but when the input list includes an ID of a record/union, <code>stm_r_mx_meaningfully_using_mx</code> will also return elements that use a field of the record/union, and not necessarily the whole record/union element.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_meaningfully_using_mx (stm_list in_list, int *status);</pre>

<b>stm_r_mx_with_combinationals_mx</b>	<p><b>Query:</b> None.</p> <p><b>Purpose:</b> Returns the elements (activities and state charts) in the input list that have combinational assignments.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_with_combinationals_mx (IN mx_list: LIST OF ELEMENT, OUT status: INTEGER)</pre>
--	---

### Input List Type: tr

<b>stm_r_tr_by_attributes_tr</b>	<p><b>Query:</b> Transitions by attributes</p> <p><b>Purpose:</b> Returns the transitions from the input list which have attribute attr_name with value attr_value in their attribute list</p> <p><b>Syntax:</b></p> <pre>stm_r_tr_by_attributes_tr(IN st_list: LIST OF TRANSITION, IN attr_name : STRING, IN attr_value : STRING, OUT status: INTEGER);</pre>
----------------------------------	--

### Input List Type: uc

<b>stm_r_uc_explicit_defined_uc</b>	<p><b>Query:</b> Use-case elements</p> <p><b>Purpose:</b> Extracts the list of use-case elements from the input list that are explicitly defined</p> <p><b>Syntax:</b></p> <pre>stm_r_uc_explicit_defined_uc(IN uc_list: LIST OF USE_CASE, OUT status: INTEGER);</pre>
<b>stm_r_uc_associates_ac</b>	<p><b>Query:</b> Use-case elements</p> <p><b>Purpose:</b> Returns the use cases that associate with activities in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_tr_by_attributes_tr(IN st_list: LIST OF TRANSITION, IN attr_name : STRING, IN attr_value : STRING, OUT status: INTEGER)</pre>

## Function Relationships

The following functions are related, but have subtle differences:

- ◆ `stm_r_mx_influenced_by_mx`
- ◆ `stm_r_mx_affected_by_mx`
- ◆ `stm_r_mx_used_by_mx`
- ◆ `stm_r_mx_affecting_mx`

The following matrix shows their relationships. In the matrix, opposite functions go from left to right, whereas cause and effect functions go up and down.

influenced by	used by
<i>Function</i>	
affected by	affecting

For example:

- ◆ If  $x$  is influenced by  $y$ , then  $y$  is used by  $x$ .
- ◆ If  $n$  is affected by  $m$ , then  $m$  is affecting  $n$ .

Consider the following statement:

```
if x is true then Function will set y=5
```

In this statement,  $x$  influences `Function` and `Function` affects  $y$ . This is shown in the matrix as follows:

- ◆ Elements above the double line influence `Function`.
- ◆ Elements below the double line are affected by `Function`.

For example,  $x$  is used by `Function` to determine whether to set the value of  $y$ , and `Function` is affecting  $y$  by setting its value.

There are four possible relationships between these functions: two opposites and two cause and effects.

- ◆ Opposite: influenced by and used by
- ◆ Opposite: affected by and affecting
- ◆ Cause and effect: influenced by and affected by
- ◆ Cause and effect: used by and affecting

To illustrate the relationships, consider the following static reaction in a state called `STATE`:

```
[D]/X=5 if D is true, then set x=y
```

The following table shows the relationships.

<b>Relation Type</b>	<b>Description</b>
Opposite: influenced and used by	STATE reads D to determine whether to perform an action, and D gives STATE the cue to set X=Y. In other words, STATE is influenced by D, and D is used by STATE.
Opposite: affected by and affecting	X's value is set by STATE and STATE sets the value of X. In other words, X is affected by STATE, and STATE is affecting X.
Cause and effect: influenced by and affected by	STATE reads Y to determine which value should be assigned to X, and while in STATE, X can be set to Y. In other words, when STATE is influenced by Y, it results in X being affected by STATE.
Cause and effect: used by and affecting	If Y is true, STATE sets the value of X. Y is influencing STATE; STATE is affecting X. In other words, when Y is used by STATE, it results in STATE affecting X.

---

**Input List Type: router**

<b>stm_r_mx_flowng_from_router</b>	<p><b>Query:</b> Elements flowing from a given router</p> <p><b>Purpose:</b> Returns the elements actually flowing from routers in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_flowng_from_router (IN router_lst: LIST OF ROUTER, OUT status: INTEGER)</pre>
<b>stm_r_mx_flowng_to_router</b>	<p><b>Query:</b> Elements flowing to a given router</p> <p><b>Purpose:</b> Returns the elements actually flowing to routers in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_flowng_to_router (IN router_lst: LIST OF ROUTER, OUT status: INTEGER)</pre>
<b>stm_r_mx_refer_to_router</b>	<p><b>Query:</b> Elements that refer to a given router.</p> <p><b>Purpose:</b> Returns the elements that directly refer to routers in the input list. This query identifies where the routers are used.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_refer_to_router (IN router_list: LIST OF ROUTER, OUT status: INTEGER)</pre>
<b>stm_r_mx_resolved_to_ext_router</b>	<p><b>Query:</b> Elements resolved to a given router.</p> <p><b>Purpose:</b> Returns the elements to which the external routers in the input list are resolved.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_resolved_to_ext_router (IN router_list: LIST OF ROUTER, OUT status: INTEGER)</pre>

### Input List Type: sb

<p><b>stm_r_mx_influenced_by_sb</b></p>	<p><b>Query:</b> Elements that refer to, or are influenced by, a given subroutine</p> <p><b>Purpose:</b> Returns the elements that directly or indirectly use the subroutines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influenced_by_sb (IN fn_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>
<p><b>stm_r_mx_refer_to_sb</b></p>	<p><b>Query:</b> Elements that refer to a given subroutine</p> <p><b>Purpose:</b> Returns the elements that directly refer to subroutines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_refer_to_sb (IN fn_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>

### Input List Type: st

<p><b>stm_r_mx_affected_by_st</b></p>	<p><b>Query:</b> Elements affected by a given state.</p> <p><b>Purpose:</b> Returns the elements (data-items, conditions, and events) that are affected (modified or generated) by states (in mini-specs and combinational assignments) in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_affected_by_st (IN st_list: LIST OF STATE,OUT status: INTEGER)</pre>
<p><b>stm_r_mx_influence_st</b></p>	<p><b>Query:</b> Elements that are referenced by or influence a given state.</p> <p><b>Purpose:</b> Returns the elements that are used in all levels by the states in the input list.</p> <p>This includes all logical descendant states, the transitions that enter or exit these states, and the elements that appear in the various fields of these states, the labels of the transitions, and their components.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influence_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</pre>



<b>stm_r_mx_influenced_by_st</b>	<p><b>Query:</b> Elements that refer to, or are influenced by a given state.</p> <p><b>Purpose:</b> Returns the elements that directly or indirectly use the states in the input list.</p> <p>This query identifies all the elements, in all levels, that refer to or affect the input states.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_influenced_by_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</pre>
<b>stm_r_mx_refer_to_st</b>	<p><b>Query:</b> Elements that refer to a given state.</p> <p><b>Purpose:</b> Returns the elements that directly refer to states in the input list.</p> <p>This query identifies where the input states are used.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_refer_to_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</pre>
<b>stm_r_mx_referenced_by_st</b>	<p><b>Query:</b> Elements that are referenced by a given state.</p> <p><b>Purpose:</b> Returns the elements that appear in the states of the input list.</p> <p>This includes all physical descendant states, the transitions that enter or exit these states, and the elements that appear in the various fields of these states and in the labels of the transitions.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_referenced_by_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</pre>
<b>stm_r_mx_used_by_st</b>	<p><b>Query:</b> Elements used by a given state.</p> <p><b>Purpose:</b> Returns the elements (data-items, conditions, and events) that are used (evaluated) by states (in mini-specs and combinational assignments) in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_used_by_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</pre>

### Input List Type: tr

<p><b>stm_r_mx_affected_by_tr</b></p>	<p><b>Query:</b> Elements affected by a given transition</p> <p><b>Purpose:</b> Returns the elements (data-items, conditions, and events) that are affected (modified, generated) by transitions (in mini-specs and combinational assignments) in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_affected_by_tr (IN tr_list: LIST OF TRANSITION,OUT status: INTEGER)</pre>
<p><b>stm_r_mx_labeling_tr</b></p>	<p><b>Query:</b> Elements labeling a given transition</p> <p><b>Purpose:</b> Returns those elements that appear in labels of the transitions in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_labeling_tr (IN tr_list: LIST OF TRANSITION, OUT status: INTEGER)</pre>
<p><b>stm_r_mx_source_of_tr</b></p>	<p><b>Query:</b> Elements that are sources of a given transition</p> <p><b>Purpose:</b> Returns the elements (states and connectors) that are sources of transitions in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_source_of_tr (IN tr_list: LIST OF TRANSITION,OUT status: INTEGER)</pre>
<p><b>stm_r_mx_target_of_tr</b></p>	<p><b>Query:</b> Elements that are targets of a given transition</p> <p><b>Purpose:</b> Returns elements (states and connectors) that are targets of transitions in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_target_of_tr (IN tr_list: LIST OF TRANSITION,OUT status: INTEGER)</pre>
<p><b>stm_r_mx_used_by_tr</b></p>	<p><b>Query:</b> Elements used by a given transition</p> <p><b>Purpose:</b> Returns the elements (data-items, conditions, and events) that are used (evaluated) by transitions (in mini-specs and combinational assignments) in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_mx_used_by_tr (IN tr_list: LIST OF TRANSITION,OUT status: INTEGER)</pre>

**Input List Type: uc**

<b>stm_r_uc_explicit_defined_uc</b>	<p><b>Query:</b> Elements from an input list.</p> <p><b>Purpose:</b> Returns the elements (data-items, conditions, and events) that are explicitly defined.</p> <p><b>Syntax:</b></p>
<b>stm_r_uc_associates_uc</b>	<p><b>Query:</b></p> <p><b>Purpose:</b></p> <p><b>Syntax:</b></p>

**Routers (router)**

This section documents the queries that return a list of routers.

**Input List Type: ac**

<b>stm_r_router_contained_in_ac</b>	<p><b>Query:</b> Routers contained in a given activity</p> <p><b>Purpose:</b> Returns the routers contained directly in activities from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_router_contained_in_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</pre>
<b>stm_r_router_in_ac</b>	<p><b>Query:</b> Router in a given activity</p> <p><b>Purpose:</b> Returns the routers directly and indirectly contained in the activities from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_router_in_ac (IN el_list: LIST OF ROUTER, OUT status: INTEGER)</pre>

### Input List Type: af

<p><b>stm_r_router_source_of_af</b></p>	<p><b>Query:</b> Routers that are sources of a given a-flow-line</p> <p><b>Purpose:</b> Returns the routers that are sources of a-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_router_source_of_af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)</pre>
<p><b>stm_r_router_target_of_af</b></p>	<p><b>Query:</b> Routers that are targets of a given a-flow-line</p> <p><b>Purpose:</b> Returns the routers that are sources of a-flow-lines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_router_target_of_af (IN af_list: LIST OF A_FLOW_LINE, OUT status: INTEGER)</pre>

### Input List Type: ch

<p><b>stm_r_router_def_or_unres_in_ch</b></p>	<p><b>Query:</b> Routers defined or unresolved in a given chart</p> <p><b>Purpose:</b> Returns the routers that are explicitly defined or unresolved in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_router_def_or_unres_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</pre>
<p><b>stm_r_router_defined_in_ch</b></p>	<p><b>Query:</b> Routers defined in a given chart</p> <p><b>Purpose:</b> Returns the routers that are explicitly defined in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_router_defined_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</pre>
<p><b>stm_r_router_unresolved_in_ch</b></p>	<p><b>Query:</b> Routers unresolved in a given chart</p> <p><b>Purpose:</b> Returns the routers that are unresolved in the charts of the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_router_unresolved_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</pre>

### Input List Type: md

<b>stm_r_router_resides_in_md</b>	<p><b>Query:</b> Routers residing in a given module.</p> <p><b>Purpose:</b> Returns the routers residing in modules from the input list. The module appears in the <b>Resides in Module</b> field of the router's form.</p> <p><b>Syntax:</b></p> <pre>stm_r_router_resides_in_md (IN md_list: LIST OF MODULE, OUT status: INTEGER)</pre>
-----------------------------------	---

### Input List Type: router

<b>stm_r_router_by_attr_router</b>	<p><b>Query:</b> Routers by attributes</p> <p><b>Purpose:</b> Returns the routers in the input list that match a given attribute name and value</p> <p><b>Syntax:</b></p> <pre>stm_r_router_by_attr_router (IN router_list: LIST OF ROUTER, IN attr_name: STRING, IN attr_value: STRING, OUT status: INTEGER)</pre>
<b>stm_r_router_exp_def_router</b>	<p><b>Query:</b> Routers that are explicitly defined</p> <p><b>Purpose:</b> Returns from the input list those routers that were explicitly defined</p> <p><b>Syntax:</b></p> <pre>stm_r_router_exp_def_router (IN router_list: LIST OF ROUTER, OUT status: INTEGER)</pre>
<b>stm_r_router_name_of_router</b>	<p><b>Query:</b> Routers whose names match a given pattern</p> <p><b>Purpose:</b> Returns all routers whose name matches a given pattern</p> <p><b>Syntax:</b></p> <pre>stm_r_router_name_of_router (IN pattern: STRING, OUT status: INTEGER)</pre>
<b>stm_r_router_res_to_ext_router</b>	<p><b>Query:</b> Routers resolved by a given external router</p> <p><b>Purpose:</b> Returns the routers (internal and external) to which the external routers in the input list are resolved</p> <p><b>Syntax:</b></p> <pre>stm_r_router_res_to_ext_router (IN router_list: LIST OF ROUTER, OUT status: INTEGER)</pre>

<b>stm_r_router_synonym_of_router</b>	<p><b>Query:</b> Routers whose synonyms match a given pattern</p> <p><b>Purpose:</b> Returns all routers whose synonyms match a given pattern</p> <p><b>Syntax:</b></p> <pre>stm_r_router_synonym_of_router (IN pattern: STRING, OUT status: INTEGER)</pre>
<b>stm_r_router_unresolved_router</b>	<p><b>Query:</b> Unresolved routers</p> <p><b>Purpose:</b> Returns the unresolved routers in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_router_unresolved_router (IN router_list: LIST OF ROUTER, OUT status: INTEGER)</pre>

## Subroutines (sb)

This section documents the queries that return a list of subroutines.

### Input List Type: ch

<b>stm_r_sb_connected_to_ch</b>	<p><b>Query:</b> Subroutines that are connected to a given procedural statechart</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are connected to the specified procedural statechart</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_connected_to_ch (IN el_list: LIST OF CHART, OUT status: INTEGER)</pre>
<b>stm_r_sb_defined_in_ch</b>	<p><b>Query:</b> Subroutines defined in a given chart</p> <p><b>Purpose:</b> Returns the subroutines that are explicitly defined in the charts in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_defined_in_ch (IN el_list: LIST OF CHART, OUT status: INTEGER)</pre>
<b>stm_r_sb_def_or_unres_in_ch</b>	<p><b>Query:</b> Subroutines defined or unresolved in a given chart</p> <p><b>Purpose:</b> Returns the subroutines that are explicitly defined or unresolved in the charts in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_def_or_unres_in_ch (IN el_list: LIST OF CHART, OUT status: INTEGER)</pre>
<b>stm_r_sb_unresolved_in_ch</b>	<p><b>Query:</b> Subroutines unresolved in a given chart</p> <p><b>Purpose:</b> Returns the subroutines that are unresolved in the charts in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_unresolved_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</pre>

### Input List Type: sb

<b>stm_r_sb_ada_sb</b>	<p><b>Query:</b> Subroutines written in Ada</p> <p><b>Purpose:</b> Returns subroutines in the input list that are written in Ada and stored in the database using the Implementation menu</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_ada_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>
<b>stm_r_sb_ansi_c_sb</b>	<p><b>Query:</b> Subroutines written in ANSI C</p> <p><b>Purpose:</b> Returns subroutines in the input list that are written in ANSI C and stored in the database using the Implementation menu</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_ansi_c_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>
<b>stm_r_sb_bit_sb</b>	<p><b>Query:</b> Subroutines by subtype</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are defined as bit</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_bit_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>
<b>stm_r_sb_bits_sb</b>	<p><b>Query:</b> Subroutines by subtype</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are defined as bit array</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_bits_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>
<b>stm_r_sb_by_attributes_sb</b>	<p><b>Query:</b> Subroutines by attributes</p> <p><b>Purpose:</b> Returns the subroutines in the input list that match the specified attribute name and value</p> <p><b>Syntax:</b> <code>stm_r_sb_by_attributes_sb (IN el_list: LIST OF SUBROUTINE, IN attr_name: STRING, IN attr_value: STRING, OUT status: INTEGER)</code></p>
<b>stm_r_sb_explicit_defined_sb</b>	<p><b>Query:</b> Subroutines that are explicitly defined</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are explicitly defined</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_explicit_defined_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>

<b>stm_r_sb_fn_with_side_effect_sb</b>	<p><b>Query:</b> Function subroutines with potential side-effects</p> <p><b>Purpose:</b> Returns the function subroutines in the input list that have potential side-effects</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_fn_with_side_effect_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>
<b>stm_r_sb_function_sb</b>	<p><b>Query:</b> Subroutines defined as functions</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are defined as functions</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_function_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>
<b>stm_r_sb_globals_usage_sb</b>	<p><b>Query:</b> Subroutines that have global data</p> <p><b>Purpose:</b> Returns all subroutines in the input list that have global data</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_globals_usage_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>
<b>stm_r_sb_imp_action_lang_sb</b>	<p><b>Query:</b> Subroutines whose selected implementation is Action Language</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are implemented in the Rhapsody Action Language using <b>Select Implementation</b></p> <p><b>Syntax:</b></p> <pre>stm_r_sb_imp_action_lang_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>
<b>stm_r_sb_imp_ada_code_sb</b>	<p><b>Query:</b> Subroutines whose selected implementation is Ada Code</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are implemented in Ada using <b>Select Implementation</b> in the properties</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_imp_ada_code_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>
<b>stm_r_sb_imp_ansi_c_code_sb</b>	<p><b>Query:</b> Subroutines whose selected implementation is ANSI C Code</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are implemented in ANSI C using <b>Select Implementation</b></p> <p><b>Syntax:</b></p> <pre>stm_r_sb_imp_ansi_c_code_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>



<b>stm_r_sb_imp_best_match_sb</b>	<p><b>Query:</b> Subroutines whose selected implementation is Best Match</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are implemented as the Best Match using <b>Select Implementation</b></p> <p><b>Syntax:</b> stm_r_sb_imp_best_match_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</p>
<b>stm_r_sb_imp_kr_c_code_sb</b>	<p><b>Query:</b> Subroutines whose selected implementation is K&amp;R C Code</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are implemented in K&amp;R C using <b>Select Implementation</b></p> <p><b>Syntax:</b> stm_r_sb_imp_kr_c_code_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</p>
<b>stm_r_sb_imp_none_sb</b>	<p><b>Query:</b> Subroutines whose selected implementation is None</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are not implemented (None) using <b>Select Implementation</b></p> <p><b>Syntax:</b> stm_r_sb_imp_none_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</p>
<b>stm_r_sb_imp_procedural_sch_sb</b>	<p><b>Query:</b> Subroutines whose selected implementation is Procedural Statechart</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are implemented as Procedural Statecharts using <b>Select Implementation</b></p> <p><b>Syntax:</b> stm_r_sb_imp_procedural_sch_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</p>
<b>stm_r_sb_imp_truth_table_sb</b>	<p><b>Query:</b> Subroutines implemented in a truth table</p> <p><b>Purpose:</b> Returns the subroutines in the input list that were implemented in a truth table</p> <p><b>Syntax:</b> stm_r_sb_imp_truth_table_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</p>
<b>stm_r_sb_integer_sb</b>	<p><b>Query:</b> Subroutines by subtype</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are defined as integer</p> <p><b>Syntax:</b> stm_r_sb_integer_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</p>

<b>stm_r_sb_kr_c_sb</b>	<p><b>Query:</b> Subroutines written in K&amp;R C</p> <p><b>Purpose:</b> Returns subroutines in the input list that are written in K&amp;R C and stored in the database using the Implementation menu</p> <p><b>Syntax:</b>  <code>stm_r_sb_kr_c_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</code></p>
<b>stm_r_sb_missing_sb</b>	<p><b>Query:</b> Subroutines by subtype</p> <p><b>Purpose:</b> Returns the subroutines in the input list for which no type is defined</p> <p><b>Syntax:</b>  <code>stm_r_sb_missing_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</code></p>
<b>stm_r_sb_name_of_sb</b>	<p><b>Query:</b> Subroutines whose names match a given pattern</p> <p><b>Purpose:</b> Returns all subroutines whose name matches the specified pattern</p> <p><b>Syntax:</b>  <code>stm_r_sb_name_of_sb (IN pattern: STRING, OUT status: INTEGER)</code></p>
<b>stm_r_sb_parameters_sb</b>	<p><b>Query:</b> Subroutines that have parameters</p> <p><b>Purpose:</b> Returns all subroutines in the input list that have parameters</p> <p><b>Syntax:</b>  <code>stm_r_sb_parameters_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</code></p>
<b>stm_r_sb_procedural_sch_sb</b>	<p><b>Query:</b> Subroutines designed as procedural statecharts</p> <p><b>Purpose:</b> Returns subroutines in the input list that are designed as procedural statecharts and stored in the database using the Implementation menu</p> <p><b>Syntax:</b>  <code>stm_r_sb_procedural_sch_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</code></p>
<b>stm_r_sb_procedure_sb</b>	<p><b>Query:</b> Subroutines defined as procedures</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are defined as procedures</p> <p><b>Syntax:</b>  <code>stm_r_sb_procedure_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</code></p>
<b>stm_r_sb_real_sb</b>	<p><b>Query:</b> Subroutines by subtype</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are defined as real</p> <p><b>Syntax:</b>  <code>stm_r_sb_real_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</code></p>

<b>stm_r_sb_statemate_action_sb</b>	<p><b>Query:</b> Subroutines written in the Rhapsody action language</p> <p><b>Purpose:</b> Returns subroutines in the input list that are written in the Rhapsody action language and stored in the database using the Implementation menu</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_statemate_action_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>
<b>stm_r_sb_string_sb</b>	<p><b>Query:</b> Subroutines by subtype</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are defined as string</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_string_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>
<b>stm_r_sb_synonym_of_sb</b>	<p><b>Query:</b> Subroutines whose synonyms match a given pattern</p> <p><b>Purpose:</b> Returns all subroutines whose synonyms match the specified pattern</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_synonym_of_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>
<b>stm_r_sb_task_sb</b>	<p><b>Query:</b> Subroutines defined as tasks</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are defined as tasks</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_task_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>
<b>stm_r_sb_unresolved_sb</b>	<p><b>Query:</b> Unresolved subroutines</p> <p><b>Purpose:</b> Returns the unresolved subroutines in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_unresolved_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>
<b>stm_r_sb_user_type_sb</b>	<p><b>Query:</b> Subroutines by subtype</p> <p><b>Purpose:</b> Returns the subroutines in the input list that are defined as user-defined type</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_user_type_sb (IN el_list: LIST OF SUBROUTINE, OUT status: INTEGER)</pre>
<b>stm_r_sb_procedural_fch_sb</b>	<p><b>Query:</b> Subroutines designed as Flowchart</p> <p><b>Purpose:</b> Returns subroutines in the input list that are designed as Flowcharts and stored in the database using the Implementation menu</p> <p><b>Syntax:</b></p> <pre>stm_r_sb_procedural_fch_sb (IN sb_list: LIST OF SUBROUTINE, OUT st: INTEGER);</pre>

<p><b>stm_r_sch_connected_to_sb</b></p>	<p><b>Query:</b> Statecharts connected to a given subroutine  <b>Purpose:</b> Returns the procedural Statecharts that are connected to the subroutines in the input list  <b>Syntax:</b>  stm_r_sch_connected_to_sb (IN sb_list: LIST OF SUBROUTINE, OUT st: INTEGER);</p>
<p><b>stm_r_fch_connected_to_sb</b></p>	<p><b>Query:</b> Flowcharts connected to a given subroutine  <b>Purpose:</b> Returns the Flowcharts that are connected to the subroutines in the input list  <b>Syntax:</b>  stm_r_fch_connected_to_sb (IN sb_list: LIST OF SUBROUTINE, OUT st: INTEGER);</p>

## States (st)

This section documents the queries that return a list of states.

### Input List Type: ac

<p><b>stm_r_st_done_throughout_ac</b></p>	<p><b>Query:</b> States in which a given activity is performed throughout  <b>Purpose:</b> Returns the states for which activities in the input list are performed throughout that state (as specified in <b>Activities Within/Throughout</b> field in the state's form)  <b>Syntax:</b>  stm_r_st_done_throughout_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>
<p><b>stm_r_st_done_within_ac</b></p>	<p><b>Query:</b> States in which a given activity is performed within them  <b>Purpose:</b> Returns the states in which activities in the input list are performed within that state (as specified in <b>Activities Within/Throughout</b> field in the state's form)  <b>Syntax:</b>  stm_r_st_done_within_ac (IN ac_list: LIST OF ACTIVITY, OUT status: INTEGER)</p>

## Input List Type: ch

<b>stm_r_st_def_or_unres_in_ch</b>	<p><b>Query:</b> States defined or unresolved in a given chart</p> <p><b>Purpose:</b> Returns the states that are explicitly defined or unresolved in the charts of the input list</p> <p><b>Syntax:</b>  <code>stm_r_st_def_or_unres_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</code></p>
<b>stm_r_st_defined_in_ch</b>	<p><b>Query:</b> States defined in a given chart</p> <p><b>Purpose:</b> Returns the states that are explicitly defined in the charts of the input list</p> <p><b>Syntax:</b>  <code>stm_r_st_defined_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>
<b>stm_r_st_instance_of_ch</b>	<p><b>Query:</b> State instances of a given chart</p> <p><b>Purpose:</b> Returns the instance states defined by the charts in the input list</p> <p><b>Syntax:</b>  <code>stm_r_st_instance_of_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>
<b>stm_r_st_root_in_ch</b>	<p><b>Query:</b> Root states of a given chart</p> <p><b>Purpose:</b> Returns the internally defined states (of type diagram) attached to the charts in the input list</p> <p><b>Syntax:</b>  <code>stm_r_st_root_in_ch (IN ch_list: LIST OF CHART, OUT status: INTEGER)</code></p>
<b>stm_r_st_top_level_in_ch</b>	<p><b>Query:</b> Top-level states of a given chart</p> <p><b>Purpose:</b> Returns the top-level states (not contained in any box) of the charts in the input list</p> <p><b>Syntax:</b>  <code>stm_r_st_top_level_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</code></p>
<b>stm_r_st_unresolved_in_ch</b>	<p><b>Query:</b> States unresolved in a given chart</p> <p><b>Purpose:</b> Returns the states that are unresolved in the charts of the input list</p> <p><b>Syntax:</b>  <code>stm_r_st_unresolved_in_ch (IN ch_list: LIST OF CHART,OUT status: INTEGER)</code></p>

### Input List Type: cn

<b>stm_r_st_containing_cn</b>	<p><b>Query:</b> States containing a given connector</p> <p><b>Purpose:</b> Returns the states that encapsulate specified connectors from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_st_containing_cn (IN cn_list: LIST OF CONNECTOR,OUT status: INTEGER)</pre>
-------------------------------	--

### Input List Type: mx

<b>stm_r_st_affecting_mx</b>	<p><b>Query:</b> States in which a given element is affected.</p> <p><b>Purpose:</b> Returns the states that affect (modify, generate, or activate) the elements (for example, events, data-items, or activities) in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_st_affecting_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_st_meaningfully_affecting_mx</b>	<p><b>Query:</b> States in which a given element is affected.</p> <p><b>Purpose:</b> Identical to <code>stm_r_st_affecting_mx</code>, but when the input list includes an ID of a record/union, <code>stm_r_st_meaningfully_affecting_mx</code> will also return elements that affect a field of the record/union, and not necessarily the whole record/union element.</p> <p><b>Syntax:</b></p> <pre>stm_r_st_meaningfully_affecting_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_st_using_mx</b>	<p><b>Query:</b> States in which a given element is used.</p> <p><b>Purpose:</b> Returns the states in static reactions that use (evaluate) the elements (basic events, conditions, data-items, states, and activities) in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_st_using_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<b>stm_r_st_meaningfully_using_mx</b>	<p><b>Query:</b> States in which a given element is used.</p> <p><b>Purpose:</b> Identical to <code>stm_r_st_using_mx</code>, but when the input list includes an ID of a record/union, <code>stm_r_st_meaningfully_using_mx</code> will also return elements that affect a field of the record/union, and not necessarily the whole record/union element.</p> <p><b>Syntax:</b></p> <pre>stm_r_st_meaningfully_using_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER);</pre>

## Input List Type: st

<b>stm_r_st_and_st</b>	<p><b>Query:</b> And states.</p> <p><b>Purpose:</b> Returns the states in the input list that are And-states.</p> <p><b>Syntax:</b> stm_r_st_and_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</p>
<b>stm_r_st_basic_st</b>	<p><b>Query:</b> Basic states.</p> <p><b>Purpose:</b> Returns the states in the input list that are basic (states that have no descendants).</p> <p><b>Syntax:</b> stm_r_st_basic_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</p>
<b>stm_r_st_by_attributes_st</b>	<p><b>Query:</b> States by attributes.</p> <p><b>Purpose:</b> Returns the states in the input list that match a given attribute name and value.</p> <p><b>Syntax:</b> stm_r_st_by_attributes_st (IN st_list: LIST OF STATE, IN attr_name: STRING, IN attr_value: STRING, OUT status: INTEGER)</p>
<b>stm_r_st_callback_binding_st</b>	<p><b>Query:</b> States with callback bindings.</p> <p><b>Purpose:</b> Returns the states in the input list that have callback bindings.</p> <p><b>Syntax:</b> stm_r_st_callback_binding_st (IN el_list: LIST OF STATE, OUT status: INTEGER)</p>
<b>stm_r_st_def_of_instance_st</b>	<p><b>Query:</b> Definition states of a given state.</p> <p><b>Purpose:</b> Returns the definition states (top-level in the definition chart) for instances in the input list.</p> <p><b>Syntax:</b> stm_r_st_def_of_instance_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</p>
<b>stm_r_st_default_entry_to_st</b>	<p><b>Query:</b> Default entry to the default state.</p> <p><b>Purpose:</b> Returns the default states of the or-states in the input list.</p> <p><b>Syntax:</b> stm_r_st_default_entry_to_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</p>

<b>stm_r_st_explicit_defined_st</b>	<p><b>Query:</b> States explicitly defined.</p> <p><b>Purpose:</b> Returns the states in the input list that were explicitly defined.</p> <p><b>Syntax:</b>  <code>stm_r_st_explicit_defined_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</code></p>
<b>stm_r_st_generic_instance_st</b>	<p><b>Query:</b> Generic instance states.</p> <p><b>Purpose:</b> Returns the states in the input list that are instances of generic charts.</p> <p><b>Syntax:</b>  <code>stm_r_st_generic_instance_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</code></p>
<b>stm_r_st_history_connector_st</b>	<p><b>Query:</b> States containing a history connector.</p> <p><b>Purpose:</b> Returns the states in the input list that contain a history connector.</p> <p><b>Syntax:</b>  <code>stm_r_st_history_connector_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</code></p>
<b>stm_r_st_instance_of_def_st</b>	<p><b>Query:</b> Instance states of a given definition state.</p> <p><b>Purpose:</b> Returns the instance states for definition states (top-level states in a definition chart) in the input list.</p> <p><b>Syntax:</b>  <code>stm_r_st_instance_of_def_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</code></p>
<b>stm_r_st_instance_st</b>	<p><b>Query:</b> Instance states.</p> <p><b>Purpose:</b> Returns those states in the input list that are instance states.</p> <p><b>Syntax:</b>  <code>stm_r_st_instance_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</code></p>
<b>stm_r_st_logical_desc_of_st</b>	<p><b>Query:</b> Logical descendants of a given state.</p> <p><b>Purpose:</b> Returns the logical descendants (children, grandchildren, and so on) of states in the input list, taking into account the translation of instances to their definition charts.</p> <p><b>Syntax:</b>  <code>stm_r_st_logical_desc_of_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</code></p>



<b>stm_r_st_logical_parent_of_st</b>	<p><b>Query:</b> Logical parent states of a given state.</p> <p><b>Purpose:</b> Returns the logical parent states of the states in the input list, taking into account the translation of instances to their definition charts.</p> <p><b>Note:</b> This query provides similar output as <code>stm_r_st_physical_parent_of_st</code>, but for states that are substates of a top-level state in a definition chart, this query also returns the instance box.</p> <p><b>Syntax:</b></p> <pre>stm_r_st_logical_parent_of_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</pre>
<b>stm_r_st_logical_sub_of_st</b>	<p><b>Query:</b> Logical substates of a given state.</p> <p><b>Purpose:</b> Returns the logical substates of the states in the input list, taking into account the translation of instances to their definition charts.</p> <p><b>Syntax:</b></p> <pre>stm_r_st_logical_sub_of_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</pre>
<b>stm_r_st_name_of_st</b>	<p><b>Query:</b> States whose names match a given pattern</p> <p><b>Purpose:</b> Returns all the states whose names match the specified pattern</p> <p><b>Syntax:</b></p> <pre>stm_r_st_name_of_st (IN pattern: STRING, OUT status: INTEGER)</pre>
<b>stm_r_st_offpage_instance_st</b>	<p><b>Query:</b> Offpage instance states.</p> <p><b>Purpose:</b> Returns the states in the input list that are instances of offpage charts.</p> <p><b>Syntax:</b></p> <pre>stm_r_st_offpage_instance_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</pre>
<b>stm_r_st_physical_desc_of_st</b>	<p><b>Query:</b> Physical descendants of a given state.</p> <p><b>Purpose:</b> Returns the physical descendants (those within the same chart) for the states in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_st_physical_desc_of_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</pre>
<b>stm_r_st_physical_parent_of_st</b>	<p><b>Query:</b> Physical parent states of a given state.</p> <p><b>Purpose:</b> Returns the physical parent states (those within the same chart) for the states in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_st_physical_parent_of_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</pre>

<p><b>stm_r_st_physical_sub_of_st</b></p>	<p><b>Query:</b> Physical substates of a given state.  <b>Purpose:</b> Returns the physical substates (those within the same chart) for the states in the input list.  <b>Syntax:</b>          stm_r_st_physical_sub_of_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</p>
<p><b>stm_r_st_reaction_activity_st</b></p>	<p><b>Query:</b> States having reactions or activities.  <b>Purpose:</b> Returns the states from the input list that have static reactions or activities performed within or throughout the state.  <b>Syntax:</b>          stm_r_st_reaction_activity_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</p>
<p><b>stm_r_st_synonym_of_st</b></p>	<p><b>Query:</b> States whose synonyms match a given pattern  <b>Purpose:</b> Returns all the states whose synonyms match the specified pattern  <b>Syntax:</b>          stm_r_st_synonym_of_st (IN pattern: STRING, OUT status: INTEGER)</p>
<p><b>stm_r_st_unresolved_st</b></p>	<p><b>Query:</b> Unresolved states.  <b>Purpose:</b> Returns the unresolved states in the input list.  <b>Syntax:</b>          stm_r_st_unresolved_st (IN st_list: LIST OF STATE, OUT status: INTEGER)</p>

### Input List Type: tr

<p><b>stm_r_st_source_of_tr</b></p>	<p><b>Query:</b> States that are sources of a given transition  <b>Purpose:</b> Returns the states that are sources of transitions in the input list  <b>Syntax:</b>          stm_r_st_source_of_tr (IN tr_list: LIST OF TRANSITION, OUT status: INTEGER)</p>
<p><b>stm_r_st_target_of_tr</b></p>	<p><b>Query:</b> States that are targets of a given transition  <b>Purpose:</b> Returns the states that are targets of transitions in the input list  <b>Syntax:</b>          stm_r_st_target_of_tr (IN tr_list: LIST OF TRANSITION, OUT status: INTEGER)</p>

## Timing Constraint (tc)

This section documents the query that returns a list of timing constraints.

### Input List Type: ch

<b>stm_r_tc_defined_in_ch</b>	<b>Query:</b> Timing constraints defined in a given chart <b>Purpose:</b> Returns the timing constraints that are explicitly defined in the charts of the input list <b>Syntax:</b> <code>stm_r_tc_defined_in_ch (IN ch_lst: LIST OF CHART, OUT status: INTEGER)</code>
-------------------------------	--

## Transitions (tr)

This section documents the queries that return a list of transitions.

### Input List Type: cn

<b>stm_r_tr_to_target_cn</b>	<b>Query:</b> Transitions whose target is a given connector <b>Purpose:</b> Returns the transition in the input list whose target is a termination or history connector <b>Syntax:</b> <code>stm_r_tr_to_target_cn (IN cn_list: LIST OF CONNECTOR,OUT status: INTEGER)</code>
------------------------------	--

### Input List Type: mx

<p><b>stm_r_tr_affecting_mx</b></p>	<p><b>Query:</b> Transitions in which a given element is affected.</p> <p><b>Purpose:</b> Returns the transitions that affect (modify, generate, or activate) the elements (for example, events, data-items, or activities) in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_tr_affecting_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<p><b>stm_r_tr_meaningfully_affecting_mx</b></p>	<p><b>Query:</b> Transitions in which a given element is affected.</p> <p><b>Purpose:</b> Identical to <code>stm_r_tr_affecting_mx</code>, but when the input list includes an ID of a record/union, <code>stm_r_tr_meaningfully_affecting_mx</code> will also return elements that affect a field of the record/union, and not necessarily the whole record/union element.</p> <p><b>Syntax:</b></p> <pre>stm_r_tr_meaningfully_affecting_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER);</pre>
<p><b>stm_r_tr_from_source_mx</b></p>	<p><b>Query:</b> Transitions whose source is a given element</p> <p><b>Purpose:</b> Returns transitions that originate at elements in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_tr_from_source_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<p><b>stm_r_tr_to_target_mx</b></p>	<p><b>Query:</b> Transitions whose target is a given element</p> <p><b>Purpose:</b> Returns the transitions whose target is an element from the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_tr_to_target_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>
<p><b>stm_r_tr_using_mx</b></p>	<p><b>Query:</b> Transitions in which a given element is used.</p> <p><b>Purpose:</b> Returns the transitions in labels that use (evaluate) the elements (basic events, conditions, data-items, states, and activities) in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_tr_using_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER)</pre>

<b>stm_r_tr_meaningly_using_mx</b>	<p><b>Query:</b> Transitions in which a given element is used.</p> <p><b>Purpose:</b> Identical to <code>stm_r_tr_using_mx</code>, but when the input list includes an ID of a record/union, <code>stm_r_tr_meaningly_using_mx</code> will also return elements that affect a field of the record/union, and not necessarily the whole record/union element.</p> <p><b>Syntax:</b></p> <pre>stm_r_tr_meaningly_using_mx (IN mx_list: LIST OF ELEMENT,OUT status: INTEGER);</pre>
------------------------------------	--

### Input List Type: st

<b>stm_r_tr_default_of_st</b>	<p><b>Query:</b> Transitions that are the default entrance of a given state</p> <p><b>Purpose:</b> Returns the default entrances (compound transitions) of the states in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_tr_default_of_st (IN st_list: LIST OF STATE,OUT status: INTEGER)</pre>
<b>stm_r_tr_from_source_st</b>	<p><b>Query:</b> Transitions whose source is the specified state</p> <p><b>Purpose:</b> Returns the transitions whose source is a state appearing in the input list</p> <p><b>Syntax:</b></p> <pre>stm_r_tr_from_source_st (IN st_list: LIST OF STATE,OUT status: INTEGER)</pre>
<b>stm_r_tr_to_target_st</b>	<p><b>Query:</b> Transitions whose target is a given state</p> <p><b>Purpose:</b> Returns the transitions whose target is a state appearing in the input list.</p> <p><b>Syntax:</b></p> <pre>stm_r_tr_to_target_st (IN st_list: LIST OF STATE,OUT status: INTEGER)</pre>

### Input List Type: tr

<b>stm_r_tr_default_tr</b>	<p><b>Query:</b> Default transition</p> <p><b>Purpose:</b> Returns, of all the transitions in the input list that are default transitions</p> <p><b>Syntax:</b></p> <pre>stm_r_tr_default_tr (IN tr_list: LIST OF TRANSITION,OUT status: INTEGER)</pre>
----------------------------	---



# Utility Functions

---

Utility functions enable you to manipulate lists. For example, you could use utility functions to determine whether a particular element exists in a list of Statemate elements. Or, you could sort these lists to make reports easier to read. You can also use utility functions to manipulate strings of characters—to locate string patterns in a given string and to extract portions of strings. Most utility functions for lists can manipulate lists of any item type, but are usually used for lists of Statemate elements.

Utility functions do not extract information from the database; however, some utility functions use database information to complete their operations. These functions enable you to manipulate the information you have already retrieved using single-element or query functions.

In this appendix, the term *item* refers to any list member; the term *element* refers to a Rhapsody element.

The topics are as follows:

- ◆ [Calling Utility Functions](#)
- ◆ [Utility Function Input Arguments](#)
- ◆ [Examples of Utility Functions](#)
- ◆ [List of Utility Functions](#)

## Calling Utility Functions

The calling sequence of the list functions is as follows:

```
stm_list_operation (list, status)
```

In this syntax:

- ◆ **stm\_list**—Designates the function as a Statemate list manipulation function
- ◆ **operation**—The kind of list operation performed
- ◆ **list**—The list to be operated on
- ◆ **status**—The return function status code

The type of value returned by the function depends on the particular function. The returned value can be a list, a Statemate element, a string, or an integer. For example:

```
stm_list_sort_by_name (event_list, status)
```

This function alphabetically sorts the events in `event_list` according to their names. Function return values for each utility function are listed in [Single-Element Functions](#).

The following sections document the utility functions that use a different calling sequence.

## Contains Element

The `contains_element` function returns a Boolean value of `TRUE` if the item of interest is contained in the list (and `FALSE` if not contained).

The syntax is as follows:

```
stm_list_contains_element (list, item, status)
```

In this syntax:

- ◆ **stm\_list**—Designates the function as a Statemate list manipulation function.
- ◆ **contains\_element**—Signifies that this function checks to see whether the specified item is a member of the list.
- ◆ **list**—Is the list in which to search for the specified item. `list` can be any DGL list type (for example, `LIST OF ELEMENT`).
- ◆ **item**—Is an item whose presence in the list you want to verify. The data type of the item must be compatible with `list`'s data type.
- ◆ **status**—Is the return function status code.



For example:

```
stm_list_contains_element (state_list, state_id, status)
```

This function returns a value of `TRUE` if the state whose ID is `state_id` is a member of the list `state_list`.

## List Extraction by Type

The `list_extraction_by_type` function extracts the StateMate elements in the input list that are of a particular data type.

The syntax is as follows:

```
stm_list_extraction_by_type (element_type, list, status)
```

In this syntax:

- ◆ **stm\_list**—Designates the function as a StateMate list manipulation function.
- ◆ **extraction\_by\_type**—Signifies that this function extracts all StateMate elements of a specified type from the input list.
- ◆ **element\_type**—Is a StateMate element type predefined constant.
- ◆ **list**—Is the list in which to search for elements of the specified type. `list` can contain a variety of StateMate elements (a mixed list).
- ◆ **status**—Is the return function status code.

For example:

```
stm_list_extraction_by_type (stm_state, elmnt_list, status);
```

This call extracts all states from the list assigned to the variable `elmnt_list`. `stm_state` is one of the predefined constants that designates an element type.

## List Extraction by Chart

The `list_extraction_by_chart` function extracts the StateMate elements in the input list that are of a particular chart.

The syntax is as follows:

```
stm_list_extraction_by_chart (chart_name, list, status)
```

In this syntax:

- ◆ **stm\_list**—Designates the function as a StateMate list manipulation function.
- ◆ **extraction\_by\_chart**—Signifies that this function extracts all StateMate elements defined in a specific chart from the input list.
- ◆ **chart\_name**—Is a string that represents the chart name.
- ◆ **list**—Is the list in which to search for elements of the specified chart. `list` can contain a variety of StateMate elements.
- ◆ **status**—Is the return function status code.

For example:

```
stm_list_extraction_by_chart ('my_chart', elmnt_list, status);
```

This call extracts all elements from the list that defined in (or unresolved in) the chart `my_chart`.

## Location of Pattern in a String

The `index` function returns the first location from the left (starting with 0), in which `pattern` appears in the input string. It returns -1 when the pattern is not found.

The syntax is as follows:

```
stm_index (string, offset, pattern, status)
```

In this syntax:

- ◆ **stm\_**—The standard prefix of StateMate functions.
- ◆ **index**—Signifies that the function looks for a location in the string.
- ◆ **string**—The input string in which the pattern is searched.
- ◆ **offset**—An integer greater than or equal to zero that represents the location at which the search begins.
- ◆ **pattern**—The string to search for.
- ◆ **status**—The return status code. The possible values are `stm_success` or `stm_null_string` (when uninitialized strings are used as parameters).

For example:

```
stm_index('ABCA',0,'C',status) = 2
stm_index('ABCA',0,'AB',status) = 0
stm_index('ABCA',2,'A',status) = 3
stm_index('ABCA',0,'AC',status) = -1
```

## Extract Portion of a String

The `string_extract` function extracts a portion of a string.

The syntax is as follows:

```
stm_string_extract (string, index, length, status)
```

In this syntax:

- ◆ **stm\_string**—Designates the function as a string manipulation function
- ◆ **extract**—Signifies that the function extracts a portion of the input string
- ◆ **string**—The input string
- ◆ **index**—The integer location from which the output string portion starts
- ◆ **length**—The output string length (integer)
- ◆ **status**—One of the following status codes: `stm_success`, `stm_null_string`, `stm_illegal_index`, and `stm_illegal_len`

For example:

```
stm_string_extract('ABCDE',0,2,status) = 'AB'
stm_string_extract('ABCDE',3,2,status) = 'DE'
stm_string_extract('ABCDE',3,7,status) = ''
and status = stm_illegal_len
```

## Utility Function Input Arguments

The following table lists the input arguments used by the utility functions.

Argument	Description	DGL Data Type
list	List of items upon which you want to perform a logical or relational operation. The list's data type must be compatible with the particular utility function in which it appears as an argument.	LIST OF . . .
item	An item whose presence in the list you want to verify. The data type of the item must be compatible with list's data type.	Any
element type	A Statemate element type predefined constant (refer to <a href="#">Statemate Element Types</a> .)	Statemate element type
string	The string on which to perform a string manipulation function.	String
offset	The position ( $\geq 0$ ) in the string from which to start the specified manipulation.	Integer
pattern	The string portion searched for in the input string.	String
length	The length of the string portion extracted from the input string.	Integer

## Examples of Utility Functions

This section shows how to use utility function calls to perform common tasks.

### Utility Functions Example 1

The following example shows how to find the number of subactivities that exist for activity A1.

```
VARIABLE
  ACTIVITY          act_id, cntrl_act;
  LIST OF ACTIVITY  act_list, cntrl_act_list;
  INTEGER           status, list_length;
  .
  .
act_id := stm_r_ac ('A1', status);
act_list := stm_r_ac_physical_sub_of_ac ({act_id},
  status);
list_length := stm_list_length (act_list, status);
  .
  .
```

The example uses a single-element function to determine the ID of A1, then uses a query function to retrieve the list of A1's subactivities. Finally, it assigns the number of A1's subactivities to `list_length`.

### Utility Functions Example 2

The following example is a continuation of the previous example. If you know that only one of the activities in the list `act_list` is a control activity and want to find out which activity it is, include the following code in your template:

```
  .
  .
cntrl_act_list := stm_r_ac_control_ac (act_list, status);
cntrl_act := stm_list_first_element (cntrl_act_list,
  status);
  .
  .
```

The list `cntrl_act_list` consists of only one element. This code extracts the first element (in this case, the only element) of the list and assigns this control activity's ID to `cntrl_act`.

## Utility Functions Example 3

The following example shows how to find the software modules in a list of modules:

```
VARIABLE
  MODULE          md_id;
  LIST OF STRING  imp_type;
  INTEGER         status;
  .
  .
  .
  imp_type := stm_r_md_attr_val (md_id, 'IMPLEMENTATION',
    status);
  IF stm_list_contains_string (imp_type, 'SOFTWARE',
    status)
  THEN
    .
    .
    .
```

First, the code finds all the values for the `IMPLEMENTATION` attribute for the module, `md_id`. Among these values, the code searches for the value `SOFTWARE`. (Multiple values can exist for a given attribute.) If it is found, the statements following `THEN` are executed.

## List of Utility Functions

In general, all the `stm_list` utilities work on lists of Rhapsody elements except strings. Strings have their own set of utilities, including:

```
stm_str_list_length (l_str, status)
stm_str_list_next_element (l_str, status)
stm_str_list_first_element (l_str, status)
stm_str_list_previous_element (l_str, status)
stm_str_list_last_element (l_str, status)
stm_list_contains_string (l_str, status)
```

The following pages document the utility functions. The functions are presented in alphabetical order, as listed in the following table.

<a href="#">stm_action_of_reaction</a>	Extracts the action part of the specified reaction
<a href="#">stm_delete_file</a>	Returns the definition type of the specified textual element
<a href="#">stm_dispose_memory</a>	Frees temporary memory used by the DOC system.
<a href="#">stm_dispose_memory</a>	Searches for the specified element in the list created by the search
<a href="#">stm_dispose_memory</a>	Looks for the specified pattern in the given string
<a href="#">stm_int</a>	Converts the specified real number to an integer
<a href="#">stm_int_to_string</a>	Returns the string representation of the specified (decimal) number
<a href="#">stm_r_is_statemate</a>	Returns TRUE if this is Statemate Classic
<a href="#">stm_list_contains_element</a>	Determines whether the specified item appears in the given list

<a href="#"><u>stm_list_contains_string</u></a>	Determines whether the specified string appears in the given list
<a href="#"><u>stm_list_extraction</u></a>	Extracts the elements from the input list
<a href="#"><u>stm_list_extraction_by_chart</u></a>	Extracts the elements from the input list that belong to the specified chart
<a href="#"><u>stm_list_extraction_by_type</u></a>	Extracts elements of the specified type from the given list of StateMate elements
<a href="#"><u>stm_list_first_element</u></a>	Returns the first element in the specified list
<a href="#"><u>stm_list_last_element</u></a>	Returns the last item in the specified list
<a href="#"><u>stm_list_length</u></a>	Returns the length of the specified list
<a href="#"><u>stm_list_next_element</u></a>	Returns the next element in the specified list
<a href="#"><u>stm_plot_ext</u></a>	Generates a plot file with the indicated parameters,
<a href="#"><u>stm_list_previous_element</u></a>	Returns the previous element in the specified list
<a href="#"><u>stm_list_sort</u></a>	Alphabetically sorts the specified list of strings
<a href="#"><u>stm_list_sort_by_attr_value</u></a>	Sorts the specified list of StateMate elements by the value of the given attribute
<a href="#"><u>stm_list_sort_by_branches</u></a>	Sorts the list of hierarchical StateMate elements by branch
<a href="#"><u>stm_list_sort_by_chart</u></a>	Alphabetically sorts the input list of named StateMate elements by the name of the chart to which they belong
<a href="#"><u>stm_list_sort_by_levels</u></a>	Sorts the list of hierarchical StateMate elements by level
<a href="#"><u>stm_list_sort_by_name</u></a>	Sorts the list of StateMate elements alphabetically by name
<a href="#"><u>stm_list_sort_by_synonym</u></a>	Sorts the list of StateMate elements alphabetically by synonym
<a href="#"><u>stm_list_sort_by_type</u></a>	Sorts the list of StateMate elements by type
<a href="#"><u>stm_multiline_to_one</u></a>	Converts the specified multiline string (with new lines) to a one-line string (without new lines)
<a href="#"><u>stm_multiline_to_strings</u></a>	Converts the specified multiline expression to a list of strings
<a href="#"><u>stm_plot</u></a>	Generates a plot file with the indicated parameters, such as plot size, output device, and so on
<a href="#"><u>stm_plot_hyper_exp</u></a>	Generates the hyperlinks in a sequence diagram
<a href="#"><u>stm_plot_with_autonumber</u></a>	Prints a sequence diagram with numbered scenarios
<a href="#"><u>stm_plot_with_break</u></a>	Breaks a sequence diagram across multiple pages
<a href="#"><u>stm_plot_with_headerline</u></a>	Prints a sequence diagram with the names of lifelines on every page
<a href="#"><u>stm_replace_string</u></a>	Replaces the specified pattern with a new pattern within a string
<a href="#"><u>stm_replace_word</u></a>	Replaces the specified word with a new word within a given string
<a href="#"><u>stm_set_rpt_formator</u></a>	Returns the static reactions defined for the specified state element.
<a href="#"><u>stm_str_list_first_element</u></a>	Returns the first item in the specified list of strings
<a href="#"><u>stm_str_list_last_element</u></a>	Returns the element ID of the last item appearing in the specified list of strings
<a href="#"><u>stm_str_list_length</u></a>	Returns the number of items in the specified list of strings
<a href="#"><u>stm_str_list_next_element</u></a>	Returns the next item in the specified list of strings
<a href="#"><u>stm_str_list_previous_element</u></a>	Returns the previous item in the specified list of strings

## Utility Functions

---

<a href="#"><u>stm_str_list_to_str</u></a>	Returns the string representation of the specified list of strings
<a href="#"><u>stm_str_to_list</u></a>	Converts the specified string to a list of strings
<a href="#"><u>stm_string_extract</u></a>	Extracts a portion of the specified string
<a href="#"><u>stm_string_free</u></a>	Frees the memory used by the specified string
<a href="#"><u>stm_string_to_int</u></a>	Returns an integer value of a decimal string representation of a number
<a href="#"><u>stm_string_retain</u></a>	Frees temporary memory used by the DOC system.
<a href="#"><u>stm_strlen</u></a>	Returns the length of the specified string
<a href="#"><u>stm_trigger_of_reaction</u></a>	Returns the trigger part of a reaction



## stm\_action\_of\_reaction

**Function type:** STRING

### Description

Extracts the action part of the specified reaction (the label of the transition or static reaction). The syntax of a reaction is *trigger/action*.

Note the following:

- ◆ The reaction is achieved by the single-element function `stm_r_st_reactions` or `stm_r_tr_labels`.
- ◆ The function returns an empty string when the action is missing.

### Syntax

```
stm_action_of_reaction (reaction, status)
```

### Arguments

Argument	Input/Output	Type	Description
reaction	In	STRING	The reaction to decompose
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`

### Example

Assume that `s1` has several static reactions and you want to list all the actions that are triggered when in this state. Include the following statements in your template:

```
VARIABLE
  STATE          st_id;
  INTEGER        status;
  LIST OF STRING reactions;
  STRING         rct;
st_id := stm_r_st ('S1', status);
reactions := stm_r_st_reactions (st_id, status);
WRITE ('\n Actions of reactions in S1:');
FOR rct IN reactions LOOP
  WRITE ('\n', stm_action_of_reaction (rct, status));
END FOR;
```

## stm\_delete\_file

### Description:

Deletes the file.

### Syntax

```
stm_delete_file (file_name_and_path, status)
```

### Arguments

Argument	Input/Output	Type	Description
file_name_and_path	In	STRING	The file name and path.
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`

## stm\_dispose\_memory

### Description:

When used in a DOC template, `STM_DISPOSE_MEMORY()` frees temporary memory used by the DOC system. If you want to avoid freeing a specific string, for example, general strings used in the template, call `STM_STRING_RETAIN(<str>)` on that string before calling `STM_DISPOSE_MEMORY()`.

### Syntax

```
stm_dispose_memory()
```

### Arguments

None

### Status Codes

None

### Example

Call the function with no parameters: `STM_DISPOSE_MEMORY();`

## stm\_index

**Function type:** INTEGER

### Description

Looks for the specified pattern in the given string. It returns the first occurrence, starting from the offset position.

Note the following:

- ◆ The first position in the string is 0.
- ◆ If the pattern is not found in the string, the function returns -1.

### Syntax

```
stm_index (string, offset, pattern, status)
```

### Arguments

Argument	Input/Output	Type	Description
string	In	STRING	The string to be searched
offset	In	INTEGER	The starting position in the string
pattern	In	STRING	The pattern to look for
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_null_string`

### Example

```
stm_index('AB@CB',0,'B',status) = 1  
stm_index('AB@CB',3,'B',status) = 4  
stm_index('AB@CB',0,'XY',status) = -1
```

## stm\_int

**Function type:** INTEGER

### Description

Converts the specified real number to an integer.

### Syntax

```
stm_int(f)
```

### Arguments

Argument	Input/Output	Type	Description
f	In	FLOAT	The floating-point number to convert

## stm\_int\_to\_string

**Function type:** STRING

### Description

Returns the string representation of the specified (decimal) number.

### Syntax

```
stm_int_to_string(int)
```

### Arguments

Argument	Input/Output	Type	Description
reaction	In	INTEGER	The integer to be converted

### Status Codes

- ◆ stm\_success

### Example

```
str := stm_int_to_string(36);  
str is '36'
```

## stm\_r\_is\_statemate

**Function type:** BOOLEAN

### Description

Returns TRUE if this is Statemate Classic.

### Syntax

```
stm_r_is_statemate()
```

## stm\_list\_contains\_element

**Function type:** BOOLEAN

### Description

Determines whether the specified item appears in the given list.

### Syntax

```
stm_list_contains_element (list, item, status)
```

### Arguments

Argument	Input/Output	Type	Description
list	In	LIST OF ELEMENT	The list to search.
item	In	ELEMENT	The item to look for. This can be a member of any element <i>except</i> strings. For strings, use <code>stm_list_contains_string</code> . Note that the item's data type must conform to the data type of the list. For example, if <code>list</code> is declared to be <code>LIST OF STATE</code> , <code>item</code> must be of type <code>STATE</code> (or <code>ELEMENT</code> ). When the list consists of Statemate elements, <code>item</code> contains the element's ID.
status	Out	INTEGER	The function status code.

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nil_list`

### Example

Suppose you want to check a list of Rhapsody elements for the presence of activity `A1`. The elements of interest are assigned to the list `elmnt_list`. Your template should contain the following statements:

```
VARIABLE
  ACTIVITY          act_id;
  LIST OF ELEMENT   elmnt_list;
  INTEGER           status;
  .
  .
  .
act_id := stm_r_ac('A1', status);
  .
  .
  .
IF stm_list_contains_element (elmnt_list, act_id, status) THEN
  .
  .
  .
```

If `A1` appears in `elmnt_list`, the statements following the `IF` statement are executed. Note that the ID of `A1` is passed to the function, not its name).

## stm\_list\_contains\_string

**Function type:** BOOLEAN

### Description

Determines whether the specified string appears in the given list.

### Syntax

```
stm_list_contains_string (list, item, status)
```

### Arguments

Argument	Input/Output	Type	Description
list	In	LIST OF ELEMENT	The list of strings to search
item	In	ELEMENT	The string to look for
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_nil\_list

### Example

Suppose you want to check a list of strings for the presence of a specific string. Your template should contain the following statements:

```
VARIABLE
  ACTIVITY          act_id;
  LIST OF STRING    string_list;
  INTEGER           status;
.
.
IF stm_list_contains_string
  (string_list, 'SPECIFICATION', status) THEN
  .
  .
```

If 'SPECIFICATION' appears in `string_list`, the statements following the `IF` statement are executed.



## stm\_list\_extraction

**Function type:** LIST OF ELEMENT

### Description

Extracts the elements from the input list.

### Syntax

```
stm_list_extraction (ex_type, el_list, status)
```

### Arguments

Argument	Input/Output	Type	Description
ex_type	In	INTEGER	The type to extract from the list
list	In	LIST OF ELEMENT	The list of Rhapsody elements
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nil_list`

## stm\_list\_extraction\_by\_chart

**Function type:** LIST OF ELEMENT

### Description

Extracts the elements from the input list that belong to the specified chart.

### Syntax

```
stm_list_extraction_by_chart (chart_name, list, status)
```

### Arguments

Argument	Input/Output	Type	Description
chart_name	In	STRING	The name of the chart
list	In	LIST OF ELEMENT	The list of Rhapsody elements
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_nil\_list
- ◆ stm\_illegal\_name
- ◆ stm\_name\_not\_found

### Example

To extract the list of all of Rhapsody elements that belong to the statechart s8 from the input list `elem_list`, use the following statements:

```
VARIABLE
LIST OF ELEMENT  elem_list, S8_elements;
INTEGER          status;
S8_elements:=
stm_list_extraction_by_chart('S8', elem_list, status);
.
```

## stm\_list\_extraction\_by\_chart\_id

**Function type:** LIST OF ELEMENT

### Description

Extracts the elements from the input list that belong to the specified chart.

### Syntax

```
stm_list_extraction_by_chart (chart_name, list, status)
```

### Arguments

Argument	Input/Output	Type	Description
chart_name	In	STRING	The name of the chart
list	In	LIST OF ELEMENT	The list of Rhapsody elements
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_nil\_list
- ◆ stm\_illegal\_name
- ◆ stm\_name\_not\_found

### Example

To extract the list of all of Rhapsody elements that belong to the statechart s8 from the input list `elem_list`, use the following statements:

```
VARIABLE
LIST OF ELEMENT  elem_list, S8_elements;
INTEGER          status;
S8_elements:=
stm_list_extraction_by_chart('S8', elem_list, status);
```

## stm\_list\_extraction\_by\_type

**Function type:** LIST OF ELEMENT

### Description

Extracts elements of the specified type from the given list of Rhapsody elements.

### Syntax

```
stm_list_extraction_by_type (element_type, list, status)
```

### Arguments

Argument	Input/Output	Type	Description
element_type	In	STRING	The type to look for. element_type is one of the possible values of the enumerated type stm_element_type. The values of this type usually take the form stm_element_type (for example, stm_state, stm_event, and so on).
list	In	LIST OF ELEMENT	The list of elements (mixed types).
status	Out	INTEGER	The function status code.

### Status Codes

- ◆ stm\_success
- ◆ stm\_nil\_list
- ◆ stm\_illegal\_extract\_type

**Example**

Suppose you want to extract a list of all the activities appearing in a list of Rhapsody elements. The input list is assigned to the variable `elmnt_list`. Your template should contain the following statements:

```
VARIABLE
  ACTIVITY          act;
  LIST OF ACTIVITY  act_list;
  LIST OF ELEMENT   elmnt_list;
  INTEGER           status;
  .
  .
  .
act_list := stm_list_extraction_by_type (stm_activity,
  elmnt_list, status);
WRITE ('\n The activities in the list are:');
FOR act IN act_list LOOP
  WRITE ('\n', stm_r_ac_name (act, status));
END LOOP;
  .
  .
  .
```

The names of all the activities in `elmnt_list` are written to your document.

## stm\_list\_first\_element

**Function type:** ELEMENT

### Description

Returns the first element in the specified list. This function can be applied to a list of any DGL data types.

### Syntax

```
stm_list_first_element (list, status)
```

### Arguments

Argument	Input/Output	Type	Description
list	In	LIST OF ELEMENT	The list of items. Items in the input list can be of any element type except string. For strings, see the function <code>stm_str_list_first_element</code> .
status	Out	INTEGER	The function status code.

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nil_list`
- ◆ `stm_list_element_does_not_exist`

### Example

Assume you have a list of activities assigned to the variable `act_list`. You know that only one of the activities in the list can be a control activity. To find out which activity this is, include the following statements in your template:

```
.
cntrl_act_list = stm_ac_control_ac (act_list, status);
cntrl_act = stm_list_first_element (cntrl_act_list,
status);
```

These statements extract the first element (in this case, the only element) of the list and assign this control activity's ID to `cntrl_act`.

## stm\_list\_last\_element

**Function type:** ELEMENT

### Description

Returns the last item in the specified list. This function can be applied to a list of any DGL data type.

### Syntax

```
stm_list_last_element (list, status)
```

### Arguments

Argument	Input/Output	Type	Description
list	In	LIST OF ELEMENT	The list of items. Items in the input list can be of any element type except string. For strings, see the function <code>stm_str_list_first_element</code> .
status	Out	INTEGER	The function status code.

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nil_list`
- ◆ `stm_list_element_does_not_exist`

### Example

Assume you have a list of states in the order `s1`, `s2`, `s3` and `s4`. The list is assigned to the variable `state_list`. To find the last item in this list (`s4`), use the following statements:

```
VARIABLE
    STATE          state_id;
    LIST OF STATE  state_list;
    INTEGER        status;
    .
    .
    .
state_id := stm_list_last_element (state_list, status);
WRITE ('\\n The last state in the list is: ',
      stm_r_st_name (state_id, status));
```

## stm\_list\_length

**Function type:** INTEGER

### Description

Returns the length of the specified list.

### Syntax

```
stm_list_length (list, status)
```

### Arguments

Argument	Input/Output	Type	Description
list	In	LIST OF ELEMENT	Items in the input list can be of any element type except string. For strings, refer to <a href="#">stm_str_list_length</a> .
status	Out	INTEGER	The function status code.

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nil_list`



**Example**

Assume you extracted all the events from the database whose name begins with EV. Before writing the list to your document, you want to make sure that it will not span more than 30 lines of text (one page length). Your template should contain the following statements:

```
VARIABLE
  EVENT          ev;
  LIST OF EVENT  ev_list;
  INTEGER        ev_list_len, status;
CONSTANT
  INTEGER page_len := 30;
          .
          .
          .
ev_list:-stm_re_ev_name_of_ev ('EV*', status);
ev_list_len := stm_list_length (ev_list, status);
IF ev_list_len < page_len
WRITE ('\n List of Events: ');
FOR ev IN ev_list LOOP
  WRITE ('\n', stm_ev_name (ev, status));
END LOOP;
END IF
```

The list is written to the document if there are less than 30 events whose name begins with “EV”.

## stm\_list\_next\_element

**Function type:** ELEMENT

### Description

Returns the next item in the specified list. This function can be applied to a list of any DGL data type.

Note that “next” refers to the item physically located after the current item in the list. The “current” item is determined using the utility function `stm_list_first_element`.

### Syntax

```
stm_list_next_element (list, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>list</code>	In	LIST OF ELEMENT	The list of items. Items in the input list can be of any element type except string. For strings, see the function <code>stm_str_list_next_element</code> .
<code>status</code>	Out	INTEGER	The function status code.

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nil_list`
- ◆ `stm_list_element_does_not_exist`

**Example**

Assume you have a list of states in the order s1, s2, s3 and s4. The list is assigned to the variable `state_list`. You locate the state s1 by calling `stm_list_first_element`. s1 becomes the “current” item. To find the next element in the list, use the following statements:

```
VARIABLE
  LIST OF STATE      state_list;
  STATE              state_id;
  INTEGER             status;
.
.
state_id := stm_list_first_element (state_list, status);
WRITE ('\\n The first state in the list is: ',
      stm_r_st_name (state_id, status));
state_id := stm_list_next_element (state_list, status);
WRITE ('\\n The second state in the list is: ',
      stm_r_st_name (state_id, status));
.
.
```

This function is often used in loop statements.

## stm\_plot\_ext

Function type:

### Description

Generates a plot file with the indicated parameters, such as plot size, output device, and so on. The plot parameters are the same for all the different plot types (statecharts, activity charts, or module charts).

The output is designated for a particular device (one of the output devices defined in Statemate). The destination of the plot output is specified by one of the parameters. If its destination is not specified, the plot is included as part of the output segment file.

The function can generate the hyperlinks in the chart, print a sequence diagram with numbered scenarios, break a sequence diagram across multiple pages and print a sequence diagram with the names of lifelines on every page.

### Syntax

```
stm_plot_ext (id, plot_file_name, width, height, device, data_position,  
title_position, title, actual_h, pages_in_x, pages_in_y, page_index_x,  
page_index_y, headerline_y, options)
```

### Arguments

Argument	Input/Output	Type	Description
id	IN	Stamate element	The ID number of the Statemate chart to be plotted.
plot_file_name	IN	STRING	The name of the file destination to which the plot is written. The operating system pathname conventions are followed. You can specify a full path name to any directory for which you have write access. If you specify a simple file name, the plot is written to your workarea. If you do not specify a value (""), the plot is included as part of the output segment
width	IN	FLOAT	The maximum possible width of the plot (in inches).
height	IN	FLOAT	The maximum possible length of the plot (in inches). If you specify a plot size (width and height parameters) that is larger than the paper size defined for the specific printer, the plot simply uses the maximum allowable height and width defined for that printer.

Argument	Input/Output	Type	Description
device	IN	STRING	Specifies the plotting device. This can be a supported formatting language if the plot is to be handled by a formatting processing system that has its own graphics language. To configure a new plotter or printer, select <b>Utilities &gt; Output Devices</b> from the main Statemate window. Plots created using the Word format in the Output Device dialog are RTF files.
data_position	IN	STRING	The position of the date. This is a string that indicates where to place the plot date. The possible values are as follows: <ul style="list-style-type: none"> <li>• <b>stm_plt_none</b> - The date is not included.</li> <li>• <b>stm_plt_top</b> - The date is placed at the top of the plot.</li> <li>• <b>stm_plt_bottom</b> - The date is placed at the bottom of the plot</li> </ul>
title_position	IN	STRING	The title position. This is a string that indicates where to place the plot title. The possible values are as follows: <ul style="list-style-type: none"> <li>• <b>stm_plt_none</b> - The title is not included.</li> <li>• <b>stm_plt_top</b> - The title is placed at the top of the plot.</li> <li>• <b>stm_plt_bottom</b> - The title is placed at the bottom of the plot.</li> </ul>
title	IN	STRING	Specifies the title to be printed with the plot
actual_h	OUT	FLOAT	Specifies the actual height (in inches) of the plotted output.
pages_in_x	OUT	INTEGER	Specifies how many pages the tool attempted to break the SD into along the x-axis. Note that if pages_in_x==0 and pages_in_y==0, the tool calculates a break pages scheme and assigns these variables so they can be read by the user after the call.
pages_in_y	OUT	INTEGER	Specifies how many pages the tool attempted to break the SD into along the y-axis.
page_index_in_x	IN	INTEGER	Plots the ith page in the x-axis.
page_index_in_y	IN	INTEGER	Plots the ith page in the y-axis.
headerline_y	IN	FLOAT	Defines the vertical coordinate on the page of the header line. This is usually 1.0.
options	IN		A list of strings of the form 'key=value'. See notes below for supported options

### Status Codes

This function may return one of the two following status codes:

- ◆ `stm_success`
- ◆ `stm_can_not_open_file`
- ◆ `stm_id_out_of_range`
- ◆ `stm_not_enough_memory`
- ◆ `stm_id_not_found`
- ◆ `stm_empty_chart`
- ◆ `stm_unknown_plotter`
- ◆ `stm_plot_failure`
- ◆ `stm_unresolved`
- ◆ `stm_illegal_parameter`
- ◆ `stm_plot_illegal_option_key`
- ◆ `stm_plot_illegal_option_val`

## Notes

The following are valid values for the 'options' argument:

- ◆ `stm_plot_option_hyperlink_ext_act_to_graphics`

**a. For External-Activity:**

When this option is 'no', the `External_activity` is hyperlinked to the 'Dictionary' description, if it exists, of the Activity it resolves to. When the 'Dictionary' description is empty, no link is created.

When this option is 'yes', the `External-Activity` is hyperlinked to the chart in which the Activity it resolves to is in. If the resolved Activity is an Off-Page Activity, the link is to the off-page chart. If the resolved Activity is an Instance of generic, the link is to the generic chart. If the `External-Activity` resolves to a higher-level unresolved `External-Activity`, then the link is to the Chart where the Upper most instance of this `External-Activity`. If the `External-Activity` does not resolve to any Activity, no hyperlink is created.

**b. For External-Router:**

When this options is 'no', `External_router` is hyperlinked to the 'Dictionary' description, if it exists, of the Router it resolves to. When the 'Dictionary' description is empty, no link is created, When this option is 'yes', `External-Router` is hyperlinked to the chart that the Router it resolves to is in.

- ◆ `hyperlink_lifeline_to_graphics`

When this option is 'no', Lifelines are hyperlinked to the 'Dictionary' description, if it exists, of the Activity they resolve to. When this option is 'yes', Lifelines are hyperlinked to the chart that the Activity they resolve to are in. If the resolved Activity is an Off-Page Activity, the link is to the off-page chart. If the resolved Activity is an Instance of generic, the link is to the generic chart. If the Lifeline resolves to an unresolved `External- Activity`, no link is created. If the Lifeline does not resolve to any Activity, no hyperlink is created.

## stm\_list\_previous\_element

**Function type:** ELEMENT

### Description

Returns the previous element in the specified list. This function can be applied to a list of any DGL data type.

Note that “previous” refers to the item physically located before the current item in the list. The “current” item is determined using the utility function `stm_list_last_element`.

### Syntax

```
stm_list_previous_element (list, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>list</code>	In	LIST OF ELEMENT	The list of items. Items in the input list can be of any element type except string. For strings, refer to <a href="#">stm_str_list_previous_element</a> .
<code>status</code>	Out	INTEGER	The function status code.

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nil_list`
- ◆ `stm_list_element_does_not_exist`



**Example**

Assume you have a list of states in the order *s1*, *s2*, *s3* and *s4*. The list is assigned to the variable `state_list`. You locate the state *s4* by calling `stm_list_last_element`. *s4* becomes the “current” item. To find the previous element in the list, use the following statements:

```
VARIABLE
  LIST OF STATE  state_list;
  STATE          state_id;
  INTEGER        status;
  .
  .
state_id := stm_list_last_element (state_list, status);
state_id := stm_list_previous_element (state_list,
  status);
WRITE ('\n State of interest is: ',
  stm_r_st_name (state_id, status));
  .
  .
```

This function is often used in loop statements.

## stm\_list\_sort

**Function type:** LIST OF STRING

### Description

Alphabetically sorts the specified list of strings.

### Syntax

```
stm_list_sort (list, status)
```

### Arguments

Argument	Input/Output	Type	Description
list	In	LIST OF ELEMENT	The list of strings to be sorted
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_nil\_list

### Example

Assume you generated a list of attributes for the state `WAIT` using single-element functions. This list is assigned to the variable `attr_list`. To sort the list items and write them to your document, use the following statements:

```
VARIABLE
    STRING          item;
    LIST OF STRING  attr_list, ord_attr_list;
    INTEGER         status;
:
:
WRITE ('\n WAIT's attributes are:');
ord_attr_list := stm_list_sort (attr_list, status);
FOR item IN ord_attr_list LOOP
    WRITE ('\n ', item);
END LOOP;
```

## stm\_list\_sort\_by\_attr\_value

**Function type:** LIST OF ELEMENT

### Description

Sorts the specified list of Rhapsody elements by the value of the given attribute.

Note that the function receives and returns a list of element IDs, not a list of element names.

### Syntax

```
stm_list_sort_by_attr_value (list, attr_name, status)
```

### Arguments

Argument	Input/Output	Type	Description
list	In	LIST OF ELEMENT	The list of Rhapsody element IDs to be sorted.
attr_name	In	STRING	The attribute to use as the sorting key.
status	Out	INTEGER	The function status code. The function returns the status code <code>stm_elements_without_attributes</code> if you apply this function to a list of elements that do not have the specified attribute.

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nil_list`
- ◆ `stm_elements_without_attributes`

### Example

Suppose you want to write a particular list of activities from the database to your document. You extract the activities of interest using single-element and query functions and build a list of such activities. This list is assigned to the variable `act_list`. To sort the activities by the value of an attribute called “code”, your template should contain the following function calls:

```
VARIABLE
  LIST OF ACTIVITY   act_list, ord_act_list;
  ACTIVITY          activ;
  INTEGER           status;
  .
  .
ord_act_list := stm_list_sort_by_attr_value (act_list,
  'code', status);
WRITE ('\n Ordered list of activities:');
FOR activ IN ord_act_list LOOP
  WRITE ('\n', stm_r_ac_name (activ, status));
END LOOP;
```

## stm\_list\_sort\_by\_branches

**Function type:** LIST OF ELEMENT

### Description

Sorts the specified list of hierarchical Rhapsody elements by branches.

Note the following:

- ◆ This function is relevant only for a list of hierarchical elements. If the function is applied to a list of non-hierarchical elements, `status` receives the value `stm_elements_not_hierarchical`.
- ◆ The order in which branches appear in the output is arbitrary. However, the order of states appearing within each branch are ordered from top to bottom.

### Syntax

```
stm_list_sort_by_branches (list, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>list</code>	In	LIST OF ELEMENT	The list of Rhapsody elements
<code>status</code>	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nil_list`
- ◆ `stm_elements_not_hierarchical`

## stm\_list\_sort\_by\_chart

**Function type:** LIST OF ELEMENT

### Description

Alphabetically sorts the input list of named Statemate elements, by the name of the chart to which they belong. The input list consists of Statemate elements.

Note that this function receives and returns a list of element IDs, not a list of element names.

### Syntax

```
stm_list_sort_by_chart (el_list, status)
```

### Arguments

Argument	Input/Output	Type	Description
el_list	In	LIST OF ELEMENT	The list of Rhapsody box elements
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nil_list`

## stm\_list\_sort\_by\_levels

**Function type:** LIST OF ELEMENT

### Description

Sorts a list of hierarchical Rhapsody elements by level.

Note that this function is relevant only for a list of hierarchical elements. If the function is applied to a list of non-hierarchical elements, `status` receives the value `stm_elements_not_hierarchical`.

### Syntax

```
stm_list_sort_by_levels (list, status)
```

### Arguments

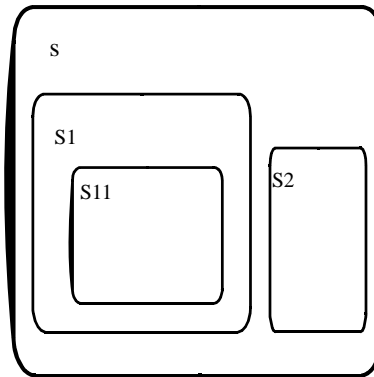
Argument	Input/Output	Type	Description
<code>list</code>	In	LIST OF ELEMENT	The list of Rhapsody box elements
<code>status</code>	Out	INTEGER	The function status code

### Status Codes

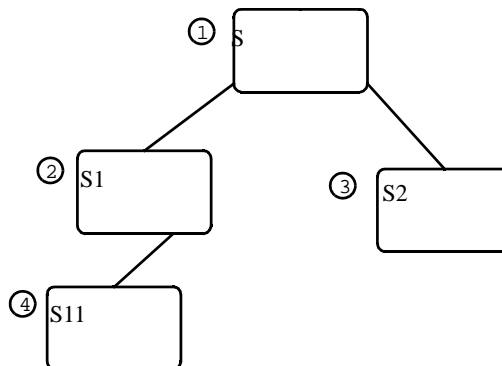
- ◆ `stm_success`
- ◆ `stm_nil_list`
- ◆ `stm_elements_not_hierarchical`

**Example**

Hierarchical elements in a chart can be ordered by *levels*. Refer to the following statechart:



Hierarchically, the states can be drawn as shown as follows:



The set of elements,  $\{s1, s2\}$ , comprise a level. Assume you perform a “`sort_by_level`” function on the states in statechart *s*. The sorted order would be *s*, *s1*, *s2*, *s11*.

**Note**

The order of elements within the same level appear in an arbitrary order in the output. For example, *s2* might appear before *s1* because they are of the same level. However, the order of levels is “top-to-bottom” (here, *s* precedes *s1* or *s2*).



## stm\_list\_sort\_by\_name

**Function type:** LIST OF ELEMENT

### Description

Sorts the specified list of Rhapsody elements alphabetically by name.

Note the following:

- ◆ The function returns the status code `stm_elements_without_name` when you attempt to apply this function to a list that contains unnamed elements.
- ◆ The function receives and returns a list of element IDs, *not* a list of element names.

### Syntax

```
stm_list_sort_by_name (list, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>list</code>	In	LIST OF ELEMENT	The list of Rhapsody elements to be sorted. This input lists consists of element IDs.
<code>status</code>	Out	INTEGER	The function status code.

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nil_list`
- ◆ `stm_elements_without_name`

### Example

Suppose you want to write a particular list of activities from the database to your document. You extract the activities of interest using single-element and query functions and build a list of such activities. This list is assigned to the variable `act_list`. To alphabetically sort the activities by their names, your template should contain the following statements:

```
VARIABLE
  LIST OF ACTIVITY  act_list, ord_act_list;
  ACTIVITY         activ;
  INTEGER          status;
  .
  .
ord_act_list := stm_list_sort_by_name (act_list, status);
WRITE ('\n Ordered list of activities:');
FOR activ IN ord_act_list LOOP
  WRITE ('\n', stm_r_ac_name (activ, status));
END LOOP;
```

## stm\_list\_sort\_by\_synonym

**Function type:** LIST OF ELEMENT

### Description

Sorts the specified list of Rhapsody elements alphabetically by their synonyms.

Note the following:

- ◆ The function returns the status code `stm_elements_without_name` when you attempt to apply this function to a list that contains unnamed elements.
- ◆ The function receives and returns a list of element IDs, *not* a list of element names.

### Syntax

```
stm_list_sort_by_synonym (list, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>list</code>	In	LIST OF ELEMENT	The list of Rhapsody elements to be sorted. This input lists consists of element IDs.
<code>status</code>	Out	INTEGER	The function status code.

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nil_list`
- ◆ `stm_elements_without_synonym`

### Example

Suppose you want to write a particular list of activities from the database to your document. You extract the activities of interest using single-element and query functions and build a list of such activities. This list is assigned to the variable `act_list`. To alphabetically sort the activities by their synonyms, your template should contain the following statements:

```
VARIABLE
  LIST OF ACTIVITY      act_list, ord_act_list;
  MODULE               act;
  INTEGER              status;
  .
  .
ord_act_list = stm_list_sort_by_synonym (act_list,
  status);
WRITE ('\n Ordered list of activities:');
FOR act IN ord_act_list LOOP
  WRITE ('\n', stm_r_ac_synonym (act, status), '\t',
  stm_r_ac_name (act, status));
END LOOP;
```

## stm\_list\_sort\_by\_type

**Function type:** LIST OF ELEMENT

### Description

Sorts the specified list of Rhapsody elements by type.

Note that the function receives and returns a list of element IDs, *not* a list of element names.

### Syntax

```
stm_list_sort_by_type (el_list, status)
```

### Arguments

Argument	Input/Output	Type	Description
el_list	In	LIST OF ELEMENT	The list of Rhapsody element IDs to be sorted
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nil_list`

## stm\_multiline\_to\_one

**Function type:** STRING

### Description

Converts the specified multiline string (with new lines) to a one-line string (without the new lines).

### Syntax

```
stm_multiline_to_one (string)
```

### Arguments

Argument	Input/Output	Type	Description
string	In	STRING	The multiline string

## stm\_multiline\_to\_strings

**Function type:** LIST OF STRING

### Description

Converts the specified multiline expression to a list of strings.

### Syntax

```
stm_multiline_to_strings (string)
```

### Arguments

Argument	Input/Output	Type	Description
string	In	STRING	The multiline expression

## stm\_plot

**Function type:** INTEGER

### Description

Generates a plot file with the indicated parameters, such as plot size, output device, and so on. The plot parameters are the same for all the different plot types (statecharts, activity charts, or module charts).

The output is designated for a particular device (one of the output devices defined in Statemate). The destination of the plot output is specified by one of the parameters. If its destination is not specified, the plot is included as part of the output segment file.

### Syntax

```
stm_plot (id, plot_file, width, height, with_labels, with_names, with_notes,  
device, date_position, title_position, title, do_rotate, with_file_header,  
actual_height)
```

### Example

Consider a template that contains the following statements:

```
VARIABLE  
  CHART      ch_id;  
  INTEGER    status;  
  FLOAT      real_ht;  
.  
ch_id:= stm_r_ch ('XL25', status);  
stm_plot (ch_id, '/sam/p_xl25', 5.0, 7.0, true, true,  
  false, 'POSTSCRIPT', stm_plt_top,  
  'SystemXL25', true, true, real_ht);  
.
```

## stm\_plot\_hyper\_exp

**Function type:** INTEGER

### Description

Generates the hyperlinks in a sequence diagram.

### Syntax

```
stm_plot_hyper_exp (id, plot_file, width, height, with_labels, with_names,  
with_notes, with_hyperlink, device, date_position, title_position, title,  
do_rotate, with_file_header, actual_height, with_breakpages, pages_in_x,  
pages_in_y, page_index_in_x, page_index_in_y, with_hyperlink_exp)
```

### Arguments

Argument	Input/Output	Type	Description
id	In	StateMate element	The ID number of the StateMate chart to be plotted.
plot_file	In	STRING	The name of the file destination to which the plot is written. The operating system path name conventions are followed. You can specify a full path name to any directory for which you have write access.  If you specify a simple file name, the plot is written to your workarea. If you do not specify a value (""), the plot is included as part of the output segment file.
width	In	FLOAT	The maximum possible width of the plot (in inches).
height	In	FLOAT	The maximum possible length of the plot (in inches).  If you specify a plot size (width and height parameters) that is larger than the paper size defined for the specific printer, the plot simply uses the maximum allowable height and width defined for that printer.
with_labels	In	BOOLEAN	Determines whether labels are plotted (TRUE) or not (FALSE).
with_names	In	BOOLEAN	Determines whether names are plotted (TRUE) or not (FALSE).
with_notes	In	BOOLEAN	Determines whether notes are plotted (TRUE) or not (FALSE).
with_hyperlink	In	BOOLEAN	Specifies whether to generate hyperlinks for lifelines and referenced sequence diagrams (TRUE).



Argument	Input/Output	Type	Description
device	In	STRING	Specifies the plotting device. This can be a supported formatting language if the plot is to be handled by a formatting processing system that has its own graphics language.  To configure a new plotter or printer, select <b>Utilities &gt; Output Devices</b> from the main Statemate window.  Plots created using the Word format in the Output Devie dialog are RTF files.
date_position	In	STRING	The position of the date.  This is a string that indicates where to place the plot date. The possible values are as follows: <ul style="list-style-type: none"> <li>• <code>stm_plt_none</code>—The date is not included.</li> <li>• <code>stm_plt_top</code>—The date is placed at the top of the plot.</li> <li>• <code>stm_plt_bottom</code>—The date is placed at the bottom of the plot.</li> </ul>
title_position	In	STRING	The title position.  This is a string that indicates where to place the plot title. The possible values are as follows: <ul style="list-style-type: none"> <li>• <code>stm_plt_none</code>—The title is not included.</li> <li>• <code>stm_plt_top</code>—The title is placed at the top of the plot.</li> <li>• <code>stm_plt_bottom</code>—The title is placed at the bottom of the plot.</li> </ul>
title	In	STRING	Specifies the title to be printed with the plot.
do_rotate	In	BOOLEAN	Determines whether the orientation of the plot is landscape (TRUE) or portrait (FALSE).
with_file_header	In	BOOLEAN	Indicates whether a header is added to the file (TRUE). Use this option if you do not want the plot as part of the document (FALSE).
actual_height	Out	FLOAT	Specifies the actual height (in inches) of the plotted output.
with_breakpages	In	BOOLEAN	Specifies whether to break the SD across multiple pages (true).
pages_in_x	Out	INTEGER	Specifies how many pages the tool attempted to break the SD into along the x-axis.  Note that if <code>pages_in_x==0</code> and <code>pages_in_y==0</code> , the tool calculates a break pages scheme and assigns these variables so they can be read by the user after the call.

## Utility Functions

---

Argument	Input/Output	Type	Description
pages_in_y	Out	INTEGER	Specifies how many pages the tool attempted to break the SD into along the y-axis.
page_index_in_x	In	INTEGER	Plots the <i>i</i> th page in the x-axis.
page_index_in_y	In	INTEGER	Plots the <i>i</i> th page in the y-axis.
with_hyperlink_exp	In	BOOLEAN	Specifies whether to generate hyperlinks for message labels (true).

### Status Codes

- ◆ `stm_success`
- ◆ `stm_can_not_open_file`
- ◆ `stm_id_out_of_range`
- ◆ `stm_not_enough_memory`
- ◆ `stm_id_not_found`
- ◆ `stm_empty_chart`
- ◆ `stm_unknown_plotter`
- ◆ `stm_plot_failure`
- ◆ `stm_unresolved`
- ◆ `stm_illegal_parameter`

## stm\_plot\_with\_autonumber

**Function type:** Integer

### Description

Prints a sequence diagram with numbered scenarios.

### Syntax

```
stm_plot_with_autonumber (id, plot_file, width, height, with_labels,
with_names, with_notes, with_hyperlink, plot_type, date_position,
title_position, title, do_rotate, with_file_header, actual_height,
with_breakpages, pages_in_x, pages_in_y, page_index_in_x, page_index_in_y,
with_hyperlink_exp, with_headerline, headerline_y, with_autonumber)
```

### Arguments

Argument	Input/Output	Type	Description
id	In	State element	The ID number of the Statechart to be plotted.
plot_file	In	STRING	The name of the file destination to which the plot is written. The operating system path name conventions are followed. You can specify a full path name to any directory for which you have write access.  If you specify a simple file name, the plot is written to your workarea. If you do not specify a value (""), the plot is included as part of the output segment file.
width	In	FLOAT	The maximum possible width of the plot (in inches).
height	In	FLOAT	The maximum possible length of the plot (in inches).  If you specify a plot size (width and height parameters) that is larger than the paper size defined for the specific printer, the plot simply uses the maximum allowable height and width defined for that printer.
with_labels	In	BOOLEAN	Determines whether labels are plotted (true) or not (false).
with_names	In	BOOLEAN	Determines whether names are plotted (true) or not (false).
with_notes	In	BOOLEAN	Determines whether notes are plotted (true) or not (false).
with_hyperlink	In	BOOLEAN	Specifies whether to generate hyperlinks for lifelines and referenced sequence diagrams (true).

Argument	Input/Output	Type	Description
plot_type	In	STRING	Specifies the plotting device. This can be a supported formatting language if the plot is to be handled by a formatting processing system that has its own graphics language.  To configure a new plotter or printer, select <b>Utilities &gt; Output Devices</b> from the main Statemate window.  Plots created using the Word format in the Output Device dialog are RTF files.
date_position	In	STRING	The date position.  This is a string that indicates where to place the plot date. The possible values are as follows: <ul style="list-style-type: none"> <li>• <code>stm_plt_none</code>—The date is not included.</li> <li>• <code>stm_plt_top</code>—The date is placed at the top of the plot.</li> <li>• <code>stm_plt_bottom</code>—The date is placed at the bottom of the plot.</li> </ul>
title_position	In	STRING	The title position.  This is a string that indicates where to place the plot title. The possible values are as follows: <ul style="list-style-type: none"> <li>• <code>stm_plt_none</code>—The title is not included.</li> <li>• <code>stm_plt_top</code>—The title is placed at the top of the plot.</li> <li>• <code>stm_plt_bottom</code>—The title is placed at the bottom of the plot.</li> </ul>
title	In	STRING	Specifies the title to be printed with the plot.
do_rotate	In	BOOLEAN	Determines whether the orientation of the plot is landscape (true) or portrait (false).
with_file_header	In	BOOLEAN	Indicates whether a header is added to the file (true). Use this option if you do not want the plot as part of the document (false).
actual_height	Out	FLOAT	Specifies the actual height (in inches) of the plotted output.
with_breakpages	In	BOOLEAN	Specifies whether to break the SD across multiple pages (true).
pages_in_x	Out	INTEGER	Specifies how many pages the tool attempted to break the SD into along the x-axis.  Note that if <code>pages_in_x==0</code> and <code>pages_in_y==0</code> , the tool calculates a break pages scheme and assigns these variables so they can be read by the user after the call.
pages_in_y	Out	INTEGER	Specifies how many pages the tool attempted to break the SD into along the y-axis.
page_index_in_x	In	INTEGER	Plots the <i>i</i> th page in the x-axis.

Argument	Input/ Output	Type	Description
page_index_in_y	In	INTEGER	Plots the <i>i</i> th page in the <i>y</i> -axis.
with_hyperlink_exp	In	BOOLEAN	Specifies whether to generate hyperlinks for message labels (true).
with_headerline	In	BOOLEAN	Specifies whether to print the names of the lifelines on every page (true).
headerline_y	In	FLOAT	Defines the vertical coordinate on the page of the headerline. This is usually 1.0.
with_autonumber	In	BOOLEAN	Specifies whether to print the SD scenario numbers (true).

### Status Codes

- ◆ stm\_success
- ◆ stm\_can\_not\_open\_file
- ◆ stm\_id\_out\_of\_range
- ◆ stm\_not\_enough\_memory
- ◆ stm\_id\_not\_found
- ◆ stm\_empty\_chart
- ◆ stm\_unknown\_plotter
- ◆ stm\_plot\_failure
- ◆ stm\_unresolved
- ◆ stm\_illegal\_parameter

## stm\_plot\_with\_break

**Function type:** INTEGER

### Description

Breaks a sequence diagram across multiple pages.

### Syntax

```
stm_plot_with_break (id, plot_file, width, height, with_labels, with_names,  
with_notes, with_hyperlink, plot_type, date_position, title_position, title,  
do_rotate, with_file_header, actual_height, with_breakpages, pages_in_x,  
pages_in_y, page_index_in_x, page_index_in_y)
```

### Arguments

Argument	Input/ Output	Type	Description
id	In	Statestate element	The ID number of the Statestate chart to be plotted.
plot_file	In	STRING	The name of the file destination to which the plot is written. The operating system path name conventions are followed. You can specify a full path name to any directory for which you have write access.  If you specify a simple file name, the plot is written to your workarea. If you do not specify a value (""), the plot is included as part of the output segment file.
width	In	FLOAT	The maximum possible width of the plot (in inches).
height	In	FLOAT	The maximum possible length of the plot (in inches).  If you specify a plot size (width and height parameters) that is larger than the paper size defined for the specific printer, the plot simply uses the maximum allowable height and width defined for that printer.
with_labels	In	BOOLEAN	Determines whether labels are plotted (TRUE) or not (FALSE).
with_names	In	BOOLEAN	Determines whether names are plotted (TRUE) or not (FALSE).
with_notes	In	BOOLEAN	Determines whether notes are plotted (TRUE) or not (FALSE).
with_hyperlink	In	BOOLEAN	Specifies whether to generate hyperlinks for lifelines and referenced sequence diagrams (TRUE).

Argument	Input/Output	Type	Description
plot_type	In	STRING	Specifies the plot type. This can be a supported formatting language if the plot is to be handled by a formatting processing system that has its own graphics language.  To configure a new plotter or printer, select <b>Utilities &gt; Output Devices</b> from the main <b>Statemate</b> window. Plots created using the Word format in the Output Device dialog are RTF files.
date_position	In	STRING	The date position. This is a string that indicates where to place the plot date. The possible values are as follows: <ul style="list-style-type: none"> <li>• <code>stm_plt_none</code>—The date is not included.</li> <li>• <code>stm_plt_top</code>—The date is placed at the top of the plot.</li> <li>• <code>stm_plt_bottom</code>—The date is placed at the bottom of the plot.</li> </ul>
title_position	In	STRING	The title position. This is a string that indicates where to place the plot title. The possible values are as follows: <ul style="list-style-type: none"> <li>• <code>stm_plt_none</code>—The title is not included.</li> <li>• <code>stm_plt_top</code>—The title is placed at the top of the plot.</li> <li>• <code>stm_plt_bottom</code>—The title is placed at the bottom of the plot.</li> </ul>
title	In	STRING	Specifies the title to be printed with the plot.
do_rotate	In	BOOLEAN	Determines whether the orientation of the plot is landscape (TRUE) or portrait (FALSE).
with_file_header	In	BOOLEAN	Indicates whether a header is added to the file (TRUE). Use this option if you do not want the plot as part of the document (FALSE).
actual_height	Out	FLOAT	Specifies the actual height (in inches) of the plotted output.
with_breakpages	In	BOOLEAN	Specifies whether to break the SD across multiple pages (TRUE).
pages_in_x	Out	INTEGER	Specifies how many pages the tool attempted to break the SD into along the x-axis.  Note that if <code>pages_in_x==0</code> and <code>pages_in_y==0</code> , the tool calculates a break pages scheme and assigns these variables so they can be read by the user after the call.

## Utility Functions

---

Argument	Input/ Output	Type	Description
pages_in_y	Out	INTEGER	Specifies how many pages the tool attempted to break the SD into along the y-axis.
page_index_in_x	In	INTEGER	Plots the <i>i</i> th page in the x-axis.
page_index_in_y	In	INTEGER	Plots the <i>i</i> th page in the y-axis.

### Status Codes

- ◆ `stm_success`
- ◆ `stm_can_not_open_file`
- ◆ `stm_id_out_of_range`
- ◆ `stm_not_enough_memory`
- ◆ `stm_id_not_found`
- ◆ `stm_empty_chart`
- ◆ `stm_unknown_plotter`
- ◆ `stm_plot_failure`
- ◆ `stm_unresolved`
- ◆ `stm_illegal_parameter`



**Notes**

Function parameters are as follows:

```
boolean with_hyperlink (IN) /* generate hyperlinks */
boolean with_breakpages (IN) /* enable break pages */
integer pages_in_x (OUT) /* try to break to # of pages in x axis */
integer pages_in_y (OUT) /* try to break to # of pages in y axis */
```

If `pages_in_x == 0` and `pages_in_y==0`, the tool calculates a break pages scheme and assigns these variables so they can be read by the user after the call.

```
integer page_index_in_x (IN) /*plot the ith page in x axis */
integer page_index_in_y (IN) /*plot the ith page in y axis */
```

Call the function `STM_PLOT_SET_DATA()` before plotting a sequence diagram using `STM_PLOT_WITH_BREAK`. Call the function `STM_PLOT_RESET_DATA()` after finishing the sequence diagram multiple pages plot.

## stm\_plot\_with\_headerline

**Function type:** Integer

### Description

Prints a sequence diagram with the names of lifelines on every page.

### Syntax

```
stm_plot_with_headerline (id, plot_file, width, height, with_labels,  
with_names, with_notes, with_hyperlink, plot_type, date_position,  
title_position, title, do_rotate, with_file_header, actual_height,  
with_breakpages, pages_in_x, pages_in_y, page_index_in_x, page_index_in_y,  
with_hyperlink_exp, with_headerline, headerline_y,)
```

### Arguments

Argument	Input/Output	Type	Description
id	In	Statmate element	The ID number of the Statmate chart to be plotted.
plot_file	In	STRING	The name of the file destination to which the plot is written. The operating system path name conventions are followed. You can specify a full path name to any directory for which you have write access.  If you specify a simple file name, the plot is written to your workarea. If you do not specify a value (""), the plot is included as part of the output segment file.
width	In	FLOAT	The maximum possible width of the plot (in inches).
height	In	FLOAT	The maximum possible length of the plot (in inches).  If you specify a plot size (width and height parameters) that is larger than the paper size defined for the specific printer, the plot simply uses the maximum allowable height and width defined for that printer.
with_labels	In	BOOLEAN	Determines whether labels are plotted (TRUE) or not (FALSE).
with_names	In	BOOLEAN	Determines whether names are plotted (TRUE) or not (FALSE).
with_notes	In	BOOLEAN	Determines whether notes are plotted (TRUE) or not (FALSE).
with_hyperlink	In	BOOLEAN	Specifies whether to generate hyperlinks for lifelines and referenced sequence diagrams (TRUE).

Argument	Input/Output	Type	Description
plot_type	In	STRING	Specifies the plot type. This can be a supported formatting language if the plot is to be handled by a formatting processing system that has its own graphics language. To configure a new plotter or printer, select <b>Utilities &gt; Output Devices</b> from the main Statemate window. Plots created using the Word format in the Output Device dialog are RTF files.
date_position	In	STRING	The date position. This is a string that indicates where to place the plot date. The possible values are as follows: <ul style="list-style-type: none"> <li>• <code>stm_plt_none</code>—The date is not included.</li> <li>• <code>stm_plt_top</code>—The date is placed at the top of the plot.</li> <li>• <code>stm_plt_bottom</code>—The date is placed at the bottom of the plot.</li> </ul>
title_position	In	STRING	The title position. This is a string that indicates where to place the plot title. The possible values are as follows: <ul style="list-style-type: none"> <li>• <code>stm_plt_none</code>—The title is not included.</li> <li>• <code>stm_plt_top</code>—The title is placed at the top of the plot.</li> <li>• <code>stm_plt_bottom</code>—The title is placed at the bottom of the plot.</li> </ul>
title	In	STRING	Specifies the title to be printed with the plot.
do_rotate	In	BOOLEAN	Determines whether the orientation of the plot is landscape (TRUE) or portrait (FALSE).
with_file_header	In	BOOLEAN	Indicates whether a header is added to the file (TRUE). Use this option if you do not want the plot as part of the document (FALSE).
actual_height	Out	FLOAT	Specifies the actual height (in inches) of the plotted output.
with_breakpages	In	BOOLEAN	Specifies whether to break the SD across multiple pages (TRUE).
pages_in_x	Out	INTEGER	Specifies how many pages the tool attempted to break the SD into along the x-axis. Note that if <code>pages_in_x==0</code> and <code>pages_in_y==0</code> , the tool calculates a break pages scheme and assigns these variables so they can be read by the user after the call.
pages_in_y	Out	INTEGER	Specifies how many pages the tool attempted to break the SD into along the y-axis.
page_index_in_x	In	INTEGER	Plots the <i>i</i> th page in the x-axis.
page_index_in_y	In	INTEGER	Plots the <i>i</i> th page in the y-axis.

## Utility Functions

---

Argument	Input/Output	Type	Description
<code>with_hyperlink_exp</code>	In	BOOLEAN	Specifies whether to generate hyperlinks for message labels (TRUE).
<code>with_headerline</code>	In	BOOLEAN	Specifies whether to print the names of the lifelines on every page (TRUE).
<code>headerline_y</code>	In	FLOAT	Defines the vertical coordinate on the page of the headerline. This is usually 1.0.

### Status Codes

- ◆ `stm_success`
- ◆ `stm_can_not_open_file`
- ◆ `stm_id_out_of_range`
- ◆ `stm_not_enough_memory`
- ◆ `stm_id_not_found`
- ◆ `stm_empty_chart`
- ◆ `stm_unknown_plotter`
- ◆ `stm_plot_failure`
- ◆ `stm_unresolved`
- ◆ `stm_illegal_parameter`

## stm\_replace\_string

**Function type:** STRING

### Description

Replaces the specified pattern with a new pattern within a string.

### Syntax

```
stm_replace_string (orig_str, find_str, repl_str)
```

### Arguments

Argument	Input/Output	Type	Description
orig_str	In	STRING	The string within which to search and replace
find_str	In	STRING	The pattern to search for
repl_str	In	STRING	The replacement pattern

## stm\_replace\_word

**Function type:** `STRING`

### Description

Replaces the specified word with a new word within a given string. A word has whitespace before and after it, preventing substitution where there is a partial match for the `find_str` parameter.

### Syntax

```
stm_replace_word (orig_str, find_str, repl_str, delimiters, format,  
replace_in_comment, is_identifier)
```

### Arguments

Argument	Input/ Output	Type	Description
<code>orig_str</code>	In	<code>STRING</code>	The string within which to search and replace
<code>find_str</code>	In	<code>STRING</code>	The word to search for
<code>repl_str</code>	In	<code>STRING</code>	The replacement word
<code>delimiters</code>	In	<code>STRING</code>	A delimiter to add whitespace
<code>format</code>	In	<code>STRING</code>	FrameMaker or Word
<code>replace_in_comment</code>	In	<code>BOOLEAN</code>	Specifies whether to replace the word in commented regions of the action language or user code
<code>is_identifier</code>	In	<code>BOOLEAN</code>	Specifies whether <code>find_str</code> is the name of a Rhapsody element

## stm\_set\_rpt\_formatter

**Function type:** INTEGER

### Description

Sets the formatter to use for Rhapsody reports. The possible values are FRAMEMAKER and WORD.

### Syntax

```
stm_set_rpt_formatter (formatter)
```

### Arguments

Argument	Input/Output	Type	Description
formatter	In	INTEGER	The formatter to use for reports

## stm\_string\_retain

### Description:

When used in a DOC template, `STM_DISPOSE_MEMORY()` frees temporary memory used by the DOC system. If you want to avoid freeing a specific string, for example, general strings used in the template, call `STM_STRING_RETAIN(<str>)` on that string before calling `STM_DISPOSE_MEMORY()`.

### Syntax

```
stm_string_retain(string)
```

### Arguments

Argument	Input/Output	Type	Description
string	In	STRING	The string to be retained in memory after performing <code>STM_DISPOSE_MEMORY()</code>

### Status Codes

None

### Example

```
STRING str;  
STM_STRING_RETAIN(str)
```



## stm\_str\_list\_first\_element

**Function type:** STRING

### Description

Returns the first item in the specified list of strings.

### Syntax

```
stm_str_list_first_element (list, status)
```

### Arguments

Argument	Input/Output	Type	Description
list	In	LIST OF STRING	The list of strings
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nil_list`
- ◆ `stm_list_element_does_not_exist`

### Example

Assume you have a list of strings `s1`, `s2`, `s3`, and `s4` assigned to the variable `str_list`. You locate the string `s1` by calling `stm_str_list_first_element`. `s1` becomes the current item. Your template should contain the following statements:

```
VARIABLE
    LIST OF STRING    str_list;
    STRING            str;
    INTEGER           status;
.
str = stm_str_list_first_element (str_list, status);
WRITE ('\n The first string in the list is: ', str);
str = stm_str_list_next_element (str_list, status);
WRITE ('\n The second string in the list is: ', str);
.
.
```

## stm\_str\_list\_last\_element

**Function type:** `STRING`

### Description

Returns the element ID of the last item appearing in the specified list of strings.

Note the following:

- ◆ The first position in the string is 0.
- ◆ If the function fails, it returns an empty string.

### Syntax

```
stm_str_list_last_element (list, status)
```

### Arguments

Argument	Input/Output	Type	Description
string	In	LIST OF STRING	The list of strings
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nil_list`
- ◆ `stm_list_element_does_not_exist`

### Example

Assume there is a list of strings (`s1`, `s2`, `s3`, and `s4`) assigned to the variable `str_list`. You locate the string `s4` by calling `stm_str_list_last_element`. `s4` becomes the “current” item. Your template should contain the following statements:

```
VARIABLE
  LIST OF STRING  str_list;
  STRING          str;
  INTEGER         status;
  .
  .
str := stm_str_list_last_element (str_list, status);
WRITE ('\n The last string in the list is: ', str);
str := stm_str_list_previous_element (str_list, status);
WRITE ('\n The third string in the list is: ', str);
  .
  .
```

## stm\_str\_list\_length

**Function type:** INTEGER

### Description

Returns the number of items in the specified list of strings.

### Syntax

```
stm_str_list_length (list, status)
```

### Arguments

Argument	Input/Output	Type	Description
list	In	LIST OF STRING	The list of strings whose length you want
status	Out	INTEGER	The function status code

### Status Codes

- ◆ stm\_success
- ◆ stm\_nil\_list

### Example

Assume you have extracted all the static reactions in state `ST1` from the database. Before writing the list of strings to your document, you want to make sure that it will not span more than 30 lines of text (one page length). Your template should contain the following statements:

```
VARIABLE
  STATE          st;
  STRING         str;
  LIST OF STRING str_list;
  INTEGER        str_list_len, status;
  CONSTANT INTEGER page_len := 30;
  .
  .
st:=stm_r_st ('ST1',status);
str_list:=stm_r_st_static_reactions (st, status);
str_list_len := stm_str_list_length (st, status);
IF str_list_len < page_len
  WRITE ('\n List of Reactions: ');
FOR str IN str_list LOOP
  WRITE ('\n', str);
END LOOP;
END IF
```

## stm\_str\_list\_next\_element

**Function type:** STRING

### Description

Returns the next item in the specified list of strings.

Note that “next” refers to the item physically located after the current item in the list of strings. The “current” item is determined using the utility function `stm_str_list_first_element`.

### Syntax

```
stm_str_list_next_element (list, status)
```

### Arguments

Argument	Input/Output	Type	Description
list	In	LIST OF STRING	The list of strings
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nil_list`
- ◆ `stm_list_element_does_not_exist`

### Example

Assume we have a list of strings `s1`, `s2`, `s3`, and `s4` assigned to the variable `str_list`. You locate the string `s1` by calling `stm_str_list_first_element`. `s1` becomes the current item. To find the next element in the list, use the following statements:

```
VARIABLE
    LIST OF STRING    str_list;
    STRING            str;
    INTEGER           status;
.
.
str = stm_str_list_first_element (str_list, status);
WRITE ('\n The first string in the list is: ', str);
str := stm_str_list_next_element (str_list, status);
WRITE ('\n The second string in the list is: ', str);
.
.
```

This function is often used in loop statements.

## stm\_str\_list\_previous\_element

**Function type:** `STRING`

### Description

Returns the previous item in the specified list of strings.

Note that “previous” refers to the item physically located before the current item in the list of strings. The “current” item is determined using the utility function `stm_str_list_last_element`.

### Syntax

```
stm_str_list_previous_element (list, status)
```

### Arguments

Argument	Input/Output	Type	Description
<code>list</code>	In	<code>LIST OF STRING</code>	The list of strings
<code>status</code>	Out	<code>INTEGER</code>	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nil_list`
- ◆ `stm_list_element_does_not_exist`

### Example

Assume you have a list of strings `s1`, `s2`, `s3`, and `s4` assigned to the variable `str_list`. You locate the string `s4` by calling `stm_str_list_last_element`. `s4` becomes the current item. Your template should contain the following statements:

```
VARIABLE
  LIST OF STRING  str_list;
  STRING          str;
  INTEGER         status;
  .
  .
str := stm_str_list_last_element (str_list, status);
WRITE ('\n The last string in the list is: ', str);
str := stm_str_list_previous_element (str_list, status);
WRITE ('\n The third string in the list is: ', str);
  .
  .
```

This function is often used in loop statements.

## stm\_str\_list\_to\_str

**Function type:** `STRING`

### Description

Returns the string representation of the specified list of strings.

### Syntax

```
stm_str_list_to_str (s_list)
```

### Arguments

Argument	Input/Output	Type	Description
<code>s_list</code>	In	LIST OF STRING	The list of strings to be converted

### Status Codes

- ◆ `stm_success`

## stm\_str\_to\_list

**Function type:** LIST OF STRING

### Description

Converts the specified string to a list of strings.

### Syntax

```
stm_str_to_list (s, delimiter)
```

### Arguments

Argument	Input/Output	Type	Description
s	In	STRING	The string to convert
delimiter	In	STRING	A delimiter for whitespace



## stm\_string\_extract

**Function type:** STRING

### Description

Extracts a portion of the specified string. The extracted portion starts at the `index` position and its length is `len`.

### Syntax

```
stm_string_extract (string, index, len, status)
```

### Arguments

Argument	Input/Output	Type	Description
string	In	STRING	The entire string
index	In	INTEGER	The starting point of the extraction
len	In	INTEGER	The number of characters to extract from the string
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_null_string`
- ◆ `stm_illegal_index`
- ◆ `stm_illegal_len`

### Example

```
stm_string_extract ('ABCDE',0,2,status) = 'AB'  
stm_string_extract ('ABCDE',3,2,status) = 'DE'  
stm_string_extract ('ABCDE',0,7,status) = ''  
    and status = stm_illegal_len
```

## stm\_string\_free

**Function type:** INTEGER

### Description

Frees the memory used by the specified string.

### Syntax

```
stm_string_free (s)
```

### Arguments

Argument	Input/Output	Type	Description
s	In	STRING	The string whose memory you want to free

## stm\_string\_retain

### Description:

When used in a DOC template, `STM_DISPOSE_MEMORY()` frees temporary memory used by the DOC system. If you want to avoid freeing a specific string, for example, general strings used in the template, call `STM_STRING_RETAIN(<str>)` on that string before calling `STM_DISPOSE_MEMORY()`.

### Syntax

```
stm_string_retain(string)
```

### Arguments

Argument	Input/Output	Type	Description
string	In	STRING	The string to be retained in memory after performing <code>STM_DISPOSE_MEMORY()</code>

### Status Codes

None

### Example

```
STRING str;  
STM_STRING_RETAIN(str0
```

## stm\_string\_to\_int

**Function type:** INTEGER

### Description

Returns an integer value of a decimal string representation of a number.

### Syntax

```
stm_string_to_int (string)
```

### Arguments

Argument	Input/Output	Type	Description
string	In	STRING	The string to be converted

### Status Codes

- ◆ `stm_success`

## stm\_strlen

**Function type:** INTEGER

### Description

Returns the length of the specified string.

### Syntax

```
stm_strlen (string)
```

### Arguments

Argument	Input/Output	Type	Description
string	In	STRING	The string whose length you want

### Status Codes

- ◆ `stm_success`

### Example

```
len := stm_strlen ('ABC');  
len is 3.
```

## stm\_trigger\_of\_reaction

**Function type:** STRING

### Description

Returns the trigger part of a reaction (label of transition or static reaction). The syntax of the reaction is `trigger/action`.

Note the following:

- ◆ The reaction is achieved by the following single-element functions:
  - `stm_r_st_reactions`
  - `stm_r_tr_labels`
- ◆ The function returns an empty string when the trigger is missing.

### Syntax

```
stm_trigger_of_reaction (reaction, status)
```

### Arguments

Argument	Input/Output	Type	Description
reaction	In	STRING	The reaction to decompose
status	Out	STRING	The function status code

### Status Codes

- ◆ `stm_success`

### Example

To list all events that have influence on `s1`, which has several static reactions, use the following statements in your template:

```
Variable
STATE          st_id;
INTEGER        status;
LIST OF STRING reactions;
STRING         rct;
st_id:=stm_r_st('S1',status);
reactions:=stm_r_st_reactions (st_id, status);
WRITE ('\n Triggers of reaction is S1:');
FOR rct IN reactions LOOP
  WRITE ('\n', stm_trigger_of_reaction (rct, status));
END FOR;
```

# Project Management

---

This appendix describes special project management functions. For each function, the following information is provided:

- ◆ Description
- ◆ Syntax
- ◆ Arguments
- ◆ Status codes

The following table lists the project management functions.

Function	Description
<a href="#"><u>stm_r_pm_member_workareas</u></a>	Returns the workareas of the specified user
<a href="#"><u>stm_r_pm_operator_projects</u></a>	Returns a list of all the projects in which the specified user is a member
<a href="#"><u>stm_r_pm_project_databank</u></a>	Returns the databank name of the specified project in the project management database
<a href="#"><u>stm_r_pm_project_manager</u></a>	Returns the manager of the specified project in the project management database
<a href="#"><u>stm_r_pm_project_members</u></a>	Returns a list of all the members of the specified project in the project management database
<a href="#"><u>stm_r_pm_projects</u></a>	Returns a list of all the projects in the project management database

## stm\_r\_pm\_member\_workareas

**Function type:** LIST OF STRING

### Description

Returns the workareas of the specified user.

### Syntax

```
stm_r_pm_member_workareas (o_name, p_name, status)
```

### Arguments

Argument	Input/Output	Type	Description
o_name	In	STRING	The name of the user
p_name	In	STRING	The name of the project
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nonexistent_project`
- ◆ `stm_not_member_of_project`



## stm\_r\_pm\_operator\_projects

**Function type:** LIST OF STRING

### Description

Returns a list of all the projects in which the specified user is a member.

### Syntax

```
stm_r_pm_operator_projects (oname, status)
```

### Arguments

Argument	Input/Output	Type	Description
oname	In	STRING	The name of the user
status	Out	STRING	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_no_projects`

## stm\_r\_pm\_project\_databank

**Function type:** `STRING`

### Description

Returns the databank name of the specified project in the project management database.

### Syntax

```
stm_r_pm_project_databank (pname, status)
```

### Arguments

Argument	Input/Output	Type	Description
pname	In	STRING	The name of the project
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nonexistent_project`

## stm\_r\_pm\_project\_manager

**Function type:** `STRING`

### Description

Returns the manager of the specified project in the project management database.

### Syntax

```
stm_r_pm_project_manager (pname, status)
```

### Arguments

Argument	Input/Output	Type	Description
pname	In	STRING	The name of the project
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nonexistent_project`

## stm\_r\_pm\_project\_members

**Function type:** LIST OF STRING

### Description

Returns a list of all the members of the specified project in the project management database.

### Syntax

```
stm_r_pm_project_members (pname, status)
```

### Arguments

Argument	Input/Output	Type	Description
pname	In	STRING	The name of the project
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_nonexistent_project`

## stm\_r\_pm\_projects

**Function type:** LIST OF STRING

### Description

Returns a list of all the projects in the project management database.

### Syntax

```
stm_r_pm_projects (status)
```

### Arguments

Argument	Input/Output	Type	Description
status	Out	INTEGER	The function status code

### Status Codes

- ◆ `stm_success`
- ◆ `stm_no_projects`



# Function Status Codes

Status codes have three severity levels:

- ◆ **S** for success
- ◆ **W** for warning
- ◆ **E** for error

When a warning or error status is returned, attempts to execute statements using the return value of the function can produce erroneous or unexpected results. Therefore, you should check the return status codes to ensure that your function call is successful before using the returned values.

The following table lists the status codes and their severity levels.

Code	Status Name	Severity Level
-4	stm_no_stm_root UNIX: The <code>STM_ROOT</code> environment variable is not defined. VMS: The <code>STM\$ROOT</code> or <code>STM\$PM</code> logical name does not exist.	E
-3	stm_obsolete_function Irrelevant function for the current version.	E
-2	stm_missing_elements_in_list Input elements do not exist in the database.	W
-1	stm_list_type_mismatch Incorrect element type used in the query.	E
0	stm_success The function call was successful.	S
1	stm_id_out_of_range The specified ID is not valid for this element type.	E
2	stm_id_not_found An element with the specified ID does not exist.	E
3	stm_illegal_name The specified name is not legal.	E
4	stm_name_not_found The specified name does not exist.	E

## Function Status Codes

---

5	<code>stm_name_not_unique</code> There is more than one element with the specified name, so a specific path name is required.	E
6	<code>stm_missing_name</code> The specified element has no name.	W
7	<code>stm_missing_synonym</code> The specified element has no synonym.	W
8	<code>stm_missing_short_description</code> The specified element has no short description.	W
9	<code>stm_missing_long_description</code> The specified element has no long description.	W
10	<code>stm_attribute_name_not_found</code> The specified element has no attribute name.	W
11	<code>stm_starting_keyword_not_found</code> The long description of the specified element does not contain the given starting keyword.	W
12	<code>stm_ending_keyword_not_found</code> The long description of the specified element does not contain the given ending keyword.	W
13	<code>stm_primitive_element</code> The element is primitive.	W
14	<code>stm_can_not_open_file</code> The operating system cannot open the file with the specified name.	E
15	<code>stm_illegal_address</code> The pointer address is illegal.	E
16	<code>stm_not_an_and_state</code> This state is not supposed to contain and-lines.	W
17	<code>stm_no_and_lines_in_and_state</code> This and-state is missing and-lines.	E
18	<code>stm_missing_graphic_data</code> Graphic data is missing from the element.	E
19	<code>stm_nil_list</code> There is no input list.	E
20	<code>stm_list_element_does_not_exist</code> The specified element does not exist.	W
21	<code>stm_cannot_compare_lists</code> The lists cannot be compared because the list types are different, or the lists are not initialized.	E
22	<code>stm_elements_without_name</code> The list cannot be sorted because its elements have no names.	E
23	<code>stm_elements_without_synonym</code> The list cannot be sorted because its elements have no synonyms.	E



24	stm_elements_not_hierarchical The list cannot be sorted because it is not hierarchical.	E
25	stm_illegal_extract_type You cannot extract this element type.	E
26	stm_no_such_list No such list exists	E
27	stm_not_diagram_connector There is no value in a connector that is not a diagram connector.	E
28	stm_implicit_element The element is defined implicitly—it is an internally defined entity.	E
29	stm_missing_label The element has no label.	W
30	stm_unknown_plotter The plotter type is unknown.	E
31	stm_unresolved The element is unresolved.	W
32	stm_elements_without_attributes The list cannot be sorted because its elements have no attributes.	E
33	stm_not_instance The element is not an instance.	E
34	stm_no_updated_pmdb The workarea database is not updated to the current version.	E
35	stm_no_updated_projdb The installation database is not updated to the current version.	E
36	stm_no_legal_operator The user is not authorized as a Statemate operator.	E
37	stm_deadlock Deadlock situation.	E
38	stm_not_member_of_project The user is not a member of the specified project.	E
39	stm_nonexistent_project The specified project does not exist.	E
40	stm_not_enough_memory The plot cannot be produced because there is not enough memory.	E
41	stm_empty_chart The plot file cannot be produced because the chart is empty.	E
42	stm_plot_failure The plot file was not produced because of a system error.	E
43	stm_no_file_of_licensed_host The file containing the name of the licensed host does not exist.	E

## Function Status Codes

---

44	stm_empty_file_of_licensed_host The file containing the name of the licensed host is empty.	E
45	stm_cannot_chdir_to_work_area Could not change directory to the workarea.	E
46	stm_cannot_write_to_file No space is left on device for writing a file.	E
47	stm_illegal_parameter An illegal parameter value was supplied.	E
50	stm_null_string The input string is null.	E
51	stm_illegal_len The length value is illegal.	E
52	stm_illegal_index The index value is illegal.	E
53	stm_cannot_read_file Cannot read from a file that was not opened.	E
54	stm_end_of_file Reached the end-of-file.	E
55	stm_not_a_parameter The specified ID is not a parameter.	E
56	stm_param_not_compatible The actual and formal parameters are not compatible.	W
57	stm_error_in_file There is an error in the requirement file.	E
58	stm_missing_field A field is missing in the requirement record.	W
59	stm_missing_user_type The specified element has no user-defined type.	E
61	stm_illegal_attribute_value The attribute value is too long.	E
62	stm_duplicate_attribute_pair The specified attribute name/value pair already exists.	E
63	stm_not_in_rw_transaction Attempt to modify the database when not in a read/write transaction.	E
64	stm_missing_of_enum_type The specified element has no enumerated type associated with its array type definition.	W
65	stm_missing_user_code The specified element has no user code.	W
66	stm_missing_subroutine_params The specified element has no subroutine parameters.	W

67	stm_missing_local_data The specified element has no local data.	W
68	stm_missing_global_data The specified element has no global data.	W
69	stm_no_connected_chart The specified element is not connected to a chart.	W
70	stm_attribute_cannot_be_deleted The specified element's attribute cannot be deleted.	E
71	stm_missing_cbk_binding The specified element has no callback binding.	W
72	stm_missing_subroutine_binding The specified element has no subroutine binding.	W
73	stm_missing_statemate_action_lang The specified element has no action language.	W
74	stm_no_projects There are no projects in the project management database.	W
121	stm_illegal_expression_n_chart There is an illegal expression in the loaded chart.	E
122	stm_error_in_chart There is an error in the loaded chart.	E
123	stm_cannot_open_chart_file Cannot open the chart file to be loaded.	E
124	stm_exceeded_max_id_number There are more than 1023 IDs in the workarea.	E
125	stm_chart_not_in_database Cannot find a chart in the database to be saved or unloaded.	E
126	stm_file_not_in_work-area Cannot find a file in the workarea to be saved or unloaded.	E
127	stm_cannot_copy_file Cannot copy a file during a save or load operation.	E
128	stm_cannot_create_file Cannot create an auxiliary file during a load to the workarea.	E
129	stm_illegal_version An illegal version was specified for the load operation.	E
130	stm_file_not_found Cannot find a source file in the load operation.	E
131	stm_not_loaded_because_modified A modified version of loaded chart or file exists in the workarea.	E
132	stm_not_loaded_because_new A new version of the loaded chart or file exists in the workarea.	E

## Function Status Codes

---

133	<code>stm_not_unloaded_modified</code> The chart or file to be unloaded is modified.	E
134	<code>stm_not_unloaded_new</code> The chart or file to be unloaded is new.	E
135	<code>stm_chart_is_active</code> The chart to be unloaded is currently being edited by a graphics editor.	E
136	<code>stm_error_in_save_operation</code> There was a write to disk error during the save operation.	E
137	<code>stm_illegal_load_mode</code> An illegal mode was specified for the load operation.	E
138	<code>stm_not_loaded_because_type</code> A chart with the same name, but of another type, exists in the workarea.	E
139	<code>stm_illegal_type</code> An illegal type of configuration item was specified.	E
140	<code>stm_illegal_parameters</code> An illegal parameter to the load function was specified.	E
141	<code>stm_illegal_bindings</code> There is an error in the loaded chart file.	E
142	<code>stm_too_long_line</code> There is a line too long in the loaded chart file.	E
143	<code>stm_instance_type_conflict</code> There is an instance type conflict in the loaded chart file.	E
144	<code>stm_usage_conflict</code> There is a usage conflict in the loaded chart file.	E
145	<code>stm_unrecognized_format</code> The loaded chart file contains an unrecognized conflict.	E
146	<code>stm_double_chart_parameters</code> There is an error in the loaded chart file.	E
147	<code>stm_double_chart_bindings</code> There is an error in the loaded chart file.	E

# DGL Reserved Words

---

The following table lists the keywords reserved for specific DGL use. Do not use these words as names in your document templates.

ACTION	FIELD	OTHERWISE
ACTIVITY	FILE	OUTPUT
AND	FIRST	PARAMETER
ANY	FLOAT	PROCEDURE
ARRAY	FOR	PROGRAM
A_FLOW_LINE	FUNCTION	READ
BEGIN	IF	RETURN
BOOLEAN	IN	SEGMENT
CLOSE	INCLUDE	SELECT
CONDITION	INFORMATION_FLOW	STATE
CONNECTOR	INPUT	STOP
CONSTANT	INTEGER	STRING
DATA_ITEM	LIST	TEMPLATE
DATA_STORE	LOOP	THEN
DATA_TYPE	MODULE	TRANSITION
ELEMENT	M_FLOW_LINE	TRUE
ELSE	NOT	TYPE
END	NULL	VARIABLE
EVENT	OF	WHEN
EXECUTE	OPEN	WHILE
EXIT	OR	WRITE
FALSE		



# BNF Syntax

---

This section lists the conventions for a widely used notational scheme for formal languages known as BNF, which stands for Bakus-Naur Form (formerly Bakus Normal Form). BNF was introduced in 1963 as a technique for defining programming languages.

In the Rhapsody documentation, a variation of the BNF notation is used to formally describe the DGL statements.

## BNF Structure and Conventions

BNF grammar follows the following general structure:

```
nonterminal_symbol ? terminal_and/or_nonterminal_
                    symbols
```

For example:

```
write_expression ? numeric_expression | string_expression
```

Symbols are delimited by spaces; underscores are frequently used for longer names.

## Symbol Types

*Terminal symbols* are basic symbols that are not parsed further to derive their meaning. *Nonterminal symbols* can be further broken down by parsing.

Examples of terminal symbols are:

- ◆ Integer numbers intrinsically recognized as a numerical value
- ◆ Language keywords recognized by the system as representing some particular operation or function.

In [BNF for DGL Statements](#), terminal symbols that are written exactly as they appear (for example, keywords of Rhapsody), are shown in all uppercase. Non-alphabetic characters not belonging to the BNF notation are also part of the syntax:

- ◆ Nonterminal symbols are written in lowercase or mixed case letters.
- ◆ Nonterminal symbols that are self-evident are not broken down further.

## BNF Notations

The | indicates a mutually exclusive choice between symbols in a nonterminal symbol definition. For example:

```
variable_name | numeric_constant | integer | function_name
```

The ? separates the nonterminal symbol on the left from its definition on the right and can be read as “is defined as...”. For example:

```
relational_operator ? = | < > | < | <= | >=
```

Square brackets ([ ]) indicate that the symbols within the brackets are optional. For example:

```
[directory_name] filename
```

Curly braces ({} ) indicate that the symbols within the braces are optional and can be repeated. For example:

```
begin {statement} end;
```

## BNF for DGL Statements

This section lists the formal syntax (in BNF) for DGL. For ease of use, the DGL statements are presented in alphabetical order.

DGL Statement	BNF
abs_integer_literal	{digit}
abs_numeric_literal	abs_integer_literal   abs_real_literal
abs_real_literal	{digit}. [{digit}]   [{digit}]. {digit}
alpha_letter	A to Z, a to zy
assignment_statement	variable := expression;
boolean_expression	boolean_term   boolean_expression OR boolean_term
boolean_function_call	function_call
boolean_identifier	boolean_parameter   boolean_constant   boolean_variable
boolean_literal	TRUE   FALSE
boolean_primary	boolean_literal   boolean_identifier   expression relation_operation expression   NOT boolean_primary   boolean_function_call   (boolean_expression)



boolean_term	boolean_primary   boolean_term AND boolean_primary
calling_sequence	string_expression
constant_declaration	CONSTANT {const_object_specification}
const_ident_spec	identifier :=literal
const_ident_spec_list	const_ident_spec [{,const_ident_spec}]
constant_object_specification	type const_ident_spec_list
const_var_declaration	constant declaration I variable declaration
control_flow_statement	if_statement   select_statement   for_statement   while_statement   exit_statement   stop_statement
copy_text_statement	/@ text @/
close_statement	CLOSE (file_identifier);
element_expression	element_variable   element_function_call
element_function_call	function_call
execute_call	EXECUTE (calling_sequence);
exit_statement	EXIT;
expression	integer_expression   real_expression   boolean_expression   string_expression   element_expression   list_expression
file_description	file_name   file_identifier
file_identifier	file_variable
file_name	string_expression
file_statement	open_statement   close_statement   read_statement
for_statement	FOR variable IN list_expression LOOP statements END LOOP;
function_call	id (expressions [{, expression}])
global_par	[parameter_declaration] [const_var_declaration] BEGIN [statements] END;
ident_spec	identifier [:= literal]
ident_spec_list	ident_spec [{,ident_spec}]
identifier	alpha_letter [{identifier_letter}]
identifier_letter	A to Z, a to z, 0 to 9, and _
if_statement	IF boolean_expression THEN statements [ELSE statements] END IF;

include_statement	INCLUDE (file_description [,integer_variable]);
integer_expression	numeric_expression
integer_identifier	integer_parameter   integer_constant   integer_variable
integer_literal	[+   -]abs_integer_literal
list_component integer_expression	expression   integer_expression..
list_component_literal	literal   integer_literal.. integer_literal
list_expression	list_term   &   list_expression + list_term   list_expression - list_term   list_expression & list_term
list_identifier	list_parameter   list_constant   list_variable
list_of_components	slist_component [{"", list_component"} <b>See Note.</b>
list_term	list_literal   list_identifier   "{" list_of_components "}" 1   list_term * list_term   (list_expression)
literal	integer_literal   real_literal   boolean_literal   string_literal   list_literal
numeric_expression	numeric_term   numeric_expression + numeric_term   numeric_expression - numeric_term
numeric_factor	numeric_factor ** numeric_primary   numeric_primary
numeric_function_call	function_call
numeric_identifier	integer_identifier   real_identifier
numeric_primary	abs_numeric_literal   numeric_identifier   numeric_function_call   (numeric_expression)
numeric_term	numeric_factor   numeric_term * numeric_factor   numeric_term / numeric_factor   + numeric_factor   -- numeric_factor
object_specification	type ident_spec_list;
open_mode	INPUT I OUTPUT
open_statement	OPEN (file_identifier, file_name, open_mode[,integer_variable]);

output_statement	copy_text_statement   write_statement   include_statement   special_procedure_call   file_statement
parameter_declaration	PARAMETER object_specification
plot_call	function_call
procedure (call)	PROCEDURE procedure_name [return type]; [parameter_object_declaration] [variable_declaration] BEGIN [statements] END;
procedure (name)	procedure proc_name
read_statement	READ (file_identifier {, simple_variable})
real_expression	numeric_expression
real_identifier	real_parameter   real_constant   real_variable
real_literal	[+   -]abs_real_literal
relation_operation	=   < >   >=   >   < =   <
report_call	function_call
segment	SEGMENT segment_name; [const_var_declaration] BEGIN [statements] END;
segment_name	identifier
select_statement	SELECT [FIRST   ANY] [when_construct_with_any] when_construct [OTHERWISE = > statements] END SELECT;
simple_type	INTEGER   REAL   BOOLEAN   STRING   FILE statemate_element
simple_variable	integer_variable   real_variable   string_variable
special_procedure_call	execute_call   report_call   plot_call   table_call

state_element	ELEMENT   STATE   ACTIVITY   DATA_STORE   MODULE   TRANSITION   A_FLOW_LINE   M_FLOW_LINE   EVENT   DATA_TYPE_FIELD   CONDITION   CONNECTOR   DATA_ITEM   ACTION   INFORMATION_FLOW
statement	assignment_statement   output_statement   control_flow_statement
stop_statement	STOP;
string_expression	string_literal   string_identifier   string_function_call   string_expression + string_expression
string_function_call	function_call
string_identifier	string_parameter   string_constant   string_variable
string_literal	'{\n \t printable character}'
structured_type	LIST OF simple_type
table_call	function_call
template	template_header global_part {segment} {procedure}
template_header	TEMPLATE template_name;
template_name	identifier
type	simple_type   structured_type
variable_declaration	VARIABLE {object specification}
when_construct	{when sentence} [WHEN ANY => statements]
when_construct_with_any	{when sentence} [WHEN ANY => statements]
when_sentence	WHEN boolean_expression => statement
while_statement	[WHILE boolean_expression] LOOP statements END LOOP;
write_expression	expression [:integer_expression]
write_statement	WRITE ([file_identifier,] {write_expression});

### Note

The quotation marks indicate that the braces here are not BNF notation, but are the actual brace characters themselves.

# Index

---

## A

Abbreviation, element type 77  
act\_interface  
  report 100  
  template 95  
Action  
  extracting from reaction 509  
  retrieving list of 382  
Activity  
  interface report 94  
  retrieving list of 362  
A-flow-line  
  retrieving lists of 375  
Assignment 114  
  statement 40, 53, 114  
  statements 40  
Attribute  
  report 141  
  sorting by value 535  
attribute report 62  
Autonumber 551

## B

BEGIN statement 115  
begin/end statement 48, 49  
binary operations 44  
BNF 37  
  conventions 595  
  notations 596  
  structure 595  
  symbol types 595  
Boolean comparisons 45  
Boolean expressions 45  
Branch, sorting by 537

## C

calling external programs 39  
Calling, convention for functions 77  
Chart  
  extracting elements by 518  
  retrieving list of 385  
CLOSE statement 117  
close statement 54

Codes, status 79  
Commands, formatting 5  
Comment 118  
comment statement 48  
Compile template 20  
Condition, retrieving list of 396  
Connector  
  retrieving list of 394  
CONSTANT statement 119  
constant statement 51  
constants 41  
control flow statements 40, 69  
  exit statement 72  
  for/loop statement 71  
  if/then/else statement 69  
  select/when statement 69  
  stop statement 73  
  while/loop statement 72  
conventional types 41  
Convert  
  integer to string 513  
  multiline to list of strings 546  
  multiline to single string 546  
  string to integer 576  
Create template 15

## D

Database extraction function  
  calling conventions 77  
  overview 75  
  status codes, list of 587  
  types 75  
  using 76  
  utility 499  
database extractions 39  
Data-item, retrieving list of 400  
Data-store  
  retrieving list of 408  
data-types 41  
declaration statement, parameter statement 50  
declaration statements 40, 50  
  constant statement 51  
  variable statement 52

- DGL 35
    - calling external programs 39
    - database extractions 39
    - extensions 39
    - features 39
    - include files 39
    - include reports and plots 39
    - overview 3
    - syntax rules 37
    - template structure 35
    - verbatim inclusion 39
  - DGL statements 48
    - assignment 40, 114
    - begin 115
    - begin/end statement 48, 49
    - close 117
    - comment 118
    - comment statements 48, 49
    - constant 119
    - constant statement 51
    - control flow statements 40
    - declaration statements 40, 50
    - end 121
    - execute 122
    - exit 123
    - file handling statements 40
    - for/loop 124
    - if/then/else 126
    - include 128
    - open 130
    - output statements 40
    - parameter 132
    - procedure 134
    - procedure statement 48, 49
    - read 138
    - report 139
    - segment 147
    - segment statement 48
    - select/when 148
    - stop 151
    - structure statements 40, 48
    - table 152
    - template 155
    - template statement 48
    - variable 156
    - verbatim 158
    - verbatim statement 55
    - while/loop 160
    - write 162
  - dictionary report 61
  - Document Generation Language, see DGL 3
  - Document segments
    - definition 3
    - editing 28
    - producing 24
    - regenerating 29
    - unformatted 24
  - Documentor tool
    - creating templates 15
    - DGL 3
    - production process 13
    - reusing templates 8
- ## E
- Edit option 28
  - Element
    - first in list 522
    - last in string 566
    - mixed, retrieving list of 450
    - next in list 526
    - previous in list 532
    - previous in string 569
    - searching in list 514
    - type abbreviations 77
  - element expressions 46
  - END statement 121
  - Entry heading 89
  - enumerated types 41, 47
  - Error code 587
  - Event, retrieving list of 418
  - EXECUTE statement 122
  - execute statement 59
  - EXIT statement 123
  - exit statement 72
  - expressions 41, 44
  - extensions to DGL 39
    - calling external programs 39
    - include files 39
    - include reports and plots 39
    - verbatim inclusion 39
  - Extract
    - action from reaction 509
    - elements, by chart 518
    - elements, by type 520
    - string 573
- ## F
- Field, retrieving list of 421
  - File
    - access 9
    - interleaf\_glob 34
  - file handling statements 40
    - close statement 54
    - open statement 53
    - read statement 54
  - First element
    - in list 522
    - in string 565
  - float values 41
  - FOR/LOOP statement 124
    - using 88
  - for/loop statement 71

- Format option 8
- Formatter
  - definition 3
  - invoking 11
- Function
  - arguments 79
  - names 77
  - retrieving list of 427
  - return status codes 79
  - return status codes, list of 587
  - return values 80
  - types 75
  - using 76
- G**
- Generate entry heading 89
- H**
- Hyperlink
  - stm\_plot\_hyper\_exp 548
  - stm\_r\_an\_expr\_hyper 230
- I**
- IF/THEN/ELSE statement 126
- if/then/else statement 69
- Import option 9
- Include file
  - access 9
  - Documentor GUI 21
  - option 21
  - using 8
- include files 39
- include plots statement 64
- include reports statement 59
  - attribute report 62
  - dictionary report 61
  - include plots statement 64
  - include table statement 67
  - interface report 62
  - list report 61
  - N2 chart report 63
  - protocol report 61
  - resolution report 64
  - structure report 61
  - tree report 61
- INCLUDE statement 128
  - using the Documentor GUI 21
- include statement 58
- include table statement 67
- Index, searching for 512
- Information-flow, retrieving list of 428
- Initiation section 86
- Input argument, to database extraction functions 79
- integer values 41
- Integer, converting to a string 513
- Interface report 142
- interface report 62
- Interleaf 34
  - templates 105
- interleaf\_glob file 34
- Invoke formatter 11
- K**
- Keyword 91
- L**
- Last element
  - in string 566
- Length
  - of list 524
  - of list of strings 567
  - of string 577
- Level
  - sort list by 539
- Lifeline
  - printing in plot 558
- List
  - extracting elements, by chart 518
  - extracting elements, by type 520
  - first element 522
  - length 524
    - of list of strings 567
  - next element 526
  - previous element 532
  - report 143
  - searching for a given element 514
  - searching for a given string 516
  - sort 534
    - by attribute value 535
    - by branches 537
  - sort by name 541
  - sorting
    - by levels 539
- list expressions 46
- list item 46
- list report 61
- M**
- meaningly 467
- Message, producing using WRITE 164
- M-flow-line, retrieving list of 433
- Mixed element
  - retrieving list of 450
- Module
  - retrieving lists of 443
- Multiline expression
  - converting to list of strings 546
  - converting to single string 546

### N

- N2 chart report 63, 143
- Name
  - function 77
  - sorting by 541
- Nested for loop 90
- Next element
  - in list 526
  - in string 568
- nroff 34
  - templates 101
- numeric expressions 44

### O

- OPEN statement 130
  - and INCLUDE 128
- open statement 53, 58
- Output mode 128
- output mode 58
- output statements 40
  - execute statement 59
  - include reports statement 59
  - include statement 58
  - verbatim statement 55
  - write statement 56

### P

- Parameter statement 50, 132
- parameters 41
- Pattern, searching for 512
- Plot
  - headerlines 558
  - hyperlinks 548
  - page breaks 554
- Previous element
  - in list 532
  - in string 569
- Procedure
  - parameters 132
  - statement 134
- procedure statement 48, 49
- produce messages
  - using write 58
- Property report
  - example 83
  - syntax 141
- Protocol report 144
- protocol report 61

### Q

- Query function
  - arguments 359
  - calling 356
  - examples 360
  - list of fields 421
  - list of states 488
  - list of subroutines 482
- Query functions 39
  - list of activities 362
  - list of a-flow-lines 375
  - list of charts 385
  - list of conditions 396
  - list of connectors 394
  - list of data-items 400
  - list of data-stores 408
  - list of events 418
  - list of functions 427
  - list of information-flow 428
  - list of m-flow-lines 433
  - list of mixed elements 450
  - list of modules 443
  - list of timing constraints 495
  - list of transitions 495
  - list of user-defined types 411
- Query functions, list of actions 382

### R

- Reaction
  - trigger of 578
- READ statement 138
- read statement 54
- Regenerate option 29
- Report
  - attribute 141
  - heading 88
  - passing files 10
  - predefined 10
  - property example 83
- Report statement 139
  - attribute report 141
  - interface report 142
  - list report 143
  - N2 chart report 143
  - property report 141
  - protocol report 144
  - resolution report 145
  - structure report 145
  - syntax 139
- reports, syntax 59
- Resolution report 145
- resolution report 64
- Return status code 79



Return value 80  
 of enumerated types 81  
 of filename 81  
 of type ELEMENT 80  
 Reuse templates 8

## S

Search

for element in list 514

Segment

definition 3

section 87

statement 147

segment statement 48

SELECT/WHEN statement 148

using 89

select/when statement 69

Sequence diagram

autonumbering 551

breaking across pages 554

simple\_type list 43

Single-element function

arguments 167

calling 166

examples 168

list of 169

Single-element functions

stm\_r\_xx\_cbk\_binding\_expression\_hyper 239

single-element functions 39

stm\_r\_rt\_note 273

Sort

by attribute value 535

by branches 537

by level 539

by name 541

by synonym 543

list 534

Standard template 9

State

retrieving list of 488

Statemate

element

abbreviations 77

Statemate element types 42

Statements 40

Status

codes 79

list of 587

stm\_action\_of\_reaction 509

stm\_delete\_file 510

stm\_dispose\_memory 511, 564

stm\_index 512

stm\_int 513

stm\_int\_to\_string 513

stm\_is\_identifier 514

stm\_list\_contains\_element 514

stm\_list\_contains\_string 516

stm\_list\_extraction 517

stm\_list\_extraction\_by\_chart 518

stm\_list\_extraction\_by\_type 520

stm\_list\_first\_element 522

stm\_list\_last\_element 523

stm\_list\_length 524

stm\_list\_next\_element 526

stm\_list\_previous\_element 532

stm\_list\_sort 534

stm\_list\_sort\_by\_attr\_value 535

stm\_list\_sort\_by\_branches 537

stm\_list\_sort\_by\_chart 537

stm\_list\_sort\_by\_levels 539

stm\_list\_sort\_by\_name 541

stm\_list\_sort\_by\_synonym 543

stm\_list\_sort\_by\_type 545

stm\_multiline\_to\_one 546

stm\_multiline\_to\_strings 546

stm\_plot\_hyper\_exp 548

stm\_plot\_with\_autonumber 551

stm\_plot\_with\_break 554

stm\_plot\_with\_headerline 558

stm\_r\_ac\_actor\_ac 368

stm\_r\_ac\_affecting\_mx 373

stm\_r\_ac\_associates\_uc 362

stm\_r\_ac\_basic\_ac 362

stm\_r\_ac\_boundary\_box\_ac 368

stm\_r\_ac\_by\_attributes\_ac 362

stm\_r\_ac\_callback\_binding\_ac 362

stm\_r\_ac\_carried\_out\_by\_md 372

stm\_r\_ac\_component\_instance\_ac 363

stm\_r\_ac\_continuous\_instance\_ac 363

stm\_r\_ac\_control\_ac 363

stm\_r\_ac\_control\_terminated\_ac 363

stm\_r\_ac\_data\_store\_ac 363

stm\_r\_ac\_def\_of\_instance\_ac 363

stm\_r\_ac\_def\_or\_unres\_in\_ch 371

stm\_r\_ac\_defined\_environment\_ac 363

stm\_r\_ac\_defined\_in\_ch 371

stm\_r\_ac\_described\_by\_ch 371

stm\_r\_ac\_explicit\_defined\_ac 364

stm\_r\_ac\_ext\_ll\_ac 368

stm\_r\_ac\_external\_ac 364

stm\_r\_ac\_external\_router\_ac 368

stm\_r\_ac\_generic\_instance\_ac 364

stm\_r\_ac\_imp\_best\_match\_ac 364

stm\_r\_ac\_imp\_mini\_spec\_ac 364

stm\_r\_ac\_imp\_none\_ac 364

stm\_r\_ac\_imp\_sb\_bind\_ac 364

stm\_r\_ac\_imp\_truth\_table\_ac 365

stm\_r\_ac\_instance\_ac 365

stm\_r\_ac\_instance\_of\_ch 371

stm\_r\_ac\_instance\_of\_def\_ac 365

stm\_r\_ac\_internal\_ac 365

stm\_r\_ac\_is\_occurrence\_of\_ac 365

stm\_r\_ac\_is\_principal\_of\_ac 365

stm\_r\_ac\_lifeline\_ac 368  
stm\_r\_ac\_logical\_desc\_of\_ac 365  
stm\_r\_ac\_logical\_parent\_of\_ac 366  
stm\_r\_ac\_logical\_sub\_of\_ac 366  
stm\_r\_ac\_meaningfully\_affecting\_mx 373  
stm\_r\_ac\_meaningfully\_using\_mx 373  
stm\_r\_ac\_mini\_spec\_ac 366  
stm\_r\_ac\_mini\_spec\_hyper 265  
stm\_r\_ac\_name\_of\_ac 366  
stm\_r\_ac\_offpage\_instance\_ac 366  
stm\_r\_ac\_parent\_of\_ds 372  
stm\_r\_ac\_parent\_of\_router 373  
stm\_r\_ac\_physical\_desc\_of\_ac 366  
stm\_r\_ac\_physical\_parent\_of\_ac 366  
stm\_r\_ac\_physical\_sub\_of\_ac 367  
stm\_r\_ac\_procedure\_like\_ac 367  
stm\_r\_ac\_resolved\_to\_ext\_ac 367  
stm\_r\_ac\_root\_in\_ch 371  
stm\_r\_ac\_router\_ac 368  
stm\_r\_ac\_self\_terminated\_ac 367  
stm\_r\_ac\_source\_of\_af 370  
stm\_r\_ac\_subroutine\_bind 313  
stm\_r\_ac\_subroutine\_bind\_enable 314  
stm\_r\_ac\_subroutine\_bind\_expr 315  
stm\_r\_ac\_subroutine\_binding\_ac 367  
stm\_r\_ac\_synonym\_of\_ac 367  
stm\_r\_ac\_target\_of\_af 370  
stm\_r\_ac\_termination 318  
stm\_r\_ac\_throughout\_st 374  
stm\_r\_ac\_top\_level\_in\_ch 371  
stm\_r\_ac\_unresolved\_ac 367  
stm\_r\_ac\_unresolved\_in\_ch 372  
stm\_r\_ac\_use\_case\_ac 368  
stm\_r\_ac\_using\_mx 373  
stm\_r\_ac\_within\_st 374  
stm\_r\_actor\_explicit\_defined\_actor 452  
stm\_r\_actual\_parameter\_exp 180  
stm\_r\_actual\_parameter\_type 181  
stm\_r\_af\_containing\_ba 376, 455  
stm\_r\_af\_containing\_laf 378  
stm\_r\_af\_from\_source\_ac 375  
stm\_r\_af\_from\_source\_ds 377  
stm\_r\_af\_from\_source\_mx 379  
stm\_r\_af\_from\_source\_router 379  
stm\_r\_af\_input\_to\_ac 375  
stm\_r\_af\_output\_from\_ac 375  
stm\_r\_af\_to\_target\_ac 375  
stm\_r\_af\_to\_target\_ds 377  
stm\_r\_af\_to\_target\_mx 379  
stm\_r\_af\_to\_target\_router 379  
stm\_r\_af\_within\_flows\_co 376  
stm\_r\_af\_within\_flows\_di 376  
stm\_r\_af\_within\_flows\_ev 377  
stm\_r\_af\_within\_flows\_if 378  
stm\_r\_af\_within\_flows\_mx 379  
stm\_r\_af\_within\_labels\_co 376  
stm\_r\_af\_within\_labels\_di 376  
stm\_r\_af\_within\_labels\_ev 377  
stm\_r\_af\_within\_labels\_if 378  
stm\_r\_af\_within\_labels\_mx 379  
stm\_r\_an\_by\_attributes\_an 382  
stm\_r\_an\_def\_or\_unres\_in\_ch 384  
stm\_r\_an\_defined\_in\_ch 384  
stm\_r\_an\_explicit\_defined\_an 382  
stm\_r\_an\_imp\_best\_match\_an 382  
stm\_r\_an\_imp\_definition\_an 383  
stm\_r\_an\_imp\_none\_an 383  
stm\_r\_an\_imp\_truth\_table\_an 383  
stm\_r\_an\_name\_of\_an 383  
stm\_r\_an\_synonym\_of\_an 383  
stm\_r\_an\_unresolved\_an 383  
stm\_r\_an\_unresolved\_in\_ch 384  
stm\_r\_ba\_contained\_in\_af 370  
stm\_r\_ba\_defined\_in\_ch 369  
stm\_r\_ba\_enter\_an 369  
stm\_r\_ba\_enter\_cn 369  
stm\_r\_ba\_enter\_ds 369  
stm\_r\_ba\_exit\_from\_ac 369  
stm\_r\_ba\_exit\_from\_cn 370  
stm\_r\_ba\_exit\_from\_ds 369  
stm\_r\_bb\_explicit\_defined\_bb 454  
stm\_r\_bf\_from\_source\_mx 435  
stm\_r\_bf\_to\_target\_mx 435  
stm\_r\_bf\_within\_flows\_co 433  
stm\_r\_bf\_within\_flows\_di 433  
stm\_r\_bf\_within\_flows\_ev 434  
stm\_r\_bf\_within\_flows\_if 434  
stm\_r\_bf\_within\_flows\_mx 435  
stm\_r\_bf\_within\_labels\_co 433  
stm\_r\_bf\_within\_labels\_di 433  
stm\_r\_bf\_within\_labels\_ev 434  
stm\_r\_bf\_within\_labels\_if 434  
stm\_r\_bf\_within\_labels\_mx 435  
stm\_r\_bm\_enter\_cn 456  
stm\_r\_bm\_enter\_md 456  
stm\_r\_bm\_exit\_from\_cn 456  
stm\_r\_bm\_exit\_from\_md 456  
stm\_r\_bm\_exit\_from\_om 457  
stm\_r\_bt\_defined\_in\_ch 455  
stm\_r\_bt\_enter\_cn 455  
stm\_r\_bt\_enter\_st 455  
stm\_r\_bt\_exit\_from\_cn 455  
stm\_r\_bt\_exit\_from\_st 454  
stm\_r\_cd\_info 246  
stm\_r\_ch\_activitychart\_ch 386  
stm\_r\_ch\_ancestors\_of\_ch 386  
stm\_r\_ch\_by\_attributes\_ch 386  
stm\_r\_ch\_connected\_to\_sb 393  
stm\_r\_ch\_define\_ac 385  
stm\_r\_ch\_define\_an 385  
stm\_r\_ch\_define\_co 388  
stm\_r\_ch\_define\_di 389  
stm\_r\_ch\_define\_ds 389  
stm\_r\_ch\_define\_dt 389

---

stm\_r\_ch\_define\_ev 390  
stm\_r\_ch\_define\_fd 390  
stm\_r\_ch\_define\_if 390  
stm\_r\_ch\_define\_md 391  
stm\_r\_ch\_define\_mx 392  
stm\_r\_ch\_define\_router 392  
stm\_r\_ch\_define\_sb 393  
stm\_r\_ch\_define\_st 393  
stm\_r\_ch\_defining\_ac 385  
stm\_r\_ch\_defining\_md 391  
stm\_r\_ch\_defining\_mx 392  
stm\_r\_ch\_defining\_st 393  
stm\_r\_ch\_descendants\_of\_ch 386  
stm\_r\_ch\_describing\_ac 385  
stm\_r\_ch\_describing\_md 391  
stm\_r\_ch\_describing\_mx 392  
stm\_r\_ch\_dictionary\_ch 386  
stm\_r\_ch\_explicit\_defined\_ch 386  
stm\_r\_ch\_generic\_ch 386  
stm\_r\_ch\_modification\_date 267  
stm\_r\_ch\_modulechart\_ch 387  
stm\_r\_ch\_name\_of\_ch 387  
stm\_r\_ch\_offpage\_ch 387  
stm\_r\_ch\_parent\_ch 387  
stm\_r\_ch\_procedural\_sch\_ch 387  
stm\_r\_ch\_referenced\_all\_by\_ch 387  
stm\_r\_ch\_referenced\_by\_ch 387  
stm\_r\_ch\_root\_ch 388  
stm\_r\_ch\_statechart\_ch 388  
stm\_r\_ch\_subchart\_ch 388  
stm\_r\_ch\_unresolved\_ch 388  
stm\_r\_ch\_usage\_type 331  
stm\_r\_ch\_version 334  
stm\_r\_cn\_deep\_history\_cn 394  
stm\_r\_cn\_history\_cn 394  
stm\_r\_cn\_history\_or\_term\_in\_st 395  
stm\_r\_cn\_source\_of\_ba 395  
stm\_r\_cn\_source\_of\_bm 394  
stm\_r\_cn\_source\_of\_bt 394  
stm\_r\_cn\_source\_of\_tr 396  
stm\_r\_cn\_target\_of\_ba 395  
stm\_r\_cn\_target\_of\_bm 394  
stm\_r\_cn\_target\_of\_bt 395  
stm\_r\_cn\_target\_of\_tr 396  
stm\_r\_cn\_termination\_cn 394  
stm\_r\_co\_array\_co 397  
stm\_r\_co\_by\_attributes\_co 397  
stm\_r\_co\_by\_structure\_type\_co 398  
stm\_r\_co\_callback\_binding\_co 398  
stm\_r\_co\_contained\_in\_di 399  
stm\_r\_co\_contained\_in\_if 399  
stm\_r\_co\_def\_or\_unres\_in\_ch 397  
stm\_r\_co\_defined\_in\_ch 397  
stm\_r\_co\_explicit\_defined\_co 398  
stm\_r\_co\_flowng\_through\_af 396  
stm\_r\_co\_flowng\_through\_mf 399  
stm\_r\_co\_labeling\_af 396  
stm\_r\_co\_labeling\_mf 399  
stm\_r\_co\_name\_of\_co 398  
stm\_r\_co\_single\_co 398  
stm\_r\_co\_synonym\_of\_co 398  
stm\_r\_co\_unresolved\_co 398  
stm\_r\_co\_unresolved\_in\_ch 397  
stm\_r\_di\_array\_di 401  
stm\_r\_di\_array\_missing\_di 401  
stm\_r\_di\_basic\_di 401  
stm\_r\_di\_bit\_di 401  
stm\_r\_di\_bit\_queue\_di 402  
stm\_r\_di\_bits\_array\_di 402  
stm\_r\_di\_bits\_di 402  
stm\_r\_di\_bits\_queue\_di 402  
stm\_r\_di\_by\_attributes\_di 402  
stm\_r\_di\_by\_structure\_type\_di 402  
stm\_r\_di\_callback\_binding\_di 402  
stm\_r\_di\_contained\_in\_if 407  
stm\_r\_di\_containing\_co 401  
stm\_r\_di\_containing\_fd 406  
stm\_r\_di\_def\_or\_unres\_in\_ch 400  
stm\_r\_di\_defined\_in\_ch 400  
stm\_r\_di\_explicit\_defined\_di 403  
stm\_r\_di\_flowng\_through\_af 400  
stm\_r\_di\_flowng\_through\_mf 407  
stm\_r\_di\_integer\_array\_di 403  
stm\_r\_di\_integer\_di 403  
stm\_r\_di\_integer\_queue\_di 403  
stm\_r\_di\_labeling\_af 400  
stm\_r\_di\_labeling\_mf 407  
stm\_r\_di\_missing\_di 403  
stm\_r\_di\_name\_of\_di 403  
stm\_r\_di\_parent\_of\_di 403  
stm\_r\_di\_queue\_di 404  
stm\_r\_di\_queue\_missing\_di 404  
stm\_r\_di\_real\_array\_di 404  
stm\_r\_di\_real\_di 404  
stm\_r\_di\_real\_queue\_di 404  
stm\_r\_di\_record\_array\_di 404  
stm\_r\_di\_record\_di 404  
stm\_r\_di\_single\_di 405  
stm\_r\_di\_string\_array\_di 405  
stm\_r\_di\_string\_di 405  
stm\_r\_di\_string\_queue\_di 405  
stm\_r\_di\_subdata\_item\_of\_di 405  
stm\_r\_di\_synonym\_of\_di 405  
stm\_r\_di\_union\_array\_di 405  
stm\_r\_di\_union\_di 406  
stm\_r\_di\_unresolved\_di 406  
stm\_r\_di\_unresolved\_in\_ch 400  
stm\_r\_di\_user\_type\_array\_di 406  
stm\_r\_di\_user\_type\_di 406  
stm\_r\_di\_user\_type\_queue\_di 406  
stm\_r\_ds\_by\_attributes\_ds 409  
stm\_r\_ds\_contained\_in\_ac 408  
stm\_r\_ds\_def\_or\_unres\_in\_ch 409  
stm\_r\_ds\_defined\_in\_ch 409

- stm\_r\_ds\_explicit\_defined\_ds 409
- stm\_r\_ds\_in\_ac 408
- stm\_r\_ds\_is\_occurrence\_of\_ds 410
- stm\_r\_ds\_is\_principal\_of\_ds 410
- stm\_r\_ds\_name\_of\_ds 410
- stm\_r\_ds\_resides\_in\_md 410
- stm\_r\_ds\_source\_of\_af 408
- stm\_r\_ds\_synonym\_of\_ds 410
- stm\_r\_ds\_target\_of\_af 408
- stm\_r\_ds\_unresolved\_ds 410
- stm\_r\_ds\_unresolved\_in\_ch 409
- stm\_r\_dt\_array\_dt 412
- stm\_r\_dt\_array\_missing\_dt 412
- stm\_r\_dt\_bit\_dt 412
- stm\_r\_dt\_bit\_queue\_dt 412
- stm\_r\_dt\_bits\_array\_dt 412
- stm\_r\_dt\_bits\_dt 412
- stm\_r\_dt\_bits\_queue\_dt 413
- stm\_r\_dt\_by\_attributes\_dt 413
- stm\_r\_dt\_by\_structure\_type\_dt 413
- stm\_r\_dt\_condition\_array\_dt 413
- stm\_r\_dt\_condition\_dt 413
- stm\_r\_dt\_condition\_queue\_dt 413
- stm\_r\_dt\_containing\_fd 417
- stm\_r\_dt\_def\_or\_unres\_in\_ch 411
- stm\_r\_dt\_defined\_in\_ch 411
- stm\_r\_dt\_enums\_dt 413
- stm\_r\_dt\_explicit\_defined\_dt 414
- stm\_r\_dt\_integer\_array\_dt 414
- stm\_r\_dt\_integer\_dt 414
- stm\_r\_dt\_integer\_queue\_dt 414
- stm\_r\_dt\_missing\_dt 414
- stm\_r\_dt\_name\_of\_dt 414
- stm\_r\_dt\_queue\_dt 414
- stm\_r\_dt\_queue\_missing\_dt 415
- stm\_r\_dt\_real\_array\_dt 415
- stm\_r\_dt\_real\_dt 415
- stm\_r\_dt\_real\_queue\_dt 415
- stm\_r\_dt\_record\_array\_dt 415
- stm\_r\_dt\_record\_dt 415
- stm\_r\_dt\_single\_dt 415
- stm\_r\_dt\_string\_array\_dt 416
- stm\_r\_dt\_string\_dt 416
- stm\_r\_dt\_string\_queue\_dt 416
- stm\_r\_dt\_synonym\_of\_dt 416
- stm\_r\_dt\_union\_array\_dt 416
- stm\_r\_dt\_union\_dt 416
- stm\_r\_dt\_unresolved\_dt 416
- stm\_r\_dt\_unresolved\_in\_ch 411
- stm\_r\_dt\_user\_type\_array\_dt 417
- stm\_r\_dt\_user\_type\_dt 417
- stm\_r\_dt\_user\_type\_queue\_dt 417
- stm\_r\_elem\_in\_ddb\_list 182
- stm\_r\_en\_parent 289
- stm\_r\_ev\_array\_ev 419
- stm\_r\_ev\_by\_attributes\_ev 419
- stm\_r\_ev\_by\_structure\_type\_ev 419
- stm\_r\_ev\_callback\_binding\_ev 419
- stm\_r\_ev\_contained\_in\_if 420
- stm\_r\_ev\_def\_or\_unres\_in\_ch 418
- stm\_r\_ev\_defined\_in\_ch 418
- stm\_r\_ev\_explicit\_defined\_ev 419
- stm\_r\_ev\_flowng\_through\_af 418
- stm\_r\_ev\_flowng\_through\_mf 421
- stm\_r\_ev\_labeling\_af 418
- stm\_r\_ev\_labeling\_mf 421
- stm\_r\_ev\_name\_of\_ev 419
- stm\_r\_ev\_single\_ev 420
- stm\_r\_ev\_synonym\_of\_ev 420
- stm\_r\_ev\_unresolved\_ev 420
- stm\_r\_ev\_unresolved\_in\_ch 418
- stm\_r\_fch\_connected\_to\_sb 488
- stm\_r\_fd\_array\_fd 422
- stm\_r\_fd\_array\_missing\_fd 422
- stm\_r\_fd\_bit\_fd 422
- stm\_r\_fd\_bit\_queue\_fd 422
- stm\_r\_fd\_bits\_array\_fd 422
- stm\_r\_fd\_bits\_fd 423
- stm\_r\_fd\_bits\_queue\_fd 423
- stm\_r\_fd\_by\_attributes\_fd 423
- stm\_r\_fd\_by\_structure\_type\_fd 423
- stm\_r\_fd\_condition\_array\_fd 423
- stm\_r\_fd\_condition\_fd 423
- stm\_r\_fd\_condition\_queue\_fd 423
- stm\_r\_fd\_contained\_in\_di 421
- stm\_r\_fd\_contained\_in\_dt 422
- stm\_r\_fd\_contained\_in\_mx 426
- stm\_r\_fd\_defined\_in\_ch 421
- stm\_r\_fd\_explicit\_defined\_fd 424
- stm\_r\_fd\_integer\_array\_fd 424
- stm\_r\_fd\_integer\_fd 424
- stm\_r\_fd\_integer\_queue\_fd 424
- stm\_r\_fd\_missing\_fd 424
- stm\_r\_fd\_name\_of\_fd 424
- stm\_r\_fd\_queue\_fd 424
- stm\_r\_fd\_queue\_missing\_fd 425
- stm\_r\_fd\_real\_array\_fd 425
- stm\_r\_fd\_real\_fd 425
- stm\_r\_fd\_real\_queue\_fd 425
- stm\_r\_fd\_string\_array\_fd 425
- stm\_r\_fd\_string\_fd 425
- stm\_r\_fd\_string\_queue\_fd 426
- stm\_r\_fd\_user\_type\_array\_fd 426
- stm\_r\_fd\_user\_type\_fd 426
- stm\_r\_fd\_user\_type\_queue\_fd 426
- stm\_r\_fn\_name\_of\_fn 427
- stm\_r\_fn\_unresolved\_in\_ch 427
- stm\_r\_formal\_parameter\_names 236
- stm\_r\_gds\_visibility\_mode 335
- stm\_r\_global\_interface\_report 238
- stm\_r\_hyper\_key 242
- stm\_r\_if\_basic\_flowng\_af 428
- stm\_r\_if\_basic\_flowng\_mf 432
- stm\_r\_if\_basic\_if 431

---

stm\_r\_if\_by\_attributes\_if 431  
stm\_r\_if\_contained\_in\_if 431  
stm\_r\_if\_containing\_co 429  
stm\_r\_if\_containing\_di 430  
stm\_r\_if\_containing\_ev 430  
stm\_r\_if\_containing\_if 431  
stm\_r\_if\_defined\_in\_ch 429  
stm\_r\_if\_explicit\_defined\_if 431  
stm\_r\_if\_flowng\_through\_af 428  
stm\_r\_if\_flowng\_through\_mf 432  
stm\_r\_if\_labeling\_af 428  
stm\_r\_if\_labeling\_mf 432  
stm\_r\_if\_name\_of\_if 431  
stm\_r\_if\_or\_unres\_in\_ch 429  
stm\_r\_if\_synonym\_of\_if 432  
stm\_r\_if\_unresolved\_if 432  
stm\_r\_if\_unresolved\_in\_ch 429  
stm\_r\_included\_gds 244  
stm\_r\_inherited\_gds 247  
stm\_r\_is\_statemate 250  
stm\_r\_laf\_contained\_in\_af 380  
stm\_r\_laf\_containing\_ba 370  
stm\_r\_laf\_from\_source\_ac 380  
stm\_r\_laf\_from\_source\_ds 381  
stm\_r\_laf\_from\_source\_mx 381  
stm\_r\_laf\_from\_source\_router 382  
stm\_r\_laf\_input\_to\_ac 380  
stm\_r\_laf\_output\_from\_ac 380  
stm\_r\_laf\_to\_target\_ac 380  
stm\_r\_laf\_to\_target\_ds 381  
stm\_r\_laf\_to\_target\_mx 381  
stm\_r\_laf\_to\_target\_router 382  
stm\_r\_lmf\_contained\_in\_mf 438  
stm\_r\_lmf\_containing\_bm 437  
stm\_r\_lmf\_from\_source\_md 437  
stm\_r\_lmf\_input\_to\_md 437  
stm\_r\_lmf\_output\_from\_md 437  
stm\_r\_lmf\_to\_target\_md 437, 441  
stm\_r\_local\_interface\_report 257  
stm\_r\_md\_basic\_md 445  
stm\_r\_md\_bus\_md 445  
stm\_r\_md\_by\_attributes\_md 445  
stm\_r\_md\_carrying\_out\_ac 443  
stm\_r\_md\_contains\_ds 444  
stm\_r\_md\_contains\_router 449  
stm\_r\_md\_control\_md 445  
stm\_r\_md\_def\_of\_instance\_md 445  
stm\_r\_md\_def\_or\_unres\_in\_ch 443  
stm\_r\_md\_defined\_environment\_md 445  
stm\_r\_md\_defined\_in\_ch 443  
stm\_r\_md\_described\_by\_ch 443  
stm\_r\_md\_environment\_md 446  
stm\_r\_md\_explicit\_defined\_md 446  
stm\_r\_md\_external\_md 446  
stm\_r\_md\_generic\_instance\_md 446  
stm\_r\_md\_implementation 243  
stm\_r\_md\_instance\_md 446  
stm\_r\_md\_instance\_of\_ch 444  
stm\_r\_md\_instance\_of\_def\_md 446  
stm\_r\_md\_library\_md 446  
stm\_r\_md\_logical\_desc\_of\_md 447  
stm\_r\_md\_logical\_parent\_of\_md 447  
stm\_r\_md\_logical\_sub\_of\_md 447  
stm\_r\_md\_name\_of\_md 447  
stm\_r\_md\_offpage\_instance\_md 447  
stm\_r\_md\_physical\_desc\_of\_md 447  
stm\_r\_md\_physical\_parent\_of\_md 448  
stm\_r\_md\_physical\_sub\_of\_md 448  
stm\_r\_md\_purpose 292  
stm\_r\_md\_regular\_md 448  
stm\_r\_md\_resolved\_to\_ext\_md 448  
stm\_r\_md\_root\_in\_ch 444  
stm\_r\_md\_source\_of\_mf 449  
stm\_r\_md\_storage\_md 448  
stm\_r\_md\_synonym\_of\_md 448  
stm\_r\_md\_target\_of\_mf 449  
stm\_r\_md\_top\_level\_in\_ch 444  
stm\_r\_md\_unresolved\_in\_ch 444  
stm\_r\_md\_unresolved\_md 448  
stm\_r\_mf\_containing\_bm 395  
stm\_r\_mf\_containing\_lmf 440  
stm\_r\_mf\_from\_source\_md 441  
stm\_r\_mf\_input\_to\_md 441  
stm\_r\_mf\_output\_from\_md 441  
stm\_r\_mf\_to\_target\_md 441  
stm\_r\_mf\_within\_flows\_co 438  
stm\_r\_mf\_within\_flows\_di 439  
stm\_r\_mf\_within\_flows\_ev 439  
stm\_r\_mf\_within\_flows\_if 440  
stm\_r\_mf\_within\_flows\_mx 442  
stm\_r\_mf\_within\_labels\_co 438  
stm\_r\_mf\_within\_labels\_di 439  
stm\_r\_mf\_within\_labels\_ev 439  
stm\_r\_mf\_within\_labels\_if 440  
stm\_r\_mf\_within\_labels\_mx 442  
stm\_r\_msg\_included\_in\_ord\_insig 245  
stm\_r\_msg\_where\_tc\_begins 336  
stm\_r\_msg\_where\_tc\_ends 337  
stm\_r\_mx\_affected\_by\_ac 451  
stm\_r\_mx\_affected\_by\_mx 467  
stm\_r\_mx\_affected\_by\_st 476  
stm\_r\_mx\_affected\_by\_tr 478  
stm\_r\_mx\_affecting\_mx 467  
stm\_r\_mx\_by\_attributes\_mx 467  
stm\_r\_mx\_callback\_binding\_mx 467  
stm\_r\_mx\_comb\_elements\_mx 468  
stm\_r\_mx\_constant\_parameter\_ch 457  
stm\_r\_mx\_containing\_fd 463  
stm\_r\_mx\_def\_of\_instance\_mx 468  
stm\_r\_mx\_def\_or\_unres\_in\_ch 457  
stm\_r\_mx\_defined\_in\_ch 457  
stm\_r\_mx\_explicit\_defined\_mx 468  
stm\_r\_mx\_flowng\_from\_router 475  
stm\_r\_mx\_flowng\_through\_af 450

- stm\_r\_mx\_flowng\_through\_mf 466
- stm\_r\_mx\_flowng\_to\_router 475
- stm\_r\_mx\_generic\_instance\_mx 468
- stm\_r\_mx\_in\_definition\_of\_an 453
- stm\_r\_mx\_in\_definition\_of\_co 459
- stm\_r\_mx\_in\_definition\_of\_di 460
- stm\_r\_mx\_in\_definition\_of\_dt 461
- stm\_r\_mx\_in\_definition\_of\_ev 462
- stm\_r\_mx\_in\_definition\_of\_fd 463
- stm\_r\_mx\_in\_definition\_of\_if 464
- stm\_r\_mx\_in\_definition\_of\_mx 468
- stm\_r\_mx\_in\_parameter\_ch 457
- stm\_r\_mx\_influence\_ac 451
- stm\_r\_mx\_influence\_md 465
- stm\_r\_mx\_influence\_st 476
- stm\_r\_mx\_influence\_value\_of\_an 453
- stm\_r\_mx\_influence\_value\_of\_ch 457
- stm\_r\_mx\_influence\_value\_of\_co 459
- stm\_r\_mx\_influence\_value\_of\_di 460
- stm\_r\_mx\_influence\_value\_of\_dt 461
- stm\_r\_mx\_influence\_value\_of\_ev 462
- stm\_r\_mx\_influence\_value\_of\_fd 463
- stm\_r\_mx\_influence\_value\_of\_if 464
- stm\_r\_mx\_influence\_value\_of\_mx 468
- stm\_r\_mx\_influenced\_by\_ac 451
- stm\_r\_mx\_influenced\_by\_an 453
- stm\_r\_mx\_influenced\_by\_co 460
- stm\_r\_mx\_influenced\_by\_di 460
- stm\_r\_mx\_influenced\_by\_dt 462
- stm\_r\_mx\_influenced\_by\_ev 462
- stm\_r\_mx\_influenced\_by\_fd 463
- stm\_r\_mx\_influenced\_by\_fn 464
- stm\_r\_mx\_influenced\_by\_if 465
- stm\_r\_mx\_influenced\_by\_md 465
- stm\_r\_mx\_influenced\_by\_mx 469
- stm\_r\_mx\_influenced\_by\_sb 476
- stm\_r\_mx\_influenced\_by\_st 477
- stm\_r\_mx\_inout\_parameter\_ch 458
- stm\_r\_mx\_instance\_mx 469
- stm\_r\_mx\_instance\_of\_ch 458
- stm\_r\_mx\_instance\_of\_def\_mx 469
- stm\_r\_mx\_labeling\_af 450
- stm\_r\_mx\_labeling\_mf 466
- stm\_r\_mx\_labeling\_msg 466
- stm\_r\_mx\_labeling\_tr 478
- stm\_r\_mx\_logical\_desc\_of\_mx 469
- stm\_r\_mx\_logical\_parent\_of\_mx 469
- stm\_r\_mx\_logical\_sub\_of\_mx 469
- stm\_r\_mx\_meaningly\_affecting\_mx 467
- stm\_r\_mx\_meaningly\_using\_mx 471
- stm\_r\_mx\_name\_of\_mx 470
- stm\_r\_mx\_offpage\_instance\_mx 470
- stm\_r\_mx\_out\_parameter\_ch 458
- stm\_r\_mx\_parameter\_mx 470
- stm\_r\_mx\_parameter\_of\_ch 458
- stm\_r\_mx\_physical\_desc\_of\_mx 470
- stm\_r\_mx\_physical\_parent\_of\_mx 470
- stm\_r\_mx\_physical\_sub\_of\_mx 470
- stm\_r\_mx\_refer\_to\_ac 451
- stm\_r\_mx\_refer\_to\_an 453
- stm\_r\_mx\_refer\_to\_co 460
- stm\_r\_mx\_refer\_to\_di 461
- stm\_r\_mx\_refer\_to\_ds 461
- stm\_r\_mx\_refer\_to\_dt 462
- stm\_r\_mx\_refer\_to\_ev 463
- stm\_r\_mx\_refer\_to\_fd 463
- stm\_r\_mx\_refer\_to\_fn 464
- stm\_r\_mx\_refer\_to\_if 465
- stm\_r\_mx\_refer\_to\_md 465
- stm\_r\_mx\_refer\_to\_mx 470
- stm\_r\_mx\_refer\_to\_router 475
- stm\_r\_mx\_refer\_to\_sb 476
- stm\_r\_mx\_refer\_to\_st 477
- stm\_r\_mx\_referenced\_by\_ac 452
- stm\_r\_mx\_referenced\_by\_ch 458
- stm\_r\_mx\_referenced\_by\_md 466
- stm\_r\_mx\_referenced\_by\_st 477
- stm\_r\_mx\_resolved\_to\_ext\_ac 452
- stm\_r\_mx\_resolved\_to\_ext\_md 466
- stm\_r\_mx\_resolved\_to\_ext\_mx 471
- stm\_r\_mx\_resolved\_to\_ext\_router 475
- stm\_r\_mx\_root\_in\_ch 458
- stm\_r\_mx\_source\_of\_af 450
- stm\_r\_mx\_source\_of\_ba 450
- stm\_r\_mx\_source\_of\_bm 456
- stm\_r\_mx\_source\_of\_bt 454
- stm\_r\_mx\_source\_of\_tr 478
- stm\_r\_mx\_synonym\_of\_mx 471
- stm\_r\_mx\_target\_of\_af 450
- stm\_r\_mx\_target\_of\_ba 450, 451
- stm\_r\_mx\_target\_of\_bm 456
- stm\_r\_mx\_target\_of\_tr 478
- stm\_r\_mx\_text\_def\_unres\_in\_ch 458
- stm\_r\_mx\_text\_unresolved\_in\_ch 459
- stm\_r\_mx\_textual\_defined\_in\_ch 459
- stm\_r\_mx\_unresolved\_in\_ch 459
- stm\_r\_mx\_unresolved\_mx 471
- stm\_r\_mx\_used\_by\_ac 452
- stm\_r\_mx\_used\_by\_mx 471
- stm\_r\_mx\_used\_by\_st 477
- stm\_r\_mx\_used\_by\_tr 478
- stm\_r\_mx\_using\_mx 471
- stm\_r\_mx\_with\_combinationals\_mx 472
- stm\_r\_next\_msg 272
- stm\_r\_ord\_insig\_defined\_in\_ch 284
- stm\_r\_parameter\_binding 285
- stm\_r\_parameter\_mode 286
- stm\_r\_pm\_operator\_projects 581
- stm\_r\_pm\_project\_databank 582
- stm\_r\_pm\_project\_manager 583
- stm\_r\_pm\_project\_members 584
- stm\_r\_pm\_project\_workareas 580
- stm\_r\_pm\_projects 585
- stm\_r\_previous\_msg 290

- 
- stm\_r\_router\_by\_attr\_router 481
  - stm\_r\_router\_contained\_in\_ac 479
  - stm\_r\_router\_def\_or\_unres\_in\_ch 480
  - stm\_r\_router\_defined\_in\_ch 480
  - stm\_r\_router\_exp\_def\_router 481
  - stm\_r\_router\_in\_ac 479
  - stm\_r\_router\_name\_of\_router 481
  - stm\_r\_router\_res\_to\_ext\_router 481
  - stm\_r\_router\_resides\_in\_md 481
  - stm\_r\_router\_source\_of\_af 480
  - stm\_r\_router\_synonym\_of\_router 482
  - stm\_r\_router\_target\_of\_af 480
  - stm\_r\_router\_unresolved\_in\_ch 480
  - stm\_r\_router\_unresolved\_router 482
  - stm\_r\_sb\_action\_lang 177
  - stm\_r\_sb\_action\_lang\_expression 178
  - stm\_r\_sb\_action\_lang\_local\_data 179
  - stm\_r\_sb\_ada\_sb 483
  - stm\_r\_sb\_ada\_user\_code 183
  - stm\_r\_sb\_ansi\_c\_sb 483
  - stm\_r\_sb\_ansi\_c\_user\_code 184
  - stm\_r\_sb\_bit\_sb 483
  - stm\_r\_sb\_bits\_sb 483
  - stm\_r\_sb\_by\_attributes\_sb 483
  - stm\_r\_sb\_connected\_statechart 352
  - stm\_r\_sb\_connected\_to\_ch 482
  - stm\_r\_sb\_def\_or\_unres\_in\_ch 482
  - stm\_r\_sb\_defined\_in\_ch 482
  - stm\_r\_sb\_explicit\_defined\_sb 483
  - stm\_r\_sb\_fn\_with\_side\_effect\_sb 484
  - stm\_r\_sb\_function\_sb 484
  - stm\_r\_sb\_global\_data 237
  - stm\_r\_sb\_globals\_usage\_sb 484
  - stm\_r\_sb\_imp\_action\_lang\_sb 484
  - stm\_r\_sb\_imp\_ada\_code\_sb 484
  - stm\_r\_sb\_imp\_ansi\_c\_code\_sb 484
  - stm\_r\_sb\_imp\_best\_match\_sb 485
  - stm\_r\_sb\_imp\_kr\_c\_code\_sb 485
  - stm\_r\_sb\_imp\_none\_sb 485
  - stm\_r\_sb\_imp\_procedural\_sch\_sb 485
  - stm\_r\_sb\_imp\_truth\_table\_sb 485
  - stm\_r\_sb\_integer\_sb 485
  - stm\_r\_sb\_kr\_c\_sb 486
  - stm\_r\_sb\_kr\_c\_user\_code 253
  - stm\_r\_sb\_missing\_sb 486
  - stm\_r\_sb\_name\_of\_sb 486
  - stm\_r\_sb\_parameters 288
  - stm\_r\_sb\_parameters\_sb 486
  - stm\_r\_sb\_proc\_sch\_local\_data 291
  - stm\_r\_sb\_procedural\_fch\_sb 487
  - stm\_r\_sb\_procedural\_sch\_sb 486
  - stm\_r\_sb\_procedure\_sb 486
  - stm\_r\_sb\_real\_sb 486
  - stm\_r\_sb\_return\_type 297
  - stm\_r\_sb\_return\_user\_type 298
  - stm\_r\_sb\_return\_user\_type\_name 299
  - stm\_r\_sb\_statemate\_action\_sb 487
  - stm\_r\_sb\_string\_sb 487
  - stm\_r\_sb\_synonym\_of\_sb 487
  - stm\_r\_sb\_task\_sb 487
  - stm\_r\_sb\_truth\_table\_local\_data 322
  - stm\_r\_sb\_unresolved\_in\_ch 482
  - stm\_r\_sb\_unresolved\_sb 487
  - stm\_r\_sb\_user\_type\_sb 487
  - stm\_r\_sch\_connected\_to\_sb 488
  - stm\_r\_sd\_scope 305
  - stm\_r\_set\_rpt\_formatter 306
  - stm\_r\_single\_fd 425
  - stm\_r\_st\_affecting\_mx 490
  - stm\_r\_st\_and\_st 491
  - stm\_r\_st\_basic\_st 491
  - stm\_r\_st\_by\_attributes\_st 491
  - stm\_r\_st\_callback\_binding\_st 491
  - stm\_r\_st\_combinationals 185
  - stm\_r\_st\_containing\_cn 490
  - stm\_r\_st\_def\_of\_instance\_st 491
  - stm\_r\_st\_def\_or\_unres\_in\_ch 489
  - stm\_r\_st\_default\_entry\_to\_st 491
  - stm\_r\_st\_defined\_in\_ch 489
  - stm\_r\_st\_done\_throughout\_ac 488
  - stm\_r\_st\_done\_within\_ac 488
  - stm\_r\_st\_explicit\_defined\_st 492
  - stm\_r\_st\_generic\_instance\_st 492
  - stm\_r\_st\_history\_connector\_st 492
  - stm\_r\_st\_instance\_of\_ch 489
  - stm\_r\_st\_instance\_of\_def\_st 492
  - stm\_r\_st\_instance\_st 492
  - stm\_r\_st\_logical\_desc\_of\_st 492
  - stm\_r\_st\_logical\_parent\_of\_st 493
  - stm\_r\_st\_logical\_sub\_of\_st 493
  - stm\_r\_st\_meaningfully\_affecting\_mx 490
  - stm\_r\_st\_meaningfully\_using\_mx 490
  - stm\_r\_st\_name\_of\_st 493
  - stm\_r\_st\_offpage\_instance\_st 493
  - stm\_r\_st\_physical\_desc\_of\_st 493
  - stm\_r\_st\_physical\_parent\_of\_st 493
  - stm\_r\_st\_physical\_sub\_of\_st 494
  - stm\_r\_st\_reaction\_activity\_st 494
  - stm\_r\_st\_root\_in\_ch 489
  - stm\_r\_st\_source\_of\_tr 494
  - stm\_r\_st\_static\_reactions 308
  - stm\_r\_st\_static\_reactions\_hyper 309
  - stm\_r\_st\_synonym\_of\_st 494
  - stm\_r\_st\_target\_of\_tr 494
  - stm\_r\_st\_top\_level\_in\_ch 489
  - stm\_r\_st\_unresolved\_in\_ch 489
  - stm\_r\_st\_unresolved\_st 494
  - stm\_r\_st\_using\_mx 490
  - stm\_r\_tc\_defined\_in\_ch 495
  - stm\_r\_tr\_affecting\_mx 496
  - stm\_r\_tr\_by\_attributes\_tr 472
  - stm\_r\_tr\_containing\_bt 454
  - stm\_r\_tr\_default\_of\_st 497
  - stm\_r\_tr\_default\_tr 497
-



- stm\_r\_tr\_from\_source\_mx 496
- stm\_r\_tr\_from\_source\_st 497
- stm\_r\_tr\_to\_target\_cn 495
- stm\_r\_tr\_to\_target\_mx 496
- stm\_r\_tr\_to\_target\_st 497
- stm\_r\_tr\_using\_mx 496
- stm\_r\_tt\_num\_of\_col 275
- stm\_r\_tt\_num\_of\_in 276
- stm\_r\_tt\_num\_of\_out 277
- stm\_r\_tt\_num\_of\_row 278
- stm\_r\_tt\_row 300
- stm\_r\_uc\_associates\_uc 479
- stm\_r\_uc\_explicit\_defined\_uc 479
- stm\_r\_uc\_ext\_point\_def 235
- stm\_r\_uc\_num\_of\_scen 279
- stm\_r\_uc\_scen 301
- stm\_r\_uc\_scen\_attr\_name 302
- stm\_r\_uc\_scen\_attr\_val 303
- stm\_r\_xx 175
- stm\_r\_xx\_array\_index 186
- stm\_r\_xx\_array\_rindex 187
- stm\_r\_xx\_attr\_enforced 188
- stm\_r\_xx\_attr\_name 190
- stm\_r\_xx\_attr\_val 192
- stm\_r\_xx\_cbk\_binding\_expression\_hyper 239
- stm\_r\_xx\_expression 231
- stm\_r\_xx\_ext\_link 233
- stm\_r\_xx\_instance\_name 248
- stm\_r\_xx\_keyword 250
- stm\_r\_xx\_labels 254
- stm\_r\_xx\_labels\_hyper 256
- stm\_r\_xx\_longdes 258
- stm\_r\_xx\_max\_val 262
- stm\_r\_xx\_min\_val 263
- stm\_r\_xx\_mini\_spec 264
- stm\_r\_xx\_mode 266
- stm\_r\_xx\_name 269
- stm\_r\_xx\_notes 273
- stm\_r\_xx\_number\_of\_bits 280
- stm\_r\_xx\_of\_enum\_type 281
- stm\_r\_xx\_of\_enum\_type\_name\_type 282
- stm\_r\_xx\_reactions 293
- stm\_r\_xx\_select\_implementation 306
- stm\_r\_xx\_string\_length 310
- stm\_r\_xx\_structure\_type 311
- stm\_r\_xx\_synonym 316
- stm\_r\_xx\_truth\_table 320
- stm\_r\_xx\_truth\_table\_expression 321
- stm\_r\_xx\_type 323
- stm\_r\_xx\_type\_expression 327
- stm\_r\_xx\_uniquename 328
- stm\_r\_xx\_user\_type 332
- stm\_r\_xx\_user\_type\_name\_type 333
- stm\_replace\_string 561
- stm\_replace\_word 562
- stm\_rpt\_attribute 141
- stm\_rpt\_dictionary 141
- stm\_rpt\_interface 142
- stm\_rpt\_list 143
- stm\_rpt\_n2chart 143
- stm\_rpt\_protocol 144
- stm\_rpt\_resolution 145
- stm\_rpt\_structure 145
- stm\_sort\_by\_levels 539
- stm\_str\_list\_first\_element 565
- stm\_str\_list\_last\_element 566
- stm\_str\_list\_length 567
- stm\_str\_list\_next\_element 568
- stm\_str\_list\_previous\_element 569
- stm\_str\_list\_to\_str 571
- stm\_str\_to\_list 572
- stm\_string\_extract 573
- stm\_string\_retain 575
- stm\_string\_to\_int 576
- stm\_strlen 577
- stm\_table\_simple 152
- stm\_trigger\_of\_reaction 578
- STOP statement 151
- stop statement 73
- String
  - convert to integer 576
  - extraction 573
  - first element 565
  - first in list 565
  - last element 566
  - length 577
  - next element 568
  - next in list 568
  - previous element 569
  - searching for a given pattern 512
  - searching for in list 516
- string expressions 45
- string values 41
- Structure report 145
- structure report 61
- structure statements 40, 48
  - begin/end statement 48, 49
  - comment statement 48
  - procedure statement 48, 49
  - segment statement 48
  - template statement 48
- Subroutine, retrieving lists of 482
- SUD specifications 75
- Synonym, sorting by 543

## T

- Table statement 152
- Template
  - act\_interface 95
  - compiling 20
  - creating 15
  - data-item properties 83
  - embedding instructions 10



---

- file access 9
- formatting commands 5
- icon 15
- Interleaf 105
- parameters 132
- reusing 8
- sample 5
- segment section 87
- standard 9
- statement 155
- structure 86
- template statement 48
- Timing constraint
  - retrieving list of 495
- Transition, retrieving list of 495
- tree report 61
- Trigger
  - of reaction 578
- Type
  - extracting elements by 520
  - Stemate 77

## U

- unary operations 44
- Unformatted segment 24
- User-defined type
  - retrieving list of 411

- Utility function
  - calling 500
  - examples 505
  - input arguments 504
  - list of 506
  - overview 499
- Utility functions 39

## V

- Value
  - return 80
- Variable
  - statement 156
- variable statement 52
- variables 41
- verbatim inclusion 39
- Verbatim statement 55, 158

## W

- WHILE/LOOP statement 72, 160
- WRITE statement 56, 162
  - producing messages 164
- write statement
  - using write to produce messages 58

