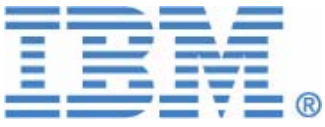
Rational. Statemate

Documentor Reference Guide

Rational 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 IBM® Rational® Statemate® 4.6 and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 1997, 2009.

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

Contents

Overview of Documentor 1

Basic Concepts 1
- Document Generation Language (DGL) 3
  - DGL Template 3
  - DGL Segments 3
  - Document Assembly 3

Designing a Document Using Templates 4
- 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

Documentor Interface with Formatting Systems 10
- Embedding Formatting Instructions 10
- Predefined Reports 10
- Plots 10
- Invoking a Formatter from Within the Documentor 11

Using Documentor 13

Document Production Process 13

Starting Documentor 14

Producing the Document Template 14
- Creating and Manipulating Templates 14
- Creating a Template 14
- Editing a Template 15
- Deleting a Template 16
- Copying a Template 17
- Exporting a Template 17

Compiling a Template . . . 18
Printing a Template . . . 19
Using Include Files . . . 19
Creating and Manipulating Include Files . . . 19
Creating an Include File . . . 20
Editing an Include File . . . 20
Deleting an Include File . . . 21
Copying an Include File . . . 21
Exporting an Include File . . . 21
Printing an Include File . . . 21
Producing the Document Segments . . . 22
Creating and Manipulating Documents . . . 22
Creating a Document . . . 23
Editing a Document . . . 26
Deleting a Document . . . 26
Regenerating Document Segments . . . 27
Producing the Final Document . . . 29
Printing a Document . . . 29
Exporting a Document . . . 30
Formatting a Document . . . 31
Working with Different Formatters . . . 31
Using nroff and troff . . . 32
Using Interleaf . . . 32
Document Templates . . . 33
Principles of DGL . . . 33
DGL Template Structure . . . 33
DGL Syntax Rules . . . 35
Special Features of DGL . . . 37
Extensions to Conventional Programming Constructs . . . 37
Database Extractions . . . 37
Overview of DGL Statements . . . 38
Data-types and Expressions . . . 39
Data-types . . . 39
Conventional Types found in Other Programming Languages . . . 39
Rational Statemate Element Types . . . 40
LIST OF simple_type . . . 41
Expressions . . . 42
Numeric Expressions . . . 42
Boolean Expressions . . . 43
String Expressions . . . 43
Rational Statemate Element Expressions . . . 44

List Expressions . . . 44
Enumerated Types—Predefined Constants . . . 45
DGL Statements . . . 46
Structure Statements . . . 46
TEMPLATE Statement . . . 46
SEGMENT Statement . . . 46
PROCEDURE Statement . . . 47
BEGIN/END Statement . . . 47
Comment Statement . . . 47
Declaration Statements . . . 48
PARAMETER Statement . . . 48
CONSTANT Statement . . . 49
VARIABLE Statement . . . 50
Assignment Statement . . . 51
File Handling Statements . . . 51
OPEN Statement . . . 51
CLOSE Statement . . . 52
READ Statement . . . 52
Output Statements . . . 53
Verbatim Statement . . . 53
WRITE Statement . . . 54
Using WRITE to Produce Messages . . . 56
INCLUDE Statement . . . 56
EXECUTE Statement . . . 57
Include Reports Statement . . . 57
Include Plots Statement . . . 63
Include Table Statement . . . 66
Control Flow Statements . . . 68
IF/THEN/ELSE Statement . . . 68
SELECT/WHEN Statement . . . 68
FOR/LOOP Statement . . . 70
WHILE/LOOP Statement . . . 71
EXIT Statement . . . 71
STOP Statement . . . 72

Documentor Functions . . . 73

Overview of the Extraction Functions . . . 73
Function Structure . . . 74
Using Database Extraction Functions . . . 74
Calling Conventions . . . 75
Function Names . . . 75
Element Type Abbreviations . . . 75
Arrow Elements . . . 77

Function Input Arguments . . . 78
Status Codes . . . 78
Function Return Values . . . 79
Return Values of Type ELEMENT . . . 79
Return Values of Filename . . . 79
Return Values of Enumerated Types . . . 80

Model Templates . . . 81

Properties . . . 81
Properties Template Structure . . . 84
Properties Initiation Section . . . 84
Declaration Part . . . 84
Body . . . 85
Properties Segment Section . . . 85
Generating the Report Heading . . . 86
Iteration: Using the FOR/LOOP Statement . . . 86
Generating the Entry Heading . . . 87
Extracting and Printing Information from the Data-Item Form . . . 87
Description and Synonym . . . 87
Using the SELECT/WHEN Construct . . . 87
Using Nested FOR Loops to Extract Attribute Names and Values . . . 88
Using Keywords to Write Portions of the Long Description . . . 89
Final Output for Data-item Properties . . . 90

Activity Interface Report . . . 91
act_interface Template . . . 92
Activity Interface Report Initiation Section . . . 94
Activity Interface Report Segment Section . . . 94
Declarations . . . 94
Producing the Headings . . . 95
Building Element Lists . . . 95
Alphabetizing and Sorting the List . . . 95
Writing the Input Elements . . . 96
Final Output for Act_Interface Report . . . 97

Template for nroff . . . 97
Template with nroff Commands . . . 98
Initiation Section (nroff) . . . 99
Segment 1: Heading and Report Overview (nroff) . . . 100
Including Global Declarations . . . 100
Producing the Heading and Overview (nroff) . . . 100
Segment 2: Activity-Chart Plot and Property Report (nroff) . . . 101

Template for Interleaf . . . 102
Initiation Section (Interleaf) . . . 103

Segment 1: Heading and Report Overview (Interleaf) . . . 103
Producing the Heading and Overview (Interleaf) . . . 104
Segment 2: Activity-Chart Plot and Property Report (Interleaf) . . . 105

Single-Element Functions . . . 107

Calling Single-Element Functions . . . 108
Single-Element Function Input Arguments . . . 109
Examples of Single-Element Function Calls . . . 110
Single-Element Function Example 1 . . . 110
Single-Element Function Example 2 . . . 110
Single-Element Function Example 3 . . . 111

List of Functions . . . 111
stm_r_xx . . . 117
stm_r_sb_action_lang . . . 119
stm_r_sb_action_lang_expression . . . 120
stm_r_sb_action_lang_local_data . . . 121
stm_r_actual_parameter_exp . . . 122
stm_r_actual_parameter_type . . . 123
stm_r_elem_in_ddb_list . . . 124
stm_r_sb_ada_user_code . . . 125
stm_r_sb_ansi_c_user_code . . . 126
stm_r_st_combinationals . . . 127
stm_r_xx_array_lindex . . . 128
stm_r_xx_array_rindex . . . 129
stm_r_xx_attr_enforced . . . 130
stm_r_xx_attr_name . . . 132
stm_r_xx_attr_val . . . 134
stm_r_xx_bit_array_lindex . . . 137
stm_r_xx_bit_array_rindex . . . 138
stm_r_xx_cbk_binding . . . 139
stm_r_xx_cbk_binding_enable . . . 140
stm_r_xx_cbk_binding_expression . . . 142
stm_r_xx_cbk_binding_expression_hyper . . . 143
stm_r_tt_cell . . . 144
stm_r_tt_cell_type . . . 145
stm_r_changes_log . . . 147
stm_r_xx_chart . . . 148
stm_r_xx_combinationals . . . 151
stm_r_sb_connected_chart . . . 152
stm_r_xx_containing_fields . . . 153
stm_r_ch_creation_date . . . 154
stm_r_ch_creator . . . 155
stm_r_xx_data_type . . . 156

stm_r_rt_date 157
stm_r_xx_definition_type 158
stm_r_xx_desc_file 161
stm_r_xx_description 162
stm_r_design_attr 164
stm_r_xx_displayed_name 165
stm_r_ddb_list_names 167
stm_r_element_type 168
stm_r_xx_expr_hyper 170
stm_r_xx_expression 171
stm_r_xx_ext_link 173
stm_r_uc_ext_point_def 175
stm_r_formal_parameter_names 176
stm_r_sb_global_data 177
stm_r_global_interface_report 178
stm_r_xx_cbk_binding_expression_hyper 179
stm_r_xx_graphic 180
stm_r_hyper_key 182
stm_r_md_implementation 183
stm_r_included_gds 184
stm_r_msg_included_in_ord_insig 185
stm_r_cd_info 186
stm_r_inherited_gds 187
stm_r_xx_instance_name 188
stm_r_xx_keyword 190
stm_r_sb_kr_c_user_code 193
stm_r_xx_labels 194
stm_r_xx_labels_hyper 196
stm_r_local_interface_report 197
stm_r_xx_longdes 198
stm_r_lookup_table_header 200
stm_r_xx_max_val 201
stm_r_xx_min_val 202
stm_r_xx_mini_spec 203
stm_r_ac_mini_spec_hyper 204
stm_r_xx_mode 205
stm_r_ch_modification_date 206
stm_r_xx_name 207
stm_r_next_msg 210
stm_r_xx_note 211
stm_r_xx_notes 212
stm_r_tt_num_of_col 213
stm_r_tt_num_of_in 214
stm_r_tt_num_of_out 215

stm_r_tt_num_of_row 216
stm_r_uc_num_of_scen 217
stm_r_xx_number_of_bits 218
stm_r_xx_of_enum_type 219
stm_r_xx_of_enum_type_name_type 220
stm_r_ord_insig_defined_in_ch 222
stm_r_parameter_binding 223
stm_r_parameter_mode 224
stm_r_sb_parameters 225
stm_r_en_parent 226
stm_r_previous_msg 227
stm_r_sb_proc_sch_local_data 228
stm_r_md_purpose 229
stm_r_xx_reactions 230
stm_r_param_binding_hyper 232
stm_r_param_binding_id 233
stm_r_sb_return_type 234
stm_r_sb_return_user_type 235
stm_r_sb_return_user_type_name 236
stm_r_tt_row 237
stm_r_uc_scen 238
stm_r_uc_scen_attr_name 239
stm_r_uc_scen_attr_val 240
stm_r_sd_scope 242
stm_r_xx_select_implementation 243
stm_r_st_static_reactions 244
stm_r_st_static_reactions_hyper 245
stm_r_xx_string_length 246
stm_r_xx_structure_type 247
stm_r_ac_subroutine_bind 249
stm_r_ac_subroutine_bind_enable 250
stm_r_ac_subroutine_bind_expr 251
stm_r_xx_synonym 252
stm_r_ac_termination 254
stm_r_xx_truth_table 256
stm_r_xx_truth_table_expression 257
stm_r_sb_truth_table_local_data 258
stm_r_xx_type 259
stm_r_xx_type_expression 263
stm_r_xx_uniquename 264
stm_r_ch_usage_type 267
stm_r_xx_user_type 267
stm_r_xx_user_type_name_type 269
stm_r_ch_version 270

stm_r_gds_visibility_mode . . . 271
stm_r_msg_where_tc_begins . . . 272
stm_r_msg_where_tc_ends . . . 273
stm_r_sb_connected_statechart . . . 274
stm_r_sb_connected_flowchart . . . 275
stm_r_sb_proc_fch_local_data . . . 276
stm_r_xx_des_attr_val . . . 277
stm_r_xx_des_attr_name . . . 279
stm_r_tt_cell_hyper . . . 281
stm_r_tt_row_hyper . . . 282
stm_r_xx_default_val . . . 283
stm_r_component_param_binding . . . 284
stm_r_component_param_mode . . . 285
stm_r_stubs_names . . . 286
stm_r_information_stub_names . . . 287
stm_r_sb_connected_statechart . . . 288
stm_r_sb_connected_flowchart . . . 289

DGL Statement Reference . . . 291

ASSIGNMENT . . . 294
BEGIN . . . 295
CLOSE . . . 297
COMMENT . . . 298
CONSTANT . . . 299
END . . . 301
EXECUTE . . . 302
EXIT . . . 303
FOR/LOOP . . . 304
IF/THEN/ELSE . . . 306
INCLUDE . . . 308
OPEN . . . 310
PARAMETER . . . 312
PROCEDURE . . . 314
READ . . . 318
REPORT . . . 319
Attribute Report . . . 321
Property Report . . . 321
Interface Report . . . 322

List Report . . . 323
N2 Chart Report . . . 323
Protocol Report . . . 324
Resolution Report . . . 325
Structure Report . . . 325
Tree Report . . . 326
SEGMENT . . . 327
SELECT/WHEN . . . 328
STOP . . . 331
TABLE . . . 332
TEMPLATE . . . 335
VARIABLE . . . 336
VERBATIM . . . 338
WHILE/LOOP . . . 340
WRITE . . . 342
Query Functions . . . 345
Calling Query Functions . . . 346
By Attributes . . . 346
By Structure Type . . . 347
Name and Synonym Patterns . . . 348
Query Function Input Arguments . . . 349
Examples of Query Functions . . . 350
Query Function Example 1 . . . 350
Query Function Example 2 . . . 350
Query Function Example 3 . . . 351
List of Query Functions . . . 351
Activities (ac) . . . 352
Input List Type: ac . . . 352
Input List Type: af . . . 359
Input List Type: ch . . . 361
Input List Type: ds . . . 362
Input List Type: md . . . 362
Input List Type: mx . . . 363
Input List Type: router . . . 363
Input List Type: st . . . 364
A-Flow-Lines (af, ba, laf) . . . 365
Output List Type: af . . . 365
Output List Type: laf . . . 370

Actions (an) 372
Input List Type: an 372
Input List Type: ch 374
Charts (ch) 375
Input List Type: ac 375
Input List Type: an 375
Input List Type: ch 376
Input List Type: co 378
Input List Type: di 379
Input List Type: ds 379
Input List Type: dt 379
Input List Type: ev 380
Input List Type: fd 380
Input List Type: if 380
Input List Type: md 381
Input List Type: mx 382
Input List Type: router 382
Input List Type: sb 383
Input List Type: st 383
Connectors (cn) 384
Input List Type: cn 384
Input List Type: st 385
Input List Type:bm 385
Input List Type: tr 386
Conditions (co) 386
Input List Type: af 386
Input List Type: ch 387
Input List Type: co 387
Input List Type: di 389
Input List Type: if 389
Input List Type: mf 389
Data-Items (di) 390
Input List Type: af 390
Input List Type: ch 390
Input List Type: co 391
Input List Type: di 391
Input List Type: fd 396
Input List Type: if 397
Input List Type: mf 397
Data-Stores (ds) 398
Input List Type: ac 398
Input List Type: af 398
Input List Type: ch 399
Input List Type: ds 399

Input List Type: md 400
User-Defined Types (dt) 401
Input List Type: ch 401
Input List Type: dt 402
Input List Type: fd 407
Events (ev) 408
Input List Type: af 408
Input List Type: ch 408
Input List Type: ev 409
Input List Type: if 410
Input List Type: mf 411
Fields (fd) 411
Input List Type: ch 411
Input List Type: di 411
Input List Type: dt 412
Input List Type: fd 412
Input List Type: mx 416
Functions (fn) 417
Input List Type: ch 417
Information-Flows (if) 418
Input List Type: af 418
Input List Type: ch 419
Input List Type: co 419
Input List Type: di 420
Input List Type: ev 420
Input List Type: if 421
Input List Type: mf 422
M-Flow-Lines (bf, bm, lmf, mf) 423
Output List Type: bf 423
Output List Type: bm 426
Output List Type: lmf 427
Output List Type: mf 428
Modules (md) 433
Input List Type: ac 433
Input List Type: ch 433
Input List Type: ds 434
Input List Type: md 435
Input List Type: mf 439
Input List Type: router 439
Mixed (mx) 440
Input List Type: af 440
Input List Type: ac 441
Input List Type: actor 442
Input List Type: an 443

Input List Type: bb 444
Input List Type: bt 444
Input List Type: bm 446
Input List Type: ch 447
Input List Type: co 449
Input List Type: di 450
Input List Type: ds 451
Input List Type: dt 451
Input List Type: ev 452
Input List Type: fd 453
Input List Type: fn 454
Input List Type: if 454
Input List Type: md 455
Input List Type: mf 456
Input List Type: msg 456
Input List Type: mx 457
Input List Type: router 465
Input List Type: sb 466
Input List Type: st 466
Input List Type: tr 468
Input List Type: uc 469
Routers (router) 469
Input List Type: ac 469
Input List Type: af 470
Input List Type: ch 470
Input List Type: md 471
Input List Type: router 471
Subroutines (sb) 472
Input List Type: ch 472
Input List Type: sb 473
States (st) 478
Input List Type: ac 478
Input List Type: ch 479
Input List Type: cn 480
Input List Type: mx 480
Input List Type: st 481
Input List Type: tr 484
Timing Constraint (tc) 485
Input List Type: ch 485
Transitions (tr) 485
Input List Type: cn 485
Input List Type: mx 486
Input List Type: st 487
Input List Type: tr 487

Utility Functions 489

Calling Utility Functions 490
Contains Element 490
List Extraction by Type 491
List Extraction by Chart 492
Location of Pattern in a String 492
Extract Portion of a String 493

Utility Function Input Arguments 494

Examples of Utility Functions 495
Utility Functions Example 1 495
Utility Functions Example 2 495
Utility Functions Example 3 496

List of Utility Functions 496
stm_action_of_reaction 499
stm_delete_file 500
stm_dispose_memory 501
stm_index 502
stm_int 503
stm_int_to_string 503
stm_r_is_statemate 504
stm_list_contains_element 504
stm_list_contains_string 506
stm_list_extraction 507
stm_list_extraction_by_chart 508
stm_list_extraction_by_chart_id 509
stm_list_extraction_by_type 510
stm_list_first_element 512
stm_list_last_element 513
stm_list_length 514
stm_list_next_element 516
stm_plot_ext 518
stm_list_previous_element 522
stm_list_sort 524
stm_list_sort_by_attr_value 525
stm_list_sort_by_branches 527
stm_list_sort_by_chart 528
stm_list_sort_by_levels 529
stm_list_sort_by_name 531
stm_list_sort_by_synonym 533
stm_list_sort_by_type 535
stm_multiline_to_one 536
stm_multiline_to_strings 536

stm_plot . . . 537
stm_plot_hyper_exp . . . 538
stm_plot_with_autonumber . . . 541
stm_plot_with_break . . . 544
stm_plot_with_headerline . . . 548
stm_replace_string . . . 551
stm_replace_word . . . 552
stm_set_rpt_formator . . . 553
stm_string_retain . . . 554
stm_str_list_first_element . . . 555
stm_str_list_last_element . . . 556
stm_str_list_length . . . 557
stm_str_list_next_element . . . 558
stm_str_list_previous_element . . . 559
stm_str_list_to_str . . . 561
stm_str_to_list . . . 562
stm_string_extract . . . 563
stm_string_free . . . 564
stm_string_retain . . . 565
stm_string_to_int . . . 566
stm_strlen . . . 567
stm_trigger_of_reaction . . . 568

Project Management . . . 569

stm_r_pm_member_workareas . . . 570
stm_r_pm_operator_projects . . . 571
stm_r_pm_project_databank . . . 572
stm_r_pm_project_manager . . . 573
stm_r_pm_project_members . . . 574
stm_r_pm_projects . . . 575

Function Status Codes . . . 577

DGL Reserved Words . . . 583

BNF Syntax . . . 585

BNF Structure and Conventions . . . 585
Symbol Types . . . 585
BNF Notations . . . 586
BNF for DGL Statements . . . 586

Index 591

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:

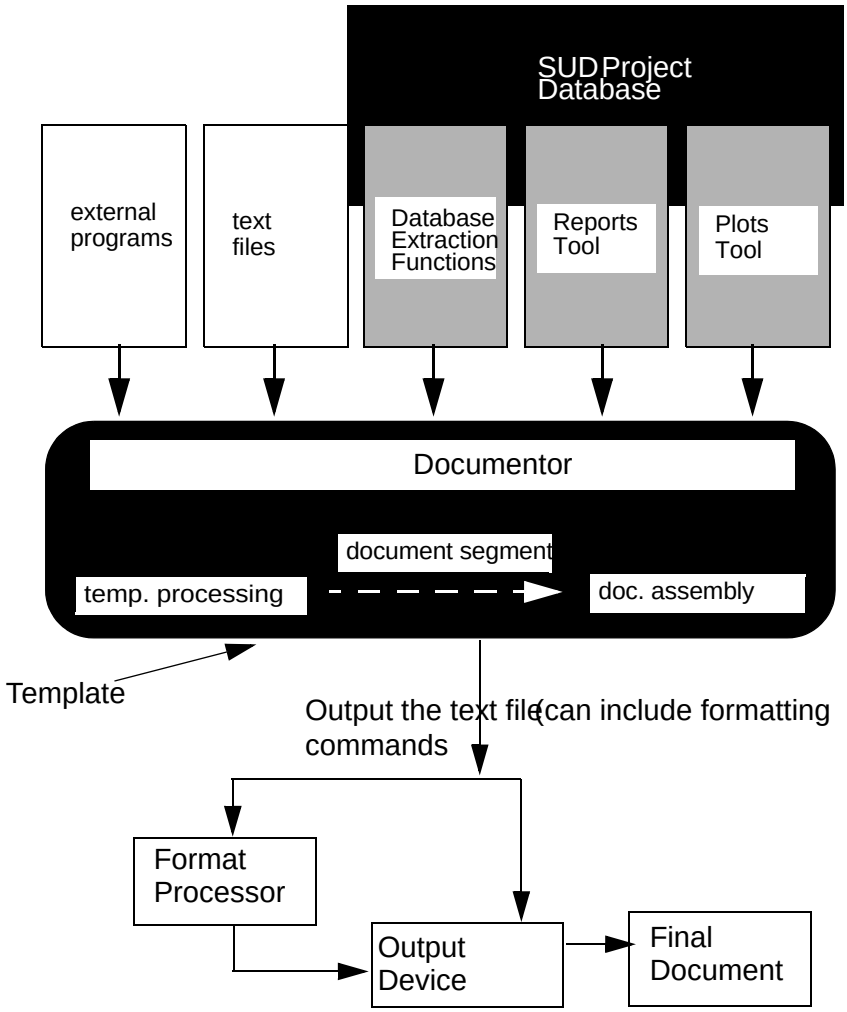
- Basic Concepts
- Designing a Document Using Templates
- Documentor Interface with Formatting Systems

Basic Concepts

You can use the Documentor to design format templates 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 in 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:

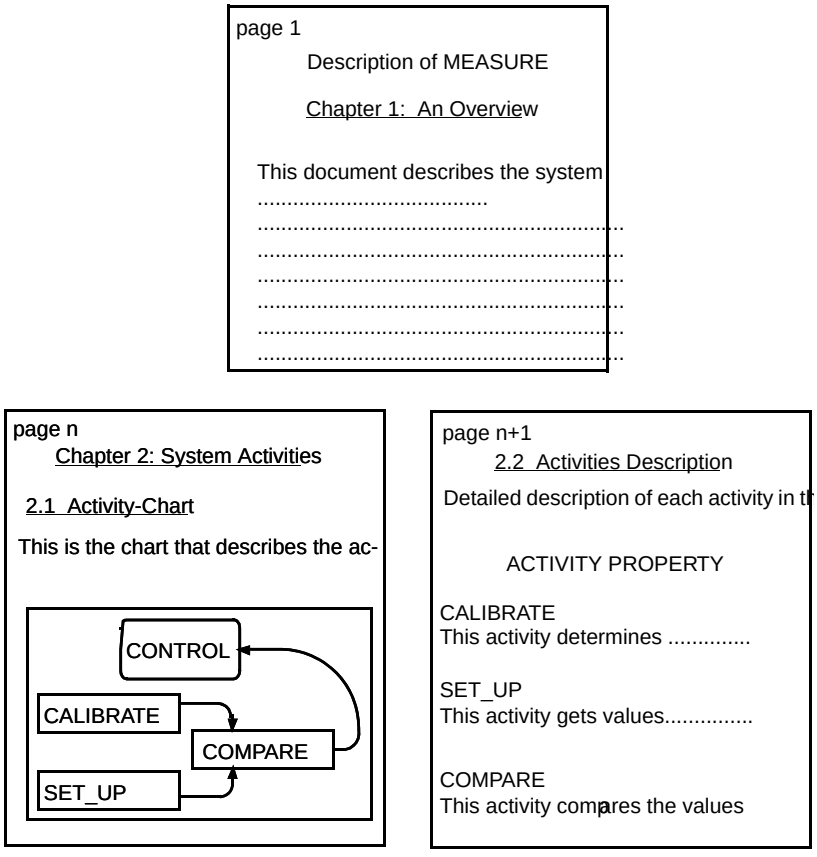
- Exports the segments to an external output device or file.
- Sends the segments to a formatting system to produce the final document.

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

Sample Template

In a template, statements beginning with "- -" are comments and are not interpreted by the tool. Formatting instructions and other text written between /@ 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 segments or 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 /@ 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 SYSDOC 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 activity vs. time 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 filename. The file is accessed via the INCLUDE statement within the template and inserted in the segment files when the template is executed.

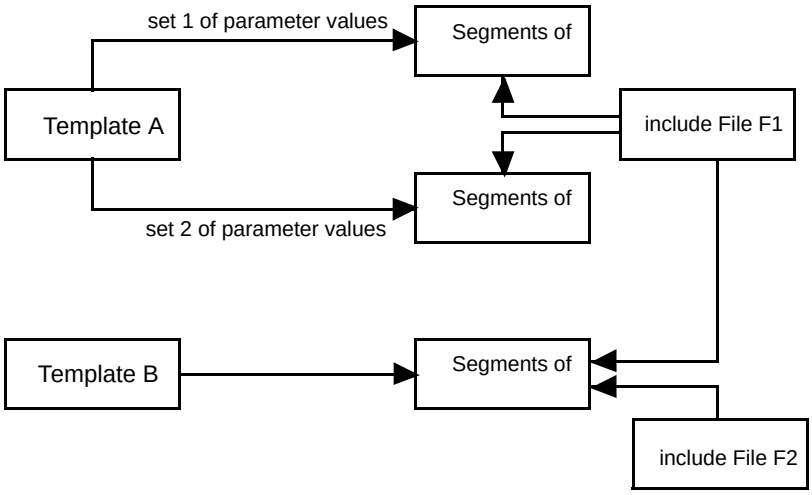
The same include files can be used in a number of different documents, and can be called from a number of different templates.

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, 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 used for document generation are stored in your workarea. You create and edit these files in your workarea using the Documentor.

You can store your templates and include files in the databank so they can be shared with other project members, and for version management purposes. In addition, you can export these files from, and import these files to, outside the system using standard storage functions.

There are templates and include files used to generate standardized documents within a company (or even within an industry) whose use is independent of any particular project. These common files must first be imported to your workarea either directly or via the databank.

Standardized templates, such as templates for the DOD-STD-2167A document set, can be supplied with Rational Statemate. You can import these templates to the project databank when the project is created, or select File > Import in the workarea browser.

The resulting segments and documents that the tool creates are not stored in the databank, but reside in your workarea. These are handled the same way as other stored files on your system.

Documentor Interface with Formatting Systems

When you create a template, you can assign a format 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, see 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 the 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, see 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 segments to the 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
Writing the template Creating new templates and editing, deleting, copying, exporting, compiling, and printing existing templates.	Edit > Templates
Preparing include files Creating new include files and editing, deleting, copying, exporting, and printing existing include files.	Edit > Include File
Producing document segments 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.	Edit > Documents
Producing the final document 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.	Edit > Documents

Starting Documentor

To start the Documentor from within the system:

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

   In addition, the Document Management dialog box opens simultaneously so you can easily manage your documents (see 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.

Creating a Template

To create a template:

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. (See 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:

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:

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:

1. Select the template from the list in the Template Management dialog box.
2. Click Copy. The Copy Template dialog box opens showing 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:

1. Select the template from the list in the Template Management dialog box.
2. Click Export. The Export Files dialog box opens.
3. 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 (see Creating a Document).

To compile a template:

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:

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

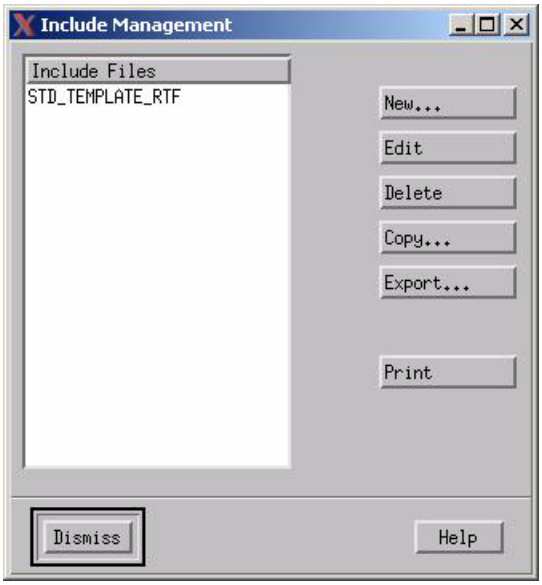
To print multiple templates, hold down the CTRL key when making your selections.

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.

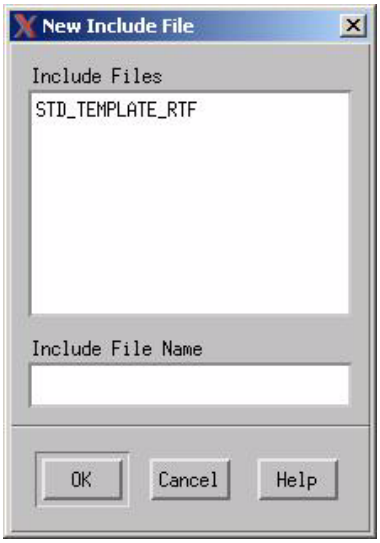


The dialog box lists the include files in your workarea.

Creating an Include File

To create an include file:

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:

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:

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:

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:

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:

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 formatted 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.

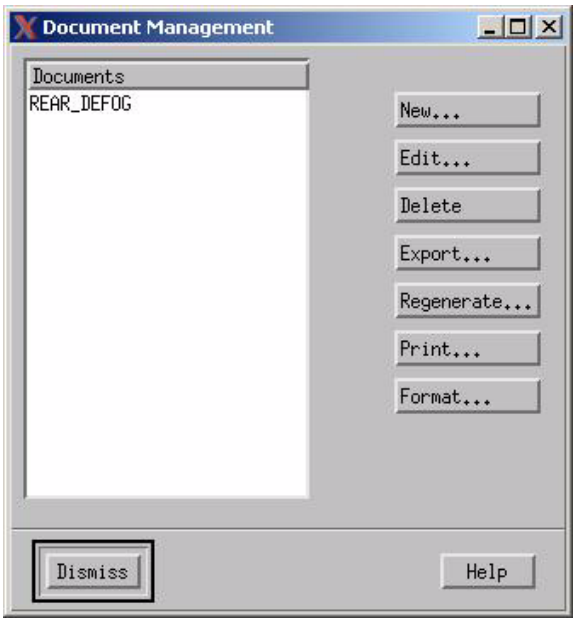
This section describes how to use the following options in the Document Management dialog box:

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

For information on the Export, Print, and Format options, see Producing the Final Document.

Creating and Manipulating Documents

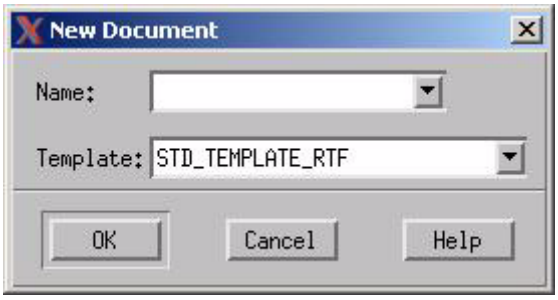
To work with documents, click the Documents icon in the toolbar or select Edit > Documents from the main menu. The Document Management dialog box lists the documents in your workarea.



Creating a Document

To create a document:

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 the 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. See Creating a Template for more information. When all 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.

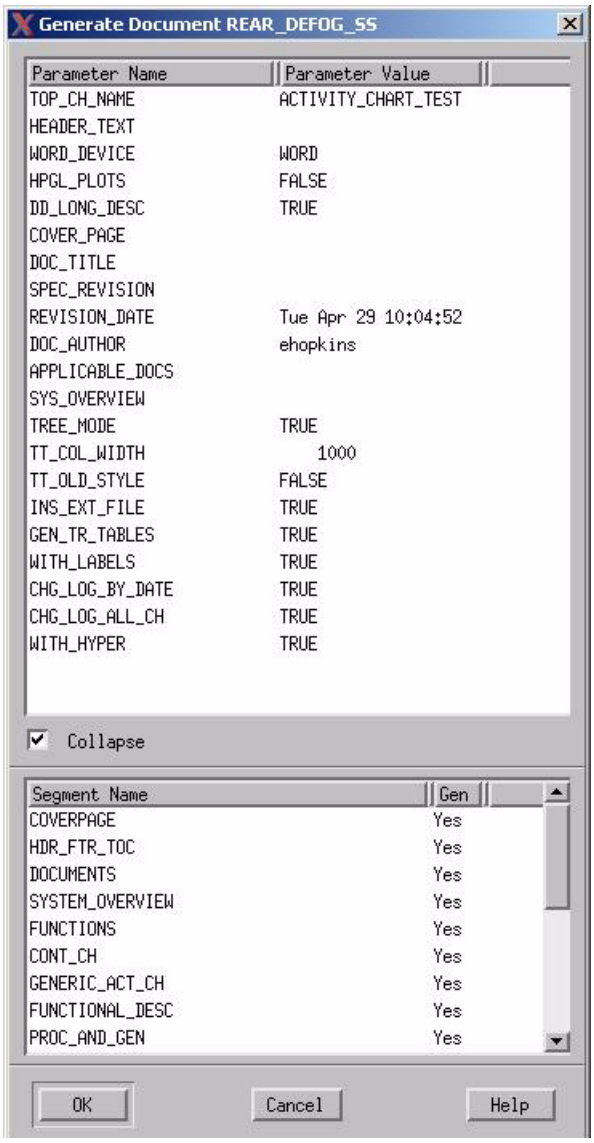
      Regardless of the template segments that you select for execution, the initiation section of the template is always executed. See Overview of Documentor for an explanation of template segments.

5. Enter values for the relevant parameters shown in the form. (For more information on template parameters, see 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)

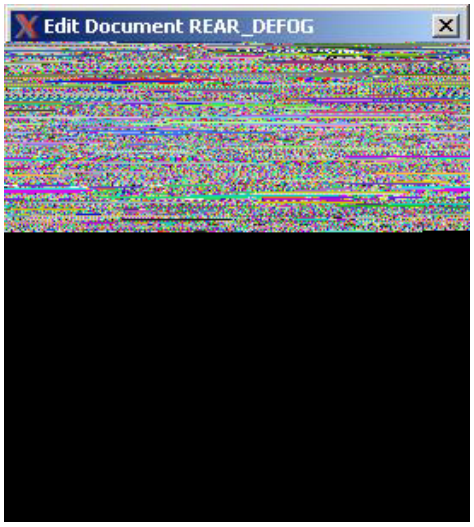


Editing a Document

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:

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:

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.

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_flowing_through_af(af_list,st);

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

 -- get input data_items
di_list := stm_r_di_flowing_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_flowing_through_af (af_list, st);

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

-- get output data_items
di_list := stm_r_di_flowing_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 activity-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 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 database 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 c_list
- A list of input events, represented by the variable e_list
- A list of input data-items, represented by the variable d_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 id_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 id_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 `SYSTEM_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 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 StateStat uses underscore 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 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 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 (nroff)

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 nroff. 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

nroff 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 nroff 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 nroff 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 host 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. It 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 plotter, and places it in a file called ac_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, see 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 template 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; see 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 before each Interleaf command and 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 (act_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).
<Autonum, List8, 2><TAB>System Activities -- writes
                       --  section header.
<subsect> -- starts a subsection.
<Autonum, 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 <subsect>, 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 it. To include the plot in the output file, you supply an empty string as the second parameter in the 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, see the 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 segment. These include instructions for a new page, a subheading, and a sentence introducing the property report. Next, you assign a value to the variable sub_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 format 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).

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. See 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 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
name	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
element ID	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).
attribute name	Name of an attribute defined in the form for a Statemate element (in the attribute field).	String
begin keyword	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
end keyword	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
filename	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, 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 value 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 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 and 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
stm_r_sb_action_lang	Returns the action language of the specified subroutine.
stm_r_sb_action_lang_expression	Returns the action language expression of the specified subroutine.
stm_r_sb_action_lang_local_data	Returns the action language local data associated with the specified subroutine.
stm_r_actual_parameter_exp	Returns the actual binding of the formal parameter name in the specified instance chart or component.
stm_r_actual_parameter_type	Returns the type of the formal parameter name in the specified instance chart or component.
stm_r_elem_in_ddb_list	Searches for the specified element in the list created by the properties browser.
stm_r_sb_ada_user_code	Returns the Ada code that was manually written by the user for the specified subroutine.

stm_r_sb_ansi_c_user_code	Returns the ANSI C code that was manually written by the user for the specified subroutine.
stm_r_xx	Retrieves the element ID of the specified element.
stm_r_xx_array_lindex	Returns the left index of an element array.
stm_r_xx_bit_array_rindex	Returns the right index of an element array.
stm_r_xx_attr_enforced	Returns the enforced attributes specified by attr_name.
stm_r_xx_attr_name	Returns the names of attributes associated with the specified element.
stm_r_xx_attr_val	Returns the attribute values associated with a particular attribute name for the specified element.
stm_r_xx_bit_array_lindex	Returns the left index of a bit array.
stm_r_xx_bit_array_rindex	Returns the right index of a bit array.
stm_r_xx_cbk_binding	Retrieves the callback binding for the specified element.
stm_r_xx_cbk_binding_enable	Retrieves the enabled callback bindings for the specified element.
stm_r_xx_cbk_binding_expression	Retrieves the callback binding expression for the specified element.
stm_r_xx_cbk_binding_expression_hyper	Retrieves the callback binding expressions.
stm_r_tt_cell	Returns the contents of the specified cell in the given truth table.
stm_r_tt_cell_type	Returns the data-type of the specified cell in the given truth table.
stm_r_changes_log	Documents all the changes made to the specified charts in a log file.
stm_r_xx_chart	Returns the chart ID for the specified element.
stm_r_xx_combinationals	Returns a list of strings. Each element of this list holds one combinational assignment, which is connected to the specified element.
stm_r_sb_connected_chart	Returns the ID of the procedural statechart connected to the specified subroutine.
stm_r_xx_containing_fields	Returns a list of union or record elements that contain fields.
stm_r_ch_creation_date	Returns the date (as a string) on which the specified chart was created.
stm_r_ch_creator	Returns the name of the Rhapsody user who created the specified chart.
stm_r_xx_data_type	Returns the element subtype, including its data type and data structure.
stm_r_xx_definition_type	Returns the definition type of the specified textual element.
stm_r_xx_desc_file	Returns the description file for the specified element.
stm_r_xx_description	Returns the short description of the specified element.
stm_r_design_attr	Retrieves the design attributes of an element in a form of a list of strings.
stm_r_xx_displayed_name	Returns the name of the chart as it appears in the graphic editor where the specified element is located.
stm_r_ddb_list_names	Returns the names of the lists created by the properties browser.

stm_r_element_type	Returns the element type of the specified element.
stm_r_xx_expr_hyper	Returns the definition expression of the specified element found in the Definition field of the element's form, including hyperlinks to referenced elements.
stm_r_xx_expression	Returns the definition expression of the specified element found in the Definition field of the element's form.
stm_r_xx_ext_link	Retrieves the external file link for the specified element.
stm_r_uc_ext_point_def	Retrieves the extension point definitions for the specified use case diagram.
stm_r_formal_parameter_names	Retrieves the list of names of formal parameters that appear in the bindings of instance boxes and components.
stm_r_global_interface_report	Creates a global interface report from the specified input list.
stm_r_sb_global_data	Returns the global data associated with the specified subroutine.
stm_r_hyper_key	Retrieves the unique key for the specified element.
stm_r_md_implementation	Retrieves the implementation type for the specified module.
stm_r_included_gds	Returns the list of global definition sets contained in the specified chart.
stm_r_msg_included_in_ord_insig	Returns a list of messages that are bounded by an order-insignificant element.
stm_r_cd_info	Retrieves the description of the specified continuous chart.
stm_r_inherited_gds	Retrieves the list of global definition sets that are "inherited" (included indirectly) by the specified chart.
stm_r_xx_instance_name	Retrieves the name of an instance as it appears in the chart for the specified hierarchical Rhapsody element.
stm_r_xx_keyword	Returns a portion of the element's long description.
stm_r_sb_kr_c_user_code	Returns the K&R C code that was manually written by the user for the specified subroutine.
stm_r_xx_labels	Returns a list of strings that consists of all the labels of the specified compound transition.
stm_r_xx_labels_hyper	Returns a list of strings of message or transition labels, with hyperlinks to referenced elements.
stm_r_local_interface_report	Creates a local interface report from the specified input list.
stm_r_xx_longdes	Returns the long description attached to the specified element.
stm_r_xx_max_val	Returns the maximum value of the specified element.
stm_r_xx_min_val	Returns the minimum value of the specified element.
stm_r_xx_mini_spec	Returns a string with mini-spec reactions or actions.
stm_r_ac_mini_spec_hyper	Returns a string with the mini-spec, including hyperlinks to referenced elements.
stm_r_xx_mode	Returns the parameter or router mode.
stm_r_ch_modification_date	Returns the date on which the version of the chart in the workarea was saved to the database.
stm_r_xx_name	Returns the element name.

stm_r_next_msg	Returns the message after (in time) the decomposed sequence diagram.
stm_r_xx_note	Returns the notes in the specified element.
stm_r_tt_num_of_col	Retrieves the number of columns in the specified truth table.
stm_r_tt_num_of_in	Retrieves the number of input columns in the specified truth table.
stm_r_tt_num_of_out	Retrieves the number of output columns in the specified truth table.
stm_r_tt_num_of_row	Retrieves the number of rows in the specified truth table.
stm_r_uc_num_of_scen	Retrieves the number of scenarios for the specified use case.
stm_r_xx_number_of_bits	Returns the number of bits in the specified element.
stm_r_xx_of_enum_type	Returns the enumeration type ID (a user-defined type) for the specified element.
stm_r_xx_of_enum_type_name_type	Returns the enumerated name and type for the specified element.
stm_r_ord_insig_defined_in_ch	Returns the origin of the specified requirement record.
stm_r_parameter_binding	Returns the parameter expression from the generic charts and components.
stm_r_parameter_mode	Retrieves the parameter mode, including subroutine parameters and the parameters of generic charts and components.
stm_r_sb_parameters	Returns the parameters of the specified subroutine.
stm_r_en_parent	Returns the local data of the procedural statechart implemented by the specified subroutine.
stm_r_previous_msg	Returns the message previous (in time) to the decomposed sequence diagram.
stm_r_sb_proc_sch_local_data	Retrieves the local data of the procedural statechart implemented by the specified subroutine.
stm_r_md_purpose	Retrieves the purpose of the module.
stm_r_xx_reactions	Returns the static reactions of the specified state.
stm_r_sb_return_type	Retrieves the subroutine's return type.
stm_r_sb_return_user_type	Returns the user-defined type ID returned by the specified subroutine.
stm_r_sb_return_user_type_name	Retrieves the subroutine's return user type and name type.
stm_r_tt_row	Retrieves the values in the specified row in the truth table.
stm_r_uc_scen	Retrieves the scenario, based on the specified index number.
stm_r_uc_scen_attr_name	Returns the names of attributes associated with the specified use case scenario.
stm_r_uc_scen_attr_val	Retrieves attribute values associated with a particular attribute name for the specified use case scenario.
stm_r_sd_scope	Retrieves the scope of the specified sequence diagram.
stm_r_xx_select_implementation	Returns the implementation type of the specified element.
stm_r_st_static_reactions	Returns the static reactions defined for the specified state element.

stm_r_st_static_reactions_hyper	Returns a string with the static reactions, including hyperlinks to referenced elements.
stm_r_xx_string_length	Returns the string length of the specified element.
stm_r_xx_structure_type	Returns the structure or type of the specified textual element.
stm_r_ac_subroutine_bind	Returns the subroutine binding connected to the specified activity.
stm_r_ac_subroutine_bind_enable	Determines whether the subroutine bound to the specified activity is enabled or disabled.
stm_r_ac_subroutine_bind_expr	Returns the subroutine binding expression connected to the specified activity.
stm_r_xx_synonym	Returns the synonym of the specified element.
stm_r_ac_termination	Returns the activity termination type specified in the activity's form.
stm_r_xx_truth_table	Returns the elements that are implemented as truth tables.
stm_r_xx_truth_table_expression	Returns the truth table expression for all the named elements.
stm_r_sb_truth_table_local_data	Returns a list of local data elements defined in truth tables that are related to the input subroutine.
stm_r_xx_type	Returns the element subtypes for the specified element.
stm_r_xx_type_expression	Returns the type expression for the specified element.
stm_r_xx_uniquename	Returns the unique path name for the specified element.
stm_r_ch_usage_type	Returns the usage type for the specified chart.
stm_r_xx_user_type	Returns the user-defined type ID referenced by the specified element.
stm_r_xx_user_type_name_type	Returns the name type of the user-defined type referenced by the specified element.
stm_r_ch_version	Returns the version of the specified chart.
stm_r_gds_visibility_mode	Returns the visibility mode for the specified global definition set (GDS).
stm_r_msg_where_tc_begins	Returns the message where the timing constraint begins.
stm_r_msg_where_tc_ends	Returns the message where the timing constraint ends.
stm_r_xx_chart	Returns the chart ID for the specified element.
stm_r_xx_longdes	Retrieves the long description attached to the specified element.
stm_r_xx_attr_name	Returns the names of attributes associated with the specified element. Attributes are associated with elements via element forms.
stm_r_xx_attr_val	Retrieves attribute values associated with a particular attribute name for the specified element.
stm_r_sb_connected_statechart	Returns the enforced attributes specified by attr_name.
stm_r_sb_connected_statechart	Returns the ID of the procedural Statechart connected to the specified subroutine.
stm_r_sb_connected_flowchart	Returns the ID of the Flowchart connected to the specified subroutine.
stm_r_sb_proc_fch_local_data	Retrieves the local data of the Flowchart implemented by the specified subroutine.

stm_r_xx_des_attr_val	Retrieves attribute values associated with a particular attribute name for the specified element.
stm_r_xx_des_attr_name	Returns the names of attributes associated with the specified element. Attributes are associated with elements via element forms.
stm_r_tt_cell_hyper	Retrieves the contents of the specified cell in the given truth table, including hyperlinks to referenced elements.
stm_r_xx_default_val	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.
stm_r_xx_default_val	Returns the default value associated with the specified element.
stm_r_component_param_binding	Returns the value bound to the formal parameter of the specified component's instance
stm_r_component_param_mode	Returns the mode of the formal parameter of the specified component's instance.
stm_r_stubs_names	Returns a list of stub names of the specified component instance.
stm_r_information_stub_names	Returns a list of stub names flowing through an info-flow of the specified component instance.

stm_r_xx

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 exttraction functions.

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

Syntax

stm_r_xx (name, status)

Arguments

Argument	Input/ Output	Type	Description
name	In	STRING	A Statemate element name or synonym. Note the following: • This can be an element name (path name) or synonym. Hierarchical elements must be identified uniquely by specifying a unique path name. • The name can include the chart name (for example, A:B). • The name is not case-sensitive.
status	Out	INTEGER	The function status code. The data type of the return value must be declared to be either a Statemate 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

Retrieves the action language of the specified subroutine.

Function type

LIST OF STRING

For elements

subroutine	sb

Syntax

stm_r_sb_action_lang (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_id_not_found
- stm_missing_statemate_action_lang
- stm_unresolved

stm_r_sb_action_ lang_expression

Retrieves the action language expression of the specified subroutine.

Function type

STRING

For elements

subroutine	sb

Syntax

stm_r_sb_action_lang_expression (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_statemate_action_lang
- stm_unresolved

stm_r_sb_action_lang_local_data

Retrieves the action language local data associated with the specified subroutine.

Function type

LIST OF STATEMATE ELEMENTS

For elements

subroutine	sb

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

Returns the actual binding of the formal parameter name in the specified instance chart or component.

Function type

INTEGER

For elements

activity	ac
condition	co
data-item	di
event	ev

Syntax

stm_r_actual_parameter_exp (xx_inst_boxid, formal_param_name, status)

Arguments

Argument	Input/ Output	Type	Description
inst_boxid	In	Statemate 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

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 .

Function type

STRING

For elements

activity	ac
condition	co
data-item	di
event	ev

Syntax

stm_r_actual_parameter_type (inst_boxid, formal_param_name, status)

Arguments

Argument	Input/ Output	Type	Description
inst_boxid	In	Statemate 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_elem_in_ddb_list

Searches for the specified element in the list created by the properties browser.

Function type

LIST OF ELEMENT stm_list

Syntax

stm_r_elem_in_ddb_list (list_name, &status)

Arguments

Argument	Input/Output	Type	Description
list_name	In	char * STRING	The input list of elements
status	Out	int INTEGER	Function status code

Status Codes

- stm_no_such_list
- stm_success

stm_r_sb_ada_user_code

Returns the Ada code that was manually written for the specified subroutine.

Function Type

LIST OF STRING

For elements

subroutine	sb

Syntax

stm_r_ada_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_sb_ansi_c_user_code

Returns the ANSI C code that was manually written for the specified subroutine.

Function type

LIST OF STRING

For elements

subroutine	sb

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

Returns a list of strings. Each element of the list holds one combination 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)

Function type

LIST OF STRING

For elements

activity	ac
chart	ch

Syntax

stm_r_xx_combinationals (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_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_lindex

Returns the left index of an element array.

You can call this function without indicating the specific element type, as follows:

```
stm_r_array_lindex (id, status)
```

Function type

STRING

For elements

condition	co
data-item	di
event	ev
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

Syntax

```
stm_r_xx_array_lindex (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_array_rindex

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)

Function type

STRING

For elements

condition	co
data-item	di
event	ev
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

Syntax

stm_r_xx_array_rindex (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_attr_enforced

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)

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

Syntax

stm_r_xx_attr_enforced (xx_id, attr_name, attr_val, status)

Arguments

Argument	Input/Output	Type	Description
xx_id	In	Statemate 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 stm_attribute_name_not_found .

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

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)

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

Syntax

stm_r_xx_attr_name (xx_id, status)

Arguments

Argument	Input/Output	Type	Description
xx_id	In	Statemate element	The element ID.
status	Out	INTEGER	The function status code. If no attributes exist for the specified element, status receives the value stm_attribute_name_not_found.

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

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)
```

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

Syntax

```
stm_r_xx_attr_val (xx_id, attr_name, status)
```

Arguments

Argument	Input/Output	Type	Description
xx_id	In	Statemate element	The element ID.
attr_name	In	STRING	The attribute name. The attribute name is not case-sensitive.
status	Out	INTEGER	The function status code. If attr_name does not exist for the specified element, status receives the value stm_attribute_name_not_found.

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

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_lindex

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)

Function type

STRING

For elements

data-item	di
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

Syntax

stm_r_xx_bit_array_lindex (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_bit_array_rindex

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)

Function type

STRING

For elements

data-item	di
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

Syntax

stm_r_xx_bit_array_rindex (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_cbk_binding

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)

Function type

LIST OF STRING

For elements

activity	ac
condition	co
data-item	di
event	ev
state	st

Syntax

stm_r_xx_cbk_binding (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_missing_cbk_binding
- stm_unresolved

stm_r_xx_cbk_binding_enable

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)

Function type

LIST OF STATEMATE ELEMENTS (predefined constant)

For elements

activity	ac
condition	co
data-item	di
event	ev
state	st

Syntax

stm_r_xx_cbk_binding_enable (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

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 predefined internally as a predefined constant in DGL.

Element	Element Subtype
activity	stm_ac_cbk_enable
	stm_ac_cbk_disable
	stm_ac_cbk_bind_missing
condition	stm_co_cbk_enable
	stm_co_cbk_disable
	stm_ac_cbk_bind_missing
data-item	stm_di_cbk_enable
	stm_di_cbk_disable
	stm_di_cbk_bind_missing
event	stm_ev_cbk_enable
	stm_ev_cbk_disable
	stm_ev_cbk_bind_missing
state	stm_st_cbk_enable
	stm_st_cbk_disable
	stm_st_cbk_bind_missing

stm_r_xx_cbk_bin ding_expression

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)

Function type

STRING

For elements

activity	ac
condition	co
data-item	di
event	ev
state	st

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

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)

Function type

STRING

For elements

activity	ac
condition	co
data-item	di
event	ev
state	st

Syntax

stm_r_xx_cbk_binding_expression_hyper (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_tt_cell

Retrieves the contents of the specified cell in the given truth table.

Function type

STRING

For elements

action	an
activity	ac
subroutine	sb

Syntax

stm_r_tt_cell (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	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

Retrieves the data-type of the specified cell in the given truth table.

Function type

INTEGER

For elements

action	an
activity	ac
subroutine	sb

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

Provides a change 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

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 returns the chart only when all the arrow segments are in the same element. Otherwise, it returns the value 0.

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

Syntax

stm_r_xx_chart (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

Example

To write the name of the chart in which state 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

Returns a list of strings. Each element of the list holds one combination 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)

Function type

LIST OF STRING

For elements

activity	ac
chart	ch

Syntax

stm_r_xx_combinationals (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_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

Returns the ID of the procedural statechart connected to the specified subroutine.

Function type

STATEMATE ELEMENT

For elements

subroutine	sb

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

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)

Function type

LIST OF FIELDS

For elements

data-item	di
user-defined type	dt

Syntax

stm_r_xx_containing_fields (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_missing_field
- stm_unresolved

stm_r_ch_creation_date

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.

Function type

STRING

For elements

chart	ch

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

Returns the name of the Statemate user who created the specified chart. This function is relevant only for charts that were explicitly created using one of the graphic editors.

Function type

STRING

For elements

chart	ch

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

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)

Function type

INTEGER

For elements

data-item	di
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

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

Returns the date field of the specified requirement record.

Function type

STRING stm_date

For elements

requirement	rt

Syntax

stm_r_rt_date (rt_id, status)

Arguments

Argument	Input/Output	Type	Description
rt_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_id_out_of_range
- stm_illegal_parameter
- stm_missing_field

stm_r_xx_definition_type

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 fn 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.

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

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	stm_an_reference
		stm_an_primitive
		stm_an_compound
condition	co	stm_co_reference
		stm_co_primitive
		stm_co_compound
		stm_co_constant
data-item	di	stm_di_reference
		stm_di_primitive
		stm_di_compound
		stm_di_constant
		stm_di_alias
event	ev	stm_ev_reference
		stm_ev_primitive
		stm_ev_compound
field	fd	stm_fd_primitive
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

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)
```

Function type

STRING

For elements

actor	actor
boundary box	bb
use case	uc

Syntax

```
stm_r_xx_desc_file (elem, status)
```

Arguments

Argument	Input/Output	Type	Description
elem	In	Statemate element	The element
status	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_boundry_box
- stm_success

stm_r_xx_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)

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

Syntax

stm_r_xx_description (xx_id, status)

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	Statemate element	The element ID.
status	Out	INTEGER	The function status code. If no description exists in the element's form, status receives the value stm_missing_short_description .

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

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.

Function type

LIST OF STRING

For elements

element ID	

Syntax

stm_r_design_attr (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_displayed_name

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)

Function type: STRING

For elements

activity	ac
data-store	ds
module	md
module-occurrence	om
off-page activity	oa
router	router
state	st

Syntax

stm_r_xx_displayed_name (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_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 functions that return different values:

- stm_r_xx_name
- stm_r_xx_uniquename

stm_r_ddb_list_names

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

Returns the element type of the specified element.

Function type

INTEGER (predefined constant)

For elements

All types

Syntax

stm_r_element_type (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

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)	stm_a_flow_line
a-flow-line (compound)	stm_compound_a_flow_line
action	stm_action
activity	stm_activity
chart	stm_chart
condition	stm_condition
connector in activity-chart	stm_a_connector
connector in module-chart	stm_m_connector
connector in statechart	stm_s_connector

Element Type	Value
data-item	stm_data_item
data-store	stm_data_store
decomposed sequence diagram	stm_decomposed_sd
event	stm_event
external lifeline	stm_external_lifeline
flow label	stm_flow_label
information-flow	stm_information_flow
lifeline	stm_lifeline
m-flow-line (basic)	stm_label
m-flow-line (compound)	stm_m_flow_line
message	stm_message
module	stm_compound_m_flow_line
module occurrence	stm_module
order insignificant	stm_order_insignificant
separator	stm_separator
subroutine	stm_subroutine
state	stm_module_occurrence
timing constraint	stm_timing_constraint
transition (basic)	stm_state
transition (compound)	stm_transition
transition label	stm_compound_transition

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

Returns the definition expression of the specified element found in the Definition field of the element's form, including hyperlinks to referenced elements.

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

Syntax

stm_r_xx_expr_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_primitive_element

stm_r_xx_expression

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 text elements, which are defined as an expression using the Definition field of its form.

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

Syntax

stm_r_xx_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. If xx_id belongs to a primitive (not a compound) element, status receives the value stm_primitive_element.

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

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)
```

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

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

Retrieves the extension point definitions for the specified use case.

Function type

STRING

For elements

use case	uc

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

Returns a list of names of formal parameters that appear in bindings of instance boxes and components.

Function type

LIST OF STRING

For elements

action	an
condition	co
data-item	di
event	ev

Syntax

stm_r_formal_parameter_names (inst_box_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_r_sb_global_data

Returns the global data associated with the specified subroutine.

Function type

LIST OF ELEMENT

For elements

subroutine	sb

Syntax

stm_r_sb_global_data (sb_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_global_data
- stm_id_out_of_range
- stm_id_not_found

stm_r_global_interface_report

Creates a global interface report from the specified input list.

Function type

LIST OF STRING

For elements

activity	ac
external router	ext_router
router	router

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

Retrieves the callback binding expression 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)

Function Type

stm_expression

For Elements

activity	ac
condition	co
data-item	di
event	ev
state	st

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

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.

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-occurence	om
note	nt
off-page activity	oa
state	st

Syntax

stm_r_xx_graphic (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_unresolved
- stm_id_not_found
- stm_missing_graphic_data

Note

When stm_unresolved is returned, no record is received.

stm_r_hyper_key

Retrieves the unique key for the specified element.

Function type

STRING

For elements

element ID	ac

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

Retrieves the implementation property for the specified module.

Function type

STRING

For elements

module	md

Syntax

stm_r_md_implementation (md_id, status)

Arguments

Argument	Input/Output	Type	Description
md_id	In	Statemate 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

Returns the list of global definitions contained in the specified chart.

Function type

LIST OF CHART

For elements

chart	ch

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

Returns a list of messages that are included by an order-insignificant element.

Function type

LIST OF MESSAGE

For elements

Last of order insignificance	ord_insig_list

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

Retrieves the description of the specified continuous chart.

Function type

STRING

For elements

chart	ch

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

Retrieves the list of global definition sets that are “inherited” (included indirectly) by the specified chart.

Function type

LIST OF CHART

For elements

chart	ch

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

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.

Function type

STRING

For elements

activity	ac
module	md
state	st

Syntax

stm_r_xx_instance_name (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_not_instance

Example

To retrieve the name of an instance for state named stm@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 stm@S1_DEF.

stm_r_xx_keyword

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)

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

Syntax

stm_r_xx_keyword (xx_id, begin_keyword, end_keyword, filename, status)

Arguments

Argument	Input/Output	Type	Description
xx_id	In	stm_id	The element ID
begin_< keyword	In	char *	The beginning of the portion of the string in the long description to extract
end_keyword	In	char *	The end of the portion of the string in the long description to extract
filename	In	stm_filename	The name of the file to contain the long description
status	Out	int	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

Returns the K&R C code that was manually written by the user for the specified subroutine.

Function type

LIST OF STRING

For elements

subroutine	sb

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

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.

Function type

LIST OF STRING

For elements

message	msg
transition	tr

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

Returns a list of strings of message or transition labels, with hyperlinks to referenced elements.

Function type

LIST OF STRING

For elements

message	msg
transition	tr

Syntax

stm_r_xx_labels_hyper (message, format, status)Arguments

Argument	Input/Output	Type	Description
elem_id	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_local_interface_report

Creates a local interface report from the specified input list.

Function type

LIST OF STRING

For elements

activity	ac
external router	ext_router
router	router

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

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)

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

Syntax

stm_r_xx_longdes (xx_id, filename, status)

Arguments

Argument	Input/Output	Type	Description
xx_id	In	Statemate element	The element ID
filename	In	stm_filename STRING	The name of the file that will contain the long description
status	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

Retrieves the header for the lookup table.

Function type

LIST OF STRING

For elements

lookup table	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

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)
```

Function type

STRING

For elements

data-item	di
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

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

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)
```

Function type

STRING

For elements

data-item	di
field	fd
local data	ld
subroutine parameter	sp
user-defined type	dt

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

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)

Function type

STRING

For elements

activity	ac

Syntax

stm_r_xx_mini_spec (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_missing_label

stm_r_ac_mini_spec_hyper

Returns a string with the mini-spec, including hyperlinks to referenced elements.

Function type

STRING

For elements

activity	ac

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