Rational. Statemate

Data Port Reference Guide

Rational Statemate
Dataport 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.

Single-Element Functions 35

Dataport Library Overview

The Dataport library provides:

- Functions to perform a wide variety of database extraction operations. Using the library's functions, you can extract information pertaining to an element from the specification database.

  You can use information extracted from the database for a variety of applications, such as:

  - To plot portions of Rational Statemate charts using your own plotter package. To do this, extract the graphic information of the relevant elements and then use this information as input data for your plotter.
  - To analyze data stored in the database. To do this, extract the relevant data and then use your own software to perform the analysis.
  - To extract textual information describing Rational Statemate elements for use in specification-related applications.
- Functions to perform a variety of operations on elements extracted from the specification database. For example, you can alphabetically order a list of extracted states using their names.
- Functions to activate Rational Statemate capabilities from your program. For example, you can generate a plot of a chart using the Rational Statemate `plot` function.

Function Types

There are four types of Dataport functions:

- **Single-element functions** - Provide information on discrete Rational Statemate elements in the specification database. For example, you can retrieve the contents of the **Description** field in a particular state's form.
- **Query functions** - Extract lists of elements from the database, that conform to a specific criterion. For example, you can extract a list of activities from the database that are control activities. Most of these functions correspond directly to queries with the search facility.
- **Utility functions** - Perform operations on lists. Most of these functions do not extract information from the database, but rather manipulate the information you have already retrieved.
- **Project management functions** - Extract information about the Rational Statemate project, manager, and members.

Dataport Interface

The Rational Statemate Dataport functions have a C language interface. They can be called from C language programs, as well as from programs written in other languages that can call C functions.

Working with the Dataport

To use the Dataport library, you extract information from the specification database by including calls to various Dataport functions in your program. An explanation of function calls, parameters, and returned values is provided in the following sections.

In addition to Dataport function calls, programs designed to extract database information must also include a file of definitions, for example a definition of data-types.

After you finish writing and compiling your program, you must link it with the Dataport Library image. These procedures are explained in **Using Dataport Functions.**

Using Dataport Functions

This section provides information on how to use Dataport functions within a program. It provides information on the following topics:

Dataport Function Calls

Dataport function calls can appear anywhere in your program once an initialization procedure is performed. Here are some examples of valid function calls.

In this example, the state named `S1` is retrieved from the database and the variable `state_id` is assigned to it. The state's ID is retrieved; ID is a value that Rational Statemate uses to identify each element in the database. The `state_id` can be used later in other function calls.

```
state_id = stm_r_st ("S1", &status);
```

Function calls are frequently used in sequence. For example, the ID for state `S1` in this function call has already been retrieved. The sample call retrieves the synonym of this state.

```
synonym = stm_r_st_synonym (state_id, &status);
```

This function creates a list (which contains the state `S1`), assigns it to the variable `state_list`, then extracts the substates of `S1`.

```
state_list = stm_list_create (state_id,
   end_of_list, &status);
sub_st = stm_r_st_physical_sub_of_st (state_list,
   &status);


for (s = (stm_id) stm_list_first_element(
     sub_st, &status);
     status == stm_success;
     s = (stm_id) stm_list_next_element(
          sub_st, &status)
)
```

Prints a list of all substates of state `S1`.

```
printf ("\n %s", stm_r_st_name (s, &status));
```

Refer to **Sample Program** for an example of how to call Dataport functions in a C program.

Calling Conventions

Dataport functions provide you with information about particular Rational Statemate elements in the database. To extract this information, you call the specific function that retrieves the information you want. You specify the particular Rational Statemate elements that interest you as input arguments to the function. The function returns a status code as an output argument, which indicates whether the function call was successful.

Function Name

Single-element and query functions use the following prefix:

```
stm_r_
```

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

Utility functions use the following prefix:

```
stm_
```

Element Type Abbreviations

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

Element	Abbreviation
A-flow-lines (basic)	ba
A-flow-lines (compound)	af
A-flow-lines (local)	laf
Actions	an
Actors	actor
Activities	ac
Boundary boxes	bb
Charts	ch
Combinational assignments	ca
Conditions	co
Connectors	cn
Data-items	di
Data-stores	ds
Enumerated value	en
Events	ev
Fields	fd
Information-flows	if
Lifelines	ll
Local data	ld
Messages	msg
Mixed (multiple types)	mx
M-flow-lines (basic)	bm
M-flow-lines (compound)	mf
M-flow-lines (local)	lmf
Modules	md
Module-occurrences	om
Notes	nt
Off-page activities	oa
Reference sequence diagrams	ref_sd
Routers	router
Separators	sep
States	st
Subroutines	sb
Subroutine parameters	sp
Timing constraints	tc

Element	Abbreviation
Transitions (basic)	bt
Transitions (compound)	tr
Use cases	uc
User-defined types	dt

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

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

Arrow elements (transitions on a statechart, a-flow-lines (control and data flow lines on an activity chart) and m-flow-lines on a module chart) can be either basic or compound:

- A basic arrow connects a box (state, activity, or module) or connector to another box or connector.
- Compound arrows result from the combination of a number of basic arrows joined together by connectors.

Function Input Arguments

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

Some functions require additional input arguments. Each argument must be declared to be of a data type recognized by the Dataport library (or by the C compiler). This document includes a complete list of input arguments for each type of database extraction function in the sections that describe the specific function type.

Refer to the appropriate function sections for the lists of arguments relevant for each function.

Example

To print out the synonym of the state `S1` (if no synonym is defined in the state's form, print `missing synonym`), use the following statements:

```
        int              status;
        stm_id           state_id;
        stm_short_name   synonym;
                            .
                            .
                            .
    state_id = stm_r_st ("S1", &status);
    synonym = stm_r_st_synonym (state_id, &status);
    if (status == stm_missing_synonym)
       printf ("\n synonym:  *missing synonym*");
    else
       printf ("\n synonym: %s", synonym);
```

The `stm_id` data-type is defined in the Dataport Library definition file, `dataport.h`. Refer to **Data Types** for a complete list of data-type definitions.

Function Return Values

Library function return values are assigned to data-types declared in the library definition file `dataport.h`. These data-types are defined in the C language. Refer **Data Types** for the maximum length of return values.

For example, a function that retrieves the name of a Rational Statemate element, returns a value of type `stm_element_name` (declared as `char *` in the library definition file), whereas a function that retrieves a state's ID returns a value of type `stm_id` (declared as `long int`).

The return values of data-types declared as strings `char *`, such as `stm_element_name`, are usually limited in length.

The returned strings are defined as `static` in the functions. You should copy them if they are needed for later use.

Special Cases of Return Values

The following are special cases of return values:

- **Return values of filename**

  A number of Dataport functions store extracted information in files, such as a function that retrieves an element's long description. This function returns the name of the file that contains the requested information. The filename returned is of type `stm_filename`. This data-type is declared as `char *` in the library definition file.

  Note: The returned string is defined as `static` in the functions. Copy the string if it is required for later use.

- **Return values of enumerated types**

  There are several functions that return a finite number of discrete values. These values are not necessarily integers, and no particular order is assumed for these values.

  For example, the function `stm_r_st_type` extracts the state type for the state specified in the function call. The possible state types are:

  ```
  stm_st_diagram
  stm_st_and
  stm_st_or
  stm_st_instance
  stm_st_reference
  stm_st_basic
  ```

  Your program can contain statements such as:

  ```
  if (stm_r_st_type (st_id, &status) == stm_st_basic)
  ```

  For such return values, there are special enumerated data-types declared in the definition file. For example, the previous state types belong to the enumerated type `stm_state_type`. Refer to for the list of enumerated data-types. Refer to **Single-Element Functions**, **Query Functions**, and **Utility Functions** for the possible values and the corresponding data-types that particular functions return.

- **Return values of pointers to records**

  There are several functions that return a pointer to records with the textual and graphical information of an element.

  Note: The records are defined as `static` in the functions; You should copy the records if they are needed for later use.

- **Return values when the function call fails**

  When a function call fails, the function status code reflects the failure by returning the following values:

Function Type	Return Value
`list`	NIL
`string`	NIL
`Boolean`	false
`stm_id`	0

  By testing the value returned by the function, you can pinpoint function call failures. To determine what went wrong, use the function status code.

Using Functions in C Language Programs

The Dataport has a C language interface. To use its functions in a program, you must follow specific procedures to access both the library and your database.

Include Files

Every program that calls Dataport functions must include the definitions for its library data-types and constants. The definitions are contained in the `dataport.h` file.

To incorporate these definitions, include the file by writing the following statement at the beginning of your program:

```
#include dir_name/include/dataport.h
```

Substitute the value of the environment variable `STM_ROOT` for the `dir_name` variable.

Information Retrieval Process

Perform the following operations when using Dataport functions:

1. Initialize the retrieval process, via the `stm_init_uad` function.
2. Call the Dataport functions to retrieve database information.
3. Include the following line in your program, after the last Dataport function call:

```
stm_finish_uad();
```

Initializing the Retrieval Process

To initialize the retrieval process, add the following statement to your program before any calls to Dataport library functions:

```
stm_init_uad (proj_name, w_area, trans_mode, &status)
```

In this call:

- **proj_name** - The name of the Rational Statemate project containing the information of interest.
- **w_area** - The directory pathname of your workarea in which the specification database is found.
- **trans_mode** (transaction on mode) - The transaction on mode. This specifies the manner in which you want to handle database transactions, using `self_transaction` or `automatic_transaction` modes.

This function returns `true` if successful, or `false` if unsuccessful. If unsuccessful, check the `status` argument for the reason the function failed.

Note

The `stm_init_uad` function automatically changes the current directory to the workarea directory. Therefore, all references to files inside the program have to take this into account. Also, when the program terminates, it does not return to the original directory.

The following example shows how to initialize the retrieval process in a C program. In this example, you are prompted for the name of the project to open.

```
main( )
{
int status, success;
char     name[32];
char     dir[30];
printf ("Enter name of Statemate project: ");
scanf ("%s", name);
printf ("Enter directory pathname
         for your Workarea: ");
scanf ("%s", dir);
success =
  stm_init_uad (name, dir, automatic_transaction,
      &status);
if (!success)
  printf ("Init function failed.
           Reason: status code %d", status);
}
```

Note

- The project name (in this case, the content of the variable `name`) is not case sensitive.
- It is recommended that you write the `init` function in the form shown in the example (`success=...; if(!success)...;`) to ensure that the `init` function succeeds before continuing.

Transaction Handling

The transaction mode determines how database modifications are reflected in your retrievals. The transaction modes are `automatic_transaction` and `self_transaction`.

Use automatic_transaction mode in the following cases:

- The database is not being updated during the information retrieval.
- The database is being updated by processes running in parallel to your program, but you are not interested in these updates.
- Your program uses the load and unload functions to change the database contents, and you want to use the updated database in your program each time it changes.

Refer to **Automatic Transaction Mode** for more information,

Use self_transaction mode if your information retrieval is to reflect database changes done by other processes at the same time your program is working, and you do not use load and unload functions in your program. Refer to **Self Transaction Mode** for more information.

Automatic Transaction Mode

In `automatic_transaction` mode, an implicit `start_transaction` is performed when you initialize the retrieval process and an implicit `commit_transaction` is performed when you finish the retrieval process.

Whenever you use load and unload functions to change the contents of the database, the program implicitly closes the read transaction, starts the read/write transaction, commits the changes, and implicitly starts a new read transaction.

The following is an example of how to use `automatic_transaction`:

```
main()
{
  int status, success;
  stm_list st_list;
  stm_id el;
   success = stm_init_uad ("PROJ", "/local/proj",
                automatic_transaction, &status);
   if (!success)
   {
     printf ("cause of failure is: %d", status);
     return;
   }
/*                                          */
/*  once initialization is done,            */
/*  retrieval can be done at any time.      */
/*                                          */
   st_list = stm_r_st_name_of_st ("*", &status);
   for (el = (stm_id)
    stm_list_first_element (st_list, 0);el!=NIL;
    el =(stm_id)stm_list_next_element (st_list, 0))
   {
     printf ("\n%s", stm_r_st_name (el, 0));
   }
/*                                          */
/*      The resulting output is:            */
/*                state_a                   */
/*                state_b                   */
/*                state_c                   */
/*                state_d                   */
/*                                          */
stm_finish_uad();
```

Self Transaction Mode

Use `self_transaction` mode when working with applications that are sensitive to database changes performed by a process that runs in parallel with your program. When you operate using the `self_transaction` mode, you must explicitly perform a `start_transaction` before you call Dataport retrieval library functions. This is done by including the following statement in your program:

```
stm_start_transaction();
```

For each `start_transaction`, you must perform an explicit `commit_transaction` to conclude the database retrievals by including the following statement in your program:

```
stm_commit_transaction();
```

You can start and commit transactions at any stage of your program. However, before retrieving additional data following a `commit_transaction`, you must first perform another `start_transaction`. Perform a new `start_transaction` whenever you want to refresh the image of the database so subsequent retrievals accurately reflect the database information.

The following is the structure of the `self_transaction` mode:

```
main()
{
  int status, success;
  stm_list st_list;
  stm_id el;
   success = stm_init_uad ("PROJ", "/local/proj",
                self_transaction, &status);
   if (!success)
   {
     printf ("cause of failure is: %d", status);
     return;
   }
/*                                          */
/*  once initialization is done,            */
/*  a start_transaction statement is        */
/*  needed in this mode.                    */
/*                                          */
  stm_start_transaction ();
  st_list = stm_r_st_name_of_st ("*", &status);
   for (el = (stm_id)
    stm_list_first_element (st_list, 0);el!=NIL;
    el = (stm_id)stm_list_next_element(st_list, 0))
   {
     printf ("\n%s", stm_r_st_name (el, 0));
   }

/*                                          */
/*   retrievals are done for now, so a      */
/*   commit_transaction is performed.       */
/*                                          */
/*    stm_commit_transaction();             */
/*      The resulting output is:            */
/*                state_a                   */
/*                state_b                   */
/*                state_c                   */
/*                state_d                   */
```

```
/*                                          */
/*   During the course of this output       */
/*   another process has updated the        */
/*   database, drawing a new state since    */
/*   this last transaction took place.      */
/*                                          */
/*   If the same retrieval is done again,   */
/*   different results should be found!     */
/*   To insure that the program "sees"      */
/*   the changes, start_transaction is      */
/*   performed again.                       */
/*                                          */
   stm_start_transaction ();
   st_list = stm_r_st_name_of_st ("*", &status);
   for (el = (stm_id)
    stm_list_first_element (st_list, 0);el!=NIL;
    el =(stm_id)stm_list_next_element (st_list, 0))
   {
     printf ("\n%s", stm_r_st_name (el, 0));
   }
/*                                          */
/*      The resulting output is:            */
/*                state_a                   */
/*                state_b                   */
/*                state_c                   */
/*                state_d                   */
/*                state_e                   */
/*                                          */
/*  The last state, e, is new to the list   */
/*                                          */
  stm_commit_transaction ();
  stm_finish_uad ();
}
```

Preparing and Executing Programs

C programs containing Dataport function calls must be linked with the Dataport library. To execute a program containing calls to Dataport functions, follow the procedure for your operating system. The definitions in the `dataport.h` file can be used for debugging purposes.

Windows Systems

Define the environment variable `STM_ROOT`, as follows:

```
SET STM_ROOT=root name
```

Contact your Rational Statemate manager for the name of the root directory of the Rational Statemate tree. For example:

```
SET STM_ROOT=C:\IBM Rational\stmm\4.6
```

Use the following command to compile and link:

```
PROGRAM= my_prog.exe
DLL= <STM_ROOT>\bin\dataport.dll
DLIB= <STM_ROOT>\lib\dataport.lib
SRCS= my_prog.c
HDRS= my_prog.h

CFLAGS= /DDLL_LINK /I<STM_ROOT>\include
LIBS= kernel32.lib
all: $(PROGRAM) $(DLL) $(HDRS)

$(PROGRAM): $(SRCS) $(DLIB)
cl $(CFLAGS) $(SRCS) $(DLIB) $(LIBS)

clean:
 -del $(PROGRAM) >nul: 2>&1
 -del *.obj >nul: 2>&1
 -del *.pdb >nul: 2>&1
 -del *.ilk >nul: 2>&1
 -del *.mdp >nul: 2>&1
 -del *.opt >nul: 2>&1
```

In this syntax:

- `prog.o`—The name you want to assign to the executable image
- `prog.h`—The header file
- `myprog.c`—The name of the file containing the C program

Use the following command to execute your program:

```
prog
```

UNIX Systems

Define the environment variable `STM_ROOT`, as follows:

```
% setenv STM_ROOT root_name
```

Contact your Rational Statemate manager for the name of the root directory of the Rational Statemate tree.

Use the following command to compile and link:

```
cc -o <program> <otherflags> <myprog.c>  \
   $STM_ROOT/lib/dataport.o \
   $STM_ROOT/lib/libgcc.a $STM_ROOT/lib/x_stubs.o \
   -lm -lsocket -lnsl -L/usr/ucblib -lucb -ldl
```

In this syntax:

- `program`—The name you want to assign to the executable image
- `otherflags`—Can include `-g` or `-O`
- `myprog.c`—The name of the file containing the C program

Use the following command to execute your program:

```
program
```

In this syntax, `program` is the name of the executable image.

Optional qualifiers, such as debug, can be added in the compile, link, and execute stages. Refer to your operating system reference manuals for the available options.

Sample Program

This section contains a sample C program that shows you how to use the Rational Statemate Dataport to extract information about activities from the specification database. The information extracted is both textual (such as activity name, synonym, short description, and so on) and graphical (such as the Cartesian coordinates of the activity's box).

Sample C Program

The sample C program is as follows:

```
#include <stdio.h>

#include "dataport.h"
#define GET_STR(S)
  {int i;for(i=0;(S[i++]=getchar())!='\n';);
       S[--i]='\0';}

int status=0;
FILE *fd;
char array[80];

void draw_line (color, x1, y1, x2, y2)
stm_color color;
stm_coordinate x1, y1, x2, y2;
{
 printf("\n line from %f, %f to %f, %f in color %d",
        x1, y1, x2, y2, color);
}

void draw_string (s, color, x1, y1)
char *s;
stm_color color;
stm_coordinate x1,y1;
{
 printf ("\n string %s at %f, %f in color %d\n",
          s, x1, y1, color);
}
static char *activity_type (search_for)
stm_activity_typesearch_for;
{

static struct search_activity_type {
  stm_activity_typeact_type;
  char    *name;
} ActivityType[] = {
  {stm_ac_diagram,"DIAGRAM"},
  {stm_ac_reference,"REFERENCE"},
```

```
  {stm_ac_internal,”INTERNAL”},
  {stm_ac_instance,”INSTANCE”},
  {stm_ac_control,”CONTROL”},
  {stm_ac_control_instance,”CONTROL_INSTANCE”},
  {stm_ac_external,”EXTERNAL”},
  {NULL,  ”NULL”},
}
struct search_activity_type*sat;

  for (sat = ActivityType; sat->name != NULL; sat++)
    if (sat->act_type == search_for)
      return sat->name;
  return ””;     /* error!*/
}

static char *activity_termination_type(search_for)
stm_activity_terminationsearch_for;
{
static struct search_activity_termination {
  stm_activity_terminationact_term_type;
  char          *name;
} ActivityTerminationType[] = {
 {stm_ac_missing, ”MISSING”      },
 {stm_ac_self_termination, ”SELF_TERMINATION”      },
 {stm_ac_controlled_termination,
                          ”CONTROLLED_TERMINATION”},
 {NULL,              ”NULL”      },
}
struct search_activity_termination*sat;

  for (sat = ActivityTerminationType;
            sat->name != NULL;sat++)
     if (sat->act_term_type == search_for)
            return sat->name;
    return "";   /* error!*/
}

void activity_text_output(ac_text)
stm_ac_text_ptr ac_text;
{
   printf("\n\n\n\n\n activity textual information\n");
   printf("============================\n");
   printf("\n activity name: %s", ac_text->ac_name);
   printf("\n activity in chart: %d",
          ac_text->ac_chart);
   printf("\n activityuniquename:%s",
          ac_text->ac_uniquename);
   printf("\n activity synonym: %s",
          ac_text->ac_synonym);
   printf("\n activity type: %s",(ac_text->ac_type));
   printf("\n activity termination: %s",
          activity_termination_type (ac_text->ac_termination));
   printf("\n activity short description: %s",
          ac_text->ac_short_des);
   printf("\n activity long description:\n\n");
   if ((fd=fopen(ac_text->ac_long_des, "r")) == ZNIL)
        printf("\n\n cannot open file for printing");
   while (fgets(array,81,fd)!=ZNIL) printf("%s",array);
}
void activity_graphic_output (ac_graphic, ac_name)
stm_ac_graphic_ptr ac_graphic;
stm_name ac_name;

{
```

```
  int i = 0;
  stm_coordinate prev_x;
  stm_coordinate prev_y;


printf("\n\n activity graphical information\n");
  printf("==============================\n");

  draw_string (ac_name, ac_graphic->ac_name_color,
       ac_graphic->ac_x_coor, ac_graphic->ac_y_coor);


  for  (i=1; i<=ac_graphic->ac_polygon.points_no;i++)
  {
     prev_x = ac_graphic->ac_polygon.outline[i-1].x;
     prev_y = ac_graphic->ac_polygon.outline[i-1].y;
     draw_line(ac_graphic->ac_color,prev_x,prev_y,
     ac_graphic->ac_polygon.outline
                [i % ac_graphic->ac_polygon.points_no].x,
            ac_graphic->ac_polygon.outline
           [i % ac_graphic->ac_polygon.points_no].y);
  }
}

void activities_info (ac_list)

stm_list ac_list;
{
  stm_ac_text_ptr    ac_textual;
  stm_ac_graphic_ptr ac_graphical;
  stm_id                el;


  for (el = (stm_id) stm_list_first_element (ac_list, &status);

      status == stm_success;
      el= (stm_id)stm_list_next_element (ac_list, &status))
  {

      ac_textual = stm_r_ac_text (el, &status);
 if (status == stm_success)
     activity_text_output (ac_textual);

 ac_graphical=stm_r_ac_graphic(el,&status);
 if (status == stm_success)
     activity_graphic_output(ac_graphical,
ac_textual->ac_name);
  }
}

void activity_boxes()
{
  char     ac_name[32];
  stm_id   ac_id;
  stm_list ac_list,acs_list;

printf("\n enter activity name: ");
  GET_STR(ac_name);
  ac_id=stm_r_ac(ac_name,&status);
  if (status!=stm_success)
  {
   printf("illegal activity name"); return;
  }
```

```
  ac_list=stm_list_create(ac_id,end_of_list,&status);
  acs_list=stm_r_ac_physical_sub_of_ac(ac_list,&status);

  if (status!=stm_success)
  {
   printf("error during execution"); return;
  }
  acs_list=stm_list_union(ac_list, acs_list, &status);
  activities_info(acs_list);
}

main( )
{
  char pname[32];

  printf("enter name of project: ");
  GET_STR(pname);
  success=stm_init_uad(pname, "/usr/sam/proj",
                       self_transaction, &status);
  if (success)
  {
    stm_start_transaction();
    activity_boxes();
    stm_commit_transaction();
  }
  else
  printf(cause of failure 15:%d,status);
  stm_finish_uad();
}
```

Program Description

For clarity, the program is explained in sections, not necessarily in the order in which it is written. The most general part, the main section of the program, is described first, then the individual functions are described.

Main Section and Program Setup

```
main( )
```

This section is the main part of the program. It calls the individual routines that are included in the C program.

```
{
 char pname[32];
 printf("enter name of project: ");
 GET_STR(pname);
```

The program begins by prompting for the name of the project.

```
success=stm_init_uad(pname,"/usr/sam/proj",
   self_transaction,&status);
```

The extraction is initialized by this function call. Your user's authorization to use the database is checked. You must be a member of the project that contains the specified activity-chart. If you are a member, the database is opened.

The parameter value `self_transaction` declares that you control the transaction handling, rather than having it done automatically. If the function is not successful, check the value of status to find out the reason for the failure.

```
stm_start_transaction();
```

This statement permits database operations. The statement must appear here because the `self_transaction` parameter was used in the `init` statement.

```
activity_boxes();
```

This calls the primary routine.

```
stm_commit_transaction();
stm_finish_uad();
}
```

This example does not have functions that write to the database; therefore, the `commit` statement here serves only to conclude the transactions.

The `finish` statement concludes the Dataport operations and closes the database.

Creating the Lists

```
void activity boxes()
```

This primary function creates a list of activities. The list is composed of a specified activity together with its subactivities. You are prompted for the parent activity.

```
{
  char     ac_name[32];
  stm_id   ac_id;
  stm_list  ac_list,acs_list;
```

`ac_name` holds the name of the activity., `ac_id` holds the `ID` number of the activity, and `ac_list` and `acs_list` hold the list of activities.

```
printf ("\n enter activity name: ");
GET_STR (ac_name);
ac_id = stm_r_ac (ac_name, &status);
if (status!=stm_success)
  {
    printf ("illegal activity name");
    return;
  }
```

The element `ID` that corresponds to the element name is retrieved. The status check determines whether the retrieval was successful. If the activity name is not unique or if it is incorrect, an error message is printed and the function is aborted.

```
ac_list = stm_list_create (ac_id, end_of_list, &status);
```

This statement creates a list of the single activity of interest. This list is then used as an input parameter for the routine `stm_r_ac_physical_sub_of_ac`.

```
acs_list = stm_r_ac_physical_sub_of_ac (ac_list, &status);
```

This statement creates a list of all the subactivities for the input activity list (which, in this case, contains only a single activity).

```
if (status != stm_success)
{
  printf ("error during execution");
  return;
}
```

The status of the operation is checked for success (in this case, the check is redundant). An error could be caused if the activity in `ac_list` is deleted from the database by some other process while the sample C program is being executed. This situation can occur when the entire program is performed under more than one transaction.

```
acs_list = stm_list_union (ac_list, acs_list, &status);
```

This statement merges the two lists that contain the parent activity and its subactivities.

```
activities_info(acs_list);
}
```

The last statement of this function calls the function `activities_info`, which retrieves information for each element in a list of activities.

Retrieving the Information

```
void activities_info (ac_list)
stm_list ac_list;
```

This function retrieves both the textual and graphical information contained in the database.

```
{
  stm_ac_text_ptr ac_textual;
  stm_ac_graphic_ptr ac_graphical;
  stm_id el;
```

These statements declare variables to point to textual and graphical records, and to include the `ID` for an activity. The structure of these records is defined in `dataport.h`.

```
for (el = (stm_id) stm_list_first_element (ac_list,
    &status);
    status == stm_success;
    el = (stm_id) stm_list_next_element (ac_list,
    &status)
)
stm_list_next_element (ac_list, &status))
```

These statements loop through each element in the activity list. The status of `stm_list_next_element` is not equal to `stm_success` when there are no more elements in the list.

```
{
  ac_textual = stm_r_ac_text (el, &status);
  if (status == stm_success)
       activity_text_output (ac_textual);
```

This statement retrieves the textual information for an activity. If successful, it writes the information into the report.

```
  ac_graphical = stm_r_ac_graphic (el, &status);
  if (status == stm_success)
      activity_graphic_output (ac_graphical,
          ac_textual->ac_name);
  }
}
```

This statement retrieves the graphical information for an activity. If successful, it calls the graphic output routine, passing the name for the activity.

Writing the Graphical Information

```
void activity_graphic_output(ac_graphic,ac_name)
```

This function defines the output of the graphical information retrieved from the database. In this example, the information is written into the report. The information can also be passed to an actual drawing routine, if desired.

The name of the activity, `ac_name`, is passed as a second parameter because the activity name is used in a graphic drawing such as a plot, although this information is part of the element's textual record.

```
stm_ac_graphic_ptr ac_graphic;
stm_name ac_name;
```

`ac_graphic` is a pointer to an activity's graphical record; `ac_name` is the name of the activity.

```
{
 int            i = 0;
 stm_coordinate prev_x;
 stm_coordinate  prev_y;
```

Declares the following variables:

- `i`—Controls the loop of the routine.
- `prev_x` and `prev_y`—Store the previous coordinates from which the line is drawn.

```
printf ("\n\n activity graphical information\n");
printf ("==============================\n");
```

These statements are the title of the output; they begin the retrieved information for each activity retrieved.

```
draw_string (ac_name, ac_graphic->ac_name_color,
   ac_graphic->ac_x_coor,ac_graphic->ac_y_coor);
```

This uses the routine `draw_string` to plot the activity name. It sends as parameters the string that holds the name, color, and X-Y coordinates where the name is drawn.

```
for  (i=1; iac_polygon.points_no; i++)
```

Plots the element. It loops through the coordinates and draw lines between the control points of the box representing the activity.

```
{
  prev_x = ac_graphic->ac_polygon.outline[i-1].x;
  prev_y = ac_graphic->ac_polygon.outline[i-1].y;
  draw_line(ac_graphic->ac_color,prev_x,prev_y,
    ac_graphic->ac_polygon.outline
         [i % ac_graphic->ac_polygon.points_no].x,
    ac_graphic->ac_polygon.outline
         [i % ac_graphic->ac_polygon.points_no].y);
  }
}
```

For each coordinate, the following parameters are passed to the `draw_line` routine:

- The color of the activity
- The X-Y coordinates to start and end the line

The `draw_line` function is called for each side of the activity individually.

Note

The lines of code beginning with `ac_graphic` to `.x` and `ac_graphic` to `.y` should be entered on a single line, which cannot be shown in the example.

Writing the Textual Information

```
void activity_text_output(ac_text)
```

This function defines how textual information retrieved from the database is output. In this example, this information is written into the report. It can also be manipulated by your own report generator, if desired.

```
stm_ac_text_ptr ac_text;
```

Declares the pointer to the activity's textual record.

```
printf("\n\n\n\n\n activity textual information\n");
printf("============================\n");
```

These statements generate a title for the output; they precede the retrieved information for each activity.

```
printf("\n activity name: %s", ac_text->ac_name);
printf("\n activity in chart: %d", ac_text->ac_chart);
printf("\n activity unique name: %s",
      ac_text->ac_uniquename);
printf("\n activity synonym: %s", ac_text->ac_synonym);
printf("\n activity type:%s",
      activity_type(ac_text->ac_type));
printf("\n activity termination: %s",
      activity_termination_type (
      ac_text->ac_termination));
printf("\n activity short description: %s",
      ac_text->ac_short_des);
printf("\n activity long description:\n\n");
```

Prints the information retrieved from the activity's fields.

```
if ((fd = fopen (ac_text->ac_long_des, "r")) == NIL)
     printf("\n\n cannot open file for printing");
while (fgets(array,81,fd) != NIL) printf("%s", array);
```

These statements loop through the element's long description, printing the lines one by one until a null string is reached.

Drawing the Names of the Elements

```
void draw_string (s, color, x1, y1)
```

This function draws the element's name.

```
char *s;
```

This is the name of the activity.

```
stm_color color;
```

This is the color to be used for the activity name.

```
stm_coordinate x1, y1;
```

This is the coordinate location for placing the activity name.

```
{
 printf("\n string %s at %f, %f in color %d\n",
        s, x1, y1, color);
}
```

The information is printed in a textual report. However, you can write a program routine to use this information in a plot.

Drawing the Activity Box

```
void draw_line(color, x1, y1, x2, y2)
```

This function draws the activity.

```
stm_color color;
```

This sets the color to be used for the activity.

```
stm_coordinate x1, y1, x2, y2;
```

This sets the coordinate locations for the activity's control points.

```
{
  printf("\n line from %f, %f to %f, %f in color %d",
         x1, y1, x2, y2, color);
}
```

The information is printed in a textual report. However, you can write a program routine to use this information in a plot.

Constructing the Activity Type

```
static char      *activity_type (search_for)
stm_activity_type  search_for;
{
static struct search_activity_type {
  stm_activity_type  act_type;
  char              *name;
}

ActivityType[] = {
  {stm_ac_diagram,          "DIAGRAM"},
  {stm_ac_reference,        "REFERENCE"},
  {stm_ac_internal,         "INTERNAL"},
  {stm_ac_instance,         "INSTANCE"},
  {stm_ac_control,          "CONTROL"},
  {stm_ac_control_instance, "CONTROL_INSTANCE"},
  {stm_ac_external,         "EXTERNAL"},
  {NULL,                     NULL},
}
struct search_activity_type  *sat;

  for (sat = ActivityType; sat->name != NULL; sat++)
    if (sat->act_type == search_for)
        return sat->name;
    return "";                /* error!  */
```

This routine matches a string for output to the type of the activity.

Constructing the Activity Termination Type

```
static char *activity_termination_type(search_for)
stm_activity_termination  search_for;
{
static struct search_activity_termination {
  stm_activity_termination  act_term_type;
  char                     *name;
}
ActivityTerminationType[] = {
  {stm_ac_missing,             "MISSING"},
  {stm_ac_self_termination,    "SELF_TERMINATION"},
  {stm_ac_controlled_termination,
       "CONTROLLED_TERMINATION"}, {NULL,NULL},
}
struct search_activity_termination   *sat;

   for (sat = ActivityTerminationType;
        sat->name != NULL; sat++)
     if (sat->act_term_type == search_for)
         return sat->name;
    return "";      /* error!  */
}
```

This routine matches a string for output to the termination type of the activity.

Global Variable Declarations

```
int status = 0;
```

This holds the status of function calls.

```
FILE *fd;
```

This is the file pointer to the file containing the long description.

```
char array[80];
```

This is an array used to hold each line of the long description for output to the report.

Program Definitions

```
#define GET_STR(S)
  {int i; for (i=0;(S[i++] = getchar())!='\n';);
   S[--i]='\0';}
```

This defines a routine to read characters entered at the keyboard.

Include File Statements

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

This includes the standard I/O library.

```
#include "dataport.h"
```

This includes the Dataport Library definitions. In this example, the sample program is assumed to have the same directory path as the file of definitions.

Program Output

The following is the output when the sample C program is executed. Information for an activity-chart in the ACCM project is extracted.

```
        enter name of project: ACCM

        enter activity name: MAIN_AC

        activity textual information

        ============================

        activity name: SC_ACTIVITIES

        activity in chart: 12

        activity unique name: SC_ACTS_CH:SC_ACTIVITIES

        activity synonym: SCA

        activity type: INTERNAL

        activity termination: MISSING

        activity short description: Speed Controller
        Activities

        activity long description:

        activity graphical information

        ==============================
        string SC_ACTIVITIES at 5.000000,14.208320 in color 7

        line from 4.625000,14.791700 to 4.625000,4.541660 in color 8

        line from 4.625000,4.541660 to 19.958320,4.541660 in color 8 line from
        19.958320,4.541660 to 19.958320,14.791700 in color 8

        line from 19.958320,14.791700 to 4.625000,14.791700 in color 8

activity textual information
============================

        activity name: CONTROL_SC

        activity in chart: 14

        activity unique name: SC_X12:CONTROL_SC

        activity synonym: CSC

        activity type: CONTROL

        activity termination: MISSING

        activity short description: Manage Speed Controller

        activity long description:

        !PURPOSE

        This activity determines when the capabilities of the Speed Controller are
        activated.
```

```
!END_PURPOSE

!BRIEF_DESCRIPTION

This control activity is described by a statechart.

!END_BRIEF_DESCRIPTION

activity graphical information

==============================

string CONTROL_SC at 10.583320,13.041700 in color 13 line from
9.958330,13.791700 to 9.958330,12.791700 in color 3 line from
9.958330,12.791700 to 14.666700,12.791700 in color 3 line from
14.666700,12.791700 to 14.666700,13.791700 in color 3 line from
14.666700,13.791700 to 9.958330,13.791700 in color 3

activity textual information

============================

activity name: SET_CRS_SPEED

activity in chart: 19

activity unique name: SETTING:SET_CRS_SPEED

activity synonym: SDCS

activity type: INTERNAL

activity termination: SELF_TERMINATION

activity short description: Set Desired Cruising Speed

activity long description:

!PURPOSE This activity stores the current speed as the desired cruising
speed.

!END_PURPOSE

!BRIEF_DESCRIPTION

The activity reads the current speed and records it in the data-store that
stores the desired cruising speed.

!END_BRIEF_DESCRIPTION

activity textual information

============================

activity name: CALIBRATE

activity in chart: 19

activity unique name: SETTING:CALIBRATE

activity synonym: CRPM

activity type: INTERNAL

activity termination: SELF_TERMINATION

activity short description: Calibrate Rotations per
Mile

activity long description:
```

```
!PURPOSE

The activity counts and updates the number of drive shaft rotations in a
measured mile for different tire sizes.

!END_PURPOSE

!BRIEF_DESCRIPTION

--- TBD

!END_BRIEF_DESCRIPTION

activity graphical information

==============================

string CALIBRATE at 15.291700,10.583320 in color 7 line from
14.791655,11.354160 to 14.791655,10.312480 in color 8 line from
14.791655,10.312480 to 19.124985,10.312480 in color 8 line from
19.124985,10.312480 to 19.124985,11.354160 in color 8 line from
19.124985,11.354160 to 14.791655,11.354160 in color 8

activity textual information

============================

activity name: MEASURE_SPEED

activity in chart: 18

activity unique name: CHECK:MEASURE_SPEED

activity synonym: MDCS

activity type: INTERNAL

activity termination: CONTROLLED_TERMINATION

activity short description: Measure Distance and Current Speed activity long
description:

!PURPOSE

This activity calculates the current speed, and updates the total mileage of
the car.

!END_PURPOSE

activity graphical information

==============================

string SET_CRS_SPEED at 5.625000,10.666700 in color 7 line from
5.333330,11.500000 to 5.333330,10.458320 in color 8 line from
5.333330,10.458320 to 9.666660,10.458320 in color 8 line from
9.666660,10.458320 to 9.666660,11.500000 in color 8 line from
9.666660,11.500000 to 5.333330,11.500000 in color 8

!BRIEF_DESCRIPTION

The activity measures the distance traveled in a brief time interval, and
calculates the average speed over this time. It adds the distance to the total
mileage and updates the mileage store.

!END_BRIEF_DESCRIPTION

activity graphical information

==============================
```

```
string MEASURE_SPEED at 15.375000,6.416660 in color 7

line from 14.875035,7.229170 to 14.875035,6.187490 in color 8 line from
14.875035,6.187490 to 19.208365,6.187490 in color 8 line from
19.208365,6.187490 to 19.208365,7.229170 in color 8 line from
19.208365,7.229170 to 14.875035,7.229170 in color 8

activity textual information

============================

activity name: MAINTAIN_SPEED

activity in chart: 25

activity unique name: OPERATE:MAINTAIN_SPEED

activity synonym: MDS

activity type: INTERNAL

activity termination: CONTROLLED_TERMINATION

activity short description: Maintain Desired Speed

activity long description:

!PURPOSE

This activity keeps the speed of the vehicle at a desired value.

!END_PURPOSE

!BRIEF_DESCRIPTION

This activity compares the current speed to the desired speed and controls
the throttle accordingly.

!END_BRIEF_DESCRIPTION

activity graphical information

============================

string MAINTAIN_SPEED at 5.916660,6.875000 in color 7 line from
5.374995,7.645840 to 5.374995,6.604160 in color 8 line from 5.374995,6.604160
to 9.708325,6.604160 in color 8 line from 9.708325,6.604160 to
9.708325,7.645840 in color 8 line from 9.708325,7.645840 to 5.374995,7.645840
in color 8

activity textual information

============================

activity name: ACCELERATE

activity in chart: 25

activity unique name: OPERATE:ACCELERATE

activity synonym: CTBAP

activity type: INTERNAL

activity termination: CONTROLLED_TERMINATION

activity short description:

Control Throttle by Accelerator Pedal
```

```
activity long description:

!PURPOSE

This activity controls the speed according to the accelerator pedal position
which is determined by the driver.

!END_PURPOSE

!BRIEF_DESCRIPTION

--- TBD

!END_BRIEF_DESCRIPTION

activity graphical information

==============================

string ACCELERATE at 7.750000,5.166660 in color 7

line from 7.249995,5.937500 to 7.249995,4.895820 in color 8 line from
7.249995,4.895820 to 11.583325,4.895820 in color 8 line from
11.583325,4.895820 to 11.583325,5.937500 in color 8

line from 11.583325,5.937500 to 7.249995,5.937500 in color 8
```

Single-Element Functions

This section provides information about the single-element extraction functions. For each function, the following information is provided:

- Return value type
- The elements for which it is relevant
- Description
- Syntax
- Arguments
- Status codes

The two characters *xx* in the function names denote element type abbreviations. Refer to **Element Type Abbreviations** for the list of element abbreviations. Refer **Data Types** for a list of the data types.

Overview of Dataport Single Element Functions

Single-element functions provide information about discrete Rational 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 Rational Statemate element, you know the element's name (path name). You can retrieve more information about an element, such as the element's synonym or what attributes are defined in the element's form, using the single-element functions.

Complete the following steps to obtain more information about a Rational Statemate element:

1. Specify the element name or synonym. Receive the element `ID`.
2. Specify the `ID` and the information requested. Receive the extracted information
3. Use the extracted information.

The element `ID` is an internal representation that Rational 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 to process your information request.

Note

- Multiple functions can be called in succession for the same element. Each extracts different types of information.
- There are functions that extract records of all information on an element. You can then use fields of this record, instead of using the individual functions for each type of information.

Calling Single-Element Functions

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 Rational 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 Rational 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 Rational 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.

Note

In addition to the Stage 1 functions, there are other ways to obtain an element's ID. In Stage 2 functions, IDs are passed as arguments to identify elements in the database.

Single-Element Function Input Arguments

The following table lists the input arguments for single-element functions.

Argument	Function	Data Type
name	The name of the Rational Statemate element. It can be an element name, path name, or synonym, including the chart name, (for example, K:L.M).	`stm_element_name, stm_short_name, or stm_pathname`
element ID	The value that Rational Statemate uses to identify each element in the database. Rational Statemate assigns a unique ID to every element.	`stm_id`
attribute name	The name of an attribute defined in the form for a Rational Statemate element (in the **Attribute** field).	`stm_attr_name`
begin keyword	The string of text appearing in the long description attached to the specified Rational Statemate element. This string represents the beginning of the portion of the long description that you want to extract from the database.	`char * (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.	`char * (string)`
filename	The path name of a system file. Long descriptions (and portions thereof) are copied to the system file specified in this argument. Information is copied to a file as follows: • If you specify a directory name and file name, the text is copied to this file. • If you specify a file name, the file is written to the current workarea. • If you specify neither a directory name nor a file name (you pass an empty string, `""`, as the argument), the file is written to the `/tmp` directory. The file is erased from this directory after you finish working with the Dataport. The name of the file is the value returned by the function.	`stm_filename`

Single-Element Function Examples

This section provides several examples of single-element functions used to extract information from the Rational Statemate database.

Example 1: Returning a State's Synonym and Description

To find the synonym and the short description for a state `S1` as it appears in the state's form (the pathname `CH:SSS.S1` uniquely identifies the state), include the following code in the C program:

```
stm_id              state_id;
stm_description     state_desc;
stm_short_name      state_syn;
int                 status;

state_id  = stm_r_st ("CH:SSS.S1", &status);
state_syn = stm_r_st_synonym (state_id, &status);
state_desc = stm_r_st_description (state_id, &status);
```

Two consecutive function calls are used to extract the synonym and the short description of the same element. The assigned variable and the function return value must have compatible data types; therefore, `state_id` is declared as `stm_id`.

Example 2: Returning Enumerated Type Values

To return enumerated type values, include the following code:

```
stm_id                state_id;
stm_element_name      state_name;
char                  *state_type;
stm_state_type        st_type;
int                   status;
                      .
                      .
                      .
state_id = stm_r_st (state_name, &status);
st_type = stm_r_st_type (state_id, &status);
   switch  (st_type) {
     case  stm_st_or:
           strcpy (state_type,"or"); break;
     case  stm_st_and:
           strcpy (state_type,"and"); break;
                      .
                      .
                      .
}

printf ("The state %s is of type %s",
        state_name, state_type);
```

Example 3: Writing a Portion of the Long Description

To output the portion of a long description appearing between the strings `"!BEGIN"` and `"!END"` for state `S1`, use the following code:

```
stm_id           state_id;
stm_filename     descr_file;
                      .
                      .
                      .
state_id = stm_r_st ("S1", &status);
descr_file = stm_r_st_keyword (state_id,
    "!BEGIN", "!END", "", &status);
                      .
                      .
                      .
```

The fourth input parameter (empty string) of the function `stm_r_st_keyword` determines the file name to which the extracted text is written. If the string is empty, as it is in this case, the function creates a temporary file. The name of this file is returned by the function (in this case, the assignment statement stores the returned file name in `descr_file`).

Example 4: Extracting Textual Information

To extract an entire record of all textual information for an activity, and then use the individual fields in subsequent calls, use the following code:

```
stm_id             act_id;
stm_ac_text_ptr    act_textual;
stm_description    act_desc;
stm_short_name     act_syn;
int                status;
                   .
                   .
                   .
act_id     = stm_r_ac ("AAA.A1", &status);
act_textual= stm_r_ac_text (act_id, &status);
act_syn    = act_textual->ac_synonym;
act_desc   = act_textual->ac_short_des;
```

Note the difference between this example and the first example. In Example 1, single functions are used for each type of information; here, the whole record is extracted and the information from each field is used later.

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`).

For ease-of-use, the functions are presented in alphabetical order, by task. The functions are as follows:

Function	Description
stm_check_out_item	Checks out a chart file (or any other configuration item file) into the current workarea.
stm_check_in_item	Checks in a chart file (or any other configuration item file) from the current workarea.
stm_lock_item	Lock a chart file (or any other configuration item file) in the current workarea.
stm_unlock_item	Unlock a chart file (or any other configuration item file) in the current workarea.
stm_r_ac_mini_spec_hyper	Returns a string with the mini-spec, including hyperlinks to referenced elements.
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 that is connected to the specified activity.
stm_r_ac_termination	Returns the activity termination type specified in the activity form.
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_inherited_gds	Retrieves the description of the specified continuous chart.
stm_r_ch_access_status	Returns the status of charts in the workarea, that is, Read, Update, or New.
stm_r_ch_creator	Returns the date (as a string) on which the specified chart was created.

stm_r_ch_creator	Returns the name of the Rational Statemate user who created the specified chart.
stm_r_ch_modification_date	Returns the date in which the version of the chart in the workarea was saved in the databank.
stm_r_ch_modification_status	Returns the chart modification status of the specified chart.
stm_r_ch_usage_type	Returns the usage type for a chart.
stm_r_ch_version	Returns the version of the specified chart.
stm_r_cn_value	Returns a value associated with a diagram connector.
stm_r_co_default_val	Returns the default value associated with the specified element.
stm_r_ddb_list_names	Returns the names of the lists created by the property sheet.
stm_r_design_attr	Retrieves the information on the element's Design-Attributes as would appear when using the **Info** tool within Rational Statemate.
stm_r_dt_enum_values	Returns a list of the enum values ids for the specified User Defined Type.
stm_r_element_type	Returns the element type of the specified element.
stm_r_elem_in_ddb_list	Return a list of stm_id's stored in a specified list_name.
stm_r_formal_parameter_names	Returns a list of names of formal parameters that appear in bindings of instance boxes and components.
stm_r_gds_visibility_mode	Returns the visibility mode for the specified global definition set (GDS).
stm_r_hyper_key	Retrieves the unique key for the specified element.
stm_r_self_hyper_key	Retrieves the unique key for the specified element.
stm_r_included_gds	Returns the list of global definition sets contained in the specified chart.
stm_r_inherited_gds	Retrieves the list of global definition sets that are "inherited" (included indirectly) by the specified chart.
stm_r_line_width	Retrieve line-width of arrows and boxes
stm_r_md_implementation	Retrieves the implementation type for the specified module.
stm_r_md_purpose	Returns the purpose of the module.
stm_r_msg_all	Returns the textual information associated with a specified element.
stm_r_msg_defined_in_scen	Returns the list of messages in chronological order that are part of a scene, defined by a separator.
stm_r_msg_graphic	Returns the graphical information associated with the specified element.
stm_r_msg_included_in_ord_insig	Returns a list of messages that are bounded by an order-insignificant element.
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_next_msg	Returns the message after (in time) the decomposed sequence diagram.

stm_r_nt_body	Returns a list of strings.
stm_r_oactor	Retrieve id of the actor that corresponds to the actor occurrence
stm_r_omd	Returns the ID of the module that corresponds to the module occurrence.
stm_r_ord_insig_all	Returns the textual information associated with a specified element
stm_r_ord_insig_graphic	Returns the graphical information associated with the specified element.
stm_r_ouc	Retrieve id of the use-case that corresponds to the use-case occurrence
stm_r_ouc	Returns the parameter expression from generic charts and components.
stm_r_previous_msg	Returns the message previous (in time) to the decomposed sequence diagram.
stm_r_sb_action_lang	Retrieves the action language of the specified subroutine.
stm_r_sb_action_lang_expression	Retrieves the action language expression of the specified subroutine.
stm_r_sb_action_lang_local_data	Retrieves the action language local data associated with the specified subroutine.
stm_r_sb_ada_user_code	Returns the Ada code that was manually written for the specified subroutine.
stm_r_sb_ansi_c_user_code	Returns the ANSI C code that was manually written for the specified subroutine.
stm_r_sb_connected_chart	Returns the ID of the procedural statechart connected to the specified subroutine.
stm_r_sb_connected_flowchart	Returns the global data associated with the specified subroutine.
stm_r_sb_global_data_mode	Returns the mode of a subroutine's global variable.
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_sb_parameters	Retrieves the parameters of the subroutine.
stm_r_sb_proc_sch_local_data	Retrieves the local data of the procedural statechart implemented by the specified subroutine.
stm_r_sb_proc_fch_local_data	Retrieves the subroutine's return type.
stm_r_sb_return_user_type	Retrieves the user-defined type ID returned by the subroutine.
stm_r_sb_return_user_type_name_type	Retrieves the subroutine's return user type and name type.
stm_r_sep_all	Returns the textual information associated with a specified element
stm_r_sep_graphic	Returns the graphical information associated with the specified element.
stm_r_st_andlines	Returns a list of the and-lines associated with the specified state.

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_stubs_name	Returns the list of stub names for an instance of a component.
stm_r_tc_all	Returns the textual information associated with a specified element.
stm_r_tc_graphic	Returns the graphical information associated with the specified element.
stm_r_tr_attr_enforced	Returns the enforced attributes specified by `attr_name`.
stm_r_tr_attr_name	Returns the names of attributes associated with the specified element. Attributes are associated with elements through element forms.
stm_r_tr_attr_val	Returns the values associated with a particular attribute name for the specified element.
stm_r_tr_longdes	Returns the long description of the specified Transition.
stm_r_tr_notes	Returns a list of strings each one is a line in the Transition Note related to the specified Transition
stm_r_tt_cell	Returns the string of the specified cell (row & column) in a Truth-Table associated with the specified element.
stm_r_tt_cell_hyper	Retrieves the contents of the specified cell in the given truth table, including hyperlinks to referenced elements.
stm_r_tt_cell_type	Returns the type of the specified cell (row & column) in a Truth-Table associated with the specified element.
stm_r_tt_num_of_col	Retrieves the number of columns (including blank ones) in the specified truth table, as viewed in the truth table editor.
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 (including blank ones) in the specified truth table, as viewed in the truth table editor
stm_r_tt_row	Retrieves the values in the specified row in the truth table.
stm_r_tt_row_hyper	Returns a list of strings that represents a row in the truth table, including hyperlinks to referenced elements.
stm_r_xx	Retrieves the element ID of the specified element.
stm_r_xx_all	Returns both the textual and graphical information associated with a specified element.
stm_r_xx_array_lindex	Returns the left index of an element array.
stm_r_xx_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	Retrieves 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 specified elements.
stm_r_xx_cbk_binding_enable	Retrieves the enabled callback bindings.
stm_r_xx_cbk_binding_expression	Retrieves the callback binding expressions.
stm_r_xx_chart	Returns the chart ID for the specified element.
stm_r_xx_combinationals	Returns a list of strings.
stm_r_xx_containing_fields	Returns the list of union or record elements that contain fields.
stm_r_xx_data_type	Returns the element subtype, including its data type and data structure.
stm_r_xx_default_val()	Returns the default value associated with the specified element.
stm_r_xx_definition_type	Returns the definition type of the specified textual element.
stm_r_xx_des_attr_name	Returns the names of Design-Attributes associated with the specified element.
stm_r_xx_des_attr_val	Retrieves the values of a given Design-Attribute values associated with the specified element.
stm_r_xx_description	Returns the short description of the specified element.
stm_r_xx_displayed_name	Returns the name of a chart, as it appears in the graphic editor where the specified element is located.
stm_r_xx_explicit_defined_xx	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	Returns the file name associated with the "Link to External File" entry in the element's properties of the specified element.
stm_r_xx_graphic	Returns the graphical information associated with the specified element.
stm_r_xx_instance_name	Returns the name of the instance as it appears in the chart for a specific hierarchical Rational Statemate element.
stm_r_xx_keyword	Retrieves a portion of the element's long description.
stm_r_xx_labels	Returns a list of strings that consists of all the labels of the specified compound transition or message.
stm_r_xx_labels_hyper	Returns a list of strings of message or transition labels, with hyperlinks to referenced elements.

stm_r_xx_longdes	Retrieves 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_xx_mode	Returns the parameter or router mode.
stm_r_xx_name	Returns the element name.
stm_r_xx_note	Returns the notes from a requirement record or timing constraint.
stm_r_xx_notes	Returns the notes in the specified element.
stm_r_xx_number_of_bits	Returns the number of bits in the element.
stm_r_xx_of_enum_type	Retrieves the enumerated type ID (a user-defined type) for the specified element.
stm_r_xx_of_enum_type_name_type	Retrieves the enumerated name type for the specified elements.
stm_r_xx_parameter_mode	Retrieves the parameter mode, including subroutine parameters and the parameters of generic charts and components.
stm_r_xx_reactions	Returns the static reactions of the specified state.
stm_r_xx_select_implementation	Retrieves the implementation type of the specified element.
stm_r_xx_string_length	Retrieves the string length of the specified element.
stm_r_xx_structure_type	Returns the structure or type of the specified textual element. The structure or type can be single, array, or queue.
stm_r_xx_synonym	Retrieves the synonym of the specified element. The synonym is defined in the element's form.
stm_r_xx_text	Returns the textual information associated with a specified element.
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 named elements.
stm_r_xx_truth_table_local_data	Returns the list of local data elements defined in the truth table related to the input subroutine.
stm_r_xx_type	Retrieves 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_xx_user_type	Returns the user-defined type ID referenced by the element.
stm_r_xx_user_type_name_type	Returns the user-defined type ID referenced by the element.

stm_check_out_item

Function Type

None

Description

Checks out a chart file (or any other configuration item file) into the current workarea.

Note

If the version string parameter is an empty string or NULL, the function regards the latest version of the chart/file in the Databank.

Syntax

```
stm_check_out_item (file_name, ext, version, with_lock, error_func, &status)
```

Status Codes

- `stm_success`
- `stm_error_in_load_operation`

Arguments

Argument Name	Input/Output	Argument Type	Argument Description
`item_name`	Input	`char *`	Name of chart/file to check-out

ext	Input	char *	Rational Statemate chart/file extensions. The possible values are as follows: • **sch –** Statechart • **ach** - Activity charts • **mch** - Module-charts • **fch –** Flowcharts • **dic** - Global Definition Set files • **qch** - Sequence-Diagrams • **uch** - Use-Case-Diagrams • **vsm -** Continuous Diagrams • **pnl –** Panel files • **scp -** Simulation SCL files • **cnf** - Simulation status files • **wpf -** Waveform Profiles • **dyn_set** - Simulation analysis profiles • **mon** - Monitor files • **chk_mdl_set -** Check Model Profiles ² **dgl** - Documentor templates • **inc** - Documentor include files • **pnl** - Prototype panels • **config** - Configuration files • **tv** - Task View files • **mak** - Makefiles • **oil** - OIL files • **cfg** - CFG files • **c** - Source(c) files • **h** - Header (h) files • **rgenset** - Rapid Prototyper Profiles • **trg** - Target files • **rtrg** - Rapid Target files • **crd** - Card files • **rconfig** - Rhapsody block Configuration files • **ccf** - Component Configuration files • **dat** - VSM Data files • **wav** - VSM Wave files • **mat** - VSM Mat. Files • **m** - VSM M. files
version	Input	char *	Version of the chart/file
with_lock	Input	int	A Boolean argument that indicates whether to lock the chart/file

`error_func`	input	`void (*) (const char* err_msg).`	pointer to a function of the following prototype: `void err_func(const char* err_msg)` If the `error_func` pointer is not NULL, this function will be called with an error message string, when the returned status is not `stm_success`. The error function may be called more than once during one checkout operation, with different error messages.
`status`	output	`int`	The function status code.

stm_check_in_item

Function Type

None

Description

Checks in a chart file (or any other configuration item file) from the current workarea.

Syntax

```
stm_check_in_item (file_name, ext, with_lock, error_func, &status)
```

Status Codes

- `stm_success`
- `stm_error_in_load_operation`

Arguments

see arguments for **stm_check_out_item**

stm_lock_item

Function Type

None

Description

Lock a chart file (or any other configuration item file) in the current workarea.

Syntax

```
stm_lock_item (file_name, ext, error_func, &status)
```

Status Codes

- `stm_success`
- `stm_error_in_load_operation`

Arguments

see arguments for **stm_check_out_item**

stm_unlock_item

Function Type

None

Description

Unlock a chart file (or any other configuration item file) in the current workarea.

Syntax

```
stm_unlock_item (file_name, ext, error_func, &status)
```

Status Codes

- `stm_success`
- `stm_error_in_load_operation`

Arguments

see arguments for **stm_check_out_item**

stm_r_ac_mini_spec_hyper

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

Function Type

```
stm_expression
```

For Elements

activity	ac

Syntax

Arguments

Argument	Input/ Output	Type	Description
elem	In	stm_id	The element ID.
format	In	char *	Either FrameMaker or Microsoft Word.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_id_not_found
- stm_unresolved

stm_r_ac_subroutine_bind

Returns the subroutine binding connected to the specified activity.

Function Type

`stm_list`

For Elements

`activity`	`ac`

Syntax

```
stm_r_ac_subroutine_bind (ac_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`ac_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_unresolved`
- `stm_id_not_found`
- `stm_missing_subroutine_binding`

stm_r_ac_subroutine_bind_enable

Determines whether the subroutine bound to the specified activity is enabled or disabled.

Function Type

```
int (predefined constant)
```

For Elements

activity	ac

Syntax

```
stm_r_ac_subroutine_bind_enable (ac_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
ac_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_unresolved
- stm_id_not_found

Return Values

Although the return value of this function is of type `int`, Dataport enables you to reference this value by name. The possible values are as follows:

- stm_ac_cbk_enable
- stm_ac_cbk_disable
- stm_ac_cbk_bind_missing

stm_r_ac_subroutine_bind_expr

Returns the subroutine binding expression that is connected to the specified activity.

Function Type

`stm_expression`

For Elements

`activity`	`ac`

Syntax

```
stm_r_ac_subroutine_bind_expr (ac_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`ac_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_id_not_found`
- `stm_unresolved`
- `stm_missing_subroutine_binding`

stm_r_ac_termination

Returns the activity termination type specified in the activity form.

Function Type

```
stm_activity_termination
```

For Elements

activity	ac

Syntax

```
stm_r_ac_termination (act_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
act_id	In	stm_id	The activity whose termination type you want to retrieve.
status	Out	int	Function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_id_not_found
- stm_unresolved

Return Values

The function output has an enumerated type, stm_activity_termination, with three possible values:

- stm_ac_self_termination
- stm_ac_controlled_termination
- stm_ac_missing

Example

To determine the termination type of the activity `A1` and if the activity is *self-terminated* write the activity's name, use the following statements:

```
stm_id                        act_id;
stm_activity_termination      act_term_type;
int                           status;
   .
   .
act_id = stm_r_ac ("A1", &status);
act_term_type = stm_r_ac_termination (act_id, &status);
if(act_term_type == stm_ac_self_termination)
   printf ("\n Self-terminated activity:", "A1");
   .
   .
   .
```

stm_r_ac_xx_ac

Provides a list of activities.

Funtion type:

```
stm_list
```

For Elements:

actor	
boundary box	bb
ext_ll	
external router	
lifeline	ll
router	router
use case	use

Syntax:

```
STM_R_AC_XX_AC (IN el_list: LIST OF ACTIVITY, OUT status: INTEGER):LIST OF
XX;
```

Arguments:

Argument	Input/ Output	Type	Description
activities_list	in	stm_list	List of Activities
status	out	int	Function Status Code

Status Codes:

- stm_success
- stm_nil_list

stm_r_actual_parameter_exp

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

Function Type

`stm_expression`

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
xx_inst_boxid	In	stm_id	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

`stm_element_type`

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	`stm_id`	The element ID.
`formal_param_name`	In	`string`	The formal parameter name.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_name_not_found`

stm_r_cd_info

Retrieves the description of the specified continuous chart.

Function Type

```
stm_expression
```

For Elements

chart	ch

Syntax

```
stm_r_cd_info (ch, &status)
```

Arguments

Argument	Input/ Output	Type	Description
cd	In	stm_id	The chart.
status	Out	int	The function status code.

Status Codes

- stm_id_out_of_range
- stm_null_string
- stm_success

stm_r_changes_log

Documents all the changes made to the specified charts in a log file.

Function Type

```
stm_list
```

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	stm_id LIST OF CHART	The list of charts to track.
ascending	In	stm_boolean BOOLEAN	Determines whether the changes are listed in ascending order TRUE).
per_date	In	stm_boolean BOOLEAN	Determines whether the changes are listed chronologically (TRUE).
dont_format	In	stm_boolean 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	int	The function status code.

Status Codes

- stm_id_out_of_range
- stm_not_chart_id
- stm_id_not_found
- stm_success

stm_r_ch_access_status

Returns the status of charts in the workarea, that is, Read, Update, or New.

Function Type

```
stm_chart_access_status
```

Syntax

```
stm_r_ch_access_status (ch_id, int *status)
```

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_name_not_found`

Return Values

- `stm_chac_readonly`
- `stm_chac_update`
- `stm_chac_new`

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

`stm_date`

For Elements

chart	ch

Syntax

```
stm_r_ch_creation_date (ch_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`ch_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_not_found`
- `stm_id_out_of_range`
- `stm_unresolved`

Example

To return the chart date, use the following statements:

```
stm_id       chart_id;
int          status;
             .
             .
chart_id = stm_r_ch ("TOP", &status);
printf ("\n Chart created on: %s",
   stm_r_ch_creation_date (chart_id, &status));
             .
             .
The date output is the date on which the chart named TOP was created.
```

stm_r_ch_creator

Returns the name of the Rational Statemate user who created the specified chart.

Note: This function is relevant only for charts that were explicitly created using one of the graphic editors.

Function Type

```
stm_user_name
```

For Elements

chart	ch

Syntax

```
stm_r_ch_creator (ch_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
ch_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_id_not_found
- stm_unresolved

Example

To return the name of the user who created the chart, use the following statements:

```
stm_id       chart_id;
int          status;
             .
             .
             .
chart_id = stm_r_ch ("TOP", &status);
printf ("\n Chart created by: %s",
   stm_r_ch_creator (chart_id, &status));
             .
             .
             .
The name output is the name of the user who created the chart named
TOP.
```

stm_r_ch_modification_date

Returns the date in which the version of the chart in the workarea was saved in the databank.

Note: This function is relevant only for charts that were explicitly defined using one of the graphics editors.

Function Type

`stm_date`

For Elements

`chart`	`ch`

Syntax

```
stm_r_ch_modification_date (ch_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`ch_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_id_not_found`
- `stm_unresolved`

Example

To return the date of the last modification for a chart, use the following statements:

```
stm_id       chart_id;
    int          status;
                    .
                    .
chart_id = stm_r_ch ("TOP", &status);
printf ("\n Chart modified on: %s",
    stm_r_ch_modification_date (chart_id, &status));
                    .
                    .
The date output is the date on which the chart named TOP was last
modified.
```

stm_r_ch_modification_status

Returns the chart modification status of the specified chart. The possible values are:

- U—Unmodified
- M—Modified
- N—New
- D—Deleted

Function Type

```
stm_chart_mod_status
```

For Elements

chart	ch

Syntax

```
stm_r_ch_modification_status (ch_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
ch_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_not_found
- stm_id_out_of_range
- stm_unresolved

stm_r_ch_usage_type

Returns the usage type for a chart.

Function Type

```
stm_chart_usage
```

For Elements

chart	ch

Syntax

```
stm_r_ch_usage_type (ch_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`ch_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_id_not_found`

Return Value

Although the return value of this function is of type `int`, Dataport enables you to reference this value by name. The possible values are:

- `stm_ch_usage_generic`
- `stm_ch_usage_normal`
- `stm_ch_usage_ref_generic`
- `stm_ch_usage_ref_offpage`
- `stm_ch_usage_ref_describing`

stm_r_ch_version

Returns the version of the specified chart.

Function Type

```
char *
```

For Elements

chart	ch

Syntax

```
stm_r_ch_version (ch, &status)
```

Arguments

Argument	Input/ Output	Type	Description
ch	In	stm_id	The chart whose version you want to retrieve.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_not_found
- stm_id_out_of_range
- stm_unresolved

stm_r_cn_value

Returns a value associated with a diagram connector. The value is a string (maximum 32 characters) reflecting an associated number or label. The input argument `cn_id` is an element ID of a diagram connector.

Function Type

```
char *
```

For Elements

diagram connector	

Syntax

```
stm_r_cn_value (cn_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`cn_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_id_not_found`
- `stm_not_diagram_connector`

Example

To assign (to `cn_value`) the label of a specific diagram connector (identified by `cn_id`), use the following statement:

```
.
        .
cn_value = stm_r_cn_value (cn_id, &status);
        .
        .
```

stm_r_co_default_val

Returns the default value associated with the specified element.

Function Type

```
char *
```

Syntax

```
stm_r_co_default_val (st_id, int &status)
```

Arguments

Argument	Input/ Output	Type	Description
`status`	Out	`int`	The function status code.

Status Codes

```
stm_success
```

stm_r_ddb_list_names

Returns the names of the lists created by the properties browser.

Function Type

```
stm_list
```

Syntax

```
stm_r_ddb_list_names (&status)
```

Arguments

Argument	Input/ Output	Type	Description
`status`	Out	`int`	The function status code.

Status Codes

```
stm_success
```

stm_r_design_attr

Retrieves the information on the element's Design-Attributes as would appear when using the **Info** tool within Rational Statemate.

Syntax

```
stm_r_design_attr (stm_id, int *status)
```

Arguments

Argument	Input/ Output	Type	Description
`stm_id`	In		The ID of the element being queried.
`int`	Out		The status of the query.
`status`	Out	`int`	The function status code.

stm_r_dt_enum_values

Returns a list of the enum values ids for the specified User Defined Type.

Function Type

```
stm_list
```

For Elements

User-defined type

Syntax

```
stm_r_dt_enum_values (dt_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`dt_id`	In	`stm_id`	The User-Defined Type ID.
`status`	Out	`int`	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range

stm_r_element_type

Returns the element type of the specified element.

Function Type

```
stm_element_type
```

For Elements

All types

Syntax

```
stm_r_element_type (id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_id_not_found`

Return Values

The return value belongs to the enumerated type `stm_element_type`. This type has the following values corresponding to the Rational Statemate element types:

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`

Element Type	Value
connector in statechart	stm_s_connector
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

Element Type	Value
router	stm_router stm_external_router
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

To list all the conditions appearing in the **Definition** field for the condition `C1`, generate a list of elements (of type mixed) using the query function `stm_r_mx_in_definition_of_co`. Elements in this list are all the elements (not necessarily conditions) appearing in the **Definition** field of the condition `C1`. Search this list for conditions and if any are found, print them.

The program contains the following statements:

```
stm_id              cond_id;
stm_list            elmnt_list, co_list;
stm_id              el;
stm_element_type    el_type;
int                 status;
     .
     .
cond_id = stm_r_co ("C1", &status);
co_list = stm_list_create (cond_id, end_of_list,
    &status);
elmnt_list = stm_r_mx_in_definition_of_co (co_list,
    &status);
for (el = (stm_id)
   stm_list_first_element (elmnt_list, &status);
   status == stm_success;
   el = (stm_id)
        stm_list_next_element (elmnt_list, &status))
   {
    el_type = stm_r_element_type (el, &status);
    if (el_type == stm_condition)
       printf ("\n Condition Name:%s",
               stm_r_co_name (el, &status));
   }
     .
     .
     .
```

stm_r_elem_in_ddb_list

Return a list of stm_id's stored in a specified list_name.

Function Type

```
stm_list
```

Syntax

```
stm_r_elem_in_ddb_list (list_name, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list_name`	In		Path name to the list.
`status`	Out	`int`	The function status code.

Status Codes

- stm_success
- stm_no_such_list

stm_r_formal_parameter_names

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

Function Type

`stm_list`

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	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range

stm_r_gds_visibility_mode

Returns the visibility mode for the specified Global Definition Set (GDS).

Function Type

```
int
```

For Elements

element ID	

Syntax

```
stm_r_gds_visibility_mode (gds_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
gds_id	In	stm_id CHART	The GDS whose visibility you want to retrieve.
status	Out	int	The function status code.

Status Codes

- stm_id_out_of_range
- stm_success

Return Values

Although the return value of this function is of type INTEGER, the Dataport enables you to reference this value by name. The name is internally defined as a predefined constant in DGL. The possible values are as follows:

- stm_explicit_usage
- stm_public_usage

stm_r_hyper_key

Retrieves the unique key for the specified element.

Function Type

```
char *
```

For Elements

element ID	ac

Syntax

```
stm_r_hyper_key (el, &status)
```

Arguments

Argument	Input/ Output	Type	Description
el	In	stm_id	The element ID whose key you want.
status	Out	int	The function status code.

Status Codes

- stm_id_out_of_range
- stm_success

stm_r_self_hyper_key

Retrieves the unique key for the specified element, but for id's of instance-boxes (offpage or generic), returns the hyper-key of the instance-box, and not of the connected chart.

Function Type

```
char *
```

For Elements

element ID	ac

Syntax

```
stm_r_self_hyper_key (el, &status)
```

Arguments

Argument	Input/ Output	Type	Description
el	In	stm_id	The element ID whose key you want.
status	Out	int	The function status code.

Status Codes

- stm_id_out_of_range
- stm_success

stm_r_included_gds

Returns the list of global definition sets contained in the specified chart.

Function Type

`stm_list`

For Elements

`chart`	`ch`

Syntax

```
stm_r_included_gds (ch_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`ch_id`	In	`stm_id`	The chart
`status`	Out	`int`	The function status code

Status Codes

- `stm_id_out_of_range`
- `stm_use_all_public_gds`
- `stm_success`

stm_r_inherited_gds

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

Function Type

```
stm_list
```

For Elements

chart	ch

Syntax

```
stm_r_inherited_gds (ch_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`ch_id`	In	`stm_id`	The chart.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_id_out_of_range`
- `stm_use_all_public_gds`
- `stm_success`

stm_r_md_implementation

Retrieves the implementation type for the specified module.

Function Type

```
char *
```

For Elements

module	md

Syntax

```
stm_r_md_implementation (md_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
md_id	In	stm_id	The module ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_id_not_found
- stm_unresolved
- stm_not_instance

stm_r_md_purpose

Returns the purpose of the module.

Function Type

```
stm_module_purpose_type
```

For Elements

`module`	`md`

Syntax

```
stm_r_md_purpose (id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_not_found`
- `stm_id_out_of_range`

stm_r_msg_all

Returns the textual information associated with a specified element.

The information is retrieved into a structured data type (record) that varies according to the type of element referenced.

Function Type

```
stm_msg_all_ptr
```

For Elements

Message

Syntax

```
stm_r_msg_all (msg_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`msg_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_unresolved`
- `stm_id_not_found`

stm_r_msg_defined_in_scen

Returns the list of messages in chronological order that are part of a scene, defined by a separator.

Function Type

`stm_list`

For Elements

`List of separators`	`sep_lst`

Syntax

```
stm_r_msg_defined_in_scen (sep_list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`sep_lst`	In	`stm_list`	A list of element IDs
`status`	Out	`int`	The function status code

Status Codes

- `stm_success`
- `stm_id_out_of_range`

stm_r_msg_graphic

Returns the graphical information associated with the specified element.

Function Type

```
stm_msg_graphic_pt
```

For Elements

Message

Syntax

```
stm_r_msg_graphic (msg_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`msg_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_unresolved`
- `stm_id_not_found`

stm_r_msg_included_in_ord_insig

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

Function Type

`stm_list`

For Elements

List 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	`stm_list`	A list of elements
`status`	Out	`int`	The function status code

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_not_order_insignificant`

stm_r_msg_where_tc_begins

Returns the message where the timing constraint begins.

Function Type

`stm_id`

For Elements

`message`	`msg`

Syntax

```
stm_r_msg_where_tc_begins (tc_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`tc_id`	In	`stm_id TIMING CONSTRAINT`	The timing constraint.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_not_timing_constraint`

stm_r_msg_where_tc_ends

Returns the message where the timing constraint ends.

Function Type

```
stm_id
```

For Elements

`message`	`msg`

Syntax

```
stm_r_msg_where_tc_ends (tc_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`tc_id`	In	`stm_id`	The timing constraint.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_not_timing_constraint`

stm_r_next_msg

Returns the message after (in time) the decomposed sequence diagram.

Function Type

`stm_id`

For Elements

decomposed SD	dec_sd

Syntax

```
stm_r_next_msg (dec_sd_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`dec_sd_id`	In	`stm_id REFERENCED_SD`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_not_decomposed_sd`
- `stm_message_not_found`

stm_r_nt_body

Returns a string, the context of the note.

Function Type

```
char *
```

For Elements

notes	nt

Syntax

```
stm_r_nt_body (id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The element ID
status	Out	int	The function status code

Status Codes

- stm_success
- stm_id_not_found
- stm_id_out_of_range

stm_r_oactor

Returns the ID of actor that corresponds to the actor occurrence.

Function Type

`stm_id`

For Elements

`actor`	`actor`

Syntax

```
stm_r_oactor (id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_id_not_found`

stm_r_omd

Returns the ID of the module that corresponds to the module occurrence.

Function Type

`stm_id`

For Elements

`module occurrence`	`om`

Syntax

```
stm_r_omd (id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_id_not_found`

Example

To determine the module occurrences for each module, use the following statements:

```
     stm_list             md_list, om_list;
     stm_id               om_id;
     int                  status;
                              .
                              .
om_list = stm_r_om_of_md (md_list, &status);
for(om_id = (stm_id)
    stm_list_first_element (om_list, &status);
    status == stm_success;
    om_id = (stm_id)
    stm_list_next_element (om_list, &status))
printf ("module occurrence %d corresponds to
    module %s", om_id, stm_r_md_name (
    stm_r_omd (om_id, &status), &status));
```

stm_r_ord_insig_all

Returns the textual information associated with a specified element.

The information is retrieved into a structured data type (record) that varies according to the type of element referenced.

Function Type

```
stm_ord_insig_all_ptr
```

For Elements

Order Insignificant Line

Syntax

```
stm_r_ord_insig_all (ord_insig_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`ord_insig_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_unresolved`
- `stm_id_not_found`

stm_r_ord_insig_graphic

Returns the graphical information associated with the specified element.

Function Type

```
stm_ord_insig_graphic_ptr
```

For Elements

Order Insignificant Line

Syntax

```
stm_r_ord_insig_graphic (ord_insig_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`ord_insig_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_unresolved`
- `stm_id_not_found`

stm_r_ouc

Returns the ID of use-case that corresponds to the use-case occurrence.

Function Type

`stm_id`

For Elements

`use-case`	`uc`

Syntax

```
stm_r_ouc (id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_id_not_found`

stm_r_parameter_binding

Returns the parameter expression from generic charts and components.

Function Type

```
stm_expression
```

For Elements

chart	ch

Syntax

```
stm_r_parameter_binding (xx_paramid_in_gen, inst_boxid,&status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_paramid_in_gen	In	stm_id	The element ID.
inst_boxid	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_param_not_compatible
- stm_name_not_found
- stm_not_a_parameter

stm_r_previous_msg

Returns the message previous (in time) to the decomposed sequence diagram.

Function Type

`stm_id`

For Elements

`decomposed SD`	`dec_sd`

Syntax

```
stm_r_previous_msg (dec_sd_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`dec_sd_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_not_decomposed_sd`
- `stm_message_not_found`

stm_r_sb_action_lang

Retrieves the action language of the specified subroutine.

Function Type

```
stm_list
```

For Elements

subroutine	sb

Syntax

```
stm_r_sb_action_lang (sb_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
sb_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_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

`stm_expression`

For Elements

`subroutine`	`sb`

Syntax

```
stm_r_sb_action_lang_expression (sb_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`sb_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_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

```
stm_list
```

For Elements

subroutine	sb

Syntax

```
stm_r_sb_action_lang_local_data (sb_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
sb_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_missing_local_data
- stm_id_not_found
- stm_id_out_of_range

stm_r_sb_ada_user_code

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

Function Type

`stm_list`

For Elements

`subroutine`	`sb`

Syntax

```
stm_r_ada_user_code (sb_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`sb_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_missing_global_data`
- `stm_missing_local_data`
- `stm_missing_subroutine_params`
- `stm_missing_user_code`
- `stm_no_connected_chart`

stm_r_sb_ansi_c_user_code

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

Function Type

```
stm_list
```

For Elements

subroutine	sb

Syntax

```
stm_r_sb_ansi_c_user_code (sb_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
sb_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_missing_user_code

stm_r_sb_connected_chart

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

Function Type

`stm_id`

For Element

`subroutine`	`sb`

Syntax

```
stm_r_sb_connected_chart (sb_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`sb_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_not_found`
- `stm_id_out_of_range`
- `stm_no_connected_chart`

stm_r_sb_connected_statechart

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

Function Type

```
stm_list
```

For Element

`subroutine`	`sb`

Syntax

```
stm_r_sb_connected_statechart(sb_id, &status);
```

Arguments

Argument	Input/ Output	Type	Description
`sb_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_not_found`
- `stm_id_out_of_range`
- `stm_no_connected_chart`

stm_r_sb_connected_flowchart

Returns the ID of the Flowchart connected to the specified subroutine.

Function Type

`stm_list`

For Element

`subroutine`	`sb`

Syntax

```
stm_r_sb_connected_flowchart (sb_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`sb_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_not_found`
- `stm_id_out_of_range`
- `stm_no_connected_chart`

stm_r_sb_global_data

Function Type

```
stm_list
```

For Elements

`subroutine`	`sb`

Syntax

```
stm_r_sb_global_data (sb_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`xx_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_missing_global_data`
- `stm_id_out_of_range`
- `stm_id_not_found`

stm_r_sb_global_data_mode

Returns the mode of a subroutine's global variable.

Function Type

```
stm_parameter_mode
```

For Elements

subroutine	sb

Syntax

```
stm_r_sb_global_data_mode (fn_id, stm_id pd_id, int *status)
```

Arguments

Argument	Input/ Output	Type	Description
fn_id			The subroutine ID.
pd_id			The global variable ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_missing_global_data
- stm_id_out_of_range
- stm_id_not_found

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

`stm_list`

For Elements

`subroutine`	`sb`

Syntax

```
stm_r_sb_kr_c_user_code (sb_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`sb_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_missing_user_code`

stm_r_sb_parameters

Retrieves the parameters of the subroutine.

Function Type

```
stm_list
```

For Elements

subroutine	sb

Syntax

```
stm_r_sb_paramaters (sb_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
sb_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_not_a_parameter
- stm_missing_subroutine_params

stm_r_sb_proc_sch_local_data

Retrieves the local data of the procedural statechart implemented by the specified subroutine.

Function Type

```
stm_list
```

For Elements

subroutine	sb

Syntax

```
stm_r_sb_proc_sch_local_data (sb_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
sb_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_missing_local_data
- stm_no_connected_chart

stm_r_sb_proc_fch_local_data

Retrieves the local data of the Flowchart implemented by the specified subroutine.

Function Type

`stm_list`

For Elements

`subroutine`	`sb`

Syntax

```
stm_r_sb_proc_fch_local_data (sb_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`sb_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_missing_local_data`
- `stm_no_connected_chart`

stm_r_sb_return_type

Retrieves the subroutine's return type.

Function Type

```
stm_sb_return_type
```

For Elements

subroutine	sb

Syntax

```
stm_r_sb_return_type (sb_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
sb_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_id_not_found

stm_r_sb_return_user_type

Retrieves the user-defined type ID returned by the subroutine.

Function Type

`stm_id`

For Elements

`subroutine`	`sb`

Syntax

```
stm_r_sb_return_user_type (sb_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`sb_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_id_not_found`
- `stm_missing_user_type`

stm_r_sb_return_user_type_name_type

Retrieves the subroutine's return user type and name type.

Function Type

```
stm_name_type
```

For Elements

subroutine	sb

Syntax

```
stm_r_sb_return_user_type_name_type (sb_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
sb_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_id_not_found
- stm_missing_user_type

stm_r_sep_all

Returns the textual information associated with a specified element.

The information is retrieved into a structured data type (record) that varies according to the type of element referenced.

Function Type

```
stm_sep_all_ptr
```

For Elements

Partition Line

Syntax

```
stm_r_sep_all (sep_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`sep_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_unresolved`
- `stm_id_not_found`

stm_r_sep_graphic

Returns the graphical information associated with the specified element.

Function Type

```
stm_sep_graphic_ptr
```

For Elements

Partition Line

Syntax

```
stm_r_sep_graphic (sep_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`sep_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_unresolved`
- `stm_id_not_found`

stm_r_st_andlines

Returns a list of the and-lines associated with the specified state. The input argument `ID` is an element ID of an `and-state`.

Function Type

`stm_and_line_list`

For Elements

`and-state`	

Syntax

```
stm_r_st_andlines (id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_not_found`
- `stm_id_out_of_range`
- `stm_not_an_and_state`
- `stm_unresolved`

Note

Refer to **Data Types** for the exact structure of the returned value.

Example

To return a list of the and-lines for a specified `state_id`, use the following call:

```
stm_id                state_id;
int                   status;
stm_and_line_list     st_list;
    .
    .
if (stm_r_st_type (state_id, &status) == stm_st_and)
st_list = stm_r_st_andlines (state_id, &status);
```

stm_r_st_static_reactions

Returns the static reactions defined for the specified state element.

Function Type

```
stm_expression
```

For Elements

`state`	`st`

Syntax

```
stm_r_st_static_reactions (st_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`st_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_not_found`
- `stm_id_out_of_range`
- `stm_unresolved`
- `stm_missing_label`

stm_r_st_static_reactions_hyper

Returns a string with the static reactions, including hyperlinks to referenced elements.

Function Type

`STRING`

For Elements

`state`	`st`

Syntax

```
stm_r_st_static_reactions_hyper (elem, format, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`elem`	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_out_of_range`
- `stm_id_not_found`
- `stm_unresolved`

stm_r_stubs_name

Returns the list of stub names for an instance of a component.

Function Type

```
stm_list
```

For Elements

activity	ac

Syntax

```
stm_r_stubs_name (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_error_in_file
- stm_id_out_of_range
- stm_illegal_parameter
- stm_id_not_found
- stm_file_not_found
- stm_missing_name
- stm_missing_field

stm_r_tc_all

Returns the textual information associated with a specified element.

The information is retrieved into a structured data type (record) that varies according to the type of element referenced.

Function Type

```
stm_tc_all_ptr
```

For Elements

Timing Constraint

Syntax

```
stm_r_tc_all (tc_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`tc_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_unresolved`
- `stm_id_not_found`

stm_r_tc_graphic

Returns the graphical information associated with the specified element.

Function Type

```
stm_tc_graphic_ptr
```

For Elements

Timing Constraint

Syntax

```
stm_r_tc_graphic (tc_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`tc_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_unresolved
- stm_id_not_found

stm_r_tr_attr_enforced

Returns the enforced attributes specified by `attr_name`.

Function Type

`stm_boolean`

For Elements

Transition

Syntax

```
stm_r_tr_attr_enfoced(stm_id st_id, stm_attr_name st_attr_name, stm_name
st_attr_val int *status)
```

Arguments

Argument	Input/ Output	Type	Description
`status`	Out	`int`	The function status code.

stm_r_tr_attr_name

Returns the names of attributes associated with the specified element. Attributes are associated with elements through element forms.

Function Type

```
stm_list
```

For Elements

Transition

Syntax

```
stm_r_tr_attr_name(stm_id st_id, int *status)
```

Arguments

Argument	Input/ Output	Type	Description
`status`	Out	`int`	The function status code.

stm_r_tr_attr_val

Returns the values associated with a particular attribute name for the specified element.

Function Type

`stm_list`

For Elements

Transition

Syntax

```
stm_r_tr_attr_val(stm_id st_id, stm_attr_name st_attr_name, int *status)
```

Arguments

Argument	Input/ Output	Type	Description
`status`	Out	`int`	The function status code.

stm_r_tr_longdes

Returns the long description of the specified Transition.

Function Type

```
char *
```

For Elements

Transition

Syntax

```
stm_r_tr_longdes(stm_id st_id, stm_filename st_file, int *status)
```

Arguments

Argument	Input/ Output	Type	Description
`status`	Out	`int`	The function status code.

stm_r_tr_notes

Returns a list of strings each one is a line in the Transition Note related to the specified Transition.

Function Type

`stm_list`

For Elements

Transition

Syntax

```
stm_r_tr_notes (tr_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`tr_id`	In	`stm_id`	The Transition ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_missing_note`

stm_r_tt_cell

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

Function Type

```
char *
```

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	stm_id	The element ID
row_num	In	int	The row number of the cell
col_num	In	int	The column number of the cell
status	Out	int	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_hyper

Retrieves the contents of the specified cell in the given truth table, including hyperlinks to referenced elements.

Function Type

```
char *
```

For Elements

action	an
activity	ac
subroutine	sb

Syntax

```
stm_r_tt_cell_hyper (el, row_num, col_num, format, &status);
```

Arguments

Argument	Input/ Output	Type	Description
el	In	stm_id	The element ID
row_num	In	int	The row number of the cell
col_num	In	int	The column number of the cell
format			
status	Out	int	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

```
char
```

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	stm_id	The element ID
row_num	In	int	The row number of the cell
col_num	In	int	The column number of the cell
status	Out	int	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 int, Dataport allows you to reference this value by name. The name is internally defined as a predefined constant. The possible values are:

- 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_tt_num_of_col

Retrieves the number of columns (including blank ones) in the specified truth table, as viewed in the truth table editor.

Function Type

```
int
```

For Elements

truth table	tt

Syntax

```
function stm_r_tt_num_of_col (el, &status)
```

Arguments

Argument	Input/ Output	Type	Description
el	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_missing_truth_table
- stm_success

stm_r_tt_num_of_in

Retrieves the number of input columns in the specified truth table.

Function Type

```
int
```

For Elements

truth table	tt

Syntax

```
stm_r_tt_num_of_in (el_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
el_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_missing_truth_table
- stm_success

stm_r_tt_num_of_out

Retrieves the number of output columns in the specified truth table.

Function Type

```
int
```

For Elements

truth table	tt

Syntax

```
stm_r_tt_num_of_out (el_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
el_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_missing_truth_table
- stm_success

stm_r_tt_num_of_row

Retrieves the number of rows (including blank ones) in the specified truth table, as viewed in the truth table editor.

Function Type

```
int
```

For Elements

truth table	tt

Syntax

```
stm_r_tt_num_of_row (el_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
el_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_missing_truth_table
- stm_success

stm_r_tt_row

Returns a list of strings that represents a row in the truth table. Each string in the list includes the text in the truth table cell. The row's index range is `[0..num_of_rows-1]`. Row 0 returns the list of table header strings.

Function Type

`stm_list`

For Elements

`truth table`	`tt`

Syntax

```
stm_r_tt_row (el, row_num, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`el`	In	`stm_id`	The element ID.
`row_num`	In	`int`	The row number to retrieve.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_truth_table_invalid_row`
- `stm_missing_truth_table`
- `stm_success`

stm_r_tt_row_hyper

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.

Function Type

```
stm_list
```

For Elements

`truth table`	`tt`

Syntax

```
stm_r_tt_row (el, row_num, format &status)
```

Arguments

Argument	Input/ Output	Type	Description
`el`	In	stm_id	The element ID.
`row_num`	In	int	The row number to retrieve.
`format`			
`status`	Out	int	The function status code.

Status Codes

- `stm_truth_table_invalid_row`
- `stm_missing_truth_table`
- `stm_success`

stm_r_xx

Retrieves the element ID of the specified element. This ID is an internal representation that Rational Statemate uses to identify each element in the database. Because Rational Statemate requires the ID to locate elements, this function is very often the first one called when using dataport functions.

Function Type

```
stm_id
```

For Elements

action	an
activity	ac
actor	actor
boundary box	bb
chart	ch
condition	co
data-item	di
data-store	ds
enumerated value	en
event	ev
field	fd
function	fn
information-flow	if
lifeline	ll
local data	ld
module	md
off-page activity chart	oac
off-page module	omd
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	`stm_element_name or stm_pathname`	A Rational 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	`int`	The function status code.

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:

```
stm_id          ev_id;
int             status;
stm_short_name  synonym;
    .
    .
ev_id = stm_r_ev ("EV1", &status);
IF (status == stm_success)
    synonym = stm_r_ev_synonym (ev_id, &status);
    .
    .
    .
```

The ID for `EV1` is assigned to the variable `ev_id`.

Note: `ev_id` is declared to be of type `stm_id`.

stm_r_xx_all

Description

Returns both the textual and graphical information associated with a specified element.

Note

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

  ```
  stm_r_all (id, &status)
  ```

- The information is retrieved into a structured data type (record) that varies according to the type of element referenced.

Function Type

```
stm_xx_all_ptr
```

For Elements

activity	ac
a-flow-line (basic)	ba
combinational assignment	ca
connector	cn
data-store	ds
m-flow-line (basic)	bm
module	md
module-occurrence	om
note	nt
off-page activity	oa
state	st
transition (basic)	bt

Syntax

```
stm_r_xx_all (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_unresolved
- stm_id_not_found

Note

When stm_unresolved is returned, a record is received with the fields name, unique name, type, and chart. The remainder of the text fields are empty. The remainder of the graphical fields contain -1.

Example

To retrieve several fields (graphical and textual) attached to a specific state whose ID is st_id, use the first statement to retrieve all the information regarding the specific state (st_id), then extract the particular fields from the record.

```
   stm_id                st_id;
   int                   status;
   stm_st_all_ptr        st_record;
   stm_element_name      name;
   stm_short_name        synonym;
   stm_color             color;
      .
      .
st_record = stm_r_st_all (st_id, &status);
name = st_record->st_name;
synonym = st_record->st_synonym;
color = st_record->st_color;
```

stm_r_xx_array_lindex

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

`stm_const_expression`

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	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_not_found
- stm_id_out_of_range

stm_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

```
stm_const_expression
```

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	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_not_found
- stm_id_out_of_range

stm_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

`stm_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
transition	tr
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	`stm_id`	The element ID.
`attr_name`	In	`string`	The attribute name.
`attr_val`	In	`string`	The attribute value.
`status`	Out	`int`	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

`stm_list`

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
transition	tr
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	stm_id	The element ID.
status	Out	int	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 its attribute names using the following statements:

```
stm_id          st_id;
     stm_attr_name   attrib;
     stm_list        attr_list;
     int             status;
        .
        .
     st_id = stm_r_st ("WAIT", &status);
     attr_list = stm_r_st_attr_name (st_id, &status);
     for (attrib = (stm_attr_name)
          stm_list_first_element (attr_list, &status);
          status == stm_success;
       attrib = (stm_attr_name)
         stm_list_next_element (attr_list, &status))
        .
        .
        .
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

```
stm_list
```

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
transition	tr
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	stm_id	The element ID.
attr_name	In	stm_attr_name	The attribute name. The attribute name is not case-sensitive.
status	Out	int	The function status code. If attr_name does not exist for the specified element, status receives the value stm_attribute_name_not_found.

- Attribute values may exist for attributes with no name. 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, Rational 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

To extract the attribute values of the attribute refer for the state WAIT and perform several operations on each of these values, use the following statements:

```
stm_id              st_id;
stm_attr_val        attrib;
stm_list            attr_list;
int                 status;
    .
    .
st_id = stm_r_st("WAIT", &status);
attr_list = stm_r_st_attr_val (st_id, "refer", &status);
for (attrib = (stm_attr_val)
     stm_list_first_element (attr_list, &status);
     status == stm_success;
     attrib = (stm_attr_val)
     stm_list_next_element (attr_list, &status))
```

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

```
char *
```

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	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_not_found
- stm_id_out_of_range

stm_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

```
char *
```

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	stm_id	The element ID
status	Out	int	The function status code

Status Codes

- stm_success
- stm_id_not_found
- stm_id_out_of_range

stm_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

`stm_list`

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	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_not_found`
- `stm_id_out_of_range`
- `stm_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

```
char
```

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	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_not_found
- stm_id_out_of_range
- stm_missing_cbk_binding

Return Values

Although the return value of this function is of type `int`, dataport enables you to reference this value by name. The following table lists the possible values allowed by each Rational Statemate element subtype.

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_binding_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

```
stm_expression
```

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	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_not_found
- stm_id_out_of_range
- stm_missing_cbk_binding
- stm_unresolved

stm_r_xx_cbk_binding_expression_hyper

Retrieves the callback binding expressions, with hyperlinks to referenced elements.

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

```
stm_r_cbk_binding_expression_hyper (id, &status)
```

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_chart

Returns the chart ID for the specified element.

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

```
stm_r_chart (id, &status)
```

- For compound arrows, this function retrieves the chart only when all the arrow segments are in the same element. Otherwise, it returns the value 0.

Function Type

```
stm_id
```

Syntax

```
stm_r_xx_chart (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The element ID
status	Out	int	The function status code

Status Codes

- stm_success
- stm_id_out_of_range
- stm_id_not_found

Example

To return the name of the chart in which state S1 is found, use the following statements:

```
stm_id       state_id, chart_id;
    int          status;
                    .
                    .
                    .
state_id = stm_r_st ("S1", &status);
chart_id = stm_r_st_chart (state_id, &status);
printf ("chart name is: %s", stm_r_ch_name (chart_id,
   &status));
                    .
                    .
```

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 (basic)	bm
m-flow-line (compound)	mf
module	md
module-occurrence	om
note	nt
router	router
state	st
subroutine	sb
subroutine parameter	sp
transition (basic)	bt
transition (compound)	tr

stm_r_xx_combinationals

Returns a list of strings. Each element of the list holds one combinational assignment, which is connected to the specified element.

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

```
stm_r_combinationals (id, &status)
```

Function Type

`stm_list`

For Elements

`activity`	`ac`
`chart`	`ch`

Syntax

```
stm_r_xx_combinationals (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`xx_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	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_containing_fields

Returns the list of union or record elements that contain fields.

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

```
stm_r_containing_fields (id, &status)
```

Function Type

`stm_list`

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	`stm_id`	The element ID
`status`	Out	`int`	The function status code

Status Codes

- `stm_success`
- `stm_id_not_found`
- `stm_id_out_of_range`
- `stm_missing_field`
- `stm_unresolved`

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:

```
stm_r_data_type (id, &status)
```

Function Type

char

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	stm_id	The element ID
status	Out	int	The function status code

Status Codes

- stm_success
- stm_error_in_file
- stm_file_not_found
- stm_illegal_parameter
- stm_missing_field

stm_r_xx_default_val()

Returns the default value associated with the specified element. You can call this function without indicating the specific element type: `stm_r_default_val (id, &status).`

Function Type

`char*`

For Elements

`di`	`data item`
`dt`	`user-defined type`
`fd`	`field`
`co`	`condition`

Syntax

`stm_r_xx_default_val(xx_id, &status)`

Arguments

Argument	Input/ Output	Type	Description
`xx_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function code status.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_id_not_found`

stm_r_xx_definition_type

Returns the definition type of the specified textual element.

Note

- You can call this function without indicating the specific element type:

```
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:
  - `stm_reference`—The element has no form.
  - `stm_primitive`—The definition field is empty.
  - `stm_compound`—The definition field contains a compound expression.
  - `stm_constant`—The definition field contains a constant.
  - `stm_alias`—The definition field contains an identifier, a bit array, a component, or a slice (relevant for `di` only).
  - `stm_explicit`—The `info_flow` has a form.
  - `stm_predefined`—Predefined function.

Function Type

`stm_definition_type`

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

Note: These types are not explicitly specified, but derived from the specification.

Syntax

```
stm_r_xx_definition_type (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`xx_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_not_found`

Return Values

Although the return value of this function is of type `int`, Dataport allows you to reference this value by name. The name is internally defined as a predefined constant The following table lists the possible values allowed for each Rational Statemate element subtype.

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_des_attr_name

Returns the names of Design-Attributes associated with the specified element.

Function Type

stm_list

For Elements

activity	ac
chart	ch
condition	co
data-item	di
data-type	dt
field	fd
subroutine	sb
data-store	ds
block	bl
event	ev
information-flow	if
actions	an
module	md
state	st
transition	tr
subroutine parameter	sp
local data	ld

Syntax

```
stm_r_xx_des_attr_name (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The Element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_id_not_found
- stm_auto_defined
- stm_attribute_name_not_found

stm_r_xx_des_attr_val

Retrieves the values of a given Design-Attribute values associated with the specified element.

Function Type

stm_list

For Elements

activity	ac
chart	ch
condition	co
data-item	di
data-type	dt
field	fd
subroutine	sb
data-store	ds
block	bl
event	ev
information-flow	if
actions	an
module	md
state	st
transition	tr
subroutine parameter	sp
local data	ld

Syntax

```
stm_r_xx_des_attr_val (xx_id st_id, xx_attr_name &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The Element ID.
xx_attr_name	In	stm_attr_name	The Design Attribute Name.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_id_not_found
- stm_auto_defined
- stm_attribute_name_not_found

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:

```
stm_r_description (id, &status)
```

Function Type

`stm_description`

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	stm_id	The element ID.
status	Out	int	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.

```
stm_id              state_id;
stm_description     state_desc;
int                 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_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

`stm_id`

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	stm_id	The element ID
status	Out	int	The function status code

Status Codes

- stm_success
- stm_id_not_found
- stm_id_out_of_range

stm_r_xx_explicit_defined_xx

Provides a status list.

Function type:

For Elements

actor	ac
boundary box	bb
use case	uc

Syntax:

```
stm_list stm_r_xx_explicit_defined_xx(stm_list xx_list, int* status)
```

Arguments

Arguement	Input/ Ouput	Type	Description
xx_list	Input	stm_list	List of elements of type xx.
status	Output	int	Function of the status code.

Status Codes:

- stm_success
- stm_nil_list

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

```
stm_expression
```

For Elements

a-flow-lines (basic)	ba
action	an
condition	co
data-item	di
event	ev
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	st_id	The element ID.
format	In	char*	Either FrameMaker or Word.
status	Out	int	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).

- You can call this function without indicating the specific element type:

```
stm_r_expression (id, &status)
```

- This function is valid for compound textual elements, which are defined as an expression using the **Definition** field of its form.

Function Type

```
stm_expression
```

For Elements

a-flow-line (basic)	ba
action	an
condition	co
data-item	di
event	ev
field	fd
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	stm_id	The element ID.
status	Out	int	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:

```
stm_id           cond_id;
stm_expression   cond_def;
int              status;
   .
   .
cond_id = stm_r_co ("C1", &status);
cond_def = stm_r_co_expression (cond_id, &status);
   .
   .
   .
```

cond_def is assigned as the string value "C2 or C3".

stm_r_xx_ext_link

Returns the file name associated with the "Link to External File" entry in the element's properties of the specified element.

Note

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

```
stm_r_ext_link (id, &status)
```

Function Type

```
char*
```

For Elements

action	an
activity	ac
actor	actor
block	bl
boundary box	bb
chart	ch
condition	co
data-item	di
data-store	ds
event	ev
field	fd
information-flow	if
lifeline	ll
router	router
subroutine	sb
use case	uc
user-defined type	dt

Syntax

```
stm_r_sb_ext_link(xx_id,&status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_is	In	stm_id	The element ID.
Status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_name_not_found
- stm_id_not_found
- stm_auto_defined
- stm_missing_external_link

stm_r_xx_graphic

Returns the graphical information associated with the specified element.

Note

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

  ```
  stm_r_graphic(id,&status)
  ```

- The information is retrieved into a structured data type (record), which varies according to the type of element referenced.
- 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

```
stm_xx_graphic_ptr
```

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	stm_id	The element ID.
status	Out	int	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.

Example

To retrieve graphical information attached to a specific state whose ID is st_id, use the first statement regarding the specific state (st_id), then extract the particular fields from the record.

```
stm_id                st_id;
   int                   status;
   stm_st_graphic_ptr    st_record;
   stm_color             color;
      .
      .
st_record = stm_r_st_graphic (st_id, &status);
      .
color = st_record->st_color;
      .
      .
```

Refer to **Sample Program** for a more detailed example of how the fields of the graphical record are used.

stm_r_xx_instance_name

Returns the name of the instance as it appears in the chart for a specific hierarchical Rational Statemate element.

- You can call this function without indicating the specific element type:

```
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

```
stm_instance_name
```

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	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_id_not_found
- stm_unresolved
- stm_not_instance

Example

To return the name of an instance for state named `S1@S1_def`, use the following statements:

```
stm_id       state_id;
int          status;
             .
             .
             .
state_id = stm_r_st ("S1", &status);
printf ("\n Instance  Name: %s",
   stm_r_st_instance_name (state_id, &status));
             .
             .
             .
The name is written to the output is S1@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:

```
stm_r_keyword (id, begin_keyword, end_keyword, filename, &status)
```

Function Type

```
char *
```

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 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, Rational Statemate 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" using the following function call:

```
stm_id          state_id;
stm_filename    descr_file;
int             status;
     .
     .
     .
state_id = stm_r_st ("WAIT", &status);
descr_file = stm_r_st_keyword (state_id, "!BHV_DESCR",
   "!END_DESCR", "", &status);
     .
     .
     .
```

The portion of the long description is written to a file. The name of the file is returned in descr_file.

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

`stm_list`

For Elements

message	msg
transition	tr

Syntax

```
stm_r_xx_labels (tr_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`xx_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	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 labels of messages exiting from state `L1`, use the following statements:

```
stm_id           lifeline_id;
int              status;
stm_list         labels, ll_lst;
stm_list         messages;
stm_id           msg;
stm_expression   lab;
                 .
                 .
lifeline_id = stm_r_ll ("L1", &status);

ll_lst = stm_list_create (lifeline_id, end_of_list,
   &status);

messages = stm_r_msg_from_source_ll (ll_lst, &status);

for (msg = (stm_id) stm_list_first_element (messages,
     &status);
     status == stm_success;
     msg = (stm_id) stm_list_next_element (messages,
         &status))
     {
          labels = stm_r_msg_labels (msg, &status);
           if (status == stm_success)

          {
               for (lab = (char*) stm_list_first_element
                   (labels, &status);
               status == stm_success;
               lab = (char*) stm_list_next_element
                   (labels, &status))
               .
               .
          }
      }
```

stm_r_xx_labels_hyper

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

Function Type

`stm_list`

For Elements

`message`	`msg`
`transition`	`tr`

Syntax

```
stm_r_xx_labels_hyper (message, format, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`elem_id`	In	`stm_id`	The element ID.
`format`	In	`string`	Either FramrMaker or Word.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_id_not_found`
- `stm_unresolved`

stm_r_xx_longdes

Retrieves the long description attached to the specified element.

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

```
stm_r_longdes (id, filename, &status)
```

Function Type

```
char *
```

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
requirement	rt
router	router
state	st
subroutine	sb
transition	tr
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	stm_id	The element ID.
filename	In	stm_filename	The name of the file that contains the long description.
status	Out	int	The function status code.

- 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, Rational 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:

```
stm_id          act_id;
stm_filename    long_des_file;
int             status;
    .
    .
act_id = stm_r_ac ("A1", &status);
long_des_file =  stm_r_ac_longdes (act_id, "text.txt",
    &status);
     .
     .
     .
```

The long description for the activity A1 is written to the system file text.txt. This file resides in the directory that is the current workarea. The variable long_des_file contains the string 'text.txt' following statement execution.

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

```
char *
```

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	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_not_found
- stm_id_out_of_range

stm_r_xx_min_val

Returns the minimum value of the specified element.

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

```
stm_r_min_val (id, &status)
```

Function Type

```
char *
```

For Elements

user-defined type	dt

Syntax

```
stm_r_xx_min_val (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_not_found
- stm_id_out_of_range

stm_r_xx_mini_spec

Returns a string with mini-spec reactions or actions.

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

```
stm_r_mini_spec (id, &status)
```

Function Type

```
stm_expression
```

For Elements

activity	ac

Syntax

```
stm_r_xx_mini_spec (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_id_not_found
- stm_unresolved
- stm_missing_label

stm_r_xx_mode

Returns the parameter or router mode.

Function Type

`stm_xx_mode`

For Elements

`parameter`	`parameter`
`router`	`router`

Syntax

```
stm_r_xx_mode (elem_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`elem_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_id_not_found`
- `stm_unresolved`

stm_r_xx_name

Returns the element name. For hierarchical elements, the function returns the name associated with the box. Because hierarchical elements can share the same name, the return value does not necessarily uniquely identify an element. To return a unique name, use the function `stm_r_xx_uniquename`.

- This function returns a pointer to a static area of memory. Subsequent calls to this procedure overwrite the old string. If the name needs to be preserved, use the `strdup()` function from the string library.
- You can call this function without indicating the specific element type:

```
stm_r_name (id, &status)
```

- For boxes that have no names, this function returns the definition chart name. For example, for box `@ABC`, this function returns `ABC`.

Function Type

```
stm_element_name
```

Syntax

```
stm_r_xx_name (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`xx_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

For Elements

action	an
activity	ac
actor	actor
boundary box	bb
chart	ch
condition	co
data-item	di
data-store	ds
enumerated value	en
event	ev
field	fd
function	fn
information-flow	if
lifeline	ll
local data	ld
module	md
router	router
state	st
subroutine	sb
subroutine parameter	sp
use case	uc
user-defined type	dt

Status Codes

- stm_success
- stm_error_in_file
- stm_id_out_of_range
- stm_id_not_found
- stm_missing_name
- stm_missing_field
- stm_illegal_parameter
- stm_file_not_found

Example

To retrieve and print the name of a state in a statechart, use the following statements:

```
stm_id            state_id;
int               status;
   .
   .
state_id = stm_r_st ("S1.S3", &status);
printf ("%s", stm_r_st_name (state_id, &status));
   .
   .
```

In this example, the state name is provided and this value is used to retrieve the same state name from the database. The purpose of this example is to demonstrate the value returned by this function, in contrast to the value returned by the function `stm_r_xx_uniquename`.

stm_r_xx_note

Returns the notes from a requirement record or timing constraint.

Function Type

```
char *
```

For Elements

requirement	rt
timing constraint	tc

Syntax

```
stm_r_xx_note (rt_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
rt_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_missing_field
- stm_illegal_parameter
- stm_file_not_found
- stm_error_in_file

stm_r_xx_notes

Returns the note for the specified input transition.

Function Type

```
stm_list
```

For Elements

`chart`	`ch`

Syntax

```
stm_r_xx_notes (tr_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`tr_id`	In	`stm_id`	The transition ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_id_out_of_range`
- `stm_missing_note`
- `stm_success`

stm_r_xx_number_of_bits

Returns the number of bits in the element.

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

```
stm_r_number_of_bits (id, &status)
```

Function Type

```
char *
```

For Elements

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

Syntax

```
stm_r_xx_number_of_bits (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_not_found
- stm_id_out_of_range

stm_r_xx_of_enum_type

Retrieves the enumerated type ID (a user-defined type) for the specified element.

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

```
stm_r_of_enum_type (id, &status)
```

Function Type

`stm_id`

For Elements

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

Syntax

```
stm_r_xx_of_enum_type (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`xx_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_id_not_found`
- `stm_missing_of_enum_type`

stm_r_xx_of_enum_type_name_type

Retrieves the enumerated name type for the specified elements.

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

```
stm_r_of_enum_type_name_type (id, &status)
```

Function Type

`stm_name_type`

For Elements

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

Syntax

```
stm_r_xx_of_enum_type_name_type (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_id_not_found
- stm_missing_of_enum_type

stm_r_xx_parameter_mode

Retrieves the parameter mode, including subroutine parameters and the parameters of generic charts and components.

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

```
stm_r_parameter_mode (xx_id, &status)
```

Function Type

`stm_parameter_mode`

For Elements

`chart`	`ch`
`subroutine parameter`	`sp`

Syntax

```
stm_r_xx_parameter_mode (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`xx_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_not_a_parameter`

Return Values

Although the return value of this function is of type `int`, Dataport enables you to reference this value by name. The possible values are:

- `stm_in_parameter`
- `stm_out_parameter`
- `stm_inout_parameter`
- `stm_constant_parameter`

stm_r_xx_reactions

Returns the static reactions of the specified state. The syntax of these reactions is `trigger/action`.

- To divide the static reactions into their trigger and action parts, use the utility routines `stm_trigger_of_reaction` and `stm_action_of_reaction`.
- You can call this function without indicating the specific element type:

```
stm_r_reactions (st_id, &status)
```

Function Type

`stm_list`

For Elements

activity	ac
state	st

Syntax

```
stm_r_xx_reactions (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_st_id	In	stm_id	The state ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_unresolved
- stm_id_out_of_range
- stm_id_not_found
- stm_missing_label

Example

To extract all static reactions of state `S1`, use the following statements:

```
  stm_id           state_id;
  int              status;
  stm_list         reactions;
  stm_expression   react;
                     .
                     .
state_id = stm_r_st ("S1", &status);
reactions = stm_r_st_reactions (state_id, &status);
if(status == stm_success)
   for(react = (string)
      stm_list_first_element (reactions, &status);
      status == stm_success;
      react = (string)
      stm_list_next_element (reactions, &status)){
                  .
                  .
       }
```

stm_r_xx_select_implementation

Retrieves the implementation type of the specified element.

Function Type

```
stm_sb_select_implementation
```

For Elements

action	an
activity	ac
subroutine	sb

Syntax

```
stm_r_xx_select_implementation (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_out_of_range
- stm_id_not_found

Return Values

Although the return value of this function is of type `int`, Dataport enables you to reference this value by name. The possible values are as follows:

- stm_sb_action_lang
- stm_sb_procedural_sch
- stm_sb_kr_c_code
- stm_sb_ansi_c_code
- stm_sb_ada_code
- stm_sb_vhdl_code
- stm_sb_verilog_code
- stm_sb_truth_table_code

- stm_sb_best_match
- stm_sb_none

stm_r_xx_string_length

Retrieves the string length of the specified element.

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

```
stm_r_string_length (id, &status)
```

Function Type

stm_const_exp

For Elements

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

Syntax

```
stm_r_xx_string_length (xx_id, &status)
```

Arguments

Argument	Input/Output	Type	Description
xx_id	In	stm_id	The element ID
status	Out	int	The function status code

Status Codes

- stm_success
- stm_missing_field
- stm_illegal_parameter
- stm_file_not_found
- stm_error_in_file

stm_r_xx_structure_type

Returns the structure or type of the specified textual element. The structure or type can be single, array, or queue.

You can call this function without specifying an element type:

```
stm_r_structure_type (id, &status)
```

Function Type

```
stm_list
```

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_structure_type (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The element ID.
status	Out	int	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 `int`, Dataport enables you to reference this value by name. The following are all possible values allowed for each Rational Statemate element subtype.

Element Type	Element Subtype
condition	`stm_co_array`
	`stm_co_missing`
	`stm_co_single`
data-item	`stm_di_array`
	`stm_di_queue`
	`stm_di_single`
event	`stm_ev_array`
	`stm_ev_missing`
	`stm_ev_single`
field	`stm_fd_array`
	`stm_fd_queue`
	`stm_fd_single`
local data	`stm_ld_array`
	`stm_ld_queue`
	`stm_ld_single`
subroutine parameter	`stm_sp_array`
	`stm_sp_queue`
	`stm_sp_single`
user-defined type	`stm_dt_array`
	`stm_dt_queue`
	`stm_dt_single`

stm_r_xx_synonym

Retrieves the synonym of the specified element. The synonym is defined in the element's form.

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

```
stm_r_synonym (id, &status)
```

Function Type

```
stm_short_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_synonym (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The element ID.
status	Out	int	The function status code. If no synonym is defined in the element's form, status receives the value stm_missing_synonym.

Status Codes

- stm_success
- stm_unresolved
- stm_missing_subroutine_params
- stm_id_out_of_range
- stm_id_not_found
- stm_missing_synonym

Example

To write out the synonym of activity A1, use the following statements:

```
stm_id    act_id;
int       status;
     .
     .
act_id = stm_r_ac ("A1", &status);
printf ("Synonym:%s", stm_r_ac_synonym (act_id,
     &status));
     .
     .
     .
```

stm_r_xx_text

Returns the textual information associated with a specified element.

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

```
stm_r_text(id,&status)
```

The information is retrieved into a structured data type (record), which varies according to the type of element referenced.

Function Type

```
stm_xx_text_ptr
```

For Elements

a-flow-line (basic)	ba
action	an
activity	ac
block	bl
chart	ch
connector	cn
combinational assignment	ca
condition	co
data-item	di
data-store	ds
event	ev
field	fd
function	fn
information-flow	if
local data	ld
module	md
note	nt
state	st
subroutine	sb
subroutine parameter	sp
user-defined type	dt

Syntax

```
stm_r_xx_text (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_file_not_found
- stm_id_out_of_range
- stm_id_not_found
- stm_missing_synonym
- stm_error_in_file
- stm_unresolved
- stm_illegal_parameter —When this status code is returned, a record is received with the fields name, unique name, type, and chart. The rest of the text fields are empty.

Example

To retrieve several fields attached to a specific state whose ID is st_id, use the first statement. Thereafter, extract from this record the particular fields.

```
    stm_id                st_id;
    int                   status;
    stm_st_text_ptr       st_record;
    stm_element_name      name;
    stm_short_name        synonym;
        .
        .
st_record = stm_r_st_text (st_id, &status);
        .
name = st_record->st_name;
synonym = st_record->st_synonym;
        .
        .
```

When retrieved, the information is assigned to a specific record that can be examined thereafter for the desired information.

stm_r_xx_truth_table

Returns the elements that are implemented as truth tables.

Function Type

```
stm_list
```

For Elements

action	an
activity	ac
subroutine	sb

Syntax

```
stm_r_xx_truth_table (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_error_in_file
- stm_id_out_of_range
- stm_illegal_parameter
- stm_id_not_found
- stm_file_not_found
- stm_missing_name
- stm_missing_field

stm_r_xx_truth_table_expression

Returns the truth table expression for all named elements.

Function Type

```
stm_expression
```

For Elements

action	an
activity	ac
subroutine	sb

Syntax

```
stm_r_xx_truth_table_expression (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_error_in_file
- stm_id_out_of_range
- stm_illegal_parameter
- stm_id_not_found
- stm_file_not_found
- stm_missing_name
- stm_missing_field

stm_r_xx_truth_table_local_data

Returns the list of local data elements defined in the truth table related to the input subroutine.

Function Type

`stm_list`

For Elements

`subroutine`	`sb`

Syntax

```
stm_r_xx_truth_table_local_data (sb_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`sb_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_error_in_file`
- `stm_id_out_of_range`
- `stm_illegal_parameter`
- `stm_id_not_found`
- `stm_file_not_found`
- `stm_missing_name`
- `stm_missing_field`

stm_r_xx_type

Function Type

```
stm_element_type
```

For Elements

a-flow-line (basic)	ba
a-flow-line (compound)	af
action	an
activity	ac
actor	actor
boundary box	bb
chart	ch
condition	co
connector	cn
data-item	di
data-store	ds
event	ev
field	fd
function	fn
information-flow	if
lifeline	ll
module	md
module-occurrence	om
note	nt
off-page activity	oa
router	router
state	st
subroutine	sb
use case	uc
user-defined type	dt

Description

Retrieves element subtypes for the specified element. Most Rational Statemate elements are divided into classes, referred to as *subtypes*. For example, a state might belong to one of a number of subtypes, such as *and*, *or*, *basic*, *diagram*, *instance*, or *reference*.

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

```
stm_r_type (id, &status)
```

Syntax

```
stm_r_xx_type (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`xx_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_id_not_found`

Return Values

The return value of the function belongs to an enumerated type. The enumerated type depends on the particular element type for which the function is performed. The enumerated type is named stm_element_type, where element varies as shown in the following table.

Element Type	Function Type	Element Subtype
a-flow-line	stm_a_flow_line_type	stm_af_control
		stm_af_data
action	stm_action_type	stm_an_compound
		stm_an_reference
activity	stm_activity_type	stm_ac_control
		stm_ac_control_instance
		stm_ac_diagram
		stm_ac_external
		stm_ac_instance
		stm_ac_internal
		stm_ac_reference
ba-flow-line	stm_ba_flow_line_type	stm_ba_control
		stm_ba_data
chart	stm_chart_type	stm_ch_activity
		stm_ch_module
		stm_ch_reference _activity
		stm_ch_reference_module
		stm_ch_reference_state
		stm_ch_state
condition	stm_condition_type	stm_co_compound
		stm_co_primitive
		stm_co_reference

Element Type	Function Type	Element Subtype
connector	stm_connector_type	stm_cn_composition
		stm_cn_condition
		stm_cn_control
		stm_cn_deep_history
		stm_cn_default
		stm_cn_diagram
		stm_cn_history
		stm_cn_joint
		stm_cn_junction
		stm_cn_selection
		stm_cn_termination
data-item	stm_data_item_type	stm_di_compound
		stm_di_alias
		stm_di_constant
		stm_di_primitive
		stm_di_reference
data-store	stm_data_store_type	stm_ds_internal
		stm_ds_reference
event	stm_event_type	stm_ev_compound
		stm_ev_primitive
		stm_ev_reference
field	stm_field_type	stm_fd_primitive
information-flow	stm_information_flow_type	stm_if_explicit
		stm_if_reference
module	stm_module_type	stm_md_diagram
		stm_md_subsystem
		stm_md_environment
		stm_md_reference
		stm_md_instance
		stm_md_storage_module
router	stm_router_type	stm_router_external
		stm_router_internal

Element Type	Function Type	Element Subtype
state	stm_state_type	stm_st_diagram
		stm_st_and
		stm_st_or
		stm_st_instance
		stm_st_reference
		stm_st_basic
subroutine	stm_subroutine_type	stm_sb_reference
user-defined type		stm_dt_primitive
		stm_dt_reference

Note

The value `stm_st_component` is not used.

Example

To retrieve the type of state `Ready` and execute some statements if the state is an `or` state, use the following statements:

```
stm_id            st_id;
stm_state_type    st_type;
int               status;
   .
   .
st_id = stm_r_st ("READY", &status);
st_type = stm_r_st_type (st_id, &status);
if(st_type == stm_st_or)
   .
   .
   .
```

stm_r_xx_type_expression

Returns the type expression for the specified element. The expression is the same as used in the properties, reports, and Info.

You can call this function without indicating the specific type:

```
stm_r_type_expression (id, &status)
```

Function Type

`stm_expression`

For Elements

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

Syntax

```
stm_r_xx_type_expression (xx_id, status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_id_not_found
- stm_unresolved

stm_r_xx_uniquename

Returns the unique path name for the specified element. The name returned by the function contains the minimum number of levels necessary to uniquely identify an element in its chart. It is especially relevant to boxes.

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

```
stm_r_uniquename (id, &status)
```

Function Type

```
har *
```

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
function	fn
information-flow	if
lifeline	ll
local data	ld
module	md
router	router
state	st
subroutine	sb
subroutine parameter	sp
use case	uc
user-defined type	dt

Syntax

```
stm_r_xx_uniquename (xx_id, &status)
```

Arguments

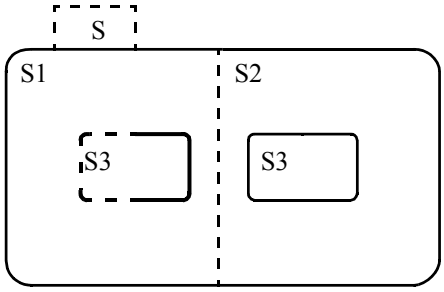
Argument	Input/ Output	Type	Description
`xx_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_id_not_found`
- `stm_missing_synonym`

Example

Consider the following statechart:



To retrieve the unique name of the highlighted state, use the following statements:

```
stm_id          state_id;
int             status;
    .
    .
state_id = stm_r_st ("S1.S3", &status);
printf ("Unique Name:%s", stm_r_st_uniquename (
    state_id, &status));
    .
    .
    .
```

The state name printed is `S1.S3` (not `S.S1.S3` or `S3`). In this example, a unique state name is provided, and this value is used to retrieve the same unique state name from the database. This example demonstrates the value returned by this function, in contrast to the value returned by the function `stm_r_xx_name`.

stm_r_xx_user_type

Function Type

```
stm_id
```

For Elements

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

Description

Returns the user-defined type ID referenced by the element.

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

```
stm_r_user_type (id, &status)
```

Syntax

```
stm_r_xx_user_type (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_id_not_found
- stm_missing_user_type
- stm_success
- stm_ntc_name
- stm_ntc_synonym
- stm_ntc_unknown

stm_r_xx_user_type_name_type

Returns the name type of the user-defined type referenced by the element.

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

```
stm_r_user_type_name_type (id, &status)
```

Function Type

`stm_name_type`

For Elements

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

Syntax

```
stm_r_xx_user_type_name_type (xx_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
xx_id	In	stm_id	The element ID.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range
- stm_id_not_found
- stm_missing_user_type

stm_open_truth_table

Opens a Truth-Table which is connected to the specified element and highlights the specifies line in it. Returns `stm_success` if request was successfully sent, and `stm_id_out_of_range` otherwise.

Function Type

```
stm_boolean
```

For Elements

action	an
activity	ac
Subroutine	sb

Syntax

```
stm_open_truth_table(stm_id id, int line, int *status)
```

Arguments

Argument	Input/ Output	Type	Description
id	In	stm_id	The element ID.
line	In	int	The line in the Truth-Table to be Highlighted.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_id_out_of_range

stm_calculate_element_magic_number

Returns a number that reflects a status of a specified element. A change in the element's definition is reflected by a change in the returned number.

Function Type

```
long
```

For Elements

All types

Syntax

```
stm_calculate_element_magic_number(el_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`el_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_missing_local_data`
- `stm_no_connected_chart`

stm_get_element_create_stamp

Returns a number that reflects a creation time of a specified element.

Function Type

```
long
```

For Elements

All types

Syntax

```
stm_get_element_create_stamp(el_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`el_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_not_found`
- `stm_id_out_of_range`
- `stm_no_connected_chart`

stm_r_line_width

Returns the line width of a specified element.

Function Type

```
int
```

For Elements

arrows and boxes

Syntax

```
stm_r_line_width(el_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`el_id`	In	`stm_id`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_not_found`
- `stm_id_out_of_range`

Query Functions

This section describes the query functions. The functions are organized into sections by element types returned by functions. Within each section, functions are organized by type of element in the input list.

For each query function, the following information is provided:

- Description
- Query (if it exists)

Each function returns an output list. The output argument `status` is the function status code.

Overview

Query functions extract lists of elements from the database that conform to a specific criterion.

The property sheet enables you to query the Rational Statemate database. This tool uses a comprehensive set of predefined queries to obtain information. All these queries operate on a list of Rational Statemate elements, called the *input list*. Each query generates an output list of elements that meet a criterion designated by the specific query. Generally, elements in the output list are related to elements in the input list in one of two ways:

- The output list is a subset of input list elements that have a specific characteristic. For example, the output list consists of all And-states in the input list.
- Elements in the output list fulfill a specific relationship to elements in the input list. For example, the output list consists of all states that are descendants of states in the input list.

Most query functions correspond to queries from the property sheet. These functions give you the same information that the corresponding queries do. Most functions require you to provide an input list as an input argument. This input list generally consists of elements of a particular type. The function returns a list of elements of the same or different type (as the input list).

The retrieval process is as follows:

1. Generate the input list.
2. Specify the query and input list. Receive the input list. Note that other procedures may be performed before you use the retrieved information.
3. Use the output list.

Most functions require you to provide an input list as an input argument. This input list generally consists of elements of a particular type. The function returns a list of elements of the same type as that of the input list, or of a different type.

Calling Query Functions

Most of the query functions use the following calling sequence:

```
stm_r_yy_relation_xx (xx_list, &status)
```

In this syntax:

- **stm_r_**—Designates the function as a Rational Statemate database retrieval function.
- **yy**—The two-character type abbreviation for elements in the output list.
- **relation**—The relationship between the input and output lists (describes the query to be applied to the input list).
- **xx**—The two-character type abbreviation for elements in the input list.
- **xx_list**—The input list to the function.
- **status**—The return function status code. There are three possible status codes: `stm_success`, `stm_nil_list`, and stm_missing_element_in_list.

For example:

```
stm_r_st_and_st (state_list, &status)
```

This function returns the states from the input list `state_list` that are and-states.

The following function returns the activities performed throughout the states in `state_list`:

```
stm_r_st_ac_throughout_st (state_list, &status)
```

The following sections document the query functions that use a different calling sequence.

By Attributes

The `by_attributes` function returns all elements in the input list that have an attribute `attr_name`, whose value is `attr_val`.

The syntax is as follows:

```
stm_r_xx_by_attributes_xx (xx_list, attr_name, attr_val, &status)
```

In this syntax:

- **stm_r_**—Designates the function as a Rational Statemate database retrieval function.
- **xx**—The two-character type abbreviation for elements in the input and output lists.
- **by_attributes**—The criterion to be met by elements in the input list.
- **xx_list**—The input list to the function.
- **attr_name**—A pattern for the attribute name.
- **attr_val**—A pattern for the attribute value to be matched.
- **status**—The return function status code. There are three possible status codes: `stm_success`, `stm_nil_list`, and `stm_missing_element_in_list`.

For example:

```
stm_r_md_by_attributes_md (module_list, "LANGUAGE", "PASCAL", &status)
```

This function returns all modules in `module_list` that have an attribute `LANGUAGE`, whose value is `PASCAL`.

Note

If you use `stm_r_xx_by_attributes_xx` to search for a specific attribute without regard to the attribute's value, enter the attribute's name with an "*" (asterisk). The search returns all elements with the attribute name and their assigned values. If you need to find an attribute that has no attribute value (empty), enter the attribute name with empty quotation marks ("").

By Structure Type

The `by_structure_type` function returns all elements in the input list that have a structure type `xx_structure_type`.

The syntax of the `by_structure_type` function is as follows:

```
stm_r_xx_by_structure_type_xx (xx_list, xx_structure_type, &status)
```

In this syntax:

- **stm_r_**—Designates the function as a Rational Statemate database retrieval function
- **xx**—The two-character type abbreviation for elements in the input and output list
- **by_structure_type**—The structure type referenced
- **xx_list**—The input list to the function
- **xx_structure_type**—The structure type referenced (array, single, or queue in the element's form)
- **status**—The return function status code

For example:

```
stm_r_di_by_structure_type_di (di_list, stm_di_array, &status)
```

This function returns all data items in `di_list` that have an array structure type.

Name and Synonym Patterns

The `name_of` and `synonym_of` functions search the entire database for elements whose name (or synonym) matches the pattern specified in the argument pattern.

The syntax is as follows:

```
stm_r_xx_name_of_xx (pattern, &status)
```

```
stm_r_xx_synonym_of_xx (pattern, &status)
```

In this syntax:

- **stm_r_**—Designates the function as a Rational Statemate database retrieval function
- **xx**—The two-character type abbreviation of elements in the output list
- **name_of** or **synonym_of**—The criterion to be met (specifies that the query "Element whose name matches a pattern" or "Element whose synonym matches a pattern" is to be applied)
- **pattern**—A character string you supply as an input argument
- **status**—The return function status code

These functions search the entire database for elements whose names (or synonyms) match the pattern specified in the argument `pattern`.

For example:

```
stm_r_ev_name_of_ev ("EV*", &status)
```

This function returns all events from the database whose names begin with the string `EV`.

In another example, to retrieve all the charts in the database use the following:

```
stm_r_ch_name_of_ch ("*", &status)
```

The output list contains all the charts in the database, including reference charts.

Query Function Input Arguments

The following table lists the input arguments for query functions.

Argument	Description	Data Type
`xx_list`	An input list of elements upon which you perform a query.	`stm_list`
`attribute name`	The name of an attribute defined in the `Attribute` field of a Rational Statemate element form. This pattern may include wildcards (`?` and `*`).	`stm_attr_name`
`attribute value`	The value of an attribute defined in the `Attribute` field of a Rational Statemate element form.	`stm_attr_val`
`pattern`	An alphanumeric string to match a Rational Statemate element name (or synonym). Two special characters can be used as wildcards: • A question mark (`?`) indicates that any character can occupy this position • An asterisk (`*`) indicates that any number of characters (including 0) can occupy this position.	`char * (string)`

Examples of Query Functions

This section provides several examples of query functions used to extract database information.

Example 1

To build an input list for query functions, use the following statements:

```
 stm_id       act_id;
 stm_list     list, act_list;
 int          status;
              ...

act_id = stm_r_ac("A1", &status);
list = stm_list_create (act_id, end_of_list, &status);
act_list = stm_r_ac_physical_sub_of_ac (list, &status);
                    .
                    .
```

The variable `act_id` contains the ID of the activity `A1`. The input list is built by calling the `stm_list_create` function. In this case, the input list consists of only one element. You would use the same function to build a list of multiple elements.

Note that the input list is built from element IDs, *not* from element names.

Example 2

To return all the basic states that are descendants of states `S1`, use the following statements:

```
  stm_id           st_id;
  int              status;
  stm_list         list;
  stm_list         descen_states, basic_states;
                    .
                    .

st_id = stm_r_st ("S1", &status);
list =
  stm_list_create (st_id, end_of_list, &status);
descen_states = stm_r_st_physical_desc_of_st (list,
   &status);
basic_states = stm_r_st_basic_st (descen_states,
   &status);
                    .
                    .
```

Example 3

To return the name of all events in the database whose names begin with the string EV, use the following statements:

```
  stm_id      eve;
  stm_list    ev_list;
  int         status;
          .
          .
          .
ev_list = stm_r_ev_name_of_ev ("EV*", &status);
for (eve = (stm_id)
       stm_list_first_element (ev_list, &status);
   status == stm_success;
   eve = (stm_id) stm_list_next_element (ev_list,
       &status))
   printf ("\n %s", stm_r_ev_name (eve, &status));
```

List of Query Functions

The query functions are grouped alphabetically first by output list type, then by input list type. The output types are as follows:

- Activities (ac)
- A-Flow-Lines (af, ba, laf)
- Actions (an)
- Actors (actor)
- Boundary Boxes (bb)
- Callbacks
- Combinational Assignments (ca)
- Charts (ch)
- Connectors (cn)
- Conditions (co)
- Data-Items (di)
- Data-Stores (ds)
- Events (ev)
- Fields (fd)
- Functions (fn)
- Information-Flows (if)
- M-Flow-Lines (bf, lmf, mf)
- Modules (md)
- Mixed (mx)
- Module-Occurrences (om)
- Routers (router)
- Subroutines (sb)
- States (st)
- Timing Constraint (tc)
- Transitions (tr)

Activities (ac)

This section documents the query functions that return a list of activities.

Input List Type: ac

stm_r_ac_actor_ac	**Query:** **Purpose:** Returns a list of activities from the input list that are of the requested type **Syntax:** `stm_r_ac_actor_ac (stm_list activities_list, int* status);`
stm_r_ac_basic_ac	**Query:** Basic activities **Purpose:** Returns the activities in the input list that have no descendants **Syntax:** `stm_r_ac_basic_ac (stm_list in_list, int *status);`
stm_r_ac_boundary_box_ac	**Query:** **Purpose:** Returns a list of activities from the input list that are of the requested type **Syntax:** `stm_r_ac_boundary_box_ac (stm_list activities_list, int* status);`
stm_r_ac_by_attributes_ac	**Query:** Activities by attributes **Purpose:** Returns the activities in the input list that match the specified attribute name and value **Syntax:** `stm_r_ac_by_attributes_ac (stm_list in_list, char* attr_name, char* attr_value, int *status);`
stm_r_ac_callback_binding_ac	**Query:** Activities with callback bindings **Purpose:** Returns the activities in the input list that have callback bindings **Syntax:** `stm_r_ac_callback_binding_ac (stm_list in_list, int *status);`

stm_r_ac_component_instance_ac	**Query:** Activities that are instances of components **Purpose:** Returns the activities in the input list that have instances of components **Syntax:** `stm_r_ac_component_ac (stm_list in_list, int *status);`
stm_r_ac_continuous_instance_ac	**Query:** Activities with continuous instances **Purpose:** Returns the activities in the input list that have continuous instances **Syntax:** `stm_r_ac_continuous_instance_ac (stm_list in_list, int *status);`
stm_r_ac_control_ac	**Query:** Control activities **Purpose:** Returns the activities in the input list that are control activities **Syntax:** `stm_r_ac_control_ac (stm_list in_list, int *status);`
stm_r_ac_control_terminated_ac	**Query:** Controlled-terminated activities **Purpose:** Returns the activities in the input list that are control-terminated activities **Syntax:** `stm_r_ac_control_terminated_ac (stm_list in_list, int *status);`
stm_r_ac_data_store_ac	**Query:** Data-stores **Purpose:** Returns the activities in the input list that are data-stores **Syntax:** `stm_r_ac_data_store_ac (stm_list in_list, int *status);`
stm_r_ac_def_of_instance_ac	**Query:** Definition activities of a given activity **Purpose:** Returns the definition activities (top-level in the definition chart) for instances in the input list **Syntax:** `stm_r_ac_def_of_instance_ac (stm_list in_list, int *status);`
stm_r_ac_defined_environment_ac	**Query:** Environment activities **Purpose:** Returns the activities in the input list that were defined as environment activities **Syntax:** `stm_r_ac_defined_environment_ac (stm_list in_list, int *status);`

stm_r_ac_explicit_defined_ac	**Query:** Activities explicitly defined **Purpose:** Returns from the input list those activities that were explicitly defined **Syntax:** `stm_r_ac_explicit_defined_ac (stm_list in_list, int *status);`
stm_r_ac_ext_11_ac	**Query:** **Purpose:** Returns a list of activities from the input list that are of the requested type **Syntax:** `stm_r_ac_ext_11_ac (stm_list activities_list, int* status);`
stm_r_ac_external_ac	**Query:** External activities **Purpose:** Returns the activities in the input list that are external **Syntax:** `stm_r_ac_external_ac (stm_list in_list, int *status);`
stm_r_ac_external_router_ac	**Query:** **Purpose:** Returns a list of activities from the input list that are of the requested type **Syntax:** `stm_r_ac_external_router_ac (stm_list activities_list, int* status);`
stm_r_ac_generic_instance_ac	**Query:** Generic instance activities **Purpose:** Returns the activities in the input list that are instances of generic charts **Syntax:** `stm_r_ac_generic_instance_ac (stm_list in_list, int *status);`
stm_r_ac_imp_best_match_ac	**Query:** Activities whose selected implementation is **Best Match** **Purpose:** Returns the activities in the input list that are implemented as the **Best Match** using **Select Implementation** in the properties **Syntax:** `stm_r_ac_imp_best_match_ac (stm_list in_list, int *status);`

stm_r_ac_imp_mini_spec_ac	**Query:** Activities implemented in a mini-spec **Purpose:** Returns the activities in the input list that are implemented in a mini-spec **Syntax:** `stm_r_ac_imp_mini_spec_ac (stm_list in_list, int *status);`
stm_r_ac_imp_none_ac	**Query:** Activities whose selected implementation is **None** **Purpose:** Returns the activities in the input list that are not implemented **Syntax:** `stm_r_ac_imp_none_ac (stm_list in_list, int *status);`
stm_r_ac_imp_sb_bind_ac	**Query:** Activities implemented with subroutine bindings **Purpose:** Returns the activities in the input list that are implemented as **Subroutine Binding** using **Select Implementation** in the properties **Syntax:** `stm_r_ac_imp_sb_bind_ac (stm_list in_list, int *status);`
stm_r_ac_imp_truth_table_ac	**Query:** Activities implemented in a truth table **Purpose:** Returns the activities in the input list that were implemented in a truth table **Syntax:** `stm_r_ac_imp_truth_table_ac (stm_list in_list, int *status);`
stm_r_ac_instance_ac	**Query:** Instance activities **Purpose:** Returns those activities in the input list that are instances **Syntax:** `stm_r_ac_instance_ac (stm_list in_list, int *status);`
stm_r_ac_instance_of_def_ac	**Query:** Instance activities of a given definition activity **Purpose:** Returns the instance activities for definition activities (top-level activities in a definition chart) in the input list **Syntax:** `stm_r_ac_instance_of_def_ac (stm_list in_list, int *status);`

stm_r_ac_internal_ac	**Query:** Internal activities **Purpose:** Returns the activities in the input list that are internal activities (not external or control) **Syntax:** `stm_r_ac_internal_ac (stm_list in_list, int *status);`
stm_r_ac_is_occurrence_of_ac	**Query:** Activity occurrences of a given activity **Purpose:** Returns the activities for which the activities in the input list appear in the **Is activity** field of their form **Syntax:** `stm_r_ac_is_occurrence_of_ac (stm_list in_list, int *status);`
stm_r_ac_is_principal_of_ac	**Query:** Principal activities of a given activity **Purpose:** Returns the activities appearing in the **Is activity** field of the activities in the input list **Syntax:** `stm_r_ac_is_principal_of_ac (stm_list in_list, int *status);`
stm_r_ac_lifeline_ac	**Query:** **Purpose:** Returns a list of activities from the input list that are of the requested type **Syntax:** `stm_r_ac_lifeline_ac (stm_list activities_list, int* status);`
stm_r_ac_logical_desc_of_ac	**Query:** Logical descendants of a given activity **Purpose:** Returns the logical descendants of the activities in the input list, taking into account the translation of instances to their definition charts **Syntax:** `stm_r_ac_logical_desc_of_ac (stm_list in_list, int *status);`
stm_r_ac_logical_parent_of_ac	**Query:** Logical parent activities of a given activity **Purpose:** Returns the logical parent activities of the activities in the input list, taking into account the translation of instances to their definition charts **Syntax:** `stm_r_ac_logical_parent_of_ac (stm_list in_list, int *status);`

stm_r_ac_logical_sub_of_ac	**Query:** Logical subactivities of a given activity **Purpose:** Returns the logical subactivities of the activities in the input list, taking into account the translation of instances to their definition charts **Syntax:** `stm_r_ac_logical_sub_of_ac (stm_list in_list, int *status);`
stm_r_ac_mini_spec_ac	**Query:** Activities having mini-specs **Purpose:** Returns the activities in the input list that have a mini-spec **Syntax:** `stm_r_ac_mini_spec_ac (stm_list in_list, int *status);`
stm_r_ac_name_of_ac	**Query:** Activities whose names match a given pattern **Purpose:** Returns all the activities whose names match a given pattern **Syntax:** `stm_r_ac_name_of_ac (char* pattern, int *status);`
stm_r_ac_offpage_instance_ac	**Query:** Offpage instance activities **Purpose:** Returns the activities in the input list that are instances of offpage charts **Syntax:** `stm_r_ac_offpage_instance_ac (stm_list in_list, int *status);`
stm_r_ac_physical_desc_of_ac	**Query:** Physical descendants of a given activity **Purpose:** Returns the physical descendants (those within the same chart) for the activities in the input list **Syntax:** `stm_r_ac_physical_desc_of_ac (stm_list in_list, int *status);`
stm_r_ac_physical_parent_of_ac	**Query:** Physical parent activities of a given activity **Purpose:** Returns the physical parent activities (those within the same chart) for the activities in the input list **Syntax:** `stm_r_ac_physical_parent_of_ac (stm_list in_list, int *status);`
stm_r_ac_physical_sub_of_ac	**Query:** Physical subactivities of a given activity **Purpose:** Returns the physical subactivities (those within the same chart) for the activities in the input list **Syntax:** `stm_r_ac_physical_sub_of_ac (stm_list in_list, int *status);`

stm_r_ac_procedure_like_ac	**Query:** Procedure-like activities **Purpose:** Returns the activities in the input list that are procedure-like activities **Syntax:** `stm_r_ac_procedure_like_ac (stm_list in_list, int *status);`
stm_r_ac_resolved_to_ext_ac	**Query:** Activities resolved to a given external activity **Purpose:** Returns the activities (internal, external, or environment) to which the external activities in the input list are resolved **Syntax:** `stm_r_ac_resolved_to_ext_ac (stm_list in_list, int *status);`
stm_r_ac_router_ac	**Query:** **Purpose:** Returns a list of activities from the input list that are of the requested type **Syntax:** `stm_r_ac_router_ac (stm_list activities_list, int* status);`
stm_r_ac_self_terminated_ac	**Query:** Self-terminated activities **Purpose:** Returns the activities in the input list that are self-terminated **Syntax:** `stm_r_ac_self_terminated_ac (stm_list in_list, int *status);`
stm_r_ac_subroutine_binding_ac	**Query:** Activities with subroutine bindings **Purpose:** Returns the activities in the input list that have subroutine bindings (regardless of the implementation setting in the properties) **Syntax:** `stm_r_ac_subroutine_binding_ac (stm_list in_list, int *status);`
stm_r_ac_synonym_of_ac	**Query:** Activities whose synonyms match a given pattern **Purpose:** Returns all the activities whose synonyms match the specified pattern **Syntax:** `stm_r_ac_synonym_of_ac (char* pattern, int *status);`

stm_r_ac_unresolved_ac	**Query:** Unresolved activities **Purpose:** Returns the unresolved activities in the input list **Syntax:** `stm_r_ac_unresolved_ac (stm_list in_list, int *status);`
stm_r_ac_use_case_ac	**Query:** **Purpose:** Returns a list of activities from the input list that are of the requested type **Syntax:** `stm_r_ac_use_case_ac (stm_list activities_list, int* status);`

Input List Type: af

stm_r_ac_source_of_af	**Query:** Activities that are sources for a given a-flow-line **Purpose:** Returns the activities that are sources of a-flow-lines in the input list **Syntax:** `stm_r_ac_source_of_af (stm_list in_list, int *status);`
stm_r_ac_target_of_af	**Query:** Activities that are targets of a given a-flow-line **Purpose:** Returns the activities that are targets for a-flow-lines in the input list **Syntax:** `stm_r_ac_target_of_af (stm_list in_list, int *status);`

Input List Type: ch

stm_r_ac_def_or_unres_in_ch	**Query:** Activities defined or unresolved in a given chart **Purpose:** Returns activities that are explicitly defined or unresolved in the charts of the input list **Syntax:** `stm_r_ac_def_or_unres_in_ ch (stm_list in_list, int *status);`
stm_r_ac_defined_in_ch	**Query:** Activities defined in a given chart **Purpose:** Returns the activities that are explicitly defined in the charts of the input list **Syntax:** `stm_r_ac_defined_in_ch (stm_list in_list, int *status);`
stm_r_ac_described_by_ch	**Query:** Control activities described by a given statechart **Purpose:** Returns the control activities described by statecharts in the input list **Syntax:** `stm_r_ac_described_by_ch (stm_list in_list, int *status);`
stm_r_ac_instance_of_ch	**Query:** Activities instance of a given chart **Purpose:** Returns the instance activities defined by the charts in the input list **Syntax:** `stm_r_ac_instance_of_ch (stm_list in_list, int *status);`
stm_r_ac_root_in_ch	**Query:** Root activities of a given chart **Purpose:** Returns the internally defined activities (of type diagram) attached to the charts in the input list **Syntax:** `stm_r_ac_root_in_ch (stm_list in_list, int *status);`

stm_r_ac_top_level_in_ch	**Query:** Top-level activities of a given chart **Purpose:** Returns the top-level activities (not contained in any box) of the charts in the input list **Syntax:** `stm_r_ac_top_level_in_ch (stm_list in_list, int *status);`
stm_r_ac_unresolved_in_ch	**Query:** Activities unresolved in a given chart **Purpose:** Returns activities that are unresolved in the charts of the input list **Syntax:** `stm_r_ac_unresolved_in_ch (stm_list in_list, int *status);`

Input List Type: ds

stm_r_ac_parent_of_ds	**Query:** Parent activities of a given data-store **Purpose:** Returns the activities that encapsulate the specified data-stores from the input list **Syntax:** `stm_r_ac_parent_of_ds (stm_list in_list, int *status);`

Input List Type: md

stm_r_ac_carried_out_by_md	**Query:** Activities carried out by a given module. **Purpose:** Returns the activities carried out by modules in the input list. The module appears in the **Implemented by Module** field of the activity's form. **Syntax:** `stm_r_ac_carried_out_by_md (stm_list in_list, int *status);`

Input List Type: mx

stm_r_ac_affecting_mx	**Query:** Activities in which a given element is affected. **Purpose:** Returns the activities that affect (modify, generate, or activate) the elements (for example, events, data-items, or activities) in the input list. **Syntax:** `stm_r_ac_affecting_mx (stm_list in_list, int *status);`
stm_r_ac_meaningly_affecting_mx	**Query:** Activities in which a given element is affected. **Purpose:** Identical to stm_r_ac_affecting_mx, but when the input list includes an ID of a record/union, stm_r_ac_meaningly_affecting_mx will also return elements that affect a field of the record/union, and not necessarily the whole record/union element. Syntax: `stm_r_ac_meaningly_affecting_mx (stm_list in_list, int *status);`
stm_r_ac_meaningly_using_mx	**Query:** Activities in which a given element is used. **Purpose:** Identical to stm_r_ac_using_mx, but when the input list includes an ID of a record/union, stm_r_ac_meaningly_using_mx will also return elements that use a field of the record/union, and not necessarily the whole record/union element. **Syntax:** `stm_r_ac_meaningly_using_mx (stm_list in_list, int *status);`
stm_r_ac_using_mx	**Query:** Activities in which a given element is used. **Purpose:** Returns the activities that use (evaluate) the elements (basic events, conditions, data-items, states, and activities) in the input list. **Syntax:** `stm_r_ac_using_mx (stm_list in_list, int *status);`

Input List Type: router

stm_r_ac_parent_of_router	**Query:** Parent activities of a given router **Purpose:** Returns the activities that encapsulate the specified routers from the input list **Syntax:** `stm_r_ac_parent_of_router (stm_list in_list, int *status);`

Input List Type: st

stm_r_ac_throughout_st	**Query:** Activities performed throughout a given state **Purpose:** Returns the activities performed throughout states in the input list (as defined in the **Activities Within/Throughout** field of the state's form) **Syntax:** `stm_r_ac_throughout_st (stm_list in_list, int *status);`
stm_r_ac_within_st	**Query:** Activities performed within a given state **Purpose:** Returns the activities performed within states in the input list (as defined in the **Activities Within/ Throughout** field of the state's form) **Syntax:** `stm_r_ac_within_st (stm_list in_list, int *status);`

Input List Type: uc

stm_r_ac_associates_uc	**Query:** Activities performed throughout a given state **Purpose:** Returns the activities that associate with use-cases in the input list. **Syntax:** `stm_r_ac_associates_uc (stm_list in_list, int *status);`
stm_r_uc_associates_ac	**Query:** Activities performed throughout a given state **Purpose:** Returns the use cases that associate with activities in the input list. **Syntax:** `stm_r_uc_associates_ac (stm_list in_list, int *status);`
stm_r_uc_explicit_defined_uc	**Query:** **Purpose:** Extracts a list of elements from the input list that are explictly defined elements of the requested type **Syntax:** `stm_r_bb_explicit_defined_uc (stm_list bb_list, int *status);`
stm_r_ouc_of_uc	**Query:** **Purpose:** Retrieve ids of all occurrences of use-cases in the input list **Syntax:** `stm_r_ouc_of_uc(stm_list in_list,int *status);`

A-Flow-Lines (af, ba, laf)

This section lists the query functions that return a list of a-flow-lines.

Two abbreviations are used in these functions:

- `af`—Global (compound) a-flow-lines
- `ba`—Basic a-flow-lines
- `laf`—Local a-flow-lines

Output List Type: af

Input List Type: ac

stm_r_af_from_source_ac	**Query:** A-flow-lines whose source is a given activity **Purpose:** Returns global compound a-flow-lines that originate at activities in the input list **Syntax:** `stm_r_af_from_source_ac (stm_list in_list, int *status);`
stm_r_af_input_to_ac	**Query:** A-flow-lines input to a given activity within chart **Purpose:** Returns all local compound a-flow-lines that originate outside and terminate at (or inside) activities in the input list **Syntax:** `stm_r_af_input_to_ac (stm_list in_list, int *status);`
stm_r_af_output_from_ac	**Query:** A-flow-lines output from a given activity **Purpose:** Returns all global compound a-flow-lines that originate at (or inside) and terminate outside activities in the input list **Syntax:** `stm_r_af_output_from_ac (stm_list in_list, int *status);`
stm_r_af_to_target_ac	**Query:** A-flow-lines whose target is a given activity **Purpose:** Returns global a-flow-lines that terminate at activities in the input list **Syntax:** `stm_r_af_to_target_ac (stm_list in_list, int *status);`

Input List Type: co

stm_r_af_within_flows_co	**Query:** A-flow-lines through which a given condition flows **Purpose:** Returns the a-flow-lines through which conditions in the input list actually flow **Syntax:** `stm_r_af_within_flows_co (stm_list in_list, int *status);`
stm_r_af_within_labels_co	**Query:** A-flow-lines labeled with a given condition **Purpose:** Returns the a-flow-lines labeled with conditions in the input list **Syntax:** `stm_r_af_within_labels_co (stm_list in_list, int *status);`

Input List Type: di

stm_r_af_within_flows_di	**Query:** A-flow-lines through which a given data-item flows **Purpose:** Returns the a-flow-lines through which data-items in the input list actually flow **Syntax:** `stm_r_af_within_flows_di (stm_list in_list, int *status);`
stm_r_af_within_labels_di	**Query:** A-flow-lines labeled by a given data-item **Purpose:** Returns the a-flow-lines labeled with data-items in the input list **Syntax:** `stm_r_af_within_labels_di (stm_list in_list, int *status);`

Input List Type: ds

stm_r_af_from_source_ds	**Query:** A-flow-lines whose source is a given data-store **Purpose:** Returns global compound a-flow-lines that originate at data-stores in the input list **Syntax:** `stm_r_af_from_source_ds (stm_list in_list, int *status);`
stm_r_af_to_target_ds	**Query:** A-flow-lines whose target is a given data-store **Purpose:** Returns global compound a-flow-lines that terminate at data-stores in the input list **Syntax:** `stm_r_af_to_target_ds (stm_list in_list, int *status);`

Input List Type: ev

stm_r_af_within_flows_ev	**Query:** A-flow-lines through which a given event flows **Purpose:** Returns the a-flow-lines through which events in the input list actually flow **Syntax:** `stm_r_af_within_flows_ev (stm_list in_list, int *status);`
stm_r_af_within_labels_ev	**Query:** A-flow-lines through which a given event flows **Purpose:** Returns the a-flow-lines labeled with events in the input list **Syntax:** `stm_r_af_within_labels_ev (stm_list in_list, int *status);`

Input List Type: if

stm_r_af_within_flows_if	**Query:** A-flow-lines through which a given information-flow flows **Purpose:** Returns the a-flow-lines through which information-flows in the input list actually flow **Syntax:** `stm_r_af_within_flows_if (stm_list in_list, int *status);`
stm_r_af_within_labels_if	**Query:** A-flow-lines labeled with a given information-flow **Purpose:** Returns the a-flow-lines labeled with information-flows in the input list **Syntax:** `stm_r_af_within_labels_if (stm_list in_list, int *status);`

Input List Type: laf

stm_r_af_containing_laf	**Query:** None **Purpose:** Returns the global a-flow-lines (which might spread over several charts) that contain the local a-flow-lines (those within charts) in the input list **Syntax:** `stm_r_af_containing_laf (stm_list l, int *status);`

Input List Type: mx

stm_r_af_from_source_mx	**Query:** A-flow-lines whose source is a given element **Purpose:** Returns global compound a-flow-lines whose source is an element from the input list **Syntax:** `stm_r_af_from_source_mx (stm_list in_list, int *status);`
stm_r_af_to_target_mx	**Query:** A-flow-lines whose target is given element **Purpose:** Returns global compound a-flow-lines whose target is an element from the input list **Syntax:** `stm_r_af_to_target_mx (stm_list in_list, int *status);`
stm_r_af_within_flows_mx	**Query:** A-flow-lines through which a given element flows **Purpose:** Returns the a-flow-lines through which elements in the input list actually flow **Syntax:** `stm_r_af_within_flows_mx (stm_list in_list, int *status);`
stm_r_af_within_labels_mx	**Query:** A-flow-lines labeled by a given element **Purpose:** Returns the a-flow-lines labeled with elements in the input list **Syntax:** `stm_r_af_within_labels_mx (stm_list in_list, int *status);`

Input List Type: router

stm_r_af_from_source_router	**Query:** A-flow-lines whose source is a given router **Purpose:** Returns global compound a-flow-lines whose source is a router from the input list **Syntax:** `stm_r_af_from_source_ router (stm_list in_list, int *status);`
stm_r_af_to_target_router	**Query:** A-flow-lines whose target is given router **Purpose:** Returns global compound a-flow-lines whose target is a router from the input list **Syntax:** `stm_r_af_to_target_router (stm_list in_list, int *status);`

Output List Type: ba

Input List Type: af

stm_r_ba_contained_in_af	**Query:** None **Purpose:** Returns the basic a-flow-lines that contain the a-flow-lines in the input list **Syntax:** `stm_r_ba_contained_in_af (stm_list l, int *status);`

Output List Type: ba

Input List Type: ch

stm_r_ba_defined_in_ch	**Query:** None **Purpose:** Returns the a-flow-lines defined in the input list of charts **Syntax:** `stm_r_ba_defined_in_ch (stm stm_list in_list, int *status);`

Output List Type: bt

Input List Type: ch

stm_r_bt_defined_in_ch	**Query:** None **Purpose:** Returns the basic transitions defined in the input list of charts **Syntax:** `stm_r_bt_defined_in_ch (stm stm_list in_list, int *status);`

Output List Type: laf

Input List Type: ac

stm_r_laf_from_source_ac	**Query:** A-flow-lines whose source is a given activity **Purpose:** Returns local compound a-flow-lines that originate at activities in the input list **Syntax:** `stm_r_laf_from_source_ac (stm_list in_list, int *status);`
stm_r_laf_input_to_ac	**Query:** A-flow-lines input to a given activity **Purpose:** Returns all the local a-flow-lines **Syntax:** `stm_r_laf_input_to_ac (stm_list in_list, int *status);`
stm_r_laf_output_from_ac	**Query:** A-flow-lines output from a given activity within chart **Purpose:** Returns all local compound a-flow-lines that originate at (or inside) and terminate outside activities in the input list **Syntax:** `stm_r_laf_output_from_ac (stm_list in_list, int *status);`
stm_r_laf_to_target_ac	**Query:** A-flow-lines whose target is a given activity within chart **Purpose:** Returns local a-flow-lines (those within charts) that terminate at activities in the input list **Syntax:** `stm_r_laf_to_target_ac (stm_list in_list, int *status);`

Input List Type: af

stm_r_laf_contained_in_af	**Query:** None **Purpose:** Returns the local a-flow-lines that contain the global a-flow-lines in the input list **Syntax:** `stm_r_laf_contained_in_af (stm_list l, int *status);`

Input List Type: ds

stm_r_laf_from_source_ds	**Query:** A-flow-lines whose source is a given data-store within chart **Purpose:** Returns local compound a-flow-lines that originate at data-stores in the input list **Syntax:** `stm_r_laf_from_source_ds (stm_list in_list, int *status);`
stm_r_laf_to_target_ds	**Query:** A-flow-lines whose target is a given data-store within chart **Purpose:** Returns local compound a-flow-lines that terminate at data-stores in the input list **Syntax:** `stm_r_laf_to_target_ds (stm_list in_list, int *status);`

Input List Type: mx

stm_r_laf_from_source_mx	**Query:** A-flow-lines whose source is a given element within chart **Purpose:** Returns local compound a-flow-lines whose source is an element from the input list **Syntax:** `stm_r_laf_from_source_mx (stm_list in_list, int *status);`
stm_r_laf_to_target_mx	**Query:** A-flow-lines whose target is given element within chart **Purpose:** Returns local compound a-flow-lines whose target is an element from the input list **Syntax:** `stm_r_laf_to_target_mx (stm_list in_list, int *status);`

Input List Type: router

stm_r_laf_from_source_router	**Query:** A-flow-lines whose source is a given router within chart **Purpose:** Returns local compound a-flow-lines whose source is a router from the input list **Syntax:** `stm_r_laf_from_source_ router (stm_list in_list, int *status);`
stm_r_laf_to_target_router	**Query:** A-flow-lines whose target is given router within chart **Purpose:** Returns local compound a-flow-lines whose target is a router from the input list **Syntax:** `stm_r_laf_to_target_router (stm_list in_list, int *status);`

Actions (an)

This section documents the query functions that return a list of actions.

Input List Type: an

stm_r_an_by_attributes_an	**Query:** Actions by attribute **Purpose:** Returns the actions in the input list that match a given attribute and value `Syntax: stm_r_an_by_attributes_an (stm_list in_list, char* attr_name, char* attr_value, int *status);`
stm_r_an_explicit_defined_an	**Query:** Actions explicitly defined **Purpose:** Returns the actions of the input list that were explicitly defined **Syntax:** `stm_r_an_explicit_defined_an (stm_list in_list, int *status);`
stm_r_an_imp_best_match_an	**Query:** Actions whose selected implementation is **Best Match** **Purpose:** Returns the actions in the input list implemented as the **Best Match** using **Select Implementation** in the properties **Syntax:** `stm_r_an_imp_best_match_an (stm_list in_list, int *status);`
stm_r_an_imp_definition_an	**Query:** Actions with a defined implementation **Purpose:** Returns the actions in the input list that have a defined implementation in the properties **Syntax:** `stm_r_an_imp_definition_an (stm_list in_list, int *status);`
stm_r_an_imp_none_an	**Query:** Actions whose selected implementation is **None** **Purpose:** Returns the actions in the input list that are not implemented using Select Implementation **Syntax:** `stm_r_an_imp_none_an (stm_list in_list, int *status);`

stm_r_an_imp_truth_table_an	**Query:** Actions implemented in a truth table **Purpose:** Returns the actions in the input list that are implemented with a truth table in the properties **Syntax:** `stm_r_an_imp_truth_table_ an (stm_list in_list, int *status);`
stm_r_an_name_of_an	**Query:** Actions whose names match a given pattern **Purpose:** Returns all the actions whose names match a specified pattern **Syntax:** `stm_r_an_name_of_an (char* pattern, int *status);`
stm_r_an_synonym_of_an	**Query:** Actions whose synonyms match a given pattern **Purpose:** Returns all the actions whose synonyms match a specified pattern **Syntax:** `stm_r_an_synonym_of_an (char* pattern, int *status);`
stm_r_an_unresolved_an	**Query:** Unresolved actions **Purpose:** Returns the unresolved actions in the input list **Syntax:** `stm_r_an_unresolved_an (stm_list in_list, int *status);`

Input List Type: ch

stm_r_an_def_or_unres_in_ch	**Query:** Actions defined or unresolved in a given chart **Purpose:** Returns the actions that are explicitly defined or unresolved in the charts of the input list **Syntax:** `stm_r_an_def_or_unres_in_ ch (stm_list in_list, int *status);`
stm_r_an_defined_in_ch	**Query:** Actions defined in a given chart **Purpose:** Returns the actions that are explicitly defined in the charts of the input list **Syntax:** `stm_r_an_defined_in_ch (stm_list in_list, int *status);`
stm_r_an_unresolved_in_ch	**Query:** Actions unresolved in a given chart **Purpose:** Returns the actions that are unresolved in the charts of the input list **Syntax:** `stm_r_an_unresolved_in_ch (stm_list in_list, int *status);`

Actors (actor)

This section documents the query function that returns a list of actors.

Input List Type: actor

stm_r_actor_explicit_defined_actor	**Query:** **Purpose:** Extracts a list of elements from the input list that are explictly defined elements of the requested type **Syntax:** `stm_r_actor_explicit_defined_actor (stm_list actor_list, int *status);`
stm_r_oactor_of_actor	**Query:** **Purpose:** Retrieve ids of all occurrences of actors in the input list **Syntax:** `stm_r_oactor_of_actor(stm_list in_list,int *status);`

Input List Type: ch

stm_r_actor_defined_in_ch	**Query:** Actors of a given chart **Purpose:** Returns the actors of the charts in the input list **Syntax:** `stm_r_actor_defined_in_ch (stm_list in_list, int *status);`

Basic relation(br)

This section documents the query functions that return a list of basic relations.

Input List Type: ch

stm_r_br_defined_in_ch	**Query:** Basic relations of a given chart **Purpose:** Retrieve ids of all relations defined in the input list charts **Syntax:** `stm_r_br_defined_in_ch(stm_list in_list, int *status)`

Input List Type: actor

stm_r_br_enter_actor	**Query:** Basic relations entering actors **Purpose:** Retrieve ids of all relations entering the actors in the input list **Syntax:** `stm_r_br_enter_actor(stm_list in_list,int *status)`
stm_r_br_exit_from_actor	**Query:** Basic relations exiting actors **Purpose:** Retrieve ids of all relations exiting the actors in the input list **Syntax:** `stm_r_br_exit_from_actor(stm_list in_list,int *status)`

Input List Type: uc

stm_r_br_enter_uc	**Query:** Basic relations entering use-cases **Purpose:** Retrieve ids of all relations entering the use-cases in the input list **Syntax:** `stm_r_br_enter_uc(stm_list in_list,int *status)`
stm_r_br_exit_from_uc	**Query:** Basic relations exiting use-cases **Purpose:** Retrieve ids of all relations exiting the use-cases in the input list **Syntax:** `stm_r_br_exit_from_uc(stm_list in_list,int *status)`

Boundary Boxes (bb)

This section documents the query function that returns a list of boundary boxes.

Output List Type: bb

stm_r_bb_explicit_defined_bb	**Query:** **Purpose:** Extracts a list of elements from the input list that are explictly defined elements of the requested type **Syntax:** `stm_r_bb_explicit_defined_bb (stm_list bb_list, int *status);`

Output List Type: ch

stm_r_bb_defined_in_ch	**Query:** Boundary boxes of a given chart **Purpose:** Returns the boundary boxes of the charts in the input list **Syntax:** `stm_r_bb_defined_in_ch (stm_list in_list, int *status);`

Combinational Assignments (ca)

This section documents the query function that returns a list of combinational assignments.

Output List Type: mx

stm_r_ca_contained_in_mx	**Syntax:** `stm_r_ca_contained_in_mx (stm_list mx_l, int *status);`

Charts (ch)

This section documents the query functions that return a list of charts.

Input List Type: ac

stm_r_ch_define_ac	**Query:** Charts in which a given activity is defined **Purpose:** Returns the charts in which the activities in the input list are explicitly defined or unresolved **Syntax:** `stm_r_ch_define_ac (stm_list in_list, int *status);`
stm_r_ch_defining_ac	**Query:** Activity-charts defining a given activity **Purpose:** Returns the activity-charts that define the instance activities in the input list **Syntax:** `stm_r_ch_defining_ac (stm_list in_list, int *status);`
stm_r_ch_defining_cd_inst_ac	**Query**: **Purpose:** **Syntax:** `stm_r_ch_defining_cd_inst_ac (stm_list in_list, int *status);`
stm_r_ch_describing_ac	**Query:** Statecharts / Flowchars describing a given control activity **Purpose:** Returns the statecharts / Flowchars that describe the control activities in the input list **Syntax:** `stm_r_ch_describing_ac (stm_list in_list, int *status);`

Input List Type: an

stm_r_ch_define_an	**Query:** Charts in which a given action is defined **Purpose:** Returns the charts in which the actions in the input list are explicitly defined or unresolved **Syntax:** `stm_r_ch_define_an (stm_list in_list, int *status);`

Input List Type: ch

stm_r_ch_activitychart_ch	**Query:** Activity-charts **Purpose:** Returns the activity-charts in the input list **Syntax:** `stm_r_ch_activitychart_ch (stm_list in_list, int *status);`
stm_r_ch_ancestors_of_ch	**Query:** Ancestors of a given chart **Purpose:** Returns the ancestors (in the static structure) of the charts in the input list **Syntax:** `stm_r_ch_ancestors_of_ch (stm_list in_list, int *status);`
stm_r_ch_by_attributes_ch	**Query:** Chart by attribute **Purpose:** Returns the charts in the input list that match the specified attribute name and value **Syntax:** `stm_r_ch_by_attributes_ch (stm_list in_list, char* attr_name, char* attr_value, int *status);`
stm_r_ch_descendants_of_ch	**Query:** Descendants of a given chart **Purpose:** Returns the descendants (in the static structure) of the charts in the input list **Syntax:** `stm_r_ch_descendants_of_ch (stm_list in_list, int *status);`
stm_r_ch_dictionary_ch	**Query:** Global definition sets (GDSs) **Purpose:** Returns the GDSs in the input list **Syntax:** `stm_r_ch_dictionary_ch (stm_list in_list, int *status);`
stm_r_ch_explicit_defined_ch	**Query:** Charts explicitly defined **Purpose:** Returns the charts of the input list that were explicitly defined **Syntax:** `stm_r_ch_explicit_defined_ch (stm_list in_list, int *status);`

stm_r_ch_flowchart_ch	**Query:** **Purpose:** **Syntax:** `stm_r_ch_flowchart_ch (stm_list in_list, int *status);`
stm_r_ch_generic_ch	**Query:** Generic charts **Purpose:** Returns the generic charts in the input list **Syntax:** `stm_r_ch_generic_ch (stm_list in_list, int *status);`
stm_r_ch_modulechart_ch	**Query:** Module-charts **Purpose:** Returns the charts in the input list that are module-charts **Syntax:** `stm_r_ch_modulechart_ch (stm_list in_list, int *status);`
stm_r_ch_name_of_ch	**Query:** Charts whose names match a given pattern **Purpose:** Returns all the charts whose names match the specified pattern **Syntax:** `stm_r_ch_name_of_ch (char* pattern, int *status);`
stm_r_ch_offpage_ch	**Query:** Offpage charts **Purpose:** Returns the offpage charts in the input list **Syntax:** `stm_r_ch_offpage_ch (stm_list in_list, int *status);`
stm_r_ch_parent_ch	**Query:** Returns the parent charts of a given chart **Purpose:** Returns the parents (in the static structure) of the charts in the input list **Syntax:** `stm_r_ch_parent_ch (stm_list in_list, int *status);`
stm_r_ch_procedural_sch_ch	**Query:** Procedural statecharts **Purpose:** Returns the charts in the input list that are procedural statecharts **Syntax:** `stm_r_ch_procedural_sch_ch (stm_list in_list, int *status);`

stm_r_ch_referenced_all_by_ch	**Query:** Charts referenced in all levels by a given chart **Purpose:** Returns all charts referenced (instantiated) by all levels of charts in the input list **Syntax:** `stm_r_ch_referenced_all_by_ch (stm_list in_list, int *status);`
stm_r_ch_referenced_by_ch	**Query:** Charts referenced by a given chart **Purpose:** Returns all charts referenced (instantiated) by the charts in the input list **Syntax:** `stm_r_ch_referenced_by_ch (stm_list in_list, int *status);`
stm_r_ch_root_ch	**Query:** Root charts **Purpose:** Returns the root-level charts (that have no parent) in the input list **Syntax:** `stm_r_ch_root_ch (stm_list in_list, int *status);`
stm_r_ch_seq_diag_ch	**Query:** **Purpose:** **Syntax:** `stm_r_ch_seq_diag_ch (stm_list in_list, int *status);`
stm_r_ch_statechart_ch	**Query:** Statecharts **Purpose:** Returns the charts in the input list that are statecharts **Syntax:** `stm_r_ch_statechart_ch (stm_list in_list, int *status);`
stm_r_ch_subchart_ch	**Query:** Subchart of the specified chart **Purpose:** Returns the subcharts (in the static structure) of the charts in the input list **Syntax:** `stm_r_ch_subchart_ch (stm_list in_list, int *status);`
stm_r_ch_unresolved_ch	**Query:** Unresolved charts **Purpose:** Returns the unresolved charts (used but not defined) in the input list **Syntax:** `stm_r_ch_unresolved_ch (stm_list in_list, int *status);`

stm_r_ch_use_case_ch	**Query:** **Purpose:** **Syntax:** `stm_r_ch_use_case_ch (stm_list in_list, int *status);`
stm_r_ch_with_notes_ch	**Query:** **Purpose:** **Syntax:** `stm_r_ch_with_notes_ch (stm_list in_list, int *status);`

Input List Type: co

stm_r_ch_define_co	**Query:** Charts in which a given condition is defined **Purpose:** Returns the charts in which the conditions in the input list are explicitly defined or unresolved **Syntax:** `stm_r_ch_define_co (stm_list in_list, int *status);`

Input List Type: di

stm_r_ch_define_di	**Query:** Charts in which a given data-item is defined **Purpose:** Returns the charts in which the data-items in the input list are explicitly defined or unresolved **Syntax:** `stm_r_ch_define_di (stm_list in_list, int *status);`

Input List Type: ds

stm_r_ch_define_ds	**Query:** Charts in which a given data-store is defined **Purpose:** Returns the charts in which the data-stores in the input list are explicitly defined or unresolved **Syntax:** `stm_r_ch_define_ds (stm_list in_list, int *status);`

Input List Type: dt

stm_r_ch_define_dt	**Query:** Charts and GDSs in which a given user-defined type is defined **Purpose:** Returns the charts in which the user-defined types in the input list are explicitly defined or unresolved **Syntax:** `stm_r_ch_define_dt (stm_list in_list, int *status);`

Input List Type: ev

stm_r_ch_define_ev	**Query:** Charts in which a given event is defined **Purpose:** Returns the charts in which the events in the input list are explicitly defined or unresolved **Syntax:** `stm_r_ch_define_ev (stm_list in_list, int *status));`

Input List Type: fd

stm_r_ch_define_fd	**Query:** Charts and GDSs in which a given field is defined **Purpose:** Returns the charts in which the fields in the input list are defined (in a structured data-item or user-defined type) **Syntax:** `stm_r_ch_define_fd (stm_list in_list, int *status);`

Input List Type: if

stm_r_ch_define_if	**Query:** Charts in which a given information-flow is defined **Purpose:** Returns the charts in which the information-flows in the input list are explicitly defined or unresolved **Syntax:** `stm_r_ch_define_if (stm_list in_list, int *status);`

Input List Type: md

stm_r_ch_define_md	**Query:** Charts in which a given module is defined **Purpose:** Returns charts in which the modules in the input list are explicitly defined or unresolved **Syntax:** `stm_r_ch_define_md (stm_list in_list, int *status);`
stm_r_ch_defining_md	**Query:** Module-charts defining a given module **Purpose:** Returns the module-charts that define the instance modules in the input list **Syntax:** `stm_r_ch_defining_md (stm_list in_list, int *status);`
stm_r_ch_describing_md	**Query:** Activity-charts describing a given module **Purpose:** Returns the activity-charts that describe the modules in the input list **Syntax:** `stm_r_ch_describing_md (stm_list in_list, int *status);`

Input List Type: mx

stm_r_ch_define_mx	**Query:** Charts in which a given element is defined **Purpose:** Returns the charts in which the elements in the input list are explicitly defined **Syntax:** `stm_r_ch_define_mx (stm_list in_list, int *status);`
stm_r_ch_defining_mx	**Query:** Charts defining a given element **Purpose:** Returns the charts that define the elements in the input list **Syntax:** `stm_r_ch_defining_mx (stm_list in_list, int *status);`
stm_r_ch_describing_mx	**Query:** Statecharts / Flowchars describing a given control activity **Purpose:** Returns the charts that describe the elements in the input list **Syntax:** `stm_r_ch_describing_mx (stm_list in_list, int *status);`

Input List Type: nt

stm_r_ch_with_nt	**Query:** Charts in which a given note is defined **Purpose:** Returns the charts in which the notes in the input list are defined **Syntax:** `stm_r_ch_with_nt (stm_list in_list, int *status);`

Input List Type: router

stm_r_ch_define_router	**Query:** Charts in which a given router is defined **Purpose:** Returns the charts in which the routers in the input list are explicitly defined or unresolved **Syntax:** `stm_r_ch_define_router (stm_list in_list, int *status);`

Input List Type: sb

stm_r_ch_connected_to_sb	**Query:** Charts connected to a given subroutine **Purpose:** Returns the procedural Statecharts that are connected to the subroutines in the input list **Syntax:** `stm_r_ch_connected_to_sb (stm_list in_list, int *status);`
stm_r_sch_connected_to_sb	**Query:** Statecharts connected to a given subroutine **Purpose:** Returns the procedural Statecharts that are connected to the subroutines in the input list **Syntax:** `stm_r_sch_connected_to_sb (stm_list in_list, int *status);`
stm_r_fch_connected_to_sb	**Query:** Flowcharts connected to a given subroutine **Purpose:** Returns the Flowcharts that are connected to the subroutines in the input list **Syntax:** `stm_r_fch_connected_to_sb (stm_list in_list, int *status);`
stm_r_ch_define_sb	**Query:** Charts in which a given subroutine is defined **Purpose:** Returns the charts in which the subroutines in the input list are explicitly defined or unresolved **Syntax:** `stm_r_ch_define_sb (stm_list in_list, int *status);`

Input List Type: st

stm_r_ch_define_st	**Query:** Charts in which a given state is defined **Purpose:** Returns the charts in which the states in the input list are explicitly defined or unresolved **Syntax:** stm_r_ch_define_st (stm_list in_list, int *status);
stm_r_ch_defining_st	**Query:** Statecharts defining a given state **Purpose:** Returns the statecharts that define the instance states in the input list **Syntax:** stm_r_ch_defining_st (stm_list in_list, int *status);

Connectors (cn)

This section documents the queries that return a list of connectors.

Input List Type: ba

stm_r_cn_source_of_ba	**Query:** History connectors sources of a given transition **Purpose:** Returns the connectors that are sources of basic a-flow-lines in the input list **Syntax:** `stm_r_cn_source_of_ba (stm_list in_list, int *status);`
stm_r_cn_target_of_ba	**Query:** Termination or history connectors targets of a given transition **Purpose:** Returns the connectors that are targets of basic a-flow-lines in the input list **Syntax:** `stm_r_cn_target_of_ba (stm_list in_list, int *status);`

Input List Type: bm

stm_r_cn_source_of_bm	**Query:** History connectors sources of a given transition **Purpose:** Returns the connectors that are sources of basic m-flow-lines in the input list **Syntax:** `stm_r_cn_source_of_bm (stm_list in_list, int *status);`
stm_r_cn_target_of_bm	**Query:** Termination or history connectors targets of a given transition **Purpose:** Returns the connectors that are targets of basic m-flow-lines in the input list **Syntax:** `stm_r_cn_target_of_bm (stm_list in_list, int *status);`

Input List Type: bt

stm_r_cn_source_of_bt	**Query:** History connectors sources of a given transition **Purpose:** Returns the connectors that are sources of basic transitions in the input list **Syntax:** `stm_r_cn_source_of_bt (stm_list in_list, int *status);`
stm_r_cn_target_of_bt	**Query:** Termination or history connectors targets of a given transition **Purpose:** Returns the connectors that are targets of basic transitions in the input list **Syntax:** `stm_r_cn_target_of_bt (stm_list in_list, int *status);`

Input List Type: cn

stm_r_cn_deep_history_cn	**Query:** Deep history connectors **Purpose:** Returns all the deep history connectors in the input list **Syntax:** `stm_r_cn_deep_history_cn (stm_list in_list, int *status);`
stm_r_cn_history_cn	**Query:** History connectors **Purpose:** Returns all the history connectors in the input list **Syntax:** `stm_r_cn_history_cn (stm_list in_list, int *status);`
stm_r_cn_termination_cn	**Query:** Termination connectors **Purpose:** Returns all the history connectors in the input list **Syntax:** `stm_r_cn_termination_cn (stm_list in_list, int *status);`

Input List Type: st

• stm_r_cn_history_or_term_in_st	**Query:** Termination or history connectors in a given state **Purpose:** Returns the termination and history connectors contained in the states in the input list **Syntax:** `stm_r_cn_history_or_term_in_st (stm_list in_list, int *status);`
• stm_r_cn_in_st	**Query:** Connectors in a given state **Purpose:** Returns the connectors contained in the states in the input list **Syntax:** `stm_r_cn_in_st (stm_list in_list, int *status);`

Input List Type: tr

stm_r_cn_source_of_tr	**Query:** History connectors sources of a given transition **Purpose:** Returns the history connectors that are sources of transitions in the input list **Syntax:** `stm_r_cn_source_of_tr (stm_list in_list, int *status);`
stm_r_cn_target_of_tr	**Query:** Termination or history connectors targets of a given transition **Purpose:** Returns the termination and history connectors that are targets of transitions in the input list **Syntax:** `stm_r_cn_target_of_tr (stm_list in_list, int *status);`

Conditions (co)

This section documents the query functions that return a list of conditions.

Input List Type: af

stm_r_co_flowing_through_af	**Query:** Conditions flowing through a given a-flow-line **Purpose:** Returns the conditions actually flowing through a-flow-lines in the input list **Syntax:** `stm_r_co_flowing_through_af (stm_list in_list, int *status);`
stm_r_co_labeling_af	**Query:** Conditions labeling a given a-flow-line **Purpose:** Returns the conditions which label the a-flow-lines in the input list **Syntax:** `stm_r_co_labeling_af (stm_list in_list, int *status);`

Input List Type: ch

stm_r_co_def_or_unres_in_ch	**Query:** Conditions defined or unresolved in a given chart **Purpose:** Returns conditions that are explicitly defined or unresolved in the charts of the input list **Syntax:** `stm_r_co_def_or_unres_in_ch (stm_list in_list, int *status);`
stm_r_co_defined_in_ch	**Query:** Conditions defined in a given chart **Purpose:** Returns the conditions that are explicitly defined in the charts of the input list **Syntax:** `stm_r_co_defined_in_ch (stm_list in_list, int *status);`
stm_r_co_unresolved_in_ch	**Query:** Conditions unresolved in a given chart **Purpose:** Returns conditions that are unresolved in the charts of the input list **Syntax:** `stm_r_co_unresolved_in_ch (stm_list in_list, int *status);`

Input List Type: co

stm_r_co_array_co	**Query:** Conditions by subtype **Purpose:** Returns the conditions in the input list that are defined as array **Syntax:** `stm_r_co_array_co (stm_list in_list, int *status);`
stm_r_co_by_attributes_co	**Query:** Conditions by attributes **Purpose:** Returns the conditions in the input list that match the specified attribute name and value **Syntax**: `stm_r_co_by_attributes_co (stm_list in_list, char* attr_name, char* attr_value, int *status);`
stm_r_co_by_structure_type_co	**Query:** None **Purpose:** Returns the conditions in the input list that have the specified structure type (for example, single or array) **Syntax:** `stm_r_co_by_structure_type_co (stm_list in_list, char structure_type, int *status);`
stm_r_co_callback_binding_co	**Query:** Conditions with callback bindings **Purpose:** Returns the conditions of the input list that have callback bindings **Syntax:** `stm_r_co_callback_binding_co (stm_list in_list, int *status);`
stm_r_co_explicit_defined_co	**Query:** Conditions explicitly defined **Purpose:** Returns the conditions of the input list that were explicitly defined **Syntax:** `stm_r_co_explicit_defined_co (stm_list in_list, int *status);`
stm_r_co_name_of_co	**Query:** Conditions whose names match a given pattern **Purpose:** Returns all the conditions whose names match the specified pattern **Syntax:** `stm_r_co_name_of_co (char* pattern, int *status);`

stm_r_co_single_co	**Query:** Conditions by subtype **Purpose:** Returns the conditions in the input list that are defined as single **Syntax:** `stm_r_co_single_co (stm_list in_list, int *status);`
stm_r_co_synonym_of_co	**Query:** Conditions whose synonyms match a given pattern **Purpose:** Returns all the conditions whose synonyms match the specified pattern **Syntax:** `stm_r_co_synonym_of_co (char* pattern, int *status);`
stm_r_co_unresolved_co	**Query:** Unresolved conditions **Purpose:** Returns the unresolved conditions in the input list **Syntax:** `stm_r_co_unresolved_co (stm_list in_list, int *status);`

Input List Type: di

stm_r_co_contained_in_di	**Query:** Conditions contained in a given data-item **Purpose:** Returns the conditions contained in data-items from the input list (conditions appearing in the **Consists of** field of a data-item) **Syntax:** `stm_r_co_contained_in_di (stm_list in_list, int *status);`

Input List Type: if

stm_r_co_contained_in_if	**Query:** Conditions contained in a given information-flow **Purpose:** Returns the conditions contained in information-flows from the input list (conditions appearing in the **Consists of** field of an information-flow) **Syntax:** `stm_r_co_contained_in_if (stm_list in_list, int *status);`

Input List Type: mf

stm_r_co_flowing_through_mf	**Query:** Conditions flowing through a given m-flow-line **Purpose:** Returns the conditions actually flowing through m-flow-lines in the input list **Syntax:** `stm_r_co_flowing_through_mf (stm_list in_list, int *status);`
stm_r_co_labeling_mf	**Query:** Conditions labeling a given m-flow-line **Purpose:** Returns the conditions that label the m-flow-lines in the input list **Syntax:** `stm_r_co_labeling_mf (stm_list in_list, int *status);`

Data-Items (di)

This section documents the query functions that return a list of data-items.

Input List Type: af

stm_r_di_flowing_through_af	**Query:** Data-items flowing through a given a-flow-line **Purpose:** Returns the data-items actually flowing through a-flow-lines in the input list **Syntax:** `stm_r_di_flowing_through_af (stm_list in_list, int *status);`
stm_r_di_labeling_af	**Query:** Data-items labeling a given a-flow-line **Purpose:** Returns the data-items which label the a-flow-lines in the input list **Syntax:** `stm_r_di_labeling_af (stm_list in_list, int *status);`

Input List Type: ch

stm_r_di_def_or_unres_in_ch	**Query:** Data-items defined or unresolved in a given chart **Purpose:** Returns the data-items explicitly defined or unresolved in the charts of the input list **Syntax:** `stm_r_di_def_or_unres_in_ch (stm_list in_list, int *status);`
stm_r_di_defined_in_ch	**Query:** Data-items defined in a given chart **Purpose:** Returns the data-items explicitly defined in the charts of the input list **Syntax:** `stm_r_di_defined_in_ch (stm_list in_list, int *status);`
stm_r_di_unresolved_in_ch	**Query:** Data-items unresolved in a given chart **Purpose:** Returns the data-items that are unresolved in the charts of the input list **Syntax:** `stm_r_di_unresolved_in_ch (stm_list in_list, int *status);`

Input List Type: co

stm_r_di_containing_co	**Query:** Data-item containing a given condition **Purpose:** Returns the data-items containing the conditions in the input list (as defined in the **Consists of** field of the data-item's form) **Syntax:** `stm_r_di_containing_co (stm_list in_list, int *status);`

Input List Type: di

stm_r_di_array_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as array **Syntax:** `stm_r_di_array_di (stm_list in_list, int *status);`
stm_r_di_array_missing_di	**Query:** Array of data-items by subtype **Purpose:** Returns the arrays of data-items in the input list for which no type is defined **Syntax:** `stm_r_di_array_missing_di (stm_list in_list, int *status);`
stm_r_di_basic_di	**Query:** Basic data-items **Purpose:** Returns the data-items in the input list that are basic (not defined using other data-items) **Syntax:** `stm_r_di_basic_di (stm_list in_list, int *status);`
stm_r_di_bit_di	**Query:** Basic data-items **Purpose:** Returns the data-items in the input list that are defined as Bit in the **Structure**/**Type** field of the data-item form **Syntax:** `stm_r_di_bit_di (stm_list in_list, int *status);`
stm_r_di_bit_queue_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as queue of bits **Syntax:** `stm_r_di_bit_queue_di (stm_list in_list, int *status);`
stm_r_di_bits_array_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as array of bit array **Syntax:** `stm_r_di_bits_array_di (stm_list in_list, int *status);`

stm_r_di_bits_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as bit-array in the **Structure**/**Type** field of the data-item form **Syntax:** `stm_r_di_bits_di (stm_list in_list, int *status);`
stm_r_di_bits_queue_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as queue of bit array **Syntax:** `stm_r_di_bits_queue_di (stm_list in_list, int *status);`
stm_r_di_by_attributes_di	**Query:** Data-items by attributes **Purpose:** Returns the data-items in the input list that match the specified attribute name and value **Syntax**: `stm_r_di_by_attributes_di (stm_list in_list, char* attr_name, char* attr_value, int *status);`
stm_r_di_by_structure_type_di	**Query:** None **Purpose:** Returns the data-items in the input list that have a particular structure type (for example, single, array, or queue) **Syntax**: `stm_r_di_by_structure_type_di (stm_list in_list, char structure_type, int *status);`
stm_r_di_callback_binding_di	**Query:** Data-items with callback bindings **Purpose:** Returns the data-items of the input list that have callback bindings **Syntax:** `stm_r_di_callback_binding_di (stm_list in_list, int *status);`
stm_r_di_explicit_defined_di	**Query:** Data-items explicitly defined **Purpose:** Returns the data-items of the input list that were explicitly defined **Syntax:** `stm_r_di_explicit_defined_di (stm_list in_list, int *status);`

stm_r_di_integer_di	**Query:** Integer subtype **Purpose:** Returns the data-items in the input list that are defined as integer in the **Structure**/**Type** field of the data-item's form **Syntax:** `stm_r_di_integer_di (stm_list in_list, int *status);`
stm_r_di_integer_array_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as array of integer **Syntax:** `stm_r_di_integer_array_di (stm_list in_list, int *status);`
stm_r_di_integer_queue_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as queue of integer **Syntax:** `stm_r_di_integer_queue_di (stm_list in_list, int *status);`
stm_r_di_list_di	**Query:** **Purpose:** **Syntax:** `stm_r_di_list_di (stm_list in_list, int *status);`
stm_r_di_missing_di	**Query:** Data-item by subtype **Purpose:** Returns the data-items in the input list for which no type is defined **Syntax:** `stm_r_di_missing_di (stm_list in_list, int *status);`
stm_r_di_name_of_di	**Query:** Data-items whose names match a given pattern **Purpose:** Returns all the data-items whose names match the specified pattern **Syntax:** `stm_r_di_name_of_di (char* pattern, int *status);`
stm_r_di_parent_of_di	**Query:** Parent data-items of a given data-item **Purpose:** Returns the data-items containing the data-items from the input list (as defined in the **Consists of** field of the data-item's form) **Syntax:** `stm_r_di_parent_of_di (stm_list in_list, int *status);`

stm_r_di_queue_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as queue **Syntax:** `stm_r_di_queue_di (stm_list in_list, int *status);`
stm_r_di_queue_missing_di	**Query:** Queues of data-items by subtype **Purpose:** Returns the queues of data-items in the input list for which no type is defined **Syntax:** `stm_r_di_queue_missing_di (stm_list in_list, int *status);`
stm_r_di_real_di	**Query:** Real subtype **Purpose:** Returns the data-items from the input list that are defined as Real (Float) in the **Structure**/**Type** field of the data-item's form **Syntax:** `stm_r_di_real_di (stm_list in_list, int *status);`
stm_r_di_real_array_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as real **Syntax:** `stm_r_di_real_array_di (stm_list in_list, int *status);`
stm_r_di_real_queue_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as queue of real **Syntax:** `stm_r_di_real_queue_di (stm_list in_list, int *status);`
stm_r_di_record_array_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as array of record **Syntax:** `stm_r_di_record_array_di (stm_list in_list, int *status);`

stm_r_di_record_di	**Query:** Record subtype **Purpose:** Returns the data-items from the input list that are defined as Record in the **Structure**/**Type** field of the data-item's form **Syntax:** `stm_r_di_record_di (stm_list in_list, int *status);`
stm_r_di_single_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as single **Syntax:** `stm_r_di_single_di (stm_list in_list, int *status);`
stm_r_di_string_array_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as array of string **Syntax:** `stm_r_di_string_array_di (stm_list in_list, int *status);`
stm_r_di_string_di	**Query:** String subtype **Purpose:** Returns the data-items from the input list that are defined as String in the **Structure**/**Type** field of the data-item's form **Syntax:** `stm_r_di_string_di (stm_list in_list, int *status);`
stm_r_di_string_queue_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as queue of string **Syntax:** `stm_r_di_string_queue_di (stm_list in_list, int *status);`
stm_r_di_subdata_item_of_di	**Query:** Subdata-item of a given data-item **Purpose:** Returns the data-items that are components of data-items in the input list (as defined in the **Consists of** field of the data-item's form) **Syntax:** `stm_r_di_subdata_item_of_ di (stm_list in_list, int *status);`

stm_r_di_synonym_of_di	**Query:** Data-items whose synonyms match a given pattern **Purpose:** Returns all the data-items whose synonyms match the specified pattern **Syntax:** `stm_r_di_synonym_of_di (char* pattern, int *status);`
stm_r_di_union_array_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as array **Syntax:** `stm_r_di_union_array_ di (stm_list in_list, int *status);`
stm_r_di_union_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as union **Syntax:** `stm_r_di_union_di (stm_list in_list, int *status);`
stm_r_di_unresolved_di	**Query:** Unresolved data-items **Purpose:** Returns the unresolved data-items in the input list **Syntax:** `stm_r_di_unresolved_di (stm_list in_list, int *status);`
stm_r_di_user_type_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as user-defined type **Syntax:** `stm_r_di_user_type_di (stm_list in_list, int *status);`
stm_r_di_user_type_array_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as array of user-defined type **Syntax:** `stm_r_di_user_type_array_ di (stm_list in_list, int *status);`
stm_r_di_user_type_queue_di	**Query:** Data-items by subtype **Purpose:** Returns the data-items in the input list that are defined as queue of user-defined type **Syntax:** `stm_r_di_user_type_queue_ di (stm_list in_list, int *status);`

Input List Type: fd

stm_r_di_containing_fd	**Query:** Data-items containing a given field **Purpose:** Returns the data-items (records or unions) in which the fields in the input list are defined **Syntax:** `stm_r_di_containing_fd (stm_list in_list, int *status);`

Input List Type: if

stm_r_di_contained_in_if	**Query:** Data-items contained in a given information-flow **Purpose:** Returns the data-items contained in information-flow from the input list (as defined in the **Consists of** field of the information-flow's form) **Syntax:** `stm_r_di_contained_in_if (stm_list in_list, int *status);`

Input List Type: mf

stm_r_di_flowing_through_mf	**Query:** Data-items flowing through a given m-flow-line **Purpose:** Returns the data-items actually flowing through m-flow-lines in the input list **Syntax:** `stm_r_di_flowing_through_mf (stm_list in_list, int *status);`
stm_r_di_labeling_mf	**Query:** Data-items labeling a given m-flow-line **Purpose:** Returns the data-items which label the m-flow-lines in the input list **Syntax:** `stm_r_di_labeling_mf (stm_list in_list, int *status);`

Data-Stores (ds)

This section documents the query functions that return a list of data-stores.

Input List Type: ac

stm_r_ds_contained_in_ac	**Query:** Data-stores contained in a given activity **Purpose:** Returns the data-stores contained directly in activities from the input list **Syntax:** `stm_r_ds_contained_in_ac (stm_list in_list, int *status);`
stm_r_ds_in_ac	**Query:** Data-stores in a given activity **Purpose:** Returns the data-stores contained in the activities from the input list **Syntax:** `stm_r_ds_in_ac (stm_list in_list, int *status);`

Input List Type: af

stm_r_ds_target_of_af	**Query:** Data-stores that are targets of a given a-flow-line **Purpose:** Returns the data-stores that are targets of a-flow-lines in the input list **Syntax:** `stm_r_ds_target_of_af (stm_list in_list, int *status);`

Input List Type: ch

stm_r_ds_def_or_unres_in_ch	**Query:** Data-stores defined or unresolved in a given chart **Purpose:** Returns the data-stores that are explicitly defined or unresolved in the charts of the input list **Syntax:** `stm_r_ds_def_or_unres_in_ch (stm_list in_list, int *status);`
stm_r_ds_defined_in_ch	**Query:** Data-stores defined in a given chart **Purpose:** Returns the data-stores that are explicitly defined in the charts of the input list **Syntax:** `stm_r_ds_defined_in_ch` (stm_list in_list, int *status);
stm_r_ds_unresolved_in_ch	**Query:** Data-stores unresolved in a given chart **Purpose:** Returns the data-stores that are unresolved in the charts of the input list **Syntax:** `stm_r_ds_unresolved_in_ch` (stm_list in_list, int *status);

Input List Type: ds

stm_r_ds_by_attributes_ds	**Query:** Data-stores by attributes **Purpose:** Returns the data-stores in the input list that match a given attribute name and value **Syntax** `stm_r_ds_by_attributes_ds (stm_list in_list, char* attr_name, char* attr_value, int *status);`
stm_r_ds_explicit_defined_ds	**Query:** Data-stores explicitly defined **Purpose:** Returns the data-stores of the input list that were explicitly defined **Syntax:** `stm_r_ds_explicit_defined_ds (stm_list in_list, int *status);`
stm_r_ds_is_occurrence_of_ds	**Query:** Data-store occurrences of a given data-store **Purpose:** Returns the data-stores for which the data-stores in the input list appear in the **Is Data-store** field of their form **Syntax:** `stm_r_ds_is_occurrence_of_ds (stm_list in_list, int *status);`
stm_r_ds_is_principal_of_ds	**Query:** Principal data-stores of a given data-store **Purpose:** Returns the data-stores for which the data-stores in the input list appear in the **Is Data-store** field of their form **Syntax:** `stm_r_ds_is_principal_of_ds (stm_list in_list, int *status);`
stm_r_ds_name_of_ds	**Query:** Data-store whose names match a given pattern **Purpose:** Returns all the data-stores whose names match the specified pattern **Syntax:** `stm_r_ds_name_of_ds (char* pattern, int *status);`
stm_r_ds_synonym_of_ds	**Query:** Data-store whose synonyms match a given pattern **Purpose:** Returns all the data-stores whose synonyms match the specified pattern **Syntax:** `stm_r_ds_synonym_of_ds (char* pattern, int *status);`

stm_r_ds_unresolved_ds	**Query:** Unresolved data-stores **Purpose:** Returns the unresolved data-stores in the input list **Syntax:** `stm_r_ds_unresolved_ds (stm_list in_list, int *status);`

Input List Type: md

stm_r_ds_resides_in_md	**Query:** Data-stores residing in a given module. **Purpose:** Returns the data-stores residing in modules from the input list. The module appears in the **Resides in Module** field of the data-store's form. **Syntax:** `stm_r_ds_resides_in_md (stm_list in_list, int *status);`

User-Defined Types (dt)

This section documents the query functions that return a list of data-types.

Input List Type: ch

stm_r_dt_def_or_unres_in_ch	**Query:** User-defined types defined or unresolved in a given chart **Purpose:** Returns the user-defined types that are explicitly defined or unresolved in the charts in the input list **Syntax:** `stm_r_dt_def_or_unres_in_ch (stm_list in_list, int *status);`
stm_r_dt_defined_in_ch	**Query:** User-defined types defined in a given chart **Purpose:** Returns the user-defined types that are explicitly defined in the charts in the input list **Syntax:** `stm_r_dt_defined_in_ch (stm_list in_list, int *status);`
stm_r_dt_unresolved_in_ch	**Query:** User-defined types unresolved in a given chart **Purpose:** Returns the user-defined types that are unresolved in the charts in the input list **Syntax:** `stm_r_dt_unresolved_in_ch (stm_list in_list, int *status);`

Input List Type: dt

stm_r_dt_array_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as array **Syntax:** `stm_r_dt_array_dt (stm_list in_list, int *status);`
stm_r_dt_array_missing_dt	**Query:** Arrays of user-defined type by subtype **Purpose:** Returns the arrays of user-defined types in the input list for which no type is defined **Syntax:** `stm_r_dt_array_missing_dt (stm_list in_list, int *status);`
stm_r_dt_bit_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as bit **Syntax:** `stm_r_dt_bit_dt (stm_list in_list, int *status);`
stm_r_dt_bit_queue_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as queue of bit **Syntax:** `stm_r_dt_bit_queue_dt (stm_list in_list, int *status);`
stm_r_dt_bits_array_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as array of bit array **Syntax:** `stm_r_dt_bits_array_dt (stm_list in_list, int *status);`
stm_r_dt_bits_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as bit array **Syntax:** `stm_r_dt_bits_dt (stm_list in_list, int *status);`

stm_r_dt_bits_queue_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as queue of bit array **Syntax:** `stm_r_dt_bits_queue_dt (stm_list in_list, int *status);`
stm_r_dt_by_attributes_dt	**Query:** User-defined types by attribute **Purpose:** Returns the user-defined types in the input list that match a given attribute and value **Syntax:** `stm_r_dt_by_attributes_dt (stm_list in_list, char* attr_name, char* attr_value, int *status);`
stm_r_dt_by_structure_type_dt	**Query:** None **Purpose:** Returns the user-defined types in the input list that have a given structure type (for example, single, array or queue) **Syntax:** `stm_r_dt_by_structure_type_dt (stm_list in_list, char structure_type, int *status);`
stm_r_dt_condition_array_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as array of condition **Syntax:** `stm_r_dt_condition_array_ dt (stm_list in_list, int *status);`
stm_r_dt_condition_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defend types in the input list that are defined as condition **Syntax:** `stm_r_dt_condition_dt (stm_list in_list, int *status);`
stm_r_dt_condition_queue_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as queue of condition **Syntax:** `stm_r_dt_condition_queue_ dt (stm_list in_list, int *status);`

stm_r_dt_enums_dt	**Query:** User-defined types defined as enumerated types **Purpose:** Returns the user-defined types in the input list that are defined as enumerated types **Syntax:** `stm_r_dt_enums_dt (stm_list in_list, int *status);`
stm_r_dt_explicit_defined_dt	**Query:** User-defined types explicitly defined **Purpose:** Returns the user-defined types in the input list that are explicitly defined **Syntax:** `stm_r_dt_explicit_defined_dt (stm_list in_list, int *status);`
stm_r_dt_integer_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as integer **Syntax:** `stm_r_dt_integer_dt (stm_list in_list, int *status);`
stm_r_dt_integer_array_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as array of integer **Syntax:** `stm_r_dt_integer_array_dt (stm_list in_list, int *status);`
stm_r_dt_integer_queue_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as queue of integer **Syntax:** `stm_r_dt_integer_queue_dt (stm_list in_list, int *status);`
stm_r_dt_missing_dt	**Query:** User-defined type by subtype **Purpose:** Returns the user-defined types in the input list for which no type is defined **Syntax:** `stm_r_dt_missing_dt (stm_list in_list, int *status);`
stm_r_dt_name_of_dt	**Query:** User-defined types whose names match a given pattern **Purpose:** Returns all user-defined types whose names match the specified pattern **Syntax:** `stm_r_dt_name_of_dt (char* pattern, int *status);`

stm_r_dt_queue_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as queue **Syntax:** `stm_r_dt_queue_dt (stm_list in_list, int *status);`
stm_r_dt_queue_missing_dt	**Query:** Queues of user-defined type by subtype **Purpose:** Returns the queues of user-defined types in the input list for which no type is defined **Syntax:** `stm_r_dt_queue_missing_dt (stm_list in_list, int *status);`
stm_r_dt_real_array_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as real **Syntax:** `stm_r_dt_real_array_dt (stm_list in_list, int *status);`
stm_r_dt_real_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as real **Syntax:** `stm_r_dt_real_dt (stm_list in_list, int *status);`
stm_r_dt_real_queue_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as queue of real **Syntax:** `stm_r_dt_real_queue_dt (stm_list in_list, int *status);`
stm_r_dt_record_array_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as array of record **Syntax:** `stm_r_dt_record_array_dt (stm_list in_list, int *status);`
stm_r_dt_record_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as record **Syntax:** `stm_r_dt_record_dt (stm_list in_list, int *status);`

stm_r_dt_single_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as single **Syntax:** `stm_r_dt_single_dt (stm_list in_list, int *status);`
stm_r_dt_string_array_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as array of string **Syntax:** `stm_r_dt_string_array_dt (stm_list in_list, int *status);`
stm_r_dt_string_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as string **Syntax:** `stm_r_dt_string_dt (stm_list in_list, int *status);`
stm_r_dt_string_queue_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as queue of string **Syntax:** `stm_r_dt_string_queue_dt (stm_list in_list, int *status);`
stm_r_dt_synonym_of_dt	**Query:** User-defined types whose synonyms match a given pattern **Purpose:** Returns all user-defined types whose synonyms match the specified pattern **Syntax**: `stm_r_dt_synonym_of_dt (char* pattern, int *status);`
stm_r_dt_union_dt	**Query:** User-defined type by subtype **Purpose:** Returns the user-defined types in the input list that are defined as union **Syntax:** `stm_r_dt_union_dt (stm_list in_list, int *status);`
stm_r_dt_union_array_dt	**Query:** User-defined type by subtype **Purpose:** Returns the user-defined types in the input list that are defined as array of union **Syntax:** `stm_r_dt_union_array_dt (stm_list in_list, int *status);`

stm_r_dt_unresolved_dt	**Query:** Unresolved user-defined types **Purpose:** Returns the unresolved user-defined types in the input list **Syntax:** `stm_r_dt_unresolved_dt (stm_list in_list, int *status);`
stm_r_dt_user_type_array_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as array of another user-defined type **Syntax:** `stm_r_dt_user_type_array_ dt (stm_list in_list, int *status);`
stm_r_dt_user_type_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as other user-defined type **Syntax:** `stm_r_dt_user_type_dt (stm_list in_list, int *status);`
stm_r_dt_user_type_queue_dt	**Query:** User-defined types by subtype **Purpose:** Returns the user-defined types in the input list that are defined as queue of another user-defined type **Syntax:** `stm_r_dt_user_type_queue_ dt (stm_list in_list, int *status);`

Input List Type: fd

stm_r_dt_containing_fd	**Query:** User-defined types containing a given field **Purpose:** Returns the user-defined types (records or unions), in which the fields in the input list are defined **Syntax:** `stm_r_dt_containing_fd (stm_list in_list, int *status);`

Events (ev)

This section documents the query functions that return a list of events.

Input List Type: af

stm_r_ev_flowing_through_af	**Query:** Events flowing through the specified a-flow-line **Purpose:** Returns the events actually flowing through a-flow-lines in the input list **Syntax:** `stm_r_ev_flowing_through_ af (stm_list in_list, int *status);`
stm_r_ev_labeling_af	**Query:** Events labeling a given a-flow-line **Purpose:** Returns the events that label the a-flow-lines in the input list **Syntax:** `stm_r_ev_labeling_af (stm_list in_list, int *status);`

Input List Type: ch

stm_r_ev_def_or_unres_in_ch	**Query:** Events defined or unresolved in a given chart **Purpose:** Returns the events that are explicitly defined or unresolved in the charts of the input list **Syntax:** `stm_r_ev_def_or_unres_in_ ch (stm_list in_list, int *status);`
stm_r_ev_defined_in_ch	**Query:** Events defined in a given chart **Purpose:** Returns the events that are explicitly defined in the charts of the input list **Syntax:** `stm_r_ev_defined_in_ch (stm_list in_list, int *status);`
stm_r_ev_unresolved_in_ch	**Query:** Events unresolved in a given chart **Purpose:** Returns the events that are unresolved in the charts of the input list **Syntax:** `stm_r_ev_unresolved_in_ch (stm_list in_list, int *status);`

Input List Type: ev

stm_r_ev_array_ev	**Query:** Events by subtype **Purpose:** Returns the events in the input list that are defined as array **Syntax:** `stm_r_ev_array_ev (stm_list in_list, int *status);`
stm_r_ev_by_attributes_ev	**Query:** Events by attributes **Purpose:** Returns the events in the input list that match the specified attribute name and value **Syntax:** `stm_r_ev_by_attributes_ev (stm_list in_list, char* attr_name, char* attr_value, int *status);`
stm_r_ev_by_structure_type_ev	**Query:** None **Purpose:** Returns the events in the input list that have the specified structure type (for example, single or array) **Syntax:** `stm_r_ev_by_structure_type_ev (stm_list in_list, char structure_type, int *status);`
stm_r_ev_callback_binding_ev	**Query:** Events with callback bindings **Purpose:** Returns the events in the input list that have callback bindings **Syntax:** `stm_r_ev_callback_binding_ev (stm_list in_list, int *status);`
stm_r_ev_explicit_defined_ev	**Query:** Events explicitly defined **Purpose:** Returns the events of the input list that were explicitly defined **Syntax:** `stm_r_ev_explicit_defined_ev (stm_list in_list, int *status);`
stm_r_ev_name_of_ev	**Query:** Events whose names match a given pattern **Purpose:** Returns all the events whose names match the specified pattern **Syntax:** `stm_r_ev_name_of_ev (char* pattern, int *status);`

stm_r_ev_single_ev	**Query:** Events by subtype **Purpose:** Returns the events in the input list that are defined as single **Syntax:** `stm_r_ev_single_ev (stm_list in_list, int *status);`
stm_r_ev_synonym_of_ev	**Query:** Events whose synonyms match a given pattern **Purpose:** Returns all the events whose synonyms match the specified pattern **Syntax:** `stm_r_ev_synonym_of_ev (char* pattern, int *status);`
stm_r_ev_unresolved_ev	**Query:** Unresolved events **Purpose:** Returns the unresolved events in the input list **Syntax:** `stm_r_ev_unresolved_ev (stm_list in_list, int *status);`

Input List Type: if

stm_r_ev_contained_in_if	**Query:** Events contained in a given information-flow **Purpose:** Returns the events contained in information-flows from the input list (events used in the **Consists of** field of the information-flow's form) **Syntax:** `stm_r_ev_contained_in_if (stm_list in_list, int *status);`

Input List Type: mf

stm_r_ev_flowing_through_mf	**Query:** Events flowing through a given m-flow-line **Purpose:** Returns the events actually flowing through m-flow-lines from the input list **Syntax:** `stm_r_ev_flowing_through_ f (stm_list in_list, int *status);`
stm_r_ev_labeling_mf	**Query:** Events labeling a given m-flow-line **Purpose:** Returns the events that label the m-flow-lines in the input list **Syntax:** `stm_r_ev_labeling_mf (stm_list in_list, int *status);`

Fields (fd)

This section documents the queries that return a list of fields.

Input List Type: ch

stm_r_fd_defined_in_ch	**Query:** Fields defined in a given chart **Purpose:** Returns the fields that are part of the structured data-items in the input list **Syntax:** `stm_r_fd_defined_in_ch (stm_list in_list, int *status);`

Input List Type: di

stm_r_fd_contained_in_di	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as array **Syntax:** `stm_r_fd_contained_in_di (stm_list in_list, int *status);`

Input List Type: dt

stm_r_fd_contained_in_dt	**Query:** Fields contained in user-defined type (UDT) **Purpose:** Returns the fields that are part of the structured UDTs in the input list **Syntax:** `stm_r_fd_contained_in_dt (stm_list in_list, int *status);`

Input List Type: fd

stm_r_fd_array_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as array **Syntax:** `stm_r_fd_array_fd (stm_list in_list, int *status);`
stm_r_fd_array_missing_fd	**Query:** Array of fields by subtype **Purpose:** Returns the array of fields in the input list for which no type is defined **Syntax:** `stm_r_fd_array_missing_fd (stm_list in_list, int *status);`
stm_r_fd_bit_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as bit **Syntax:** `stm_r_fd_bit_fd (stm_list in_list, int *status);`
stm_r_fd_bit_queue_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as queue of bit **Syntax:** `stm_r_fd_bit_queue_fd (stm_list in_list, int *status);`
stm_r_fd_bits_array_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as bit array **Syntax:** `stm_r_fd_bits_array_fd (stm_list in_list, int *status);`
stm_r_fd_bits_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as bit array **Syntax:** `stm_r_fd_bits_fd (stm_list in_list, int *status);`

stm_r_fd_bits_queue_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as queue of bit array **Syntax:** `stm_r_fd_bits_queue_fd (stm_list in_list, int *status);`
stm_r_fd_by_attributes_fd	**Query:** Fields by attribute **Purpose:** Returns the fields in the input list that match the specified attribute and value **Syntax**: `stm_r_fd_by_attributes_fd (stm_list in_list, char* attr_name, char* attr_value, int *status);`
stm_r_fd_by_structure_type_fd	**Query:** None **Purpose:** Returns the fields in the input list that have the specified structure type (for example, single or array) **Syntax:** `stm_r_fd_by_structure_type_fd (stm_list in_list, char structure_type, int *status);`
stm_r_fd_condition_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as condition **Syntax:** `stm_r_fd_condition_fd (stm_list in_list, int *status);`
stm_r_fd_condition_array_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as array of condition **Syntax:** `stm_r_fd_condition_array_ fd (stm_list in_list, int *status);`
stm_r_fd_condition_queue_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as queue of condition **Syntax:** `stm_r_fd_condition_ queue_fd (stm_list in_list, int *status);`
stm_r_fd_explicit_defined_fd	**Query:** Fields explicitly defined **Purpose:** Returns the fields in the input list that are explicitly defined **Syntax:** `stm_r_fd_explicit_defined_fd (stm_list in_list, int *status);`

stm_r_fd_integer_array_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as array of integer **Syntax:** `stm_r_fd_integer_array_fd (stm_list in_list, int *status);`
stm_r_fd_integer_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as integer **Syntax:** `stm_r_fd_integer_fd (stm_list in_list, int *status);`
stm_r_fd_integer_queue_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as queue of integer **Syntax:** `stm_r_fd_integer_queue_fd (stm_list in_list, int *status);`
stm_r_fd_missing_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list for which no type is defined **Syntax:** `stm_r_fd_missing_fd (stm_list in_list, int *status);`
stm_r_fd_name_of_fd	**Query:** Fields whose names match a given pattern **Purpose:** Returns all fields whose name matches the specified pattern **Syntax:** `stm_r_fd_name_of_fd (char* pattern, int *status);`
stm_r_fd_queue_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as queue **Syntax:** `stm_r_fd_queue_fd (stm_list in_list, int *status);`
stm_r_fd_queue_missing_fd	**Query:** Queues of field by subtype **Purpose:** Returns the queues of fields in the input list for which no type is defined **Syntax:** `stm_r_fd_queue_missing_fd (stm_list in_list, int *status);`

stm_r_fd_real_array_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as array of real **Syntax:** `stm_r_fd_real_array_fd (stm_list in_list, int *status);`
stm_r_fd_real_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as real **Syntax:** `stm_r_fd_real_fd (stm_list in_list, int *status);`
stm_r_fd_real_queue_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as queue of real **Syntax:** `stm_r_fd_real_queue_fd (stm_list in_list, int *status);`
stm_r_fd_single_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as single **Syntax:** `stm_r_fd_single_fd (stm_list in_list, int *status);`
stm_r_fd_string_array_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as array of string **Syntax:** `stm_r_fd_string_array_fd (stm_list in_list, int *status);`
stm_r_fd_string_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as string **Syntax:** `stm_r_fd_string_fd (stm_list in_list, int *status);`
stm_r_fd_string_queue_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as queue of string **Syntax:** `stm_r_fd_string_queue_fd (stm_list in_list, int *status);`

stm_r_fd_user_type_array_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as array of user-defined type **Syntax:** `stm_r_fd_user_type_array_ fd (stm_list in_list, int *status);`
stm_r_fd_user_type_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as user-defined type **Syntax:** `stm_r_fd_user_type_fd (stm_list in_list, int *status);`
stm_r_fd_user_type_queue_fd	**Query:** Fields by subtype **Purpose:** Returns the fields in the input list that are defined as queue of user-defined type **Syntax:** `stm_r_fd_user_type_queue_ fd (stm_list in_list, int *status);`

Input List Type: mx

stm_r_fd_contained_in_mx	**Query:** Fields contained in a given element **Purpose:** Returns the fields that are part of the structured elements (data-items and user-defined types) in the input list **Syntax:** `stm_r_fd_contained_in_mx (stm_list in_list, int *status);`

Functions (fn)

This section documents the queries that return a list of functions.

Input List Type: ch

stm_r_fn_name_of_fn	**Query:** Function names that match a given pattern **Purpose:** Returns all the functions whose names match the specified pattern **Syntax:** `stm_r_fn_name_of_fn (char* pattern, int *status);`
stm_r_fn_unresolved_in_ch	**Query:** Functions unresolved in a given chart **Purpose:** Returns the functions that are unresolved in the charts of the input list **Syntax**: `stm_r_fn_unresolved_in_ch (stm_list in_list, int *status);`

Information-Flows (if)

This section documents the queries that return a list of information-flows.

Input List Type: af

stm_r_if_basic_flowing_af	**Query:** Basic information-flows flowing through a given a-flow-line. **Purpose:** Returns information-flows that are not decomposed to other information items, and are flowing through a-flow-lines in the input list. **Syntax:** `stm_r_if_basic_flowing_af (stm_list in_list, int *status);`
stm_r_if_flowing_through_af	**Query:** Information-flows flowing through a given a-flow-line. **Purpose:** Returns the information-flows flowing through a-flow-lines in the input list. **Note:** This function returns the highest information-flows, as opposed to `stm_r_if_basic_ flowing_af`, which returns the lowest level. **Syntax:** `stm_r_if_flowing_through_ af (stm_list in_list, int *status);`
stm_r_if_labeling_af	**Query:** Information-flows labeling a given a-flow-line. **Purpose:** Returns the information-flows that label a-flow-lines in the input list. **Syntax:** `stm_r_if_labeling_af (stm_list in_list, int *status);`

Input List Type: ch

stm_r_if_def_or_unres_in_ch	**Query:** Information-flows defined or unresolved in a given chart **Purpose:** Returns the information-flows that are explicitly defined or unresolved in the charts of the input list **Syntax:** `stm_r_if_def_or_unres_in_ ch (stm_list in_list, int *status);`
stm_r_if_defined_in_ch	**Query:** Information-flows defined in a given chart **Purpose:** Returns the information-flows that are explicitly defined in the charts of the input list **Syntax:** `stm_r_if_defined_in_ch (stm_list in_list, int *status);`
stm_r_if_unresolved_in_ch	**Query:** Information-flows unresolved in a given chart **Purpose:** Returns the information-flows that are unresolved in the charts of the input list **Syntax:** `stm_r_if_unresolved_in_ch (stm_list in_list, int *status);`

Input List Type: co

stm_r_if_containing_co	**Query:** Information-flows containing a given condition **Purpose:** Returns the information-flows containing conditions from the input list (conditions appearing in the **Consists of** field of the information-flow's form) **Syntax:** `stm_r_if_containing_co (stm_list in_list, int *status);`

Input List Type: di

stm_r_if_containing_di	**Query:** Information-flows containing a given data-item **Purpose:** Returns the information-flows containing data-items from the input list (data-items appearing in the **Consists of** field of the information-flow's form) **Syntax:** `stm_r_if_containing_di (stm_list in_list, int *status);`

Input List Type: ev

stm_r_if_containing_ev	**Query:** Information-flows containing a given event **Purpose:** Returns the information-flows containing events from the input list (events appearing in the **Consists of** field of the information-flow's form) **Syntax:** `stm_r_if_containing_ev (stm_list in_list, int *status);`

Input List Type: if

stm_r_if_basic_if	**Query:** Basic information-flows **Purpose:** Returns the information-flows in the input list that are basic (those not defined using other information-flows) **Syntax:** `stm_r_if_basic_if (stm_list in_list, int *status);`
stm_r_if_by_attributes_if	**Query:** Information-flows by attributes **Purpose:** Returns the information-flows in the input list that match a particular attribute name and value **Syntax:** `stm_r_if_by_attributes_if (stm_list in_list, char* attr_name, char* attr_value, int *status);`
stm_r_if_contained_in_if	**Query:** Information-flows contained in a given information-flow **Purpose:** Returns the information-flows that are contained in information-flows from the input list (as defined in the **Consists of** field) **Syntax:** `stm_r_if_contained_in_if (stm_list in_list, int *status);`
stm_r_if_containing_if	**Query:** Information-flows containing a given information-flow **Purpose:** Returns the information-flows that contain information-flows from the input list (as defined in the **Consists of** field) **Syntax:** `stm_r_if_containing_if (stm_list in_list, int *status);`
stm_r_if_explicit_defined_if	**Query:** Information-flows explicitly defined **Purpose:** Returns the information-flows of the input list that were explicitly defined **Syntax:** `stm_r_if_explicit_defined_if (stm_list in_list, int *status);`

stm_r_if_name_of_if	**Query:** Information-flow names that match a given pattern **Purpose:** Returns all the information-flows whose names match the specified pattern **Syntax:** `stm_r_if_name_of_if (char* pattern, int *status);`
stm_r_if_synonym_of_if	**Query:** Information-flow synonyms that match a given pattern **Purpose:** Returns all the information-flows whose synonyms match the specified pattern **Syntax:** `stm_r_if_synonym_of_if (char* pattern, int *status);`
stm_r_if_unresolved_if	**Query:** Unresolved information-flows **Purpose:** Returns the unresolved information-flows in the input list **Syntax:** `stm_r_if_unresolved_if (stm_list in_list, int *status);`

Input List Type: mf

stm_r_if_basic_flowing_mf	**Query:** Basic information-flows flowing through a given m-flow-line. **Purpose:** Returns the information-flows that are not decomposed to other information items and are flowing through m-flow-lines in the input list. **Syntax:** `stm_r_if_basic_flowing_mf (stm_list in_list, int *status);`
stm_r_if_flowing_through_mf	**Query:** Information-flows flowing through a given m-flow-line. **Purpose:** Returns the information-flow flowing through m-flow-lines in the input list. **Note:** This function returns the highest information-flows as opposed to `stm_r_if_basic_flowing_mf`, which returns the lowest level. **Syntax:** `stm_r_if_flowing_through_ mf (stm_list in_list, int *status);`
stm_r_if_labeling_mf	**Query:** Information-flows labeling a given m-flow-line. **Purpose:** Returns the information-flow labeling m-flow-lines in the input list. **Syntax:** `stm_r_if_labeling_mf (stm_list in_list, int *status);`

M-Flow-Lines (bf, lmf, mf)

This section documents the queries that return a list of m-flow-lines. The types are as follows:

- `bf`—Basic m-flow-lines
- `lmf`—Local m-flow-lines
- `mf`—Global (compound) m-flow-lines

Output List Type: bf

Input List Type: co

stm_r_bf_within_flows_co	**Query:** A-flow-lines through which a given condition flows **Purpose:** Returns the a-flow-lines through which conditions from the input list actually flow **Syntax:** `stm_r_af_within_flows_co (stm_list in_list, int *status);`
stm_r_bf_within_labels_co	**Query:** A-flow-lines labeled by a given condition **Purpose:** Returns the a-flow-lines labeled with conditions in the input list **Syntax:** `stm_r_af_within_labels_co (stm_list in_list, int *status);`

Input List Type: di

stm_r_bf_within_flows_di	**Query:** A-flow-lines through which a given data-item flows **Purpose:** Returns the a-flow-lines through which data-items from the input list actually flow **Syntax:** `stm_r_af_within_flows_di (stm_list in_list, int *status);`
stm_r_bf_within_labels_di	**Query:** A-flow-lines labeled by a given data-item **Purpose:** Returns the a-flow-lines labeled with data-items in the input list **Syntax:** `stm_r_af_within_labels_di (stm_list in_list, int *status);`

Input List Type: ev

stm_r_bf_within_flows_ev	**Query:** A-flow-lines through which a given event flows **Purpose:** Returns the a-flow-lines through which events from the input list actually flow **Syntax:** `stm_r_bf_within_flows_ev (stm_list in_list, int *status);`
stm_r_bf_within_labels_ev	**Query:** A-flow-lines labeled by a given event **Purpose:** Returns the a-flow-lines labeled with events in the input list **Syntax:** `stm_r_bf_within_labels_ev (stm_list in_list, int *status);`

Input List Type: if

stm_r_bf_within_flows_if	**Query:** A-flow-lines through which a given information-flow flows **Purpose:** Returns the a-flow-lines through which information-flows from the input list actually flow **Syntax:** `stm_r_bf_within_flows_if (stm_list in_list, int *status);`
stm_r_bf_within_labels_if	**Query:** A-flow-lines labeled by a given information-flow **Purpose:** Returns the a-flow-lines labeled with information-flows in the input list **Syntax:** `stm_r_bf_within_labels_if (stm_list in_list, int *status);`

Input List Type: mx

stm_r_bf_from_source_mx	**Query:** A-flow-lines whose source is a given element **Purpose:** Returns basic a-flow-lines that originate at elements in the input list **Syntax:** `stm_r_af_from_source_mx (stm_list in_list, int *status);`
stm_r_bf_to_target_mx	**Query:** A-flow-lines whose target is a given element **Purpose:** Returns the basic a-flow-lines whose target is an element from the input list **Syntax:** `stm_r_bf_to_target_mx (stm_list in_list, int *status);`
stm_r_bf_within_flows_mx	**Query:** A-flow-lines through which a given element flows **Purpose:** Returns the a-flow-lines through which elements from the input list actually flow **Syntax:** `stm_r_af_within_flows_mx (stm_list in_list, int *status);`
stm_r_bf_within_labels_mx	**Query:** A-flow-lines labeled by a given elements **Purpose:** Returns the a-flow-lines labeled with elements in the input list **Syntax:** `stm_r_af_within_labels_mx (stm_list in_list, int *status);`

Output List Type: lmf

Input List Type: md

stm_r_lmf_from_source_md	**Query:** M-flow-lines whose source is a given module within chart **Purpose:** Returns the local compound m-flow-lines (those within charts) whose source is a module from the input list **Syntax:** `stm_r_lmf_from_source_md (stm_list in_list, int *status);`
stm_r_lmf_input_to_md	**Query:** M-flow-lines input to a given module within the chart **Purpose:** Returns all the local compound m-flow-lines that originate outside and terminate at (or inside) modules in the input list **Syntax:** `stm_r_lmf_input_to_md (stm_list in_list, int *status);`
stm_r_lmf_output_from_md	**Query:** M-flow-lines output from a given module within that chart **Purpose:** Returns all the local compound m-flow-lines that originate at (or inside) and terminate outside modules in the input list **Syntax:** `stm_r_lmf_output_from_md (stm_list in_list, int *status);`
stm_r_lmf_to_target_md	**Query:** M-flow-lines whose target is a given module **Purpose:** Returns the local m-flow-lines whose target is a module from the input list **Syntax:** `stm_r_lmf_to_target_md (stm_list in_list, int *status);`

Input List Type: mf

stm_r_lmf_contained_in_mf	**Query:** None **Purpose:** Returns the local m-flow-lines that contain the global m-flow-lines in the input list **Syntax:** `stm_r_lmf_contained_in_mf (stm_list l, int *status);`

Output List Type: mf

Input List Type: co

stm_r_mf_within_flows_co	**Query:** M-flow-lines through which a given condition flows **Purpose:** Returns the m-flow-lines through which conditions from the input list actually flow **Syntax:** `stm_r_mf_within_flows_co (stm_list in_list, int *status);`
stm_r_mf_within_labels_co	**Query:** M-flow-lines labeled by a given condition **Purpose:** Returns the m-flow-lines that are labeled by the conditions in the input list **Syntax:** `stm_r_mf_within_labels_co (stm_list in_list, int *status);`

Input List Type: di

stm_r_mf_within_flows_di	**Query:** M-flow-lines through which a given data-item flows **Purpose:** Returns the m-flow-lines through which data-items from the input list actually flow **Syntax:** `stm_r_mf_within_flows_di (stm_list in_list, int *status);`
stm_r_mf_within_labels_di	**Query:** M-flow-lines labeled by a given data-item **Purpose:** Returns the m-flow-lines that are labeled by the data-items in the input list **Syntax:** `stm_r_mf_within_labels_di (stm_list in_list, int *status);`

Input List Type: ev

stm_r_mf_within_flows_ev	**Query:** M-flow-lines through which a given event flows **Purpose:** Returns the m-flow-lines through which events from the input list actually flow **Syntax:** `stm_r_mf_within_flows_ev (stm_list in_list, int *status);`
stm_r_mf_within_labels_ev	**Query:** M-flow-lines labeled by a given event **Purpose:** Returns the m-flow-lines that are labeled by the events in the input list **Syntax:** `stm_r_mf_within_labels_ev (stm_list in_list, int *status);`

Input List Type: if

stm_r_mf_within_flows_if	**Query:** M-flow-lines through which a given information-flow flows **Purpose:** Returns the m-flow-lines through which information-flows from the input list actually flow **Syntax:** `stm_r_mf_within_flows_if (stm_list in_list, int *status);`
stm_r_mf_within_labels_if	**Query:** M-flow-lines labeled with a given information-flow **Purpose:** Returns the m-flow-lines that are labeled with information-flows in the input list **Syntax:** `stm_r_mf_within_labels_if (stm_list in_list, int *status);`

Input List Type: lmf

stm_r_mf_containing_lm	**Query:** None **Purpose:** Returns the global m-flow-lines (which might spread over several charts) that contain the local m-flow-lines (those within charts) in the input list **Syntax:** `stm_r_mf_containing_lmf (stm_list l, int *status);`
stm_r_mf_containing_lmf	**Query:** None **Purpose:** Returns the global m-flow-lines (which might spread over several charts) that contain the local m-flow-lines (those within charts) in the input list **Syntax:** `stm_r_mf_containing_lmf (stm_list l, int *status);`

Input List Type: md

stm_r_mf_from_source_md	**Query:** M-flow-lines whose source is a given module **Purpose:** Returns the global compound m-flow-lines (those that might spread over several charts) whose source is a module from the input list **Syntax:** `stm_r_mf_from_source_md (stm_list in_list, int *status));`
stm_r_mf_input_to_md	**Query:** M-flow-lines input to a given module **Purpose:** Returns all the global compound m-flow-lines that originate outside and terminate at (or inside) modules in the input list **Syntax:** `stm_r_mf_input_to_md (stm_list in_list, int *status);`
stm_r_mf_output_from_md	**Query:** M-flow-lines output from a given module **Purpose:** Returns all the global compound m-flow-lines that originate at (or inside) and terminate outside modules in the input list **Syntax:** `stm_r_mf_output_from_md (stm_list in_list, int *status);`

stm_r_mf_to_target_md	**Query:** M-flow-lines whose target is a given module **Purpose:** Returns the global compound m-flow-lines whose target is a module from the input list **Syntax:** `stm_r_mf_to_target_md (stm_list in_list, int *status);`
stm_r_lmf_to_target_md	**Query:** M-flow-lines whose target is a given module **Purpose:** Returns the local compound m-flow-lines whose target is a module from the input list **Syntax:** `stm_r_lmf_to_target_md (stm_list in_list, int *status);`

Input List Type: mx

stm_r_mf_within_flows_mx	**Query:** M-flow-lines through which a given element flows **Purpose:** Returns the m-flow-lines through which elements from the input list actually flow **Syntax:** `stm_r_mf_within_flows_mx (stm_list in_list, int *status);`
stm_r_mf_within_labels_mx	**Query:** M-flow-lines that are labeled by a given information-flow **Purpose:** Returns the m-flow-lines that are labeled with elements in the input list **Syntax:** `stm_r_mf_within_labels_mx (stm_list in_list, int *status);`

Modules (md)

This section documents the queries that return a list of modules.

Input List Type: ac

stm_r_md_carrying_out_ac	**Query:** Modules carrying out a given activity. **Purpose:** Returns the modules carrying out activities in the input list. The modules appear in the **Implemented by Module** field of an activity's form. **Syntax:** `stm_r_md_carrying_out_ac (stm_list in_list, int *status);`

Input List Type: ch

stm_r_md_def_or_unres_in_ch	**Query:** Modules defined or unresolved in a given chart **Purpose:** Returns the modules that are explicitly defined or unresolved in the charts of the input list **Syntax:** `stm_r_md_def_or_unres_in ch (stm_list in_list, int *status);`
stm_r_md_defined_in_ch	**Query:** Modules defined in a given chart **Purpose:** Returns the modules that are explicitly defined in the charts of the input list **Syntax:** `stm_r_md_defined_in_ch (stm_list in_list, int *status);`
stm_r_md_described_by_ch	**Query:** Modules described by a given activity-chart **Purpose:** Returns the modules described by activity-charts in the input list **Syntax:** `stm_r_md_described_by_ch (stm_list in_list, int *status);`

stm_r_md_instance_of_ch	**Query:** Modules instance of a given chart **Purpose:** Returns the instance modules defined by the charts in the input list **Syntax:** `stm_r_md_instance_of_ch (stm_list in_list, int *status);`
stm_r_md_root_in_ch	**Query:** Root modules of a given chart **Purpose:** Returns the internally defined modules (of type `diagram`) attached to the charts in the input list **Syntax:** `stm_r_md_root_in_ch (stm_list in_list, int *status);`
stm_r_md_top_level_in_ch	**Query:** Top-level modules of a given chart **Purpose:** Returns the top level modules (not contained in any box) of the charts in the input list **Syntax:** `stm_r_md_top_level_in_ch (stm_list in_list, int *status);`
stm_r_md_unresolved_in_ch	**Query:** Modules unresolved in a given chart **Purpose:** Returns the modules that are unresolved in the charts of the input list **Syntax:** `stm_r_md_unresolved_in_ch (stm_list in_list, int *status);`

Input List Type: ds

stm_r_md_contains_ds	**Query:** Modules in which a given data-store resides. **Purpose:** Returns the modules in which data-stores from the input list resides. The modules appear in the **Resides in Module** field of a data-store's form. **Syntax:** `stm_r_md_contains_ds (stm_list in_list, int *status);`

Input List Type: md

stm_r_md_basic_md	**Query:** Basic modules **Purpose:** Returns the modules in the input list that are basic modules (those that have no descendants) **Syntax:** `stm_r_md_basic_md (stm_list in_list, int *status);`
stm_r_md_bus_md	**Query:** Bus modules **Purpose:** Returns the modules in the input list that are bus modules **Syntax:** `stm_r_md_bus_md (stm_list in_list, int *status);`
stm_r_md_by_attributes_md	**Query:** Modules by attributes **Purpose:** Returns the modules in the input list that match a particular attribute name and value **Syntax:** `stm_r_md_by_attributes_md (stm_list in_list, char* attr_name, char* attr_value, int *status);`
stm_r_md_control_md	**Query:** Control modules **Purpose:** Returns the modules in the input list that are control modules **Syntax:** `stm_r_md_control_md (stm_list in_list, int *status);`
stm_r_md_def_of_instance_md	**Query:** Definition modules of a given module **Purpose:** Returns the definition modules (top level modules in a definition chart) for instances in the input list **Syntax:** `stm_r_md_def_of_instance_ md (stm_list in_list, int *status);`
stm_r_md_defined_environment_md	**Query:** Environment modules **Purpose:** Returns the modules from the input list that were defined as environment modules **Syntax:** `stm_r_md_defined_ environment_md (stm_list in_list, int *status);`

stm_r_md_environment_md	**Query:** Environment modules **Purpose:** Returns the modules in the input list that are environment modules **Syntax:** `stm_r_md_environment_md (stm_list in_list, int *status);`
stm_r_md_explicit_defined_md	**Query:** Modules explicitly defined **Purpose:** Returns the modules of the input list that were explicitly defined **Syntax:** `stm_r_md_explicit_defined_md (stm_list in_list, int *status);`
stm_r_md_external_md	**Query:** External modules **Purpose:** Returns the modules in the input list that are external **Syntax:** `stm_r_md_external_md (stm_list in_list, int *status);`
stm_r_md_generic_instance_md	**Query:** Generic instance modules **Purpose:** Returns the modules in the input list that are instances of generic charts **Syntax:** `stm_r_md_generic_instance_md (stm_list in_list, int *status);`
stm_r_md_instance_md	**Query:** Instance modules **Purpose:** Returns the instance modules from the modules in the input list **Syntax:** `stm_r_md_instance_md (stm_list in_list, int *status);`
stm_r_md_instance_of_def_m	**Query:** Instance modules of a given definition module **Purpose:** Returns the instance modules for definition modules (top-level modules in a definition chart) in the input list **Syntax:** `stm_r_md_instance_of_def_ md (stm_list in_list, int *status);`
stm_r_md_library_md	**Query:** Library modules **Purpose:** Returns the modules from the input list that are library modules **Syntax:** `stm_r_md_library_md (stm_list in_list, int *status);`

stm_r_md_logical_desc_of_md	**Query:** Logical descendants of a given module **Purpose:** Returns the logical descendants of the modules in the input list, taking into account the translation of instances to their definition charts **Syntax:** `stm_r_md_logical_ desc_of_md (stm_list in_list, int *status);`
stm_r_md_logical_parent_of_md	**Query:** Logical parent modules of a given module **Purpose:** Returns the logical parent modules of the modules in the input list, taking into account the translation of instances to their definition charts **Syntax:** `stm_r_md_logical_parent_of_md (stm_list in_list, int *status);`
stm_r_md_logical_sub_of_md	**Query:** Logical submodules of a given module **Purpose:** Returns the logical submodules of the modules in the input list, taking into account the translation of instances to their definition charts **Syntax:** `stm_r_md_logical_sub_of_md (stm_list in_list, int *status);`
stm_r_md_name_of_md	**Query:** Modules whose names match a given pattern **Purpose:** Returns all the modules whose names match the specified pattern **Syntax:** `stm_r_md_name_of_md (char* pattern, int *status);`
stm_r_md_offpage_instance_md	**Query:** Offpage instance modules **Purpose:** Returns the modules in the input list that are instances of offpage charts **Syntax:** `stm_r_md_offpage_instance_md (stm_list in_list, int *status);`
stm_r_md_physical_desc_of_md	**Query:** Physical descendants of a given module **Purpose:** Returns the physical descendants (those within the same chart) for the modules in the input list **Syntax:** `stm_r_md_physical_desc_of_md (stm_list in_list, int *status);`

stm_r_md_physical_parent_of_md	**Query:** Physical parent modules of a given module **Purpose:** Returns the physical parent modules (those within the same chart) for the modules in the input list **Syntax:** `stm_r_md_physical_parent_ of_md (stm_list in_list, int *status);`
stm_r_md_physical_sub_of_md	**Query:** Physical submodules of a given module **Purpose:** Returns the physical submodules (those within the same chart) for the modules in the input list **Syntax:** `stm_r_md_physical_sub_of_ md (stm_list in_list, int *status);`
stm_r_md_regular_md	**Query:** Regular modules **Purpose:** Returns the modules from the input list that are regular modules (not environment or storage) **Syntax:** `stm_r_md_regular_md (stm_list in_list, int *status);`
stm_r_md_resolved_to_ext_md	**Query:** Modules resolved to a given external module **Purpose:** Returns the modules (internal, external, or environment) to which the external modules in the input list are resolved **Syntax:** `stm_r_md_resolved_to_ext_ md (stm_list in_list, int *status);`
stm_r_md_storage_md	**Query:** Storage modules **Purpose:** Returns the modules from the input list that are storage modules **Syntax:** `stm_r_md_storage_md (stm_list in_list, int *status);`
stm_r_md_synonym_of_md	**Query:** Modules whose synonyms match a given pattern **Purpose:** Returns all the modules whose synonyms match the specified pattern **Syntax:** `stm_r_md_synonym_of_md (char* pattern, int *status);`
stm_r_md_unresolved_md	**Query:** Unresolved modules **Purpose:** Returns the unresolved modules in the input list **Syntax:** `stm_r_md_unresolved_md (stm_list in_list, int *status);`

Input List Type: mf

stm_r_md_source_of_mf	**Query:** Modules that are sources of a given m-flow-line **Purpose:** Returns the modules that are sources of m-flow-lines from the input list **Syntax:** `stm_r_md_source_of_mf (stm_list in_list, int *status);`
stm_r_md_target_of_mf	**Query:** Modules that are targets of a given m-flow-line **Purpose:** Returns the modules that are targets of m-flow-lines from the input list **Syntax:** `stm_r_md_target_of_mf (stm_list in_list, int *status);`

Input List Type: router

stm_r_md_contains_router	**Query:** Modules in which a given router resides. **Purpose:** Returns the modules in which routers from the input list resides. The modules appear in the **Resides in Module** field of a router's form. **Syntax:** `stm_r_md_contains_router (stm_list in_list, int *status);`

Mixed (mx)

This section documents the queries that return a list of elements.

Input List Type: af

stm_r_mx_flowing_through_af	**Query:** Elements flowing through a given a-flow-line **Purpose:** Returns the information elements (conditions, events, data-items, and basic information-flows) that actually flow through the a-flow-lines in the input list **Syntax:** `stm_r_mx_flowing_through af (stm_list in_list, int *status);`
stm_r_mx_labeling_af	**Query:** Elements labeling a given a-flow-line **Purpose:** Returns the elements that label a-flow-lines in the input list **Syntax:** `stm_r_mx_labeling_af (stm_list in_list, int *status);`
stm_r_mx_source_of_af	**Query:** Elements that are sources of a given a-flow-line **Purpose:** Returns the elements (activities and data-stores) that are sources of a-flow-lines in the input list **Syntax:** `stm_r_mx_source_of_af (stm_list in_list, int *status);`
stm_r_mx_target_of_af	**Query:** Elements that are targets of a given a-flow-line **Purpose:** Returns the elements (activities and data-stores) that are sources of a-flow-lines in the input list **Syntax:** `stm_r_mx_target_of_af (stm_list in_list, int *status);`

Input List Type: ac

stm_r_mx_affected_by_ac	**Query:** Elements affected by a given activity. **Purpose:** Returns the elements (data-items, conditions, and events) affected (modified or generated) by activities, in mini-specs, and combinational assignments in the input list. **Syntax:** `stm_r_mx_affected_by_ ac (stm_list in_list, int *status);`
stm_r_mx_influence_ac	**Query:** Elements that are referenced by or influence a given activity. **Purpose:** Returns the elements used in all levels by the activities in the input list. This includes all logical descendant activities, a-flow-lines that enter or exit these activities, elements that appear in the various fields of these activities, and in the labels of the flow-lines and their components. **Syntax:** `stm_r_mx_influence_ac (stm_list in_list, int *status);`
stm_r_mx_influenced_by_ac	**Query:** Elements that refer to, or are influenced by, a given activity. **Purpose:** Returns the elements that directly or indirectly use the activities in the input list. This query identifies all the elements, in all levels, that refer to or affect the input activities. **Syntax:** `stm_r_mx_influenced_by_ac (stm_list in_list, int *status);`
stm_r_mx_refer_to_ac	**Query:** Elements that refer to a given activity. **Purpose:** Returns the elements that directly refer to activities in the input list. This query identifies where input activities are used. **Syntax:** `stm_r_mx_refer_to_ac (stm_list in_list, int *status);`

stm_r_mx_referenced_by_ac	**Query:** Elements that are referenced by a given activity. **Purpose:** Returns the elements that appear in the activities of the input list. This includes all physical descendant activities, a-flow-lines that enter or exit these activities, elements that appear in the various fields of these activities, and in the labels of the flow-lines. **Syntax:** `stm_r_mx_referenced_by_ac (stm_list in_list, int *status);`
stm_r_mx_resolved_to_ext_ac	**Query:** Elements resolved to a given external activity. **Purpose:** Returns the activities and modules (internal, external, or environment) to which the external activities in the input list are resolved. **Syntax:** `stm_r_mx_resolved_to_ext_ ac (stm_list in_list, int *status);`
stm_r_mx_used_by_ac	**Query:** Elements used by a given activity. **Purpose:** Returns the elements (data-items, conditions, and events) used (evaluated) by activities in mini-specs and combinational assignments in the input list. **Syntax:** `stm_r_mx_used_by_ac (stm_list in_list, int *status);`

Input List Type: an

stm_r_mx_in_definition_of_an	**Query:** Elements appearing in the definition of a given action. **Purpose:** Returns the elements that appear in the **Definition** field of actions (in the action's form) in the input list. This query identifies the elements that are used directly by the actions in the input list. **Syntax:** `stm_r_mx_in_definition_of_an (stm_list in_list, int *status);`
stm_r_mx_influence_value_of_an	**Query:** Elements that influence the value of a given action. **Purpose:** Returns the elements that appear in the **Definition** field of actions in the input list, and those that appear in the definitions of these elements (for all levels). This query identifies the elements that are used directly or indirectly by the actions in the input list. **Syntax:** `stm_r_mx_influence_value_ of_an (stm_list in_list, int *status);`
stm_r_mx_influenced_by_an	**Query:** Elements that refer to, or are influenced by, a given action. **Purpose:** Returns the elements that directly or indirectly use the actions in the input list. This query identifies all the elements, in all levels, that use the input actions. **Syntax:** `stm_r_mx_influence_value_ of_an (stm_list in_list, int *status);`
stm_r_mx_refer_to_an	**Query:** Elements that refer to a given action. **Purpose:** Returns the elements that directly use the actions in the input list. This query identifies where the input actions appear. **Syntax:** `stm_r_mx_influence_value_ of_an (stm_list in_list, int *status);`

Input List Type: ba

stm_r_mx_source_of_ba	**Query:** Elements affected by a given activity. **Purpose:** Returns the elements that are sources of basic a-flow-lines in the input list. **Syntax:** `stm_r_mx_source_of_ ba (stm_list in_list, int *status);`
stm_r_mx_target_of_ba	**Query:** Elements affected by a given activity. **Purpose:** Returns the elements that are targets of basic a-flow-lines in the input list. **Syntax:** `stm_r_mx_influence_value_ of_an (stm_list in_list, int *status);`

Input List Type: bm

stm_r_mx_source_of_bm	**Query:** Elements affected by a given activity. **Purpose:** Returns the elements that are sources of basic m-flow-lines in the input list **Syntax:** `stm_r_mx_source_of_ bm (stm_list in_list, int *status);`
stm_r_mx_target_of_bm	**Query:** Elements affected by a given activity. **Purpose:** Returns the elements that are targets of basic m-flow-lines in the input list **Syntax:** `stm_r_mx_target_of_ bm (stm_list in_list, int *status);`

Input List Type: bt

stm_r_mx_source_of_bt	**Query:** Elements affected by a given activity. **Purpose:** Returns the elements that are sources of basic transitions in the input list **Syntax:** `stm_r_mx_source_of_ bt (stm_list in_list, int *status);`
stm_r_mx_target_of_bt	**Query:** Elements affected by a given activity. **Purpose:** Returns the elements that are targets of basic transitions in the input list **Syntax:** `stm_r_mx_target_of_ bt (stm_list in_list, int *status);`

Input List Type: ch

stm_r_mx_constant_parameter_ch	**Query:** Constant parameters in a given chart **Purpose:** Returns the constant formal parameters of generic charts and components in the input list **Syntax:** `stm_r_mx_constant_ parameter_ch (stm_list in_list, int *status);`
stm_r_mx_def_or_unres_in_ch	**Query:** Elements that are defined or unresolved in a given chart **Purpose:** Returns the elements that are explicitly defined or unresolved in the charts of the input list **Syntax:** `stm_r_mx_def_or_unres_in_ ch (stm_list in_list, int *status);`
stm_r_mx_defined_in_ch	**Query:** Elements that are defined in a given chart **Purpose:** Returns the elements that are explicitly defined in the charts of the input list **Syntax:** `stm_r_mx_defined_in_ch (stm_list in_list, int *status);`

stm_r_mx_in_parameter_ch	**Query:** In parameters in a given chart **Purpose:** Returns the formal in parameters of generic charts and components in the input list **Syntax:** `stm_r_mx_in_parameter_ch (stm_list in_list, int *status);`
stm_r_mx_influence_value_of_ch	**Query:** Elements referenced or influenced by a given chart **Purpose:** Returns the elements that are used directly or indirectly (referenced or affected) by the charts of the input list **Syntax:** `stm_r_mx_influence_value_ of_ch (stm_list in_list, int *status);`
stm_r_mx_inout_parameter_ch	**Query:** Inout parameters in a given chart **Purpose:** Returns the formal inout parameters of generic charts and components in the input list **Syntax:** `stm_r_mx_inout_parameter_ ch (stm_list in_list, int *status);`
stm_r_mx_instance_of_ch	**Query:** Element instance of a given chart **Purpose:** Returns the element instances defined by the charts in the input list **Syntax:** `stm_r_mx_instance_of_ch (stm_list in_list, int *status);`
stm_r_mx_out_parameter_ch	**Query:** Out parameters in a given chart **Purpose:** Returns the formal out parameters of generic charts and components in the input list **Syntax:** `stm_r_mx_out_parameter_ch (stm_list in_list, int *status);`
stm_r_mx_parameter_of_ch	**Query:** Parameters in a given chart **Purpose:** Returns the formal parameters of generic charts and components in the input list **Syntax:** `stm_r_mx_parameter_of_ch (stm_list in_list, int *status);`

stm_r_mx_referenced_by_ch	**Query:** Elements referenced by a given chart **Purpose:** Returns the elements that appear in the charts of the input list **Syntax:** `stm_r_mx_referenced_by_ch (stm_list in_list, int *status);`
stm_r_mx_root_in_ch	**Query:** Root elements of a given chart **Purpose:** Returns the internally defined elements (of type diagram) attached to the charts in the input list **Syntax:** `stm_r_mx_root_in_ch (stm_list in_list, int *status);`
stm_r_mx_text_def_unres_in_ch	**Query:** Textual elements defined or unresolved in a given chart **Purpose:** Returns the textual elements that are explicitly defined or unresolved in the charts of the input list **Syntax:** `stm_r_mx_text_def_unres_in_ch (stm_list in_list, int *status);`
stm_r_mx_text_unresolved_in_ch	**Query:** Textual elements unresolved in a given chart **Purpose:** Returns the textual elements that are unresolved in the charts of the input list **Syntax:** `stm_r_mx_text_def_unres_ in_ch (stm_list in_list, int *status);`
stm_r_mx_textual_defined_in_ch	**Query:** Textual elements that are defined in a given chart **Purpose:** Returns the textual elements that are explicitly defined in the charts of the input list **Syntax:** `stm_r_mx_textual_defined_ in_ch (stm_list in_list, int *status);`
stm_r_mx_unresolved_in_ch	**Query:** Elements unresolved in a given chart **Purpose:** Returns elements that are unresolved in the charts of the input list **Syntax:** `stm_r_mx_unresolved_in_ch (stm_list in_list, int *status);`

Input List Type: co

stm_r_mx_in_definition_of_co	**Query:** Elements appearing in the definition of a given condition. **Purpose:** Returns the elements that appear in the **Definition** field of conditions (in the condition's form) in the input list. This query identifies the elements that are used directly by the conditions in the input list. **Syntax:** `stm_r_mx_in_definition_of_co (stm_list in_list, int *status);`
stm_r_mx_influence_value_of_co	**Query:** Elements that influence the value of a given condition. **Purpose:** Returns the elements that appear in the **Definition** field of the conditions in the input list, and those that appear in the definitions of these elements (for all levels). This query identifies the elements that directly or indirectly influence the conditions in the input list. **Syntax:** `stm_r_mx_influence_value_ of_co (stm_list in_list, int *status);`
stm_r_mx_influenced_by_co	**Query:** Elements that refer to, or are influenced by, a given condition. **Purpose:** Returns the elements that directly or indirectly use the conditions in the input list. **Syntax:** `stm_r_mx_influenced_by_co (stm_list in_list, int *status);`
stm_r_mx_refer_to_co	**Query:** Elements that refer to a given condition. **Purpose:** Returns the elements that directly use the conditions in the input list. This query identifies where the input conditions appear. **Syntax:** `stm_r_mx_refer_to_co (stm_list in_list, int *status);`

Input List Type: di

stm_r_mx_in_definition_of_di	**Query:** Elements appearing in the definition of a given data-item. **Purpose:** Returns the elements that appear in the **Definition** and the **Consists of** fields of data-items (in the data-item's form) in the input list. This query identifies the elements that are directly used by the data-items in the input list. **Syntax:** `stm_r_mx_in_definition_of_di (stm_list in_list, int *status);`
stm_r_mx_influence_value_of_di	**Query:** Elements that influence the value of a given data-item. **Purpose:** Returns the elements that appear in the **Definition** and **Consists** of fields of data-items in the input list, and those that appear in the fields of these elements (for all levels). This query identifies the elements that directly or indirectly influence the data-items in the input list. **Syntax:** `stm_r_mx_influence_value_ of_dt (stm_list in_list, int *status);`
stm_r_mx_influenced_by_di	**Query:** Elements that refer to, or are influenced by, a given data-item. **Purpose:** Returns the elements that directly or indirectly use the data-items in the input list. This query identifies all the elements, in all levels, that refer to or affect the input data-items. **Syntax:** `stm_r_mx_influenced_by_di (stm_list in_list, int *status);`
stm_r_mx_refer_to_di	**Query:** Elements that refer to the specified data-item. **Purpose:** Returns the elements that directly use the data-items in the input list. This query identifies where the input data-items appear. **Syntax:** `stm_r_mx_refer_to_di (stm_list in_list, int *status);`

Input List Type: ds

stm_r_mx_refer_to_ds	**Query:** Elements that refer to a given data-store **Purpose:** Returns the elements directly affected by data-stores in the input list **Syntax:** `stm_r_mx_refer_to_ds (stm_list in_list, int *status);`

Input List Type: dt

stm_r_mx_in_definition_of_dt	**Query:** Elements that appear in the definition of a given user-defined type **Purpose:** Returns the elements that appear in the definition form of the user-defined types in the input list **Syntax:** `stm_r_mx_in_definition_of_dt (stm_list in_list, int *status);`
stm_r_mx_influence_value_of_dt	**Query:** Elements that influence the definition of a given user-defined type **Purpose:** Returns the elements, data-items and user-defined types, that appear in the definition form of the user-defined types in the input list, and those that appear in the definition form of these elements—in all levels **Syntax:** `stm_r_mx_influence_ value_of_dt (stm_list in_list, int *status);`
stm_r_mx_influenced_by_dt	**Query:** Elements that refer to or influenced by a given user-defined type **Purpose:** Returns the elements that directly or indirectly use in their definition the user-defined types in the input list. **Syntax:** `stm_r_mx_influenced_by_dt (stm_list in_list, int *status);`
stm_r_mx_refer_to_dt	**Query:** Elements that refer to a given user-defined type **Purpose**: Returns the elements that use in their definition form the user-defined types in the input list **Syntax:** `stm_r_mx_refer_to_dt (stm_list in_list, int *status);`

Input List Type: ev

stm_r_mx_in_definition_of_ev	**Query:** Elements appearing in the definition of a given event. **Purpose:** Returns the elements that appear in the **Definition** field of events (in the event's form) in the input list. This query identifies the elements that are directly used by the events in the input list. **Syntax:** `stm_r_mx_in_definition_of_ev (stm_list in_list, int *status);`
stm_r_mx_influence_value_of_ev	**Query:** Elements that influence the value of a given event. **Purpose:** Returns the elements that appear in the **Definition** field of events in the input list, and those that appear in the definitions of these elements (for all levels). This query identifies the elements that are used directly or indirectly by the events in the input list. **Syntax:** `stm_r_mx_influence_value_ of_ev (stm_list in_list, int *status);`
stm_r_mx_influenced_by_ev	**Query:** Elements that refer to, or are influenced by, a given event. **Purpose:** Returns the elements that directly or indirectly use the events in the input list. This query identifies all the elements, in all levels, that refer to or affect the input events. **Syntax:** `stm_r_mx_influenced_by_ev (stm_list in_list, int *status);`
stm_r_mx_refer_to_ev	**Query:** Elements that refer to a given event. **Purpose:** Returns the elements that directly use the events in the input list. This query identifies where the input events appear. **Syntax:** `stm_r_mx_refer_to_ev (stm_list in_list, int *status);`

Input List Type: fd

stm_r_mx_containing_fd	**Query:** Elements containing a given field. **Purpose:** Returns the data-items and user-defined types in which the fields in the input list are defined. **Syntax:** `stm_r_mx_containing_fd (stm_list in_list, int *status);`
stm_r_mx_in_definition_of_fd	**Query:** Elements that appear in the definition of a given field. **Purpose:** Returns the elements that appear in the type definition of the fields in the input list. **Syntax:** `stm_r_mx_in_definition_of_fd (stm_list in_list, int *status);`
stm_r_mx_influence_value_of_f	**Query:** Elements that influence the definition of a given field. **Purpose:** Returns the elements, data-items, and user-defined types that appear in the type definition of the fields in the input list, and those that appear in the definition form of these elements (in all levels). **Syntax:** `stm_r_mx_influence_value_ of_fd (stm_list in_list, int *status);`
stm_r_mx_influenced_by_fd	**Query:** Elements that refer to, or are influenced by, a given field. **Purpose:** Returns the elements that directly refer to the fields in the input list. This query identifies where the fields in the input list are used. **Syntax:** `stm_r_mx_influenced_by_fd (stm_list in_list, int *status);`
stm_r_mx_refer_to_fd	**Query:** Elements that refer to a given field. **Purpose:** Returns the elements that directly refer to the fields in the input list. This query identifies where the fields in the input list are used. **Syntax:** `stm_r_mx_refer_to_fd (stm_list in_list, int *status);`

Input List Type: fn

stm_r_mx_influenced_by_fn	**Query:** Elements that refer to, or are influenced by, a given function **Purpose:** Returns the elements that indirectly or directly use the functions in the input list. This query identifies all the elements, in all levels, that refer to the input functions. **Syntax:** `stm_r_mx_influenced_by_fn (stm_list in_list, int *status);`
stm_r_mx_refer_to_fn	**Query:** Elements that refer to a given function. **Purpose:** Returns the elements that directly use the functions in the input list. This query identifies where the input functions appear. **Syntax:** `stm_r_mx_refer_to_fn (stm_list in_list, int *status);`

Input List Type: if

stm_r_mx_in_definition_of_if	**Query:** Elements appearing in the definition of a given a information-flow. **Purpose:** Returns the elements listed in the **Consists of** field (in the information-flow's forms) for information-flows in the input list. This query identifies the elements that are used directly by the information-flows in the input list. **Syntax:** `stm_r_mx_in_definition_of_if (stm_list in_list, int *status);`
stm_r_mx_influence_value_of_if	**Query:** Elements that influence the value of a given information-flow. **Purpose:** Returns the elements contained in the information-flows in the input list (as listed in the **Consists of** field), for all levels of decomposition. This query identifies the elements that are directly or indirectly contained in the information-flows of the input list. **Syntax:** `stm_r_mx_influence_value_ of_if (stm_list in_list, int *status);`

stm_r_mx_influenced_by_if	**Query:** Elements that refer to, or are influenced by, a given information-flow **Purpose:** Returns the elements that directly or indirectly use the information-flows in the input list. This query identifies all the elements, in all levels, that refer to the input information-flows. **Syntax:** `stm_r_mx_influenced_by_if (stm_list in_list, int *status);`
stm_r_mx_refer_to_if	**Query:** Elements that refer to a given information-flow. **Purpose:** Returns the elements that directly use the information-flows in the input list. This query identifies where the input information-flows appear. **Syntax:** `stm_r_mx_refer_to_if (stm_list in_list, int *status);`

Input List Type: md

stm_r_mx_influence_md	**Query:** Elements that are referenced by or influence a given module. **Purpose:** Returns the elements that are used in all levels by the modules in the input list. **Syntax:** `stm_r_mx_influence_md (stm_list in_list, int *status);`
stm_r_mx_influenced_by_md	**Query:** Elements that refer to, or are influenced by, a given module **Purpose:** Returns the elements that directly or indirectly use the modules in the input list **Syntax:** `stm_r_mx_influenced_by_ev (stm_list in_list, int *status);`
stm_r_mx_refer_to_md	**Query:** Elements that refer to a given module. **Purpose:** Returns the elements that directly refer to modules in the input list. **Syntax:** `stm_r_mx_refer_to_md (stm_list in_list, int *status);`

stm_r_mx_referenced_by_md	**Query:** Elements that are referenced by a given module. **Purpose:** Returns the elements that appear in the modules of the input list. **Syntax:** `stm_r_mx_referenced_by_md (stm_list in_list, int *status);`
stm_r_mx_resolved_to_ext_md	**Query:** Elements resolved to a given external module. **Purpose:** Returns the elements to which the external modules in the input list are resolved. **Syntax:** `stm_r_mx_resolved_to_ext_ md (stm_list in_list, int *status);`

Input List Type: mf

stm_r_mx_flowing_through_mf	**Query:** Elements flowing through a given m-flow-line **Purpose:** Returns the information elements (conditions, events, data-items and basic information-flows) that actually flow through the m-flow-lines in the input list **Syntax:** `stm_r_mx_flowing_through_ mf (stm_list in_list, int *status);`
stm_r_mx_information_through_mf	**Query:** **Purpose:** **Syntax:** `stm_r_mx_information_ through_mf (stm_list in_list, int *status);`
stm_r_mx_labeling_mf	**Query:** Elements labeling a given m-flow-line **Purpose:** Returns the elements that label m-flow-lines in the input list **Syntax:** `stm_r_mx_labeling_mf (stm_list in_list, int *status);`

Input List Type: msg

stm_r_mx_labeling_msg	**Query:** Elements labeling a given message **Purpose:** Returns those elements that appear in labels of messages in the input list **Syntax:** `stm_r_mx_labeling_msg (stm_list in_list, int *status);`

Input List Type: mx

stm_r_mx_affected_by_mx	**Query:** Elements affected by a given element. **Purpose:** Returns the elements (primitive data-items, conditions, events, and activities) that are affected (modified, generated, started, stopped, and so on) by elements (states in static reactions or transitions in labels) in the input list. **Syntax:** `stm_r_mx_affected_by_mx (stm_list in_list, int *status);`
stm_r_mx_affecting_mx	**Query:** Elements in which a given element is affected. **Purpose:** Returns the elements (states and transitions) that affect (modify, generate, or activate) the elements (for example, events, data-items, or activities) in the input list. **Syntax:** `stm_r_mx_affecting_mx (stm_list in_list, int *status);`
stm_r_mx_by_attributes_mx	**Query:** Elements by attributes **Purpose:** Returns the elements in the input list that match a particular attribute name and value **Syntax:** `stm_r_mx_by_attributes_mx (stm_list in_list, char* attr_name, char* attr_value, int *status);`
stm_r_mx_callback_binding_mx	**Query:** Elements with callback bindings. **Purpose:** Returns the elements in the input list that have callback bindings. **Syntax:** `stm_r_mx_callback_binding_mx (stm_list in_list, int *status);`

stm_r_mx_comb_elements_mx	**Query:** None. **Purpose:** Returns the elements (data-items and conditions) in the input list that are combinational elements. **Syntax:** `stm_r_mx_comb_elements_mx (stm_list mx_l, int *status);`
stm_r_mx_component_ instance_mx	**Query:** Activities or blocks that are instances of components. **Purpose:** Returns the activities or blocks in the input list that have instances of components. **Syntax:** `stm_r_mx_component_ instance_mx (stm_list in_list, int *status);`
stm_r_mx_def_of_instance_mx	**Query:** Definition elements of a given element. **Purpose:** Returns the definition elements (top-level) for instances in the input list. **Syntax:** `stm_r_mx_def_of_instance_ mx (stm_list in_list, int *status);`
stm_r_mx_explicit_defined_mx	**Query:** Elements explicitly defined. **Purpose:** Returns the elements of the input list that were explicitly defined. **Syntax:** `stm_r_mx_explicit_defined_mx (stm_list in_list, int *status);`
stm_r_mx_generic_instance_mx	**Query:** None. **Purpose:** Returns the boxes (states, activities, and modules) in the input list that are instances of generic charts. **Syntax:** `stm_r_mx_generic_instance_mx (stm_list in_list, int *status);`
stm_r_mx_in_definition_of_mx	**Query:** Elements that appear in the definition of a given element. **Purpose:** Returns the elements that appear in the various fields of the element's form, or in labels of elements in the input list. This query identifies the elements that are used directly by the elements in the input list. **Syntax:** `stm_r_mx_in_definition_of_mx (stm_list in_list, int *status);`

stm_r_mx_influence_value_of_mx	**Query:** Elements that influence the value of a given element. **Purpose:** Returns the elements that appear in various form's fields or labels of elements in the input list, and those that appear in the fields of these elements (for all levels). This query identifies the elements that directly or indirectly influence the elements in the input list. **Syntax:** `stm_r_mx_influence_value_ of_mx (stm_list in_list, int *status);`
stm_r_mx_influenced_by_mx	**Query:** Elements that refer to or influenced by a given element. **Purpose:** Returns the elements that directly or indirectly use the elements in the input list. This query identifies all the elements, in all levels, that refer to or affect the input elements. **Syntax:** `stm_r_mx_influenced_by_mx (stm_list in_list, int *status);`
stm_r_mx_instance_mx	**Query:** Element instance of a given element **Purpose:** Returns the instance elements defined by the elements in the input list **Syntax:** `stm_r_mx_instance_mx (stm_list in_list, int *status);`
stm_r_mx_instance_of_def_mx	**Query:** Instance elements **Purpose:** Returns the instance elements for definition elements (top-level) in the input list. **Syntax:** `stm_r_mx_instance_of_def_ mx (stm_list in_list, int *status);`
stm_r_mx_logical_desc_of_mx	**Query:** Logical descendants of a given element **Purpose:** Returns the logical descendants of the elements in the input list **Syntax:** `stm_r_mx_logical_desc_of_ mx (stm_list in_list, int *status);`
stm_r_mx_logical_parent_of_mx	**Query:** Logical parent elements of a given element. **Purpose:** Returns the logical parent elements of the elements in the input list, taking into account the translation of instances to their definition charts. **Syntax:** `stm_r_mx_logical_sub_of_mx (stm_list in_list, int *status);`

stm_r_mx_logical_sub_of_mx	**Query:** Logical subelements of a given element **Purpose:** Returns the logical subelements of the elements in the input list **Syntax:** `stm_r_mx_logical_sub_of_mx (stm_list in_list, int *status);`
stm_r_mx_meaningly_affecting_mx	**Query:** Activities in which a given element is affected. **Purpose:** Identical to stm_r_mx_affecting_mx, but when the input list includes an ID of a record/union, stm_r_mx_meaningly_affecting_mx will also return elements that affect a field of the record/union, and not necessarily the whole record/union element. **Syntax:** `stm_r_mx_meaningly_affecting_mx (stm_list in_list, int *status);`
stm_r_mx_meaningly_using_mx	**Query:** Activities in which a given element is used. **Purpose:** Identical to stm_r_mx_using_mx, but when the input list includes an ID of a record/union, stm_r_mx_meaningly_using_mx will also return elements that use a field of the record/union, and not necessarily the whole record/union element. **Syntax:** `stm_r_mx_meaningly_using_mx (stm_list in_list, int *status);`
stm_r_mx_name_of_mx	**Query:** Element whose names match a given pattern **Purpose:** Returns all the elements whose names match the specified pattern **Syntax:** `stm_r_mx_name_of_mx (char* pattern, int *status);`
stm_r_mx_offpage_instance_mx	**Query:** None. **Purpose:** Returns the boxes (states, activities, and modules) in the input list that are instances of offpage charts. **Syntax:** `stm_r_mx_offpage_instance_mx (stm_list in_list, int *status);`
stm_r_mx_parameter_mx	**Query:** Elements that are parameters. **Purpose:** Returns the elements in the input list that are declared as formal parameters of a generic chart. **Syntax:** `stm_r_mx_parameter_mx (stm_list in_list, int *status);`

stm_r_mx_physical_desc_of_mx	**Query:** Physical descendants of a given element **Purpose:** Returns the physical descendants (those within the same chart) for the elements in the input list **Syntax:** `stm_r_mx_physical_desc_of_mx (stm_list in_list, int *status);`
stm_r_mx_physical_parent_of_mx	**Query:** Physical parent elements of a given element **Purpose:** Returns the physical parent elements (those within the same chart) for the elements in the input list **Syntax:** `stm_r_mx_physical_parent_ of_mx (stm_list in_list, int *status);`
stm_r_mx_physical_sub_of_mx	**Query:** Physical subelements of a given element **Purpose:** Returns the physical subelements (those within the same chart) for the elements in the input list **Syntax:** `stm_r_mx_physical_sub_of_ mx (stm_list in_list, int *status);`
stm_r_mx_refer_to_mx	**Query:** Elements that refer to a given element. **Purpose:** Returns the elements that directly refer to elements in the input list. This query identifies where the input elements are used. **Syntax:** `stm_r_mx_refer_to_mx (stm_list in_list, int *status);`
stm_r_mx_resolved_to_ext_mx	**Query:** Elements resolved to a given external box. **Purpose:** Returns the activities and modules (internal, external, or environment) to which the external activities and modules in the input list are resolved. **Syntax:** `stm_r_mx_resolved_to_ext_ mx (stm_list in_list, int *status);`
stm_r_mx_synonym_of_mx	**Query:** Elements whose synonyms match a given pattern **Purpose:** Returns all the elements whose synonyms match the specified pattern **Syntax:** `stm_r_mx_synonym_of_mx (char* pattern, int *status);`

stm_r_mx_unresolved_mx	**Query:** Unresolved elements. **Purpose:** Returns the unresolved elements in the input list. **Syntax:** `stm_r_mx_unresolved_mx (stm_list in_list, int *status);`
stm_r_mx_used_by_mx	**Query:** Elements used by a given element. **Purpose:** Returns the elements (primitive events, conditions, data-items, states, and activities) that are used (evaluated by the elements, such as states in static reactions and transitions in labels) in the input list. **Syntax:** `stm_r_mx_used_by_mx (stm_list in_list, int *status);`
stm_r_mx_using_mx	**Query:** Elements in which a given element is used. **Purpose:** Returns the elements (states in static reactions and transitions in labels) that use (evaluate) the elements (basic events, conditions, data-items, states, and activities) in the input list. **Syntax:** `stm_r_mx_using_mx (stm_list in_list, int *status);`
stm_r_mx_with_combinationals_mx	**Query:** None. **Purpose:** Returns the elements (activities and state charts) in the input list that have combinational assignments. **Syntax:** `stm_r_mx_with_ combinationals_mx (stm_list mx_l, int *status);`

Function Relationships

The following functions are related, but have subtle differences:

- `stm_r_mx_influenced_by_mx`
- `stm_r_mx_affected_by_mx`
- `stm_r_mx_used_by_mx`
- `stm_r_mx_affecting_mx`

The following matrix shows their relationships. In the matrix, opposite functions go from left to right, whereas cause and effect functions go up and down.

influenced by	used by
Function	
affected by	affecting

For example:

- If `x` is influenced by `y`, then `y` is used by `x`.
- If `n` is affected by `m`, then `m` is affecting `n`.

Consider the following statement:

```
if x is true then Function will set y=5
```

In this statement, `x` influences `Function` and `Function` affects `y`. This is shown in the matrix as follows:

- Elements above the double line influence `Function`.
- Elements below the double line are affected by `Function`.

For example, `x` is used by `Function` to determine whether to set the value of `y`, and `Function` is affecting `y` by setting its value.

There are four possible relationships between these functions: two opposites and two cause and effects.

- Opposite: influenced by and used by
- Opposite: affected by and affecting
- Cause and effect: influenced by and affected by
- Cause and effect: used by and affecting

To illustrate the relationships, consider the following static reaction in a state called STATE:

```
[D]/X=5 if D is true, then set x=y
```

The following table shows the relationships.

Relation Type	Description
Opposite: influenced and used by	STATE reads D to determine whether to perform an action, and D gives STATE the cue to set X=Y. In other words, STATE is influenced by D, and D is used by STATE.
Opposite: affected by and affecting	X's value is set by STATE and STATE sets the value of X. In other words, X is affected by STATE, and STATE is affecting X.
Cause and effect: influenced by and affected by	STATE reads y to determine which value should be assigned to X, and while in STATE, X can be set to y. In other words, when STATE is influenced by y, it results in X being affected by STATE.
Cause and effect: used by and affecting	If y is true, STATE sets the value of X. y is influencing STATE; STATE is affecting X. In other words, when y is used by STATE, it results in STATE affecting X.

Input List Type: router

stm_r_mx_flowing_from_router	**Query:** Elements flowing from a given router **Purpose:** Returns the elements actually flowing from routers in the input list **Syntax:** `stm_r_mx_flowing_from_ router (stm_list in_list, int *status);`
stm_r_mx_flowing_to_router	**Query:** Elements flowing to a given router **Purpose:** Returns the elements actually flowing to routers in the input list **Syntax:** `stm_r_mx_flowing_to_router (stm_list in_list, int *status);`
stm_r_mx_refer_to_router	**Query:** Elements that refer to a given router. **Purpose:** Returns the elements that directly refer to routers in the input list. This query identifies where the routers are used. **Syntax:** `stm_r_mx_refer_to_router (stm_list in_list, int *status);`
stm_r_mx_resolved_to_ext_router	**Query:** Elements resolved to a given router. **Purpose:** Returns the elements to which the external routers in the input list are resolved. **Syntax:** `stm_r_mx_resolved_to_ext_ router (stm_list in_list, int *status);`

Input List Type: sb

stm_r_mx_influenced_by_sb	**Query:** Elements that refer to, or are influenced by, a given subroutine **Purpose:** Returns the elements that directly or indirectly use the subroutines in the input list **Syntax:** `stm_r_mx_influenced_by_sb (stm_list in_list, int *status);`
stm_r_mx_refer_to_sb	**Query:** Elements that refer to a given subroutine **Purpose:** Returns the elements that directly refer to subroutines in the input list **Syntax:** `stm_r_mx_refer_to_sb (stm_list in_list, int *status);`

Input List Type: st

stm_r_mx_affected_by_st	**Query:** Elements affected by a given state. **Purpose:** Returns the elements (data-items, conditions, and events) that are affected (modified or generated) by states (in mini-specs and combinational assignments) in the input list. **Syntax:** `stm_r_mx_affected_by_st (stm_list in_list, int *status);`
stm_r_mx_influence_st	**Query:** Elements that are referenced by or influence a given state. **Purpose:** Returns the elements that are used in all levels by the states in the input list. This includes all logical descendant states, the transitions that enter or exit these states, and the elements that appear in the various fields of these states, the labels of the transitions, and their components. **Syntax:** `stm_r_mx_influence_st (stm_list in_list, int *status);`
stm_r_mx_influenced_by_st	**Query:** Elements that refer to, or are influenced by a given state. **Purpose:** Returns the elements that directly or indirectly use the states in the input list. This query identifies all the elements, in all levels, that refer to or affect the input states. **Syntax:** `stm_r_mx_influenced_by_st (stm_list in_list, int *status);`
stm_r_mx_refer_to_st	**Query:** Elements that refer to a given state. **Purpose:** Returns the elements that directly refer to states in the input list. This query identifies where the input states are used. **Syntax:** `stm_r_mx_refer_to_st (stm_list in_list, int *status);`

stm_r_mx_referenced_by_st	**Query:** Elements that are referenced by a given state. **Purpose:** Returns the elements that appear in the states of the input list. This includes all physical descendant states, the transitions that enter or exit these states, and the elements that appear in the various fields of these states and in the labels of the transitions. **Syntax:** `stm_r_mx_referenced_by_st (stm_list in_list, int *status);`
stm_r_mx_used_by_st	**Query:** Elements used by a given state. **Purpose:** Returns the elements (data-items, conditions, and events) that are used (evaluated) by states (in mini-specs and combinational assignments) in the input list. **Syntax:** `stm_r_mx_used_by_st (stm_list in_list, int *status);`

Input List Type: tr

stm_r_mx_affected_by_tr	**Query:** Elements affected by a given transition **Purpose:** Returns the elements (data-items, conditions, and events) that are affected (modified, generated) by transitions (in mini-specs and combinational assignments) in the input list **Syntax:** `stm_r_mx_affected_by_tr (stm_list in_list, int *status);`
stm_r_mx_labeling_tr	**Query:** Elements labeling a given transition **Purpose:** Returns those elements that appear in labels of the transitions in the input list **Syntax:** `stm_r_mx_labeling_tr (stm_list in_list, int *status);`
stm_r_mx_source_of_tr	**Query:** Elements that are sources of a given transition **Purpose:** Returns the elements (states and connectors) that are sources of transitions in the input list **Syntax:** `stm_r_mx_source_of_tr (stm_list in_list, int *status);`
stm_r_mx_target_of_tr	**Query:** Elements that are targets of a given transition **Purpose:** Returns elements (states and connectors) that are targets of transitions in the input list **Syntax:** `stm_r_mx_target_of_tr (stm_list in_list, int *status);`
stm_r_mx_used_by_tr	**Query:** Elements used by a given transition **Purpose:** Returns the elements (data-items, conditions, and events) that are used (evaluated) by transitions (in mini-specs and combinational assignments) in the input list **Syntax:** `stm_r_mx_used_by_tr (stm_list in_list, int *status);`

Module-Occurrences (om)

This section documents the query that returns a list of module-occurrences.

Input List Type: md

stm_r_om_in_md	**Query:** Module-occurrences contained in a given module **Purpose:** Returns the module-occurrences contained in modules from the input list **Syntax:** `stm_r_om_in_md (stm_list in_list, int *status);`

Routers (router)

This section documents the queries that return a list of routers.

Input List Type: ac

stm_r_router_contained_in_ac	**Query:** Routers contained in a given activity **Purpose:** Returns the routers contained directly in activities from the input list **Syntax:** `stm_r_router_contained_in_ac (stm_list in_list, int *status);`
stm_r_router_in_ac	**Query:** Router in a given activity **Purpose:** Returns the routers directly and indirectly contained in the activities from the input list **Syntax:** `stm_r_router_in_ac (stm_list in_list, int *status);`

Input List Type: af

stm_r_router_source_of_af	**Query:** Routers that are sources of a given a-flow-line **Purpose:** Returns the routers that are sources of a-flow-lines in the input list **Syntax:** `stm_r_router_source_of_af (stm_list in_list, int *status);`
stm_r_router_target_of_af	**Query:** Routers that are targets of a given a-flow-line **Purpose:** Returns the routers that are sources of a-flow-lines in the input list **Syntax:** `stm_r_router_target_of_af (stm_list in_list, int *status);`

Input List Type: ch

stm_r_router_def_or_unres_in_ch	**Query:** Routers defined or unresolved in a given chart **Purpose:** Returns the routers that are explicitly defined or unresolved in the charts of the input list **Syntax:** `stm_r_router_def_or_unres_in_ch (stm_list in_list, int *status);`
stm_r_router_defined_in_ch	**Query:** Routers defined in a given chart **Purpose:** Returns the routers that are explicitly defined in the charts of the input list **Syntax:** `stm_r_router_defined_in_ch (stm_list in_list, int *status);`
stm_r_router_unresolved_in_ch	**Query:** Routers unresolved in a given chart **Purpose:** Returns the routers that are unresolved in the charts of the input list **Syntax:** `stm_r_router_unresolved_in_ch (stm_list in_list, int *status);`

Input List Type: md

stm_r_router_resides_in_md	**Query:** Routers residing in a given module. **Purpose:** Returns the routers residing in modules from the input list. The module appears in the **Resides in Module** field of the router's form. **Syntax:** `stm_r_router_resides_in_md (stm_list in_list, int *status);`

Input List Type: router

stm_r_router_by_attr_router	**Query:** Routers by attributes **Purpose:** Returns the routers in the input list that match a given attribute name and value **Syntax:** `stm_r_router_resides_in_md (stm_list in_list, int *status);`
stm_r_router_exp_def_router	**Query:** Routers that are explicitly defined **Purpose:** Returns from the input list those routers that were explicitly defined **Syntax:** `stm_r_router_exp_def_ router (stm_list in_list, int *status);`
stm_r_router_name_of_router	**Query:** Routers whose names match a given pattern **Purpose:** Returns all routers whose name matches a given pattern **Syntax:** `stm_r_router_name_of_ router (char* pattern, int *status));`
stm_r_router_res_to_ext_router	**Query:** Routers resolved by a given external router **Purpose:** Returns the routers (internal and external) to which the external routers in the input list are resolved **Syntax:** `stm_r_router_res_to_ext_ router (stm_list in_list, int *status);`
stm_r_router_synonym_of_router	**Query:** Routers whose synonyms match a given pattern **Purpose:** Returns all routers whose synonyms match a given pattern **Syntax:** `stm_r_router_synonym_of_ router (char* pattern, int *status);`

stm_r_router_unresolved_router	**Query:** Unresolved routers **Purpose:** Returns the unresolved routers in the input list **Syntax:** `stm_r_router_unresolved_ router (stm_list router_list, int *status);`
stm_r_router_unresolved_in_ch	**Query**: Unresolved in a given chart. **Purpose**: Returns routers that are unresolved in the charts of the input list. **Syntax**: `stm_r_router_unresolved_in_ch (stm_list in_list, int*status);`
stm_r_mx_refer_to_router	**Query**: Elements that refer to a given router. **Purpose**: Returns the elements that directly refer to routers in the input list. This query identifies where input routers are used. **Syntax**: `stm_r_mx_refer_to_router (stm_list in_list, int *status);`
stm_r_md_contains_router	**Query**: Modules in which a given datastoreresides. **Purpose**: Returns the modules in which routers from the input list resides. The modules appear in the Resides in Module field of a Router's form **Syntax**: `stm_r_md_contains_router (stm_list in_list, int *status);`
stm_r_ch_define_router	**Query**: Charts in which a given router is defined **Purpose**: Returns the charts in which the routers in the input list are explicitly defined or unresolved. **Syntax**: `stm_r_ch_define_router (stm_list in_list, int *status);`
stm_r_laf_from_source_router	**Query**: A-flow-lines whose source is a given router **Purpose**: Returns local compound aflow- lines that originate at routers in the input list **Syntax**: `stm_r_laf_from_source_router (stm_list in_list, int *status);`
stm_r_laf_to_target_router	**Query**: A-flow-lines whose target is a given router within chart **Purpose**: Returns local a-flow-lines (those within charts) that terminate at routers in the input list **Syntax**: `stm_r_laf_to_target_router (stm_list in_list, int *status);`

stm_r_router_name_of_router	**Query**: Routers whose names match a given pattern. **Purpose**: Returns all the routers whose names match a given pattern **Syntax**: `stm_r_router_name_of_router (char* pattern, int *status);`
stm_r_router_synonym_of_router	**Query**: Routers whose synonyms match a given pattern, **Purpose**: Returns all the routers whose synonyms match the specified pattern **Syntax**: `stm_r_ac_synonym_of_ac (char* pattern, int *status);`

Subroutines (sb)

This section documents the queries that return a list of subroutines.

Input List Type: ch

stm_r_sb_connected_to_ch	**Query:** Subroutines that are connected to a given procedural statechart **Purpose:** Returns the subroutines in the input list that are connected to the specified procedural statechart **Syntax:** `stm_r_sb_connected_to_ch (stm_list in_list, int *status);`
stm_r_sb_connected_to_sch	**Query:** Subroutines that are connected to a given procedural statechart **Purpose:** Returns the subroutines in the input list that are connected to the specified procedural statechart **Syntax:** `stm_r_sb_connected_to_sch (stm_list in_list, int *status);`
stm_r_sb_connected_to_fch	**Query:** Subroutines that are connected to a given Flowcharts **Purpose:** Returns the subroutines in the input list that are connected to the specified Flowchart **Syntax:** `stm_r_sb_connected_to_fch (stm_list in_list, int *status);`
stm_r_sb_defined_in_ch	**Query:** Subroutines defined in a given chart **Purpose:** Returns the subroutines that are explicitly defined in the charts in the input list **Syntax:** `stm_r_sb_defined_in_ch (stm_list in_list, int *status);`
stm_r_sb_def_or_unres_in_ch	**Query:** Subroutines defined or unresolved in a given chart **Purpose:** Returns the subroutines that are explicitly defined or unresolved in the charts in the input list **Syntax:** `stm_r_sb_def_or_unres_in_ ch (stm_list in_list, int *status);`

stm_r_sb_unresolved_in_ch	**Query:** Subroutines unresolved in a given chart **Purpose:** Returns the subroutines that are unresolved in the charts in the input list **Syntax:** `stm_r_sb_unresolved_in_ch (stm_list in_list, int *status);`

Input List Type: sb

stm_r_sb_ada_sb	**Query:** Subroutines written in Ada **Purpose:** Returns subroutines in the input list that are written in Ada and stored in the database using the Implementation menu **Syntax:** `stm_r_sb_ada_sb (stm_list in_list, int *status);`
stm_r_sb_ansi_c_sb	**Query:** Subroutines written in ANSI C **Purpose:** Returns subroutines in the input list that are written in ANSI C and stored in the database using the Implementation menu **Syntax:** `stm_r_sb_ansi_c_sb (stm_list in_list, int *status);`
stm_r_sb_bit_sb	**Query:** Subroutines by subtype **Purpose:** Returns the subroutines in the input list that are defined as bit **Syntax:** `stm_r_sb_bit_sb (stm_list in_list, int *status);`
stm_r_sb_bits_sb	**Query:** Subroutines by subtype **Purpose:** Returns the subroutines in the input list that are defined as bit array **Syntax:** `stm_r_sb_bits_sb (stm_list in_list, int *status);`
stm_r_sb_by_attributes_sb	**Query:** Subroutines by attributes **Purpose:** Returns the subroutines in the input list that match the specified attribute name and value **Syntax:** `stm_r_sb_by_attributes_sb (stm_list in_list, char* attr_name, char* attr_value, int *status);`

stm_r_sb_explicit_defined_sb	**Query:** Subroutines that are explicitly defined **Purpose:** Returns the subroutines in the input list that are explicitly defined **Syntax:** `stm_r_sb_explicit_defined_sb (stm_list in_list, int *status);`
stm_r_sb_fn_with_side_effect_sb	**Query:** Function subroutines with potential side-effects **Purpose:** Returns the function subroutines in the input list that have potential side-effects **Syntax:** `stm_r_sb_fn_with_side_ effect_sb (stm_list in_list, int *status);`
stm_r_sb_function_sb	**Query:** Subroutines defined as functions **Purpose:** Returns the subroutines in the input list that are defined as functions **Syntax:** `stm_r_sb_function_sb (stm_list in_list, int *status);`
stm_r_sb_globals_usage_sb	**Query:** Subroutines that have global data **Purpose:** Returns all subroutines in the input list that have global data **Syntax:** `stm_r_sb_globals_usage_sb (stm_list in_list, int *status);`
stm_r_sb_imp_action_lang_sb	**Query:** Subroutines whose selected implementation is Action Language **Purpose:** Returns the subroutines in the input list that are implemented in the Rational Statemate Action Language using **Select Implementation** **Syntax:** `stm_r_sb_imp_action_lang_ sb (stm_lis in_list, int *status);`
stm_r_sb_imp_ada_code_sb	**Query:** Subroutines whose selected implementation is Ada Code **Purpose:** Returns the subroutines in the input list that are implemented in Ada using **Select Implementation** in the properties **Syntax:** `stm_r_sb_imp_ada_code_sb (stm_list in_list, int *status);`

stm_r_sb_imp_ansi_c_code_sb	**Query:** Subroutines whose selected implementation is ANSI C Code **Purpose:** Returns the subroutines in the input list that are implemented in ANSI C using **Select Implementation** **Syntax:** `stm_r_sb_imp_ansi_c_code_ sb (stm_list in_list, int *status);`
stm_r_sb_imp_best_match_sb	**Query:** Subroutines whose selected implementation is Best Match **Purpose:** Returns the subroutines in the input list that are implemented as the Best Match using **Select Implementation** **Syntax:** `stm_r_sb_imp_best_match_sb (stm_list in_list, int *status);`
stm_r_sb_imp_kr_c_code_sb	**Query:** Subroutines whose selected implementation is K&R C Code **Purpose:** Returns the subroutines in the input list that are implemented in K&R C using **Select Implementation** **Syntax:** `stm_r_sb_imp_kr_c_code_sb (stm_list in_list, int *status);`
stm_r_sb_imp_none_sb	**Query:** Subroutines whose selected implementation is None **Purpose:** Returns the subroutines in the input list that are not implemented (None) using **Select Implementation** **Syntax:** `stm_r_sb_imp_none_sb (stm_list in_list, int *status);`
stm_r_sb_imp_procedural_sch_sb	**Query:** Subroutines whose selected implementation is Procedural Statechart **Purpose:** Returns the subroutines in the input list that are implemented as Procedural Statecharts using **Select Implementation** **Syntax:** `stm_r_sb_imp_procedural_ sch_sb (stm_list in_list, int *status);`
stm_r_sb_integer_sb	**Query:** Subroutines by subtype **Purpose:** Returns the subroutines in the input list that are defined as integer **Syntax:** `stm_r_sb_integer_sb (stm_list in_list, int *status);`

stm_r_sb_kr_c_sb	**Query:** Subroutines written in K&R C **Purpose:** Returns subroutines in the input list that are written in K&R C and stored in the database using the Implementation menu **Syntax:** `stm_r_sb_kr_c_sb (stm_list in_list, int *status);`
stm_r_sb_missing_sb	**Query:** Subroutines by subtype **Purpose:** Returns the subroutines in the input list for which no type is defined **Syntax:** `stm_r_sb_missing_sb (stm_list in_list, int *status);`
stm_r_sb_name_of_sb	**Query:** Subroutines whose names match a given pattern **Purpose:** Returns all subroutines whose name matches the specified pattern **Syntax:** `stm_r_sb_name_of_sb (char* pattern, int *status);`
stm_r_sb_parameters_sb	**Query:** Subroutines that have parameters **Purpose:** Returns all subroutines in the input list that have parameters **Syntax:** `stm_r_sb_parameters_sb (stm_list in_list, int *status);`
stm_r_sb_procedural_sch_sb	**Query:** Subroutines designed as procedural statecharts **Purpose:** Returns subroutines in the input list that are designed as procedural statecharts and stored in the database using the Implementation menu **Syntax:** `ch_sch_sb (stm_list in_list, int *status);`
stm_r_sb_procedural_fch_sb	**Query:** Subroutines designed as Flowchart **Purpose:** Returns subroutines in the input list that are designed as Flowcharts and stored in the database using the Implementation menu **Syntax**: `stm_r_sb_procedural_fch_sb (stm_list in_list, int *status);`
stm_r_sb_procedure_sb	**Query:** Subroutines defined as procedures **Purpose:** Returns the subroutines in the input list that are defined as procedures **Syntax:** `stm_r_sb_procedure_sb (stm_list in_list, int *status);`

stm_r_sb_real_sb	**Query:** Subroutines by subtype **Purpose:** Returns the subroutines in the input list that are defined as real **Syntax:** `stm_r_sb_real_sb (stm_list in_list, int *status);`
stm_r_sb_statemate_action_sb	**Query:** Subroutines written in the Rational Statemate action language **Purpose:** Returns subroutines in the input list that are written in the Rational Statemate action language and stored in the database using the Implementation menu **Syntax:** `stm_r_sb_statemate_action_sb (stm_list in_list, int *status);`
stm_r_sb_string_sb	**Query:** Subroutines by subtype **Purpose:** Returns the subroutines in the input list that are defined as string **Syntax:** `stm_r_sb_string_sb (stm_list in_list, int *status);`
stm_r_sb_synonym_of_sb	**Query:** Subroutines whose synonyms match a given pattern **Purpose:** Returns all subroutines whose synonyms match the specified pattern **Syntax:** `stm_r_sb_synonym_of_sb (char* pattern, int *status);`
stm_r_sb_task_sb	**Query:** Subroutines defined as tasks **Purpose:** Returns the subroutines in the input list that are defined as tasks **Syntax:** `stm_r_sb_task_sb (stm_list in_list, int *status);`

stm_r_sb_unresolved_sb	**Query:** Unresolved subroutines **Purpose:** Returns the unresolved subroutines in the input list **Syntax:** `stm_r_sb_unresolved_sb (stm_list in_list, int *status);`
stm_r_sb_user_type_sb	**Query:** Subroutines by subtype **Purpose:** Returns the subroutines in the input list that are defined as user-defined type **Syntax:** `stm_r_sb_user_type_sb (stm_list in_list, int *status);`

States (st)

This section documents the queries that return a list of states.

Input List Type: ac

stm_r_st_done_throughout_ac	**Query:** States in which a given activity is performed throughout **Purpose:** Returns the states for which activities in the input list are performed throughout that state (as specified in **Activities Within/Throughout** field in the state's form) **Syntax:** `stm_r_st_done_throughout_ ac (stm_list in_list, int *status);`
stm_r_st_done_within_ac	**Query:** States in which a given activity is performed within them **Purpose:** Returns the states in which activities in the input list are performed within that state (as specified in **Activities Within/Throughout** field in the state's form) **Syntax:** `stm_r_st_done_within_ac (stm_list in_list, int *status);`

Input List Type: ch

stm_r_st_def_or_unres_in_ch	**Query:** States defined or unresolved in a given chart **Purpose:** Returns the states that are explicitly defined or unresolved in the charts of the input list **Syntax:** `stm_r_st_def_or_unres_in_ ch (stm_list in_list, int *status);`
stm_r_st_defined_in_ch	**Query:** States defined in a given chart **Purpose:** Returns the states that are explicitly defined in the charts of the input list **Syntax:** `stm_r_st_defined_in_ch (stm_list in_list, int *status);`
stm_r_st_instance_of_ch	**Query:** State instances of a given chart **Purpose:** Returns the instance states defined by the charts in the input list **Syntax:** `stm_r_st_instance_of_ch (stm_list in_list, int *status));`
stm_r_st_root_in_ch	**Query:** Root states of a given chart **Purpose:** Returns the internally defined states (of type diagram) attached to the charts in the input list **Syntax:** `stm_r_st_root_in_ch (stm_list in_list, int *status);`
stm_r_st_top_level_in_ch	**Query:** Top-level states of a given chart **Purpose:** Returns the top-level states (not contained in any box) of the charts in the input list **Syntax:** `stm_r_st_top_level_in_ch (stm_list in_list, int *status);`
stm_r_st_unresolved_in_ch	**Query:** States unresolved in a given chart **Purpose:** Returns the states that are unresolved in the charts of the input list **Syntax:** `stm_r_st_unresolved_in_ch (stm_list in_list, int *status);`

Input List Type: cn

stm_r_st_containing_cn	**Query:** States containing a given connector **Purpose:** Returns the states that encapsulate specified connectors from the input list **Syntax:** `stm_r_st_containing_cn (stm_list in_list, int *status);`

Input List Type: mx

stm_r_st_affecting_mx	**Query:** States in which a given element is affected. **Purpose:** Returns the states that affect (modify, generate, or activate) the elements (for example, events, data-items, or activities) in the input list. **Syntax:** `stm_r_st_affecting_mx (stm_list in_list, int *status);`
stm_r_st_meaningly_affecting_mx	**Query:** Activities in which a given element is affected. **Purpose:** Identical to `stm_r_st_affecting_mx`, but when the input list includes an ID of a record/union, `stm_r_st_meaningly_affecting_mx` will also return elements that affect a field of the record/union, and not necessarily the whole record/union element. **Syntax**: `stm_r_st_meaningly_affecting_mx (stm_list in_list, int *status);`
stm_r_st_meaningly_using_mx	**Query:** Activities in which a given element is used. **Purpose:** Identical to `stm_r_st_using_mx`, but when the input list includes an ID of a record/union, `stm_r_st_meaningly_using_mx` will also return elements that use a field of the record/union, and not necessarily the whole record/union element. **Syntax:** `stm_r_st_meaningly_using_mx (stm_list in_list, int *status);`

stm_r_st_using_mx	**Query:** States in which a given element is used. **Purpose:** Returns the states in static reactions that use (evaluate) the elements (basic events, conditions, data-items, states, and activities) in the input list. **Syntax:** `stm_r_st_using_mx (stm_list in_list, int *status);`

Input List Type: st

stm_r_st_and_st	**Query:** And states. **Purpose:** Returns the states in the input list that are And-states. **Syntax:** `stm_r_st_and_st (stm_list in_list, int *status);`
stm_r_st_basic_st	**Query:** Basic states. **Purpose:** Returns the states in the input list that are basic (states that have no descendants). **Syntax:** `stm_r_st_basic_st (stm_list in_list, int *status);`
stm_r_st_by_attributes_st	**Query:** States by attributes. **Purpose:** Returns the states in the input list that match a given attribute name and value. **Syntax:** `stm_r_st_by_attributes_st (stm_list in_list, char* attr_name, char* attr_value, int *status);`
stm_r_st_callback_binding_st	**Query:** States with callback bindings. **Purpose:** Returns the states in the input list that have callback bindings. **Syntax:** `stm_r_st_callback_binding_st (stm_list in_list, int *status);`
stm_r_st_def_of_instance_st	**Query:** Definition states of a given state. **Purpose:** Returns the definition states (top-level in the definition chart) for instances in the input list. **Syntax:** `stm_r_st_def_of_instance_ st (stm_list in_list, int *status);`

stm_r_st_default_entry_to_st	**Query:** Default entry to the default state. **Purpose:** Returns the default states of the or-states in the input list. **Syntax:** `stm_r_st_default_entry_to_st (stm_list in_list, int *status);`
stm_r_st_explicit_defined_st	**Query:** States explicitly defined. **Purpose:** Returns the states in the input list that were explicitly defined. **Syntax:** `stm_r_st_explicit_defined_st (stm_list in_list, int *status);`
stm_r_st_generic_instance_st	**Query:** Generic instance states. **Purpose:** Returns the states in the input list that are instances of generic charts. **Syntax:** `stm_r_st_generic_instance_st (stm_list in_list, int *status);`
stm_r_st_history_connector_st	**Query:** States containing a history connector. **Purpose:** Returns the states in the input list that contain a history connector. **Syntax:** `stm_r_st_history_connector_st (stm_list in_list, int *status);`
stm_r_st_instance_of_def_st	**Query:** Instance states of a given definition state. **Purpose:** Returns the instance states for definition states (top-level states in a definition chart) in the input list. **Syntax:** `stm_r_st_instance_of_def_ st (stm_list in_list, int *status);`
stm_r_st_instance_st	**Query:** Instance states. **Purpose:** Returns those states in the input list that are instance states. **Syntax:** `stm_r_st_instance_st (stm_list in_list, int *status);`

stm_r_st_logical_desc_of_st	**Query:** Logical descendants of a given state. **Purpose:** Returns the logical descendants (children, grandchildren, and so on) of states in the input list, taking into account the translation of instances to their definition charts. **Syntax:** `stm_r_st_logical_desc_of_ st (stm_list in_list, int *status);`
stm_r_st_logical_parent_of_st	**Query:** Logical parent states of a given state. **Purpose:** Returns the logical parent states of the states in the input list, taking into account the translation of instances to their definition charts. **Note:** This query provides similar output as `stm_r_st_physical_ parent_of_st`, but for states that are substates of a top-level state in a definition chart, this query also returns the instance box. **Syntax:** `stm_r_st_logical_parent_of_st (stm_list in_list, int *status);`
stm_r_st_logical_sub_of_st	**Query:** Logical substates of a given state. **Purpose:** Returns the logical substates of the states in the input list, taking into account the translation of instances to their definition charts. **Syntax:** `stm_r_st_logical_sub_of_st (stm_list in_list, int *status);`
stm_r_st_name_of_st	**Query:** States whose names match a given pattern **Purpose:** Returns all the states whose names match the specified pattern **Syntax:** `stm_r_st_name_of_st (char* pattern, int *status);`
stm_r_st_offpage_instance_st	**Query:** Offpage instance states. **Purpose:** Returns the states in the input list that are instances of offpage charts. **Syntax:** `stm_r_st_offpage_instance_st (stm_list in_list, int *status);`
stm_r_st_physical_desc_of_st	**Query:** Physical descendants of a given state. **Purpose:** Returns the physical descendants (those within the same chart) for the states in the input list. **Syntax:** `stm_r_st_physical_desc_of_st (stm_list in_list, int *status);`

stm_r_st_physical_parent_of_st	**Query:** Physical parent states of a given state. **Purpose:** Returns the physical parent states (those within the same chart) for the states in the input list. **Syntax:** `stm_r_st_physical_desc_of_st (stm_list in_list, int *status);`
stm_r_st_physical_sub_of_st	**Query:** Physical substates of a given state. **Purpose:** Returns the physical substates (those within the same chart) for the states in the input list. **Syntax:** `stm_r_st_physical_sub_of_ st (stm_list in_list, int *status);`
stm_r_st_reaction_activity_st	**Query:** States having reactions or activities. **Purpose:** Returns the states from the input list that have static reactions or activities performed within or throughout the state. **Syntax:** `stm_r_st_reaction_activity_st (stm_list in_list, int *status);`
stm_r_st_synonym_of_st	**Query:** States whose synonyms match a given pattern **Purpose:** Returns all the states whose synonyms match the specified pattern **Syntax:** `stm_r_st_synonym_of_st (char* pattern, int *status);`
stm_r_st_unresolved_st	**Query:** Unresolved states. **Purpose:** Returns the unresolved states in the input list. **Syntax:** `stm_r_st_unresolved_st (stm_list in_list, int *status);`

Input List Type: tr

stm_r_st_source_of_tr	**Query:** States that are sources of a given transition **Purpose:** Returns the states that are sources of transitions in the input list **Syntax:** `stm_r_st_source_of_tr (stm_list in_list, int *status);`
stm_r_st_target_of_tr	**Query:** States that are targets of a given transition **Purpose:** Returns the states that are targets of transitions in the input list **Syntax:** `stm_r_st_target_of_tr (stm_list in_list, int *status);`

Timing Constraint (tc)

This section documents the query that returns a list of timing constraints.

Input List Type: ch

stm_r_tc_defined_in_ch	**Query:** Timing constraints defined in a given chart **Purpose:** Returns the timing constraints that are explicitly defined in the charts of the input list **Syntax:** `stm_r_tc_defined_in_ch (stm_list in_list, int *status);`

Transitions (tr)

This section documents the queries that return a list of transitions. The following abbreviations are used:

- bt—Basic transition
- tr—Compound transition

Output List: tr

Input List Type: cn

stm_r_tr_from_source_cn	**Query:** Transitions whose source is a given connector **Purpose:** Returns the transitions in the input list whose source is a termination or history connector **Syntax:** `stm_r_tr_from_source_cn (stm_list in_list, int *status);`
stm_r_tr_to_target_cn	**Query:** Transitions whose target is a given connector **Purpose:** Returns the transition in the input list whose target is a termination or history connector **Syntax:** `stm_r_tr_to_target_cn (stm_list in_list, int *status);`

Input List Type: enforced

stm_r_tr_by_attributes_enforced	**Query:** Transitions whose source is a given connector **Purpose:** Returns the transitions in the input list that match the specified attribute name and value **Syntax:** `stm_r_tr_by_attributes_enforced (stm_list in_list, int char* attr_name, char* attr_value, int *status);`

Input List Type: mx

stm_r_tr_affecting_mx	**Query:** Transitions in which a given element is affected. **Purpose:** Returns the transitions that affect (modify, generate, or activate) the elements (for example, events, data-items, or activities) in the input list. **Syntax:** `stm_r_tr_affecting_mx (stm_list in_list, int *status);`
stm_r_tr_from_source_mx	**Query:** Transitions whose source is a given element **Purpose:** Returns transitions that originate at elements in the input list **Syntax:** `stm_r_tr_from_source_mx (stm_list in_list, int *status);`
stm_r_tr_meaningly_affecting_mx	**Query:** Activities in which a given element is affected. **Purpose:** Identical to stm_r_tr_affecting_mx, but when the input list includes an ID of a record/union, stm_r_tr_meaningly_affecting_mx will also return elements that affect a field of the record/union, and not necessarily the whole record/union element. **Syntax:** `stm_r_tr_meaningly_affecting_mx (stm_list in_list, int *status);`
stm_r_tr_meaningly_using_mx	**Query:** Activities in which a given element is used. **Purpose:** Identical to stm_r_tr_using_mx, but when the input list includes an ID of a record/union, stm_r_tr_meaningly_using_mx will also return elements that use a field of the record/union, and not necessarily the whole record/union element. **Syntax:** `stm_r_tr_meaningly_using_mx (stm_list in_list, int *status);`
stm_r_tr_to_target_mx	**Query:** Transitions whose target is a given element **Purpose:** Returns the transitions whose target is an element from the input list **Syntax:** `stm_r_tr_to_target_mx (stm_list in_list, int *status);`

stm_r_tr_using_mx	**Query:** Transitions in which a given element is used. **Purpose:** Returns the transitions in labels that use (evaluate) the elements (basic events, conditions, data-items, states, and activities) in the input list. **Syntax:** `stm_r_tr_using_mx (stm_list in_list, int *status);`

Input List Type: st

stm_r_tr_default_of_st	**Query:** Transitions that are the default entrance of a given state **Purpose:** Returns the default entrances (compound transitions) of the states in the input list **Syntax:** `stm_r_tr_default_of_st (stm_list in_list, int *status));`
stm_r_tr_from_source_st	**Query:** Transitions whose source is the specified state **Purpose:** Returns the transitions whose source is a state appearing in the input list **Syntax:** `stm_r_tr_from_source_st (stm_list in_list, int *status);`
stm_r_tr_to_target_st	**Query:** Transitions whose target is a given state **Purpose:** Returns the transitions whose target is a state appearing in the input list. **Syntax:** `stm_r_tr_to_target_st (stm_list in_list, int *status);`

Input List Type: tr

stm_r_tr_by_attributes_tr	**Query:** **Purpose:** Returns the Transitions in the input list that match the specified attribute name and value. **Syntax:** `stm_r_tr_by_attributes_tr (stm_list in_list, char* attr_name, char* attr_value, int *status);`
stm_r_tr_default_tr	**Query:** Default transition **Purpose:** Returns, of all the transitions in the input list that are default transitions **Syntax:** `stm_r_tr_default_tr (stm_list in_list, int *status);`

Utility Functions

Utility functions enable you to manipulate lists. For example, you could use utility functions to determine whether a particular element exists in a list of Rational Statemate elements. Or, you could sort these lists to make reports easier to read. You can also use utility functions to manipulate strings of characters—to locate string patterns in a given string and to extract portions of strings. Most utility functions for lists can manipulate lists of any item type, but are usually used for lists of Rational Statemate elements.

Utility functions do not extract information from the database; however, some utility functions use database information to complete their operations. These functions enable you to manipulate the information you have already retrieved using single-element or query functions.

Generating Lists

To perform operations on lists, it is sometimes necessary for you to prepare the lists yourself. (Lists are also generated as output from other Dataport functions, such as query functions.) There are two such situations:

- Creating a list from a number of discrete elements
- Loading a list that was stored in the specification database via queries

The created lists are of type `stm_list`. You can store them (using an assignment statement) in a variable of this type to be used in subsequent statements.

Creating a List

Call the following function to create a list of Rational Statemate elements:

```
stm_list_create (e1, e2,..., end_of_list, &status);
```

In this syntax:

- `e1` and `e2`—The element IDs that constitute the list
- `end_of_list`—A constant, defined in `dataport.h`, that signifies the end of the parameter sequence of list items
- `status`—The function return status

Loading a List

You can perform operations on lists that you stored in the workarea using the property sheet. To access these lists, call the following function in your C program:

```
stm_list_load (list_name, &status);
```

In this syntax:

- `list_name`—A string identifying the list that you stored using the property sheet
- `&status`—The function return status

Calling List Utility Functions

Most of the utility functions operate on lists using the following calling sequence:

```
stm_list_operation (list, &status);
```

In this syntax:

- **stm_list**—Designates the function as a Rational Statemate list manipulation function
- **operation**—The kind of list operation performed
- **list**—The list to be operated on
- **status**—The return function status code

The type of value returned by the function depends on the particular function. The returned value can be a list, a Rational Statemate element, a string, or an integer. For example:

```
stm_list_sort_by_name (event_list, &status)
```

This function alphabetically sorts the events in `event_list` according to their names.

The following sections document the utility functions that use a different calling sequence.

Calling Report and Plot Functions

Some utility functions enable you to produce predefined Rational Statemate reports and generate plots of charts. All these functions produce an output addressed to a specific plotter or word processor. This means that the output is written in a specific language, determined by one of the arguments of the function call.

Producing Predefined Reports

There is a set of routines that generate and write Rational Statemate predefined reports of the Reports tool, such as Tree, Property, Interface, and so on. The output contains commands for a word processor that is determined by one of the input arguments. There are different calling sequences for each type of report; the following is the general form:

```
stm_uad_report_name (report_specific_arguments,
   file_name, wp, append, with_header, p_width, p_height)
```

In this syntax:

- `report_name`—One of the predefined report types, such as `tree`.
- `report_specific_arguments`—A list of different arguments for the various types of reports.
- `file_name`—A string that includes the file into which the output is written.
- `wp`—A string that includes the word processor name whose commands are included in the output. The possible values for this parameter are: `troff`, `nroff`, and `interleaf`.
- `append`—A Boolean value which when true indicates that the output is appended to the contents of the output file. This parameter also determines whether or not a page header is omitted.
- `with_header`—A Boolean value which when true indicates that the set-up commands of the work processor is included in the output. These commands usually appear only once in a file to be processed by the word processor.
- `p_width`—The width of the output page in characters. For Interleaf, the width is given in inches.
- `p_height` - The length of the output pages in lines. For Interleaf, the length is given in inches.

For example, the following sequence produces a tree report:

```
stm_uad_rpt_tree (elist, 5, "my_file", "runoff", false,
   true, 80, 60)
```

The tree reportis produced for all elements in `elist`, to a depth of `5` in the hierarchy.

Generating Chart Plots

The following function generates plots of charts:

```
stm_plot
```

Refer to **stm_plot** for more information.

Calling Functions on Reactions

The following two functions enable you to extract the trigger part and the action part from a reaction string, *trigger/action*:

```
stm_trigger_of_reaction (reaction, &status)
stm_action_of_reaction (reaction, &status)
```

They operate on a reaction string that was extracted by single-element functions that return lists of transition labels and static reactions of states.

The return values of the two functions are a string of type `stm_expression`. Because they are defined as `static` in the functions, you should copy them for later use.

Calling Functions of the Workarea

The following four functions deal with the contents of the workarea and enable you to load, unload, or save charts and other configuration items:

- `stm_load`
- `stm_save`
- `stm_unload`
- `stm_unload_all`

In general, use these functions when you want to change the contents of the workarea while running the program, not interactively.

Utility Function Examples

This section shows how to use utility function calls to perform common tasks.

Example 1

To return the number of subactivities existing for the activity `A1`, use the following statements:

```
stm_id          act_id, cntrl_act;
stm_list        list, act_list, cntrl_act_list;
int             status, list_length;
                 .
                 .
                 .
act_id=stm_r_ac("A1", &status);
list=stm_list_create (act_id, end_of_list, &status);
act_list=stm_ac_physical_sub_of_ac (list, &status);
list_length=stm_list_length (act_list, &status);
```

Example 2

To return the activity from Example 1, which is the control activity, use the following statements:

```
                 .
                 .
                 .
cntrl_act_list = stm_ac_control_ac (act_list &status);
cntrl_act = (stm_id)stm_list_first_element
            (cntrl_act_list, &status);
                 .
                 .
                 .
The list cntrl_act_list consists of only one element. Extract the first
element (in this case, the only element) of the list and assign this control
activity's ID to cntrl_act.
```

List of Utility Functions

The following pages document the utility functions. The functions are presented in alphabetical order, as listed in the following table.

Function	Description
stm_action_of_reaction	Extracts the action part of the specified reaction
stm_add_attribute	Enables you to add new attributes to a property element.
stm_backup	Creates a back up of the workarea in a session to a selected directory.
stm_commit_transaction	Closes any open database transactions. It is done explicitly if you are working in the `self_transaction` mode.
stm_decode_color	Decodes the the color's value as it is retrieved from the database.
stm_delete_attributes	Removes an attribute entry from the properties for an element.
stm_dispose_all	Disposes of all the records that were previously allocated and retrieved.
stm_dispose_graphic	Disposes of the record of type `stm_xx_graphic` that was previously allocated and retrieved by the function `stm_r_xx_graphic ()`.
stm_dispose_text	Disposes of the record of type `stm_xx_text`, which was previously allocated and retrieved by the function `stm_r_xx_text()`.
stm_do_command_line	Sends a message to the open Rational Statemate main tool to execute a command line using the same syntax of STMM CLI.
stm_exit_simulation	Allows exit of a Simulation session by profile name.
stm_finish_uad	Completes the information retrieval session so the database is closed for transactions. This is performed after the last Dataport function call.
stm_frm_Reset_id	Resets the Framemaker ID's counter (after calling `stm_plot` using Framemaker).
stm_get_db_status	Returns a number of type `stm_id`. This number is changed when the database is changed.
stm_init_uad	Initializes the database for information retrieval by the Dataport functions, and checks the user access rights and user license.
stm_internal_refresh	Notifies STM that data was changed outside the tool.
stm_list_add_id_element	Adds a new element to a list of element IDs.
stm_list_add_id_element_to_list	Adds a new element to a list of element IDs.

stm_list_add_ptr_element	Adds a new element to a list of element pointers.
stm_list_add_ptr_element_to_list	Adds a new element to a list of element pointers.
stm_list_contains_id_element	Determines whether the specified ID appears in the given list.
stm_list_contains_ptr_element	Determines whether the specified item appears in the given list.
stm_list_create_ids_list	Creates a list of items using their IDs.
stm_list_create_ptr_list	Creates a list of items.
stm_list_create_id_list_with_args	Creates a list of items using the specified IDs.
stm_list_create_ptr_list_with_args	Creates a list of items.
stm_list_delete_id_element	Deletes the specified element from a list of element IDs.
stm_list_delete_id_element_from_list	Deletes the specified element from a list of element pointers.
stm_list_delete_ptr_element_from_list	Deallocates the memory used by the specified list.
stm_list_extraction	Extracts the elements from the input list.
stm_list_extraction_by_chart	Extracts the elements from the input list that belong to the specified chart.
stm_list_extraction_by_chart_id	Extracts the elements from the input list that are defined in the pecified chart.
stm_list_extraction_by_type	Extracts elements of the specified type from the given list of Rational Statemate elements.
stm_list_first_id_element	Returns the first item appearing in the list passed as an input argument.
stm_list_first_ptr_element	Returns the first item appearing in the list passed as an input argument.
stm_list_intersect_ids_lists	Extracts elements that are common to the two specified input lists.
stm_list_intersect_ptr_lists	Extracts elements that are common to the two specified ptr lists.
stm_list_last_id_element	Returns the last item appearing in the list passed as an input argument.
stm_list_last_ptr_element	Returns the last item appearing in the list passed as an input argument.
stm_list_length	Returns the length of the specified list.
stm_list_load	Loads a previously saved list into memory to be used by the program.
stm_list_next_id_element	Returns the next item appearing in the list passed as an input argument.
stm_list_next_ptr_element	Returns the next item appearing in the list passed as an input argument.
stm_list_previous_id_element	Returns the previous item appearing in the list passed as an input argument.

stm_list_previous_ptr_element	Returns the previous item appearing in the list passed as an input argument.
stm_list_purge	Erases the input list's pointers and the list elements.
stm_list_sort	Alphabetically sorts the specified list of strings.
stm_list_sort_alphabetically_by_branches	Alphabetically sorts the specified list of strings by branches.
stm_list_sort_alphabetically_by_levels	Alphabetically sorts the specified list of strings by levels.
stm_list_sort_by_attr_value	Sorts the specified list of Rational Statemate elements by the value of the given attribute.
stm_list_sort_by_branches	Sorts the specified list of hierarchical Rational Statemate elements by branches.
stm_list_sort_by_chart	Alphabetically sorts the input list of named Rational Statemate elements, by the name of the chart to which they belong.
stm_list_sort_by_levels	Alphabetically sorts the input list of named Rational Statemate elements, by the name of the chart to which they belong.
stm_list_sort_by_name	Sorts the specified list of Rational Statemate elements alphabetically by name.
stm_list_sort_by_synonym	Sorts the specified list of Rational Statemate elements alphabetically by their synonyms.
stm_list_sort_by_type	Sorts the specified list of Rational Statemate elements by type.
stm_list_subtract_ids_lists	Creates a new list of those elements of the first input list that are not found in the second input list.
stm_list_subtraction_ptr_lists	Creates a new list of those elements of the first input list that are not found in the second input list.
stm_list_subtract_ids_lists	Creates a new list of those elements of the first input list that are not found in the second input list.
stm_list_union_ids_lists	Merges the elements of two specified ids lists.
stm_list_union_ptr_lists	Merges the elements of two specified ptr lists.
stm_load	Loads a chart file (or any other configuration item file) into the current workarea.
stm_multiline_to_one	Converts the specified multiline string (with new lines) to a one-line string (without the new lines).
stm_multiline_to_strings	Converts the specified multiline expression to a list of strings.
stm_open_truth_table	Opens a Truth Table that is connected to the specified element and highlights the specified line in it.
stm_plot	Generates a plot file with the indicated parameters, such as plot size, output device, and so on.
stm_plot_ext	May return one of two status codes.
stm_plot_hyper_exp	Generates the hyperlinks in a sequence diagram.
stm_plot_with_autonumber	Prints a sequence diagram with numbered scenarios.

stm_plot_with_break	Breaks a sequence diagram across multiple pages.
stm_plot_with_headerline	Prints a sequence diagram with the names of lifelines on every page.
stm_r_global_interface_report	Return the global interface report for the elements in the input list
stm_r_local_interface_report	Return the local interface report for the elements in the input list.
stm_run_simulation_profile	Sends a message to Rational Statemate to open and execute a Simulation profile by the name passed as a parameter.
stm_save	Saves a chart (or any other configuration item file) from the current workarea to an external file.
stm_start_transaction	Enables transaction operations on the database.
stm_start_transaction_rw	Enables read/write transaction operations on the database.
stm_trigger_of_reaction	Returns the trigger part of a reaction (label of transition or static reaction).
stm_uad_attribute	Writes the predefined attribute report to the specified output file.
stm_uad_dictionary	Writes the predefined property report to the specified output file.
stm_uad_interface	Writes the predefined attribute report to the specified output file.
stm_uad_list	Writes the predefined list report to the specified output file.
stm_uad_n2	Writes the predefined N2-chart report to the specified output file.
stm_uad_protocol	Writes the predefined protocol report to the specified output file.
stm_uad_resolution	Writes the predefined resolution report to the specified output file.
stm_uad_state_interface	Writes the predefined state interface report to the specified output file.
stm_uad_structure	Writes the predefined structure report to the specified output file.
stm_uad_tree	Writes the predefined tree report to the specified output file.
stm_unload	Unloads (deletes from the current workarea) a chart or any other configuration item file.
stm_unload_all	Unloads all charts from the current workarea and clears all database fields.

stm_action_of_reaction

Extracts the action part of the specified reaction (the label of the transition or static reaction). The syntax of a reaction is *trigger/action*.

- The reaction is achieved by the single-element function `stm_r_st_reactions` or `stm_r_tr_label`s.
- The function returns an empty string when the action is missing.

Function Type

`stm_expression`

Syntax

```
stm_action_of_reaction (reaction, &status)
```

Arguments

Argument	Input/Output	Type	Description
`reaction`	In	`char *`	The reaction to decompose
`status`	Out	`int`	The function status code

Status Codes

- `stm_success`

Example

To list all actions that are triggered when `S1` is in a static reactions (assume that `S1` has several static reactions), include the following calls in your program:

```
stm_id          st_id;
    int             status;
    stm_list        reactions;
    stm_expression  rct;

    st_id = stm_r_st ("S1", status);
    reactions = stm_r_st_reactions (st_id, status);
    printf ("\n Actions of reactions in S1:");
    for (rct = (string)
        stm_list_first_element (reactions, &status);
        status == stm_success;
        rct = (string)
        stm_list_next_element (reactions, &status))
        printf ("\n %S", stm_action_of_reaction (
            rct, &status));
```

stm_add_attribute

Enables you to add new attributes to a properties element.

- Initialization of the program must be performed in `self_transaction` mode; `stm_init_uad` with `self_transaction`.
- Use `stm_start_transaction_rw()` instead of `stm_start_transaction()`.
- Use `stm_commit_transaction()` at the end of each transaction.

Function Type

```
void
```

For Elements

activity	ac
block	bl
chart	ch
condition	co
data-item	di
data-store	ds
event	ev
field	fd
function	fn
information-flow	if
module	md
state	st
transition	
user-defined type	dt

Syntax

```
stm_add_attribute (id, attr_name, attr_val, &status)
```

Arguments

Argument	Input/ Output	Type	Description
id	In	stm_id	The element ID
attr_name	In	char *	The attribute name. This name must be uppercase, alpha-numeric, or empty (with a maximum length of 64).
Attr_value	In	char *	The attribute value. The value can be any text string, with a maximum length of 300.
status	Out	int	Function status code

Status Codes

- stm_success
- stm_id_out_of_range
- stm_id_not_found
- stm_illegal_attribute_name
- stm_illegal_attribute_value
- stm_duplicate_attribute_pair
- stm_not_in_rw_transaction

Example

The following example inserts an attribute into state S1.

```
#include <dataport.h>

main(argc, argv)
char **argv;
int    argc;
{
   int      status;
   stm_id    state_id;

   if (argc!=3)
   {
       printf ("Usage %s PROJECT workarea\n", argv[0]);

      exit (0);
   }
   if (!stm_init_uad(argv[1], argv[2], self_transaction,
       &status))
   {
      printf ("can't open workarea %s\n", argv[2]);
      exit(1);
   }
   stm_start_transaction_rw ();
   state_id = stm_r_st ("S1", &status);
```

```
        stm_add_attribute (state_id, "FRED", "A Value",
           &status);
        stm_commit_transaction();
    }
```

stm_backup

Creates a back up of the workarea in a session to a selected directory.

Syntax

```
stm_backup (char* destination, char* mess, int *status)
```

Arguments

Argument	Input/ Output	Type	Description
`char*`	In	`destination`	The destination directory for the backup.
`char*`	Out	`mess`	Messages related to the backup operation.
`int`	Out	`status`	The status of the query (`stm_error_in_backup` or `stm_success`).
`status`	Out	`int`	Function status code

stm_commit_transaction

Closes any open database transactions. It is done explicitly if you are working in the `self_transaction` mode.

Note

In `self_transaction mode`, each start-transaction must have a corresponding commit. In `automatic_transaction mode`, the commit is performed automatically.

Function Type

```
void
```

Syntax

```
stm_commit_transaction()
```

Example

To close transactions, use the following statement:

```
            .
            .
stm_r...                    -- a retrieval function
stm_commit_transaction();
            .
            .
```

stm_decode_color

Decodes the color's value as it is retrieved from te datavbase, suing APIs like stm_r_ac_graphic(). The stm_decode_color API receives the color value received from the database and converts it into a structure (struct stm_color_all) that has the true color values in its fields, as well as the fill style of the graphical shape.

Syntax

```
stm_dolor_all stm_decode_color (unsigned long color, int *status
```

stm_delete_attributes

Removes an attribute entry from the properties for an element.

- Initialization of the program must be done in `self_transaction` mode, that is use `stm_init_uad with self_transaction`.
- Use s`tm_start_transaction_rw()`, instead of `stm_start_transaction()`.
- Use `stm_commit_transaction()` at the end of each transaction.

Function Type

```
void
```

For Elements

action	an
activity	ac
block	bl
chart	ch
condition	co
data-item	di
data-store	ds
event	ev
field	fd
function	fn
information-flow	if
module	md
state	st
transition	
user-defined type	dt

Syntax

```
stm_delete_attributes (id, attr_name, attr_val, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`id`	In	`stm_id`	The element ID.
`attr_name`	In	`string`	The attribute name. This name must be uppercase, alpha-numeric, or empty (with a maximum length of 31).
`attr_value`	In	`string`	The attribute value. The value can be any text string, with a maximum length of 300.
`status`	Out	`int`	Function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_id_not_found`
- `stm_illegal_attribute_name`
- `stm_illegal_attribute_value`

- `stm_elements_without_attributes`
- `stm_attribute_cannot_be_deleted`
- `stm_not_in_rw_transaction`

stm_dispose_all

Disposes of all the records that were previously allocated and retrieved.

Function Type

```
void
```

Syntax

```
stm_dispose_all (gen_all_ptr)
```

Arguments

Argument	Input/ Output	Type	Description
`gen_all_ptr`	In	`void **`	A pointer to the data to be disposed of.

stm_dispose_graphic

Disposes of the record of type `stm_xx_graphic` that was previously allocated and retrieved by the function `stm_r_xx_graphic ()`.

Function Type

`void`

Syntax

```
stm_dispose_graphic (elm_graphic_data)
```

Arguments

Argument	Input/ Output	Type	Description
`elm_graphic_data`	In	`stm_xx_graphic_ptr`	A pointer to the graphical data of the element to be disposed.

Status Codes

- `stm_dispose_rt_allocation`
- `stm_dispose_rt_tex`

stm_dispose_text

Disposes of the record of type `stm_xx_text`, which was previously allocated and retrieved by the function `stm_r_xx_text()`.

Function Type

`void`

Syntax

```
stm_dispose_text (elm_textual_data)
```

Arguments

Argument	Input/ Output	Type	Description
`elm_textual_data`	In	`stm_xx_text_ptr`	A pointer to the textual data to be disposed.

stm_do_command_line

Sends a message to the open Rational Statemate main tool to exectue a command line using the same syntax of STMM CLI.

There is no need to include the '-wa' and '-project' switches - the CLI command will be executed on the workarea currently open by Rational Statemate main.

A new switch '-queue' can be used in the command line for this API, in conjunction with one of the supported CLI tools (e.g. '-simulation', '- checkmodel', and so forth), to allow queuing of several tool profiles. The queued profiles will be executed consequentially (When running Simulation, the input_file (specified after "-i") should have a 'quit' command at the end, to terminate the simulation session and allow execution of the next profile in the queue).

Function Type

```
stm_boolean
```

Syntax

```
stm_do_command_line(string command_line , int* status);
```

stm_exit_simulation

Allows exit of a Simulation session by profile name.

Function Type

```
stm_boolean
```

Syntax

```
stm_exit simulation (profile_name, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`profile_name`	In	`char`	The name of the Simulation.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_error_in_open_socket_to_statemate`

stm_exit_graphic_editor

Allows exit of a graphic-editor session by chart ID.

Function Type

```
stm_boolean
```

Syntax

```
stm_exit graphic_editor (chart_id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`chart_id`	In	`stm_chart_id`	ID of chart.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_error_in_open_socket_to_statemate`

stm_finish_uad

Completes the information retrieval session so the database is closed for transactions. This is performed after the last Dataport function call.

Function Type

```
void
```

Syntax

```
stm_finish_uad ()
```

Example

The information retrieval process in a program concludes with the following:

```
stm_finish_uad();
                .
                .
```

stm_frm_Reset_id

Resets the Framemaker ID's counter (after calling `stm_plot` using Framemaker).

Function type:

```
void
```

Syntax:

```
stm_frm_reset_id()
```

stm_get_db_status

Returns a number of type `stm_id`. This number is changed when the database is changed.

Note

The return type is `int`, which is the same type as `stm_id`.

Syntax

```
stm_get_db_status (stm_id)
```

stm_init_uad

Initializes the database for information retrieval by the Dataport functions, and checks the user access rights and user license.

The `stm_init_uad` function automatically changes the current directory to the workarea directory. All references to files inside the program have to take this into account. When the program terminates, it does *not* return to the original directory.

Function Type

```
stm_boolean
```

Syntax

```
stm_init_uad (proj_name, workarea, trans_mode, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`proj_name`	In	`char *`	The name of the project.
`workarea`	In	`char *`	The pathname to the workarea directory.
`trans_mode`	In	`stm_transaction`	The transaction mode. The possible values are as follows: • **automatic_transaction** - An implicit `start_transaction` is performed when you initialize and an implicit `commit_transaction` is performed when you finish the retrieval process • **self_transaction** - You can control when the `start_transaction` and `commit_transaction` is performed for an accurate picture of the database when the functions are used.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_no_updated_pmdb`
- `stm_no_updated_projdb`
- `stm_no_legal_operator`
- `stm_deadlock`
- `stm_not_member_of_project`
- `stm_nonexistent_project`
- `stm_empty_file_of_licensed_host`
- `stm_no_file_of_licensed_host`
- `stm_cannot_chdir_to_work_area`
- stm_workarea_does_not_exist

Example

To initialize the database, use the following statements:

```
int      success, status;
           .
           .
success = stm_init_uad ("A5S700", "/a5/general",
            automatic_transaction, &status);
           .
           .
           .
This function initializes the workarea for project A5S700, which is
found in the directory /a5/general. The transactions are started
automatically and finished automatically by the init and finish
```

```
functions. The status should be checked in case the function fails and
returns false.
```

stm_internal_refresh

Notifies Rational Statemate that data was changed outside the tool, such that in the next refresh, values are recalculated. The function return value is stm_true and the status value is stm_success if the message was successfully sent to Rational Statemate.

Function Type

```
stm_boolean
```

Syntax

```
stm_internal_refresh (int *status)
```

stm_list_add_id_element

Adds a new element to a list of element IDs.

Function Type

```
stm_list
```

Syntax

```
stm_list_add_id_element (list, element, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list`	The list of Rational Statemate elements.
`element`	In	`stm_list_id_elemen`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_nil_list`

Example

To add a new element use the following statement (the element ID number has been assigned to `st_id`):

```
stm_id       st_id;
     stm_list     st_list;
     int          status;
                  .
                  .
     st_id = stm_r_st ("S1", &status);
     st_list = stm_list_add_id_element (st_list, st_id,
        &status);
                  .
                  .
```

stm_list_add_id_element_to_list

Adds a new element to a list of element IDs.

Function Type

`void`

Syntax

```
stm_list_add_id_element_to_list (list, element, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In/out	`stm_list`	The list of Rational Statemate elements.
`element`	In	`stm_list_id_elemen`	The element ID.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`
- `stm_nil_list`

Example

To add a new element use the following statement (the element ID number has been assigned to `st_id`):

```
stm_id       st_id;
     stm_list     st_list;
     int          status;
                  .
                  .
     st_id = stm_r_st ("S1", &status);
     stm_list_add_id_element_to_list (st_list, st_id,
        &status);
                  .
                  .
```

stm_list_add_ptr_element

Adds a new element to a list of element pointers.

Function Type

`stm_list`

Syntax

```
stm_list_add_ptr_element (list, element, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list`	The list of Rational Statemate elements.
`element`	In	`stm_list_ptr_elm`	The element pointer.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`

Example

To add a new elemen,t use the following statement (the element id has been assigned to `st_id` and the element name has been assigned to `st_name`):

```
stm_id st_id;stm_element_name st_name;stm_list st_list;
int status;..st_name = stm_r_st_name(st_id, &status);
st_list = stm_list_add_ptr_element (st_list, st_name, &status);
```

```
.
.
```

stm_list_add_ptr_element_to_list

Adds a new element to a list of element pointers.

Function Type

```
void
```

Syntax

```
stm_list_add_ptr_element_to_list (list, element, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In/out	`stm_list`	The list of Rational Statemate elements.
`element`	In	`stm_list_ptr_elm`	The element pointer.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`

Example

To add a new elemen,t use the following statement (the element id has been assigned to `st_id` and the element name has been assigned to `st_name`):

```
stm_id st_id;stm_element_name st_name;stm_list st_list;
int status;..st_name = stm_r_st_name(st_id, &status);
st_list = stm_list_add_ptr_element (st_list, st_name, &status);
.
.
```

stm_list_contains_id_element

Determines whether the specified ID appears in the given list.

Function Type

```
int
```

Syntax

```
stm_list_contains_id_element (list, item, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list`	The list to search.
`item`	In	`stm_list_id_elm`	The item to look for.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`

Example

To check a list of Rational Statemate elements for the presence of activity `A1` and assign the elements to the list `elmnt_list`, use the following statement:

```
stm_list      elmnt_list;
   stm_id        st_id;
   int           status;
         .
         .
   if (stm_list_contains_id_element (elmnt_list,
      (stm_list_id_elm)st_id, &status))
         .
         .
If A1 appears in elmnt_list, the statements following the if statement
are executed. Note that the ID of A1 (and not its name) is passed to
the function.
```

stm_list_contains_ptr_element

Determines whether the specified item appears in the given list.

Function Type

```
int
```

Syntax

```
stm_list_contains_ptr_element (list, item, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list`	The list to search.
`item`	In	`stm_list_ptr_elm`	The item to look for.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`

Example

To check a list of Rational Statemate elements for the presence of activity `A1` and assign the elements to the list `elmnt_list`, use the following statement:

```
stm_list            elmnt_list;
   stm_list_ptr_elm    st_name;
   int                 status;
        .
        .
   if (stm_list_contains_ptr_element (elmnt_list,
      (stm_list_elm)st_name, &status))
        .
        .
If A1 appears in elmnt_list, the statements following the if statement
are executed. Note that the name of A1 (and not its ID) is passed to
the function.
```

stm_list_create_ids_list

Creates a list of items using their IDs. The number of arguments varies according to the number of elements to be included in the list. The list is terminated by the predefined constant `end_of_id_list`.

Function Type

`stm_list`

Syntax

```
stm_list_create_ids_list (item1, item2..., end_of_id_list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`item1`	In	`stm_id`	The first element in the list.
`item2`	In	`stm_id`	The next element in the list (and so on).
`end_of_id_list`	In	`stm_id`	The end of the list.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`

Example

To create a list that contains the activity named `A2` and the state `S8`, retrieve their IDs from the database and create the list using the following statements:

```
stm_id          ac_id, st_id;
int             status;
stm_list        list;
                .
                .
ac_id = stm_r_ac ("A2", &status);
st_id = stm_r_st ("S8", &status);
list = stm_list_create_ids_list (st_id, ac_id, end_of_id_list,
   &status);
                .
                .
```

stm_list_create_ptr_list

Creates a list of items. The number of arguments varies according to the number of elements to be included in the list. The list is terminated by the predefined constant `end_of_ptr_list`.

Function Type

`stm_list`

Syntax

```
stm_list_create_ptr_list (item1, item2..., end_of_list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`item1`	In	`stm_list_ptr_id`	The first element in the list.
`item2`	In	`stm_list_ptr_id`	The next element in the list (and so on).
`end_of_ptr_list`	In	`char*`	The end of the list.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`

Example

To create a list that contains the names of the activity named `A2` and the state `S8`, create the list using the following statements:

```
stm_id                ac_id, st_id;
int                   status;
stm_element_name      ac_name,st_name
stm_list              list;
                      .
                    .
ac_name = stm_r_ac_name ("ac_id", &status);
st_name = stm_r_st_name ("st_id", &status);
list = stm_list_create_ptr_list (st_name, ac_name, end_of_ptr_list,
&status);
                      .
                  .
```

stm_list_create_id_list_with_args

Creates a list of items using the specified IDs.

Function Type

```
stm_list
```

Syntax

```
stm_list_create_id_list_with_args (args, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`args`	In	`stm_id*`	Array of stm_id's to insert to the list
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_id_out_of_range`

stm_list_create_ptr_list_with_args

Creates a list of items.

Function Type

`stm_list`

Syntax

```
stm_list_create_ptr_list_with_args (args, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`args`	In	`stm_list_ptr_elm*`	Array of stm_list_ptr_elm's to insert to the list.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`

stm_list_delete_id_element

Deletes the specified element from a list of element IDs.

Function Type

`stm_list`

Syntax

```
stm_list_delete_id_element (list, element, &status)
```

Arguments

Argument	Input/Output	Type	Description
`list`	In	`stm_list`	The list of Rational Statemate elements
`element`	In	`stm_id`	The ID of the element to delete
`status`	Out	`int`	The function status code

Status Codes

- `stm_success`
- `stm_list_element_does_not_exist`
- `stm_nil_list`

Example

To remove the element ID `st_id` from a list, use the following statement:

```
stm_list   st_list;
stm_id     st_id;
int        status;
           .
           .
st_list = stm_list_delete_id_element (st_list, st_id,
    &status);
           .
           .
```

stm_list_delete_id_element_from_list

Deletes the specified element from a list of element IDs.

Function Type

```
void
```

Syntax

```
stm_list_delete_id_element_from_list (list, element, &status)
```

Arguments

Argument	Input/Output	Type	Description
`list`	In/out	`stm_list`	The list of Rational Statemate elements
`element`	In	`stm_id`	The ID of the element to delete
`status`	Out	`int`	The function status code

Status Codes

- `stm_success`
- `stm_list_element_does_not_exist`
- `stm_nil_list`

Example

To remove the element ID `st_id` from a list, use the following statement:

```
stm_list   st_list;
stm_id     st_id;
int        status;
           .
           .
stm_list_delete_id_element_from_list (st_list, st_id,
    &status);
           .
           .
```

stm_list_delete_ptr_element

Deletes the specified element from a list of element pointers.

Function Type

`stm_list`

Syntax

```
stm_list_delete_ptr_element (list, element, &status)
```

Arguments

Argument	Input/Output	Type	Description
`list`	In	`stm_list`	The list of Rational Statemate elements
`element`	In	`stm_list_ptr_elm`	The element to delete
`status`	Out	`int`	The function status code

Status Codes

- `stm_success`
- `stm_list_element_does_not_exist`
- `stm_nil_list`

Example

To remove the element `st_ptr` from a list, use the following statement:

```
stm_list            st_list;
stm_list_ptr_elm    st_ptr;
int                 status;
                .
                .
st_list = stm_list_delete_ptr_element (st_list, st_ptr,
    &status);
                .
                .
```

stm_list_delete_ptr_element_from_list

Deletes the specified element from a list of element pointers.

Function Type

```
void
```

Syntax

```
stm_list_delete_ptr_element_from_list (list, element, &status)
```

Arguments

Argument	Input/Output	Type	Description
`list`	In/out	`stm_list`	The list of Rational Statemate elements
`element`	In	`stm_list_ptr_elm`	The element to delete
`status`	Out	`int`	The function status code

Status Codes

- `stm_success`
- `stm_list_element_does_not_exist`
- `stm_nil_list`

Example

To remove the element `st_ptr` from a list, use the following statement:

```
stm_list            st_list;
stm_list_ptr_elm    st_ptr;
int                 status;
            .
            .
stm_list_delete_ptr_element_from_list (st_list, st_ptr,
    &status);
            .
            .
```

stm_list_destroy

Deallocates the memory used by the specified list.

- The returned value is false when the input list is a nil list.
- After the function operation, `list` cannot be used as an input argument in list functions.
- The function does *not* free the entire memory space used by the list when the list members occupy more than a single memory location. For example, with strings only the pointers to the strings are destroyed. (Compare with `stm_list_purge`.)

Function Type

```
int
```

Syntax

```
stm_list_destroy (list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list`	The list to destroy
`status`	Out	`int`	The function status code

Status Codes

- `stm_success`
- `stm_nil_list`

Example

To use a list and then make its associated space available, use the following statements:

```
stm_list    list;
int         status;
            .
            .
if (stm_list_destroy (list, &status))
    printf ("list destroyed");
                    .
                    .
```

stm_list_extraction

Extracts the elements from the input list.

Function Type

```
stm_list
```

Syntax

```
stm_list_extraction (ex_type, list, &status)
stm_list_extraction (ex_type, el_list, status)
```

Arguments

Argument	Input/ Output	Type	Description
`ex_type`	In	`int`	The type to extract from the list.
`list`	In	`stm_list`	The list of Rational Statemate elements.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`

stm_list_extraction_by_chart

Extracts the elements from the input list that belong to the specified chart.

Function Type

`stm_list`

Syntax

```
stm_list_extraction_by_chart (chart_name, list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`chart_name`	In	`stm_element_name`	The name of the chart.
`list`	In	`stm_list`	The list of Rational Statemate elements.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_illegal_name`
- `stm_name_not_found`

Example

To extract a list of all Rational Statemate elements that belong to the Statechart `S8` from the input list `elem_list`, use the following statements:

```
stm_list   elem_list,S8_elements;
           .
           .
S8_elements = stm_list_extraction_by_chart ("S8",
                elem_list, &status);
           .
           .
```

stm_list_extraction_by_chart_id

Extracts the elements from the input list that are defined in the pecified chart.

Function Type

```
stm_list
```

Syntax

```
stm_list_extraction_by_chart_id (stm_id chart_id, stm_list list, int* status)
```

Arguments

Argument	Input/ Output	Type	Description
`chart_id`	In	`stm_chart_id`	Chart from which to extract elements.
`list`	In	`stm_list`	The list of Rational Statemate elements.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`

stm_list_extraction_by_type

Extracts elements of the specified type from the given list of Rational Statemate elements.

Function Type

`stm_list`

Syntax

```
stm_list_extraction_by_type (element_type, list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`element_type`	In	`int`	The type to look for. `element_type` is one of the possible values of the enumerated type `stm_element_type`. The values of this type usually take the form `stm_element_type` (for example, `stm_state`, `stm_event`, and so on).
`list`	In	`stm_list`	The list of elements (mixed types).
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_illegal_extract_type`

Example

To extract a list of all the activities appearing in a list of Rational Statemate elements, the input list is assigned to the variable `act_list` using the following statement:

```
stm_list    list, act_list;
int         status;
            .
            .
act_list = stm_list_extraction_by_type (stm_activity,
    list, &status);
            .
            .
```

Note that `stm_activity` is a constant value, not a variable.

stm_list_first_id_element

Returns the first item appearing in the list passed as an input argument. This function may be applied to lists containing Rational Statemate ids.

Function Type

```
stm_list_id_elm
```

Syntax

```
stm_list_first_id_element (list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list`	The list of ids.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_list_ielement_does_not_exist`

Example

To find out which activity in the list of activities assigned to the variable `act_list` is a control activity, use the following statements:

```
stm_list     cntrl_act_list, act_list;
stm_id       cntrl_act;
int          status;
             .
             .
cntrl_act_list = stm_ac_control_ac (act_list, &status);
cntrl_act = (stm_id)
stm_list_first_id_element (cntrl_act_list, &status);
             .
             .
```

First, extract all control activities from the input list `act_list`. The list `cntrl_act_list` consists of only one element. Extract the first element (in this case the only element) of the list and assign this control activity's ID to `cntrl_act`.

stm_list_first_ptr_element

Function Type

```
stm_list_elm
```

Description

Returns the first item appearing in the list passed as an input argument. This function may be applied to lists containing pointers.

Syntax

```
stm_list_first_ptr_element (list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list`	The list of items.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_list_element_does_not_exist`

stm_list_intersect_ids_lists

Function Type

`stm_list`

Description

Extracts elements that are common to the two specified input lists.

Note

The two lists must be both lists of stm_id's

Syntax

```
stm_list_intersect_ids_lists (list1, list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list1`	In	`stm_list`	The first list to compare
`list2`	In	`stm_list`	The second list to compare
`status`	Out	`int`	The function status code

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_cannot_compare_lists`

Example

To produce the intersection list `list3` from `list1` and `list2`, use the following statements:

```
stm_list     list1,list2,list3;
int          status;
             .
             .
list3 = stm_list_intersect_ids_list (list1, list2, &status);
             .
             .
```

stm_list_intersect_ptr_lists

Function Type

`stm_list`

Description

Extracts elements that are common to the two specified ptr lists.

Note

The two lists must be both lists of stm_list_ptr_elm's.

Syntax

```
stm_list_intersect_ptr_lists (list1, list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list1`	In	`stm_list`	The first list to compare
`list2`	In	`stm_list`	The second list to compare
`status`	Out	`int`	The function status code

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_cannot_compare_lists`

Example

To produce the intersection list `list3` from `list1` and `list2`, use the following statements:

```
stm_list     list1,list2,list3;
int          status;
             .
             .
list3 = stm_list_intersect_ptr_lists (list1, list2, &status);
             .
             .
```

stm_list_last_id_element

Function Type

```
stm_list_id_elm
```

Description

Returns the last item appearing in the list passed as an input argument. This function can be applied to lists containing stm_id's.

Syntax

```
stm_list_last_id_element (list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
list	In	stm_list	The list of items. Items in the input list must be stm_id's.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_nil_list
- stm_list_element_does_not_exist

Example

To find the last item in the list of states, S1, S2, S3, and S4 assigned to the variable state_list, use the following statements:

```
stm_list    state_list;
stm_id      state_id;
int         status;
            .
            .
state_id = (stm_id) stm_list_last_id_element (state_list,
    &status);
printf ("The last state in the list is: %s\n",
    stm_r_st_name (state_id, &status));
            .
            .
```

stm_list_last_ptr_element

Function Type

```
stm_list_ptr_elm
```

Description

Returns the last item appearing in the list passed as an input argument. This function can be applied to lists containing stm_list_ptr_elm's.

Syntax

```
stm_list_last_ptr_element (list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list`	The list of items. Items in the input list must be stm_list_ptr_elm's.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_list_element_does_not_exist`

stm_list_length

Function Type

`int`

Description

Returns the length of the specified list.

Syntax

```
stm_list_length (list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list`	The list of items. Items in the input list can be of any element type.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`

Example

To verify that an extracted list of all the events from the database whose name matched a certain pattern (assigned to `st_list`) contains no more than 3000 elements, use the following statements:

```
stm_list   st_list;
int        status;
           .
           .
if (stm_list_length (st_list, &status) < 3000)
           .
           .
```

stm_list_load

Function Type

`stm_list`

Description

Loads a previously saved list into memory to be used by the program. Lists can be saved by several tools in Rational Statemate, such as search.

Syntax

```
stm_list_load (list_name, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list_name`	In	`char *`	The list to load
`status`	Out	`int`	The function status code

Status Codes

- `stm_success`
- `stm_no_such_list`

Example

To load a list saved as the name `EXT_SIGNALS` into memory, use the following call:

```
stm_list  list;
int       status;
          .
          .
list = stm_list_load ("EXT_SIGNALS", &status);
          .
          .
```

Once a list has been loaded, you can operate on it using any of the Dataport functions.

stm_list_next_id_element

Function Type

```
stm_list_id_elm
```

Description

Returns the next item appearing in the list passed as an input argument. This function can be applied to lists containing stm_list_id_elm's..

Syntax

```
stm_list_next_id_element (list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list`	The list of items. Items in the input list must be stm_list_id_elm's.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_list_element_does_not_exist`

Example

To find the next element in the list of states, `S1`, `S2`, `S3`, and `S4` (appearing in this order) assigned to the variable `state_list`, use the following statements:

```
stm_list  state_list;
stm_id    state_id;
int       status;
          .
          .
state_id =  (stm_id)  stm_list_first_id_element (
   state_list, &status);
printf ("The first state in the list is: %s\n",
   stm_r_st_name (state_id, &status));
state_id = (stm_id) stm_list_next_id_element (state_list,
   &status);
printf ("The second state in the list is: %s\n",
   stm_r_st_name (state_id, &status));
          .
          .
```

This function can be used in a `for` loop (in conjunction with `stm_list_first_id_element`) to perform operations on all elements in the list

stm_list_next_ptr_element

Function Type

```
stm_list_ptr_elm
```

Description

Returns the next item appearing in the list passed as an input argument. This function can be applied to lists containing stm_list_ptr_elm's..

Syntax

```
stm_list_next_ptr_element (list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list`	The list of items. Items in the input list must be stm_list_ptr_elem's..
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_list_element_does_not_exist`

stm_list_previous_id_element

Function Type

```
stm_list_id_elm
```

Description

Returns the previous item appearing in the list passed as an input argument. This function can be applied to lists containing stm_list_id_elm's.

Note that "previous" refers to the item physically located before the current item in the list. The "current" item is determined using the utility function `stm_list_last_id_element`.

Syntax

```
stm_list_previous_id_element (list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list`	The list of items. Items in the input list must be stm_list_id_elm's.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_list_element_does_not_exist`

Example

In the list of states `S1`, `S2`, `S3`, and `S4` (appearing in this order) assigned to the variable `state_list`, locate the state `S4` by calling `stm_list_last_ids_element`; `S4` becomes the current item. To find the previous element in the list, use the following statements:

```
stm_list    state_list;
stm_id      state_id;
int         status;
            .
            .
state_id = (stm_id) stm_list_last_element (
   state_list, &status);
state_id =  (stm_id)  stm_list_previous_id_element (
   state_list, &status);
printf ("State of interest is: %s\n",
   stm_r_st_name (state_id, &status));
            .
            .
```

This function can be used in a `for` loop (in conjunction with `stm_list_last_id_element`) to perform operations on all elements in the list in reverse order.

stm_list_previous_ptr_element

Function Type

```
stm_list_ptr_elm
```

Description

Returns the previous item appearing in the list passed as an input argument. This function can be applied to lists containing stm_list_id_elm's.

Note that "previous" refers to the item physically located before the current item in the list. The "current" item is determined using the utility function `stm_list_last_ptr_element`.

Syntax

```
stm_list_previous_ptr_element (list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list`	The list of items. Items in the input list cmust be stm_list_ptr_elm's.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_list_element_does_not_exist`

stm_list_purge

Erases the input lis pointers and the list elements.

Note

- This function is intended for use only with lists of strings. You should not purge a list of Rational Statemate elements (IDs) because it can cause serious problems in your program. (Compare with `stm_list_destroy`.)
- The returned value is `false` when the input list is a nil list.
- After the function operation, `list` cannot be used as an input argument in list functions.

Function Type

```
int
```

Syntax

```
stm_list_purge (list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list`	The list to purge.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`

Example

To deallocate the space associated with the list of names `name_list` (and the strings included in this list are not referenced by other pointers in your program), use the following statements:

```
.
    .
list = stm_r_st_attr_name (st_id, &status);
if (stm_list_purge (list, &status))
   printf ("list purged");
      .
      .
```

stm_list_sort

Function Type

`stm_list`

Description

Alphabetically sorts the specified list of strings.

Syntax

```
stm_list_sort (list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list_elm`	The list of strings to be sorted
`status`	Out	`int`	The function status code

Status Codes

- `stm_success`
- `stm_nil_list`

Example

To sort a list of strings alphabetically, use the following statement:

```
stm_list  unsorted_list, sorted_list;
int       status;
          .
          .
sorted_list = stm_sort_list (unsorted_list, &status);
          .
          .
```

stm_list_sort_alphabetically_by_branches

Sorts a list of hierarchical Rational Statemate elements by branch. Elements appearing within each branch are ordered alphabetically.

This function is relevant only for a list of hierarchical elements. If the function is applied to a list of non-hierarchical elements, `status` receives the value `stm_elements_not_hierarchical`.

Function Type

```
stm_list
```

Syntax

```
stm_list_sort_alphabetically_by_branches (list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	Input	`stm_list`	The list of Rational Statemate hierarchical elements
`status`	Input	`int`	The function status code

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_elements_not_hierarchical`

stm_list_sort_alphabetically_by_levels

Sorts a list of hierarchical Rational Statemate elements by level. Elements appearing within each level are ordered alphabetically.

This function is relevant only for a list of hierarchical elements. If the function is applied to a list of non-hierarchical elements, `status` receives the value `stm_elements_not_hierarchical`.

Function Type

```
stm_list
```

Syntax

```
stm_list_sort_alphabetically_by_levels (list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	Input	`stm_list`	The list of Statemate hierarchical elements
`status`	Input	`int`	The function status code

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_elements_not_hierarchical`

stm_list_sort_by_attr_value

Sorts the specified list of Rational Statemate elements by the value of the given attribute.

Note that the function receives and returns a list of element IDs, *not* a list of element names.

Function Type

```
stm_list
```

Syntax

```
stm_list_sort_by_attr_value (list, attr_name, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list LIST OF ELEMENT`	The list of Rational Statemate element IDs to be sorted.
`attr_name`	In	`stm_attr_name STRING`	The attribute to use as the sorting key.
`status`	Out	`int INTEGER`	The function status code. The function returns the status code. `stm_elements_without_attributes` if you apply this function to a list of elements that do not have the specified attribute.

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_elements_without_attributes`

Example

To sort activities by the name of an attribute called code, use the following function calls:

```
stm_list   act_list, ord_act_list;
stm_id     el;
int        status;
           .
           .
ord_act_list = stm_list_sort_by_attr_value (
   act_list,"code", &status);
printf("\n Ordered list of activities:");
           .
           .
```

This example prints a particular list of activities from the database. Assume you extracted the activities of interest using single-element and query functions and built a list of such activities. This list is assigned to the variable `act_list`.

stm_list_sort_by_branches

When Rational Statemate uses this function to sort a specified list of elements by branches and it encounters two or more charts at the same level of hierarchy, it sorts them alphabetically by name.

Function Type

`stm_list`

Description

Sorts the specified list of hierarchical Rational Statemate elements by branches.

Note

This function is relevant only for a list of hierarchical elements. If the function is applied to a list of non-hierarchical elements, `status` receives the value `stm_elements_not_hierarchical`.

Syntax

```
stm_list_sort_by_branches (list, &status)
```

Arguments

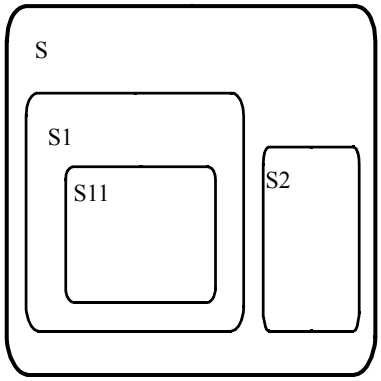
Argument	Input/ Output	Type	Description
`list`	In	`stm_list`	The list of Rational Statemate elements.
`status`	Out	`int`	The function status code.

Status Codes

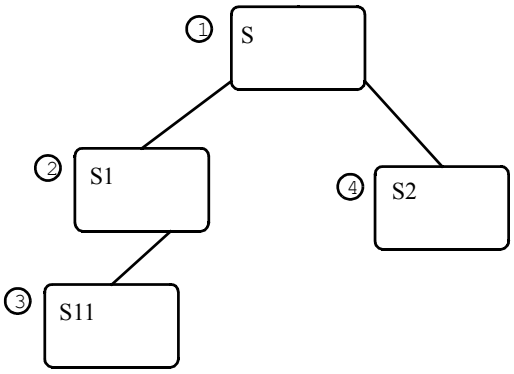
- `stm_success`
- `stm_nil_list`
- `stm_elements_not_hierarchical`

Example

Hierarchical elements in a chart can be ordered by *branches*. Consider the following statechart:



Hierarchically, the states can be drawn as shown in the following figure.



The set of elements, {`S,S1,S11`}, comprise a branch. Assume you perform a `sort_by_branch` function on statechart S. The sorted order would be: `S`, `S1`, `S11`, `S2`.

The order in which branches appear in the output is arbitrary. However, the order of states appearing within each branch are ordered from top-to-bottom (`S` to `S11`, for example).

stm_list_sort_by_chart

Alphabetically sorts the input list of named Rational Statemate elements, by the name of the chart to which they belong. The input list consists of Rational Statemate elements.

This function receives and returns a list of element IDs, *not* a list of element names.

Function Type

`stm_list`

Syntax

```
stm_list_sort_by_chart (el_list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`el_list`	In	`stm_list`	The list of Rational Statemate box elements.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`

Example

To sort a list of items by the chart they belong to, use the following statements:

```
stm_list   st_list, sorted_by_chart
int        status;
           .
           .
st_list = stm_r_st_name_of_st ("*", &status);
sorted_by_chart = stm_list_sort_by_chart (st_list,&status);
```

A list of all named statuses is retrieved, then the sort function orders the list by chart name.

stm_list_sort_by_levels

When Rational Statemate uses this function to sort a specified list of elements by levels and it encounters two or more charts at the same level of hierarchy, it sorts them alphabetically by name.

Function Type

`stm_list`

Description

Sorts a list of hierarchical Rational Statemate elements by level.

This function is relevant only for a list of hierarchical elements. If the function is applied to a list of non-hierarchical elements, `status` receives the value `stm_elements_not_hierarchical`.

Syntax

```
stm_list_sort_by_levels (list, &status)
```

Arguments

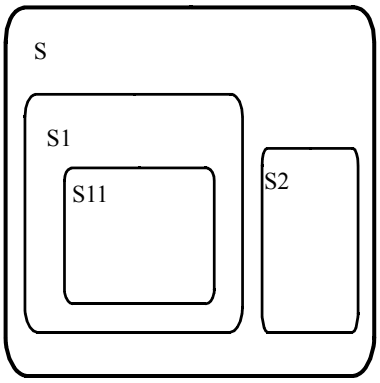
Argument	Input/ Output	Type	Description
`list`	In	`stm_list`	The list of Rational Statemate box elements.
`status`	Out	`int`	The function status code.

Status Codes

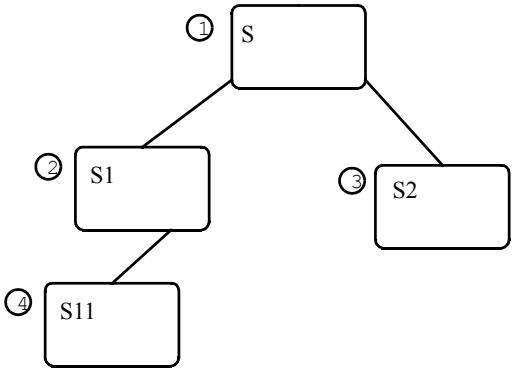
- `stm_success`
- `stm_nil_list`
- `stm_elements_not_hierarchical`

Example

Hierarchical elements in a chart can be ordered by levels, as shown in the following statechart.



Hierarchically, the states can be drawn as shown in the following figure.



The set of elements, {`S1,S2`}, comprise a level. If you perform a `sort_by_level` function on statechart `S`, the sorted order would be: `S`, `S1`, `S2`, `S11`.

The order of elements within the same level appear in an arbitrary order in the output. For example, `S2` might appear before `S1` because they are of the same level. However, the order of levels is top-to-bottom.

stm_list_sort_by_name

Sorts the specified list of Rational Statemate elements alphabetically by name.

Note

- The function returns the status code `stm_elements_without_name` when you attempt to apply this function to a list that contains unnamed elements.
- The function receives and returns a list of element IDs, *not* a list of element names.

Function Type

```
stm_list
```

Syntax

```
stm_list_sort_by_name (list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list`	The list of Statemate elements to be sorted. This input lists consists of element IDs.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_elements_without_name`

Example

To sort a list of items by name, use the following statements:

```
stm_list   st_list, sorted_st_list;
int        status;
           .
           .
st_list = stm_r_st_name_of_st ("*", &status);
sorted_st_list = stm_list_sort_by_name (st_list,
   &status);
           .
           .
```

A list of all named states in an unspecified order is retrieved from the database, then the sort function orders the list.

stm_list_sort_by_synonym

Sorts the specified list of Rational Statemate elements alphabetically by their synonyms.

Note

- The function returns the status code `stm_elements_without_name` when you attempt to apply this function to a list that contains unnamed elements.
- The function receives and returns a list of element IDs, not a list of element names.

Function Type

`stm_list`

Syntax

```
stm_list_sort_by_synonym (list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list`	In	`stm_list LIST OF ELEMENT`	The list of Statemate elements to be sorted. This input lists consists of element IDs.
`status`	Out	`int INTEGER`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_elements_without_synonym`

Example

To sort a list of items by synonym, use the following statements:

```
stm_list      st_list, sorted_st_list;
int           status;
              .
              .
st_list = stm_r_st_name_of_st ("*", &status);
sorted_st_list = stm_list_sort_by_synonym (st_list,
   &status);
              .
              .
```

A list of all named states in an unspecified order is retrieved from the database, then the sort function orders the list alphabetically, according to the synonyms.

stm_list_sort_by_type

Sorts the specified list of Rational Statemate elements by type.

Note that the function receives and returns a list of element IDs, *not* a list of element names.

Syntax

```
stm_list_sort_by_type (el_list, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`el_list`	In	`stm_list`	The list of Statemate element IDs to be sorted
`status`	Out	`int`	The function status code

Status Codes

- `stm_success`
- `stm_nil_list`

stm_list_subtract_ids_lists

Creates a new list of those elements of the first input list that are not found in the second input list.

Note

The two input lists must both lists stm_list_id_elm's.

Function Type

`stm_list`

Syntax

```
stm_list_id_subtract-ids_lists (list1, list2, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list1`	In	`stm_list`	The first list of Rational Statemate elements.
`list2`	In	`stm_list`	The second list of Statemate elements.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_cannot_compare_lists`

Example

The following example creates list `list3` by subtracting `list2` from `list1`:

```
stm_list  list1, list2, list3;
int       status;
          .
          .
list3 = stm_list_subtract_ids_lists (list1, list2, &status);
          .
          .
```

stm_list_subtraction_ptr_lists

Creates a new list of those elements of the first input list that are not found in the second input list.

Note

The two input lists must stm_list_ptr_elm's.

Function Type

`stm_list`

Syntax

```
stm_list_subtract_ptr_lists (list1, list2, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list1`	In	`stm_list`	The first list of Rational Statemate elements.
`list2`	In	`stm_list`	The second list of Statemate elements.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_cannot_compare_lists`

Example

The following example creates list `list3` by subtracting `list2` from `list1`:

```
stm_list  list1, list2, list3;
int       status;
          .
          .
list3 = stm_list_subtract_ptr_lists (list1, list2, &status);
          .
          .
```

stm_list_union_ids_lists

Function Type

`stm_list`

Description

Merges the elements of two specified ids lists.

Note

- Elements in both input lists appear in the output list only once.
- The two input lists must be both lists of stm_id_elm's.

Syntax

```
stm_list_union_ids_lists (list2, list2, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list1`	In	`stm_list`	The first list.
`list2`	In	`stm_list`	The second list.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_cannot_compare_lists`

Example

To produce the union list `list3` from `list1` and `list2`, use the following statements:

```
stm_list    list1,list2,list3;
int         status;
               .
               .
list3 = stm_list_union_ids_list (list1, list2, &status);
```

stm_list_union_ptr_lists

Function Type

`stm_list`

Description

Merges the elements of two specified ptr lists.

Note

- Elements in both input lists appear in the output list only once.
- The two input lists must be both lists of stm_list_ptr_elm's.

Syntax

```
stm_list_union_ptr_lists (list2, list2, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`list1`	In	`stm_list`	The first list.
`list2`	In	`stm_list`	The second list.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nil_list`
- `stm_cannot_compare_lists`

Example

To produce the union list `list3` from `list1` and `list2`, use the following statements:

```
stm_list    list1,list2,list3;
int         status;
               .
               .
list3 = stm_list_union_ptr_lists (list1, list2, &status);
               .
```

stm_load

Function Type

None

Description

Loads a chart file (or any other configuration item file) into the current workarea. It is one of the four utility functions (`stm_load`, `stm_save`, `stm_unload`, and `stm_unload_all`) that provide an interface between the Rational Statemate user workarea and external files.

Note: You must work in automatic transaction mode when using this function by specifying `automatic_transaction` as the `trans_mode` (third) argument of the `stm_init_uad` function. Your program should contain lines similar to the following:

```
int success, status;
   ...
success = stm_init_uad ("MY_PROJECT",
    "/local/my_work_area", automatic_transaction,
    &status);
if (!success)
...
```

Syntax

```
stm_load (file_name, item_name, version, mode, enforce,
   message, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`file_name`	In	`char *`	The full path name for the file (which is usually a chart file). This file can reside in any directory.
`item_name`	In	`char *`	The possible item name and type values are as follows: **sch –** Statechart **ach** - Activity-charts **mch** - Module-charts **fch –** Flowcharts **dic** - Global Definition Set files **qch** - Sequence-Diagrams **uch** - Use-Case-Diagrams **vsm** - Continuous Diagrams **pnl –** Panel files **scp** - Simulation SCL files **cnf** - Simulation status files **wpf** - Waveform Profiles **dyn_set** - Simulation analysis profiles **mon** - Monitor files **chk_mdl_set** - Check Model Profiles **dgl** - Documentor templates **inc** - Documentor include files **pnl** - Prototype panels **config** - Configuration files **tv** - Task View files **mak** - Makefiles **oil** - OIL files **cfg** - CFG files **c** - Source(c) files **h** - Header (h) files **rgenset** - Rapid Prototyper Profiles **trg** - Target files **rtrg** - Rapid Target files **crd** - Card files **rconfig** - Rhapsody block Configuration files **ccf** - Component Configuration files **dat** - VSM Data files **wav** - VSM Wave files **mat** - VSM Mat. files **m** - VSM M. files
`version`	In	`char *`	The version number. The version contained in this string is recorded in the workarea.

Argument	Input/ Output	Type	Description
mode	In	char *	The mode - u for update, r for read. No lock files are created by this function, even when mode is u (update).
enforce	In	int	If this is 1, it enforces the load - even in cases when the new or modified item with the same name already exists in the workarea. If this is 0, the load operation fails, in these cases, with the corresponding status code and error message.
message	Out	char *	A buffer that holds the error message, if an error occurs. This buffer can hold 127 characters.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_illegal_expression_in_chart
- stm_error_in_chart
- stm_exceeded_max_id_number
- stm_illegal_version
- stm_not_loaded_because_type
- stm_not_loaded_because_modified
- stm_not_loaded_because_new
- stm_cannot_create_file
- stm_cannot_open_chart_file
- stm_illegal_load_mode
- stm_cannot_copy_file
- stm_file_not_found
- stm_chart_is_active

Example

To load the first version of a statechart named `MY_CHART` from the databank located at `/local/my_bank` in read mode and enforce the operation, use the following statement:

```
#define ENFORCE 1
int     status;
char    mess[128];
.
.
stm_load ("/local/my_bank/chart/my_chart.sch.1",
   "my_chart.sch", "1", 'r', ENFORCE, message, &status);
if (status != stm_success)
.
.
```

stm_multiline_to_one

Function Type

```
stm_expression
```

Description

Converts the specified multiline string (with new lines) to a one-line string (without the new lines).

Syntax

```
stm_multiline_to_one (string)
```

Arguments

Argument	Input/ Output	Type	Description
`string`	In	`char *`	The multiline string.

stm_multiline_to_strings

Function Type

```
stm_list
```

Description

Converts the specified multiline expression to a list of strings.

Syntax

```
stm_multiline_to_strings (string)
```

Arguments

Argument	Input/Output	Type	Description
`string`	In	`char *`	The multiline expression

stm_open_truth_table

Function Type

```
stm_boolean
```

Description

Opens a Truth Table that is connected to the specified element and highlights the specified line in it. Returns `stm_success` if request was successfully sent. Otherwise, it returns `stm_id_out_of_range`.

Syntax

```
stm_open_truth_table(stm_id id, int line,int *status)
```

stm_plot

Generates a plot file with the indicated parameters, such as plot size, output device, and so on. The plot parameters are the same for all the different plot types (statecharts, activity charts, or module charts).

The output is designated for a particular device (one of the output devices defined in Rational Statemate). The destination of the plot output is specified by one of the parameters. If its destination is not specified, the plot is included as part of the output segment file.

When working with Interleaf, the plot uses the following definitions, which should be included at the beginning of the file:

```
<!Font Definitions F46 = Typewriter 10 >
<!Class, caption, Font = F46>
<!Class, plot, Font = F46>
<!Master Frame,
    Name = PltFrm,
    Placement = Following Anchor,
    Horizontal Alignment =Center,
    Same Page = Yes,
    Diagram = V6, (g9,0,0)>
```

Function Type

```
int
```

Syntax

```
stm_plot (id, plot_file, width, height, with_labels,with_names, with_notes,
device, date_position,title_position, title, do_rotate, with_file_header,
actual_height)
```

Arguments

Argument	Input/ Output	Type	Description
id	In	stm_id	The ID number of the Rational Statemate chart to be plotted.
plot_file	In	stm_filename	The name of the file destination to which the plot is written. The operating system path name conventions are followed. You can specify a full path name to any directory for which you have write access. If you specify a simple file name, the plot is written to your workarea.
width	In	double	The maximum possible width of the plot (in inches).

Argument	Input/ Output	Type	Description
`height`	In	`double`	The maximum possible length of the plot (in inches). If you specify a plot size (width and height parameters) that is larger than the paper size defined for the specific printer, the plot simply uses the maximum allowable height and width defined for that printer.
`with_labels`	In	`stm_boolean`	Determines whether labels are plotted (TRUE) or not (FALSE).
`with_names`	In	`stm_boolean`	Determines whether names are plotted (TRUE) or not (FALSE).
`with_notes`	In	`stm_boolean`	Determines whether notes are plotted (TRUE) or not (FALSE).
`device`	In	`char*`	Specifies the plotting device. This can be a supported formatting language if the plot is to be handled by a formatting processing system that has its own graphics language. To configure a new plotter or printer, select **Utilities > Output Devices** from the main Rational Statemate window. Plots created using Word format in the Output Device dialog box are HPGL files. To import these files into Word, rename them as .HGL or .PLT files.
`date_position`	In	`stm_plt_position`	The position of the date. This is an integer parameter of type `stm_plt_position` That indicates where to place the plot date. The possible values are as follows: • `stm_plt_none` - The date is not included. • `stm_plt_top` - The date is placed at the top of the plot. • `stm_plt_bottom` - The date is placed at the bottom of the plot.

Argument	Input/ Output	Type	Description
title_position	In	stm_plt_position	The title position. This is an integer parameter of type stm_plt_position that indicates where to place the plot title. The possible values are as follows: • stm_plt_none - The title is not included. • stm_plt_top - The title is placed at the top of the plot. • stm_plt_bottom - The title is placed at the bottom of the plot.
title	In	char*	Specifies the title to be printed with the plot.
do_rotate	In	stm_boolean	Determines whether the orientation of the plot is landscape (TRUE) or portrait (FALSE).
with_file_header	In	stm_boolean	Indicates whether a header is added to the file (TRUE). Use this option if you do not want the plot as part of the document (FALSE).
actual_height	Out	double	Specifies the actual height (in inches) of the plotted output.

Status Codes

- stm_success
- stm_can_not_open_file
- stm_id_out_of_range
- stm_not_enough_memory
- stm_id_not_found
- stm_empty_chart
- stm_unknown_plotter
- stm_plot_failure
- stm_unresolved
- stm_illegal_parameter

Example

To create a plot of a chart within a file, use the following statements:

```
stm_id              chart_id;
int                 status;
double              real_ht;
stm_plt_position    date_position;
stm_plt_position    title_position;
date_position = stm_plt_bottom;
title_position = stm_plt_bottom;
chart_id = stm_r_ch ("XL25", &status);
stm_plot (chart_id, "sam/p_xl25", 5.0, 7.0,
    stm_true, stm_true, stm_false, "eps100h",
    date_position, title_position, "System XL25",
    stm_true, stm_false, &real_ht);
```

This produces a plot for the activity-chart `XL25` in landscape orientation, limited to a maximum size of 5x7 inches, that prints full labels and names, but excludes notes. The plot is output to the file specified by the path `sam/p_xl25`. This file is suitable for printing on an Epson FX100 graphics printer. The date and the title System XL25 appear at the bottom of the plot. The actual scaled height of the plot is returned in the variable `real_ht`.

stm_plot_ext

Generates a plot file with the indicated parameters, such as plot size, output device, and so on. The plot parameters are the same for all the different plot types (statecharts, activity charts, or module charts). The output is designated for a particular device (one of the output devices defined in Rational Statemate). The destination of the plot output is specified by one of the parameters. If its destination is not specified, the plot is included as part of the output segment file.

The function can generate the hyperlinks in the chart, print a sequence diagram with numbered scenarios, break a sequence diagram across multiple pages and print a sequence diagram with the names of lifelines on every page.

Function Type

```
int
```

Syntax

```
stm_plot_ext (id, plot_file_name, width, height, device, data_position,
title_position, title, actual_h, pages_in_x, pages_in_y, page_index_in_x,
page_index_in_y, headerline_y, options)
```

Arguments

Argument	Input/ Output	Type	Description
`id`	In	`stm_id`	The ID number of the Rational Statemate chart of be plotted
`plot_file_name`	IN	`stm_filename`	The name of the file destination to which the plot is written. The operating system pathname conventions are followed. You can specify a full path name to any directory for which you have writeaccess.If you specify a simple file name, the plot is written to your workarea.
`width`	In	`double`	The maximum possible width of the plot (in inches).
`height`	In	`double`	The maximum possible length of the plot (ininches).If you specify a plot size (width and height parameters) that is larger than the paper size defined for the specific printer, the plot simply uses the maximum allowable height and width defined for that printer.

Argument	Input/ Output	Type	Description
`device`	In	`char*`	Specifies the plotting device. This can be a supported formatting language if the plot is to be handled by a formatting processing system that has its own graphics language.To configure a new plotter or printer, select **Utilities > Output Devices** from the main Statemate window.Plots created using Word format in the Output Device dialog box are HPGL files. To import these files into Word, rename them as .HGL or .PLT files.
`data_position`	In	`stm_plt_position`	The position of the date.This is an integer parameter of type stm_plt_position That indicates where to place the plot date. The possible values are as follows: • `stm_plt_none` - The date is not included. • `stm_plt_top` - The date is placed at the top of the plot. • `stm_plt_bottom` - The date is placed atthe bottom of the plot.
`title_position`	In	`stm_plot_position`	This is an integer parameter of type stm_plt_position that indicates where to place the plot title. The possible values are as follows: • `stm_plt_none` - The title is not included. • `stm_plt_top` - The title is placed at the top of the plot. • `stm_plt_bottom` - The title is placed at the bottom of the plot.
`title`	In	`char*`	Specifies the title to be printed with the plot.
`actual_h`	Out	`double`	Specifies the actual height (in inches) of the plotted output.

Argument	Input/ Output	Type	Description
pages_in_x	Out	int	Specifies how many pages the tool attempted to break the SD into along the x-axis. Note that ifpages_in_x==0 and pages_in_y==0, the tool calculates a break pages scheme and assigns these variables so they can be read by the user after the call.
page_index_in_x	Out	int	Specifies how many pages the tool attempted to break the SD into along the x-axis. Note that ifpages_in_x==0 and pages_in_y==0, the tool calculates a break pages scheme and assigns these variables so they an be read by the user after the call.
page_index_in_y	Out	int	Specifies how many pages the tool attempted to break the SD into along the y-axis.
headerline_y	In	double	Defines the vertical coordinate on the page of the header line. This is usually 1.0.
options	In	list	A list of strings of the form 'key=value'. See notes below for supported options

Status Codes

This function may return one of the two following status codes:

- `stm_success`
- `stm_can_not_open_file`
- `stm_id_out_of_range`
- `stm_not_enough_memory`
- `stm_id_not_found`
- `stm_empty_chart`
- `stm_unknown_plotter`
- `stm_plot_failure`
- `stm_unresolved`
- `stm_illegal_parameter`
- `stm_plot_illegal_option_key`
- `stm_plot_illegal_option_val`

Note: The following are valid values for the “options” argument

- stm_plot_option_hyperlink_ext_act_to_graphics
  - **For External-Activity**:

    When this option is 'no', the External_activity is hyperlinked to the 'Dictionary' description, if it exists, of the Activity it resolves to. When the 'Dictionary' description is empty, no link is created.

    When this option is 'yes', the External-Activity is hyperlinked to the chart in which the Activity it resolves to is in. If the resolved Activity is an Off-Page Activity, the link is to the off-page chart. If the resolved Activity is an Instance of generic, the link is to the generic chart. If the External-Activity resolves to a higher-level unresolved External-Activity, then the link is to the Chart where the Upper most instance of this External-Activity. If the External-Activity does not resolve to any Activity, no hyperlink is created.
  - **For External-Router**:

    When this options is 'no', External_router is hyperlinked to the 'Dictionary' description, if it exists, of the Router it resolves to. When the 'Dictionary' description is empty, no link is created, When this option is 'yes', External-Router is hyperlinked to the chart that the Router it resolves to is in.

- hyperlink_lifeline_to_graphics

  When this option is 'no', Lifelines are hyperlinked to the 'Dictionary' description, if it exists, of the Activity they resolve to.

  When this option is 'yes', Lifelines are hyperlinked to the chart that the Activity they resolve to are in. If the resolved Activity is an Off-Page Activity, the link is to the off-page chart. If the resolved Activity is an Instance of generic, the link is to the generic chart. If the Lifeline resolves to an unresolved External- Activity, no link is created. If the Lifeline does not resolve to any Activity, no hyperlink is created.

stm_plot_hyper_exp

Function Type

```
int
```

Description

Generates the hyperlinks in a sequence diagram.

Syntax

```
stm_plot_hyper_exp (id, plot_file, width, height, with_labels, with_names,
with_notes, with_hyperlink, device, date_position, title_position, title,
do_rotate, with_file_header, actual_height, with_breakpages, pages_in_x,
pages_in_y, page_index_in_x, page_index_in_y, with_hyperlink_exp)
```

Arguments

Argument	Input/ Output	Type	Description
`id`	In	`stm_id`	The ID number of the Rational Statemate chart to be plotted.
`plot_file`	In	`stm_filename`	The name of the file destination to which the plot is written. The operating system path name conventions are followed. You can specify a full path name to any directory for which you have write access. If you specify a simple file name, the plot is written to your workarea.
`width`	In	`double`	The maximum possible width of the plot (in inches).
`height`	In	`double`	The maximum possible length of the plot (in inches). If you specify a plot size (width and height parameters) that is larger than the paper size defined for the specific printer, the plot simply uses the maximum allowable height and width defined for that printer.
`with_labels`	In	`stm_boolean`	Determines whether labels are plotted (TRUE) or not (FALSE).
`with_names`	In	`stm_boolean`	Determines whether names are plotted (TRUE) or not (FALSE).
`with_notes`	In	`stm_boolean`	Determines whether notes are plotted (TRUE) or not (FALSE).

Argument	Input/ Output	Type	Description
with_hyperlink	In	stm_boolean	Specifies whether to generate hyperlinks for lifelines and referenced sequence diagrams (TRUE).
device	In	char*	Specifies the plotting device. This can be a supported formatting language if the plot is to be handled by a formatting processing system that has its own graphics language. To configure a new plotter or printer, select **Utilities > Output Devices** from the main Rational Statemate window. Plots created using Word format in the Output Device dialog box are HPGL files. To import these files into Word, rename them as .HGL or .PLT files.
date_position	In	stm_plt_ position	The position of the date. This is an integer parameter of type stm_plt_position that indicates where to place the plot date. The possible values are as follows: • stm_plt_none - The date is not included. • stm_plt_top - The date is placed at the top of the plot. • stm_plt_bottom - The date is placed at the bottom of the plot.
title_position	In	stm_plt_ position	The title position. This is an integer parameter of type stm_plt_position that indicates where to place the plot title. The possible values are as follows: • stm_plt_none - The title is not included. • stm_plt_top - The title is placed at the top of the plot. • stm_plt_bottom - The title is placed at the bottom of the plot.
title	In	char*	Specifies the title to be printed with the plot.

Argument	Input/ Output	Type	Description
do_rotate	In	stm_boolean	Determines whether the orientation of the plot is landscape (TRUE) or portrait (FALSE).
with_file_header	In	stm_boolean	Indicates whether a header is added to the file (TRUE). Use this option if you do not want the plot as part of the document (FALSE).
actual_height	Out	double	Specifies the actual height (in inches) of the plotted output.
with_breakpages	In	stm_boolean	Specifies whether to break the SD across multiple pages (true).
pages_in_x	Out	int	Specifies how many pages the tool attempted to break the SD into along the x-axis. Note that if pages_in_x==0 and pages_in_y==0, the tool calculates a break pages scheme and assigns these variables so they can be read by the user after the call.
pages_in_y	Out	int	Specifies how many pages the tool attempted to break the SD into along the y-axis.
page_index_in_x	In	int	Plots the *i*th page in the x-axis.
page_index_in_y	In	int	Plots the *i*th page in the y-axis.
with_hyperlink_exp	In	stm_boolean	Specifies whether to generate hyperlinks for message labels (true).

Status Codes

- stm_success
- stm_can_not_open_file
- stm_id_out_of_range
- stm_not_enough_memory
- stm_id_not_found
- stm_empty_chart
- stm_unknown_plotter
- stm_plot_failure
- stm_unresolved
- stm_illegal_parameter

stm_plot_with_autonumber

Function Type

```
int
```

Description

Prints a sequence diagram with numbered scenarios.

Syntax

```
stm_plot_with_autonumber (id, plot_file, width, height, with_labels,
with_names, with_notes, with_hyperlink, plot_type, title_position, title,
do_rotate, with_file_header, actual_y, with_breakpages, pages_in_x,
pages_in_y, page_index_in_x, page_index_in_y, with_hyperlink_exp,
with_headerline, headerline_y, with_autonumber)
```

Arguments

Argument	Input/ Output	Type	Description
`id`	In	`stm_id`	The ID number of the Rational Statemate chart to be plotted.
`plot_file`	In	`stm_filename`	The name of the file destination to which the plot is written. The operating system path name conventions are followed. You can specify a full path name to any directory for which you have write access. If you specify a simple file name, the plot is written to your workarea.
`width`	In	`double`	The maximum possible width of the plot (in inches).
`height`	In	`double`	The maximum possible length of the plot (in inches). If you specify a plot size (width and height parameters) that is larger than the paper size defined for the specific printer, the plot simply uses the maximum allowable height and width defined for that printer.

Argument	Input/ Output	Type	Description
`with_labels`	In	`stm_boolean`	Determines whether labels are plotted (true) or not (false).
`with_names`	In	`stm_boolean`	Determines whether names are plotted (true) or not (false).
`with_notes`	In	`stm_boolean`	Determines whether notes are plotted (true) or not (false).
`with_hyperlink`	In	`stm_boolean`	Specifies whether to generate hyperlinks for lifelines and referenced sequence diagrams (true).
`plot_type`	In	`char*`	Specifies the plotting device. This can be a supported formatting language if the plot is to be handled by a formatting processing system that has its own graphics language. To configure a new plotter or printer, select **Utilities > Output Devices** from the main Rational Statemate window. Plots created using Word format in the Output Device dialog box are HPGL files. To import these files into Word, rename them as .HGL or .PLT files.
`date_position`	In	`stm_plt_position`	The date position. This is an integer parameter of type `stm_plt_position` that indicates where to place the plot date. The possible values are as follows: • `stm_plt_none` - The date is not included. • `stm_plt_top` - The date is placed at the top of the plot. • `stm_plt_bottom` - The date is placed at the bottom of the plot.

Argument	Input/ Output	Type	Description
title_position	In	stm_plt_position	The title position. This is an integer parameter of type stm_plt_position that indicates where to place the plot title. The possible values are as follows: • stm_plt_none - The title is not included. • stm_plt_top - The title is placed at the top of the plot. • stm_plt_bottom - The title is placed at the bottom of the plot.
title	In	char*	Specifies the title to be printed with the plot.
do_rotate	In	stm_boolean	Determines whether the orientation of the plot is landscape (true) or portrait (false).
with_file_header	In	stm_boolean	Indicates whether a header is added to the file (true). Use this option if you do not want the plot as part of the document (false).
actual_height	Out	double	Specifies the actual height (in inches) of the plotted output.
with_breakpages	In	stm_boolean	Specifies whether to break the SD across multiple pages (true).
pages_in_x	Out	int	Specifies how many pages the tool attempted to break the SD into along the x-axis. Note that if pages_in_x==0 and pages_in_y==0, the tool calculates a break pages scheme and assigns these variables so they can be read by the user after the call.
pages_in_y	Out	int	Specifies how many pages the tool attempted to break the SD into along the y-axis.
page_index_in_x	In	int	Plots the *i*th page in the x-axis.

Argument	Input/ Output	Type	Description
page_index_in_y	In	int	Plots the *i*th page in the y-axis.
with_hyperlink_ exp	In	stm_boolean	Specifies whether to generate hyperlinks for message labels (true).
with_headerline	In	stm_boolean	Specifies whether to print the names of the lifelines on every page (true).
headerline_y	In	double	Defines the vertical coordinate on the page of the headerline. This is usually 1.0.
with_autonumber	In	stm_boolean	Specifies whether to print the SD scenario numbers (true).

Status Codes

- stm_success
- stm_can_not_open_file
- stm_id_out_of_range
- stm_not_enough_memory
- stm_id_not_found
- stm_empty_chart
- stm_unknown_plotter
- stm_plot_failure
- stm_unresolved
- stm_illegal_parameter

stm_plot_with_break

Function Type

```
int
```

Description

Breaks a sequence diagram across multiple pages.

Syntax

```
stm_plot_with_break (id, plot_file, width, height,
   with_labels, with_names, with_notes,
   with_hyperlink, plot_type, date_position,
   title_position, title, do_rotate, with_file_header,
   actual_height, with_breakpages, pages_in_x,
   pages_in_y, page_index_in_x, page_index_in_y)
```

Arguments

Argument	Input/ Output	Type	Description
`id`	In	`stm_id`	The ID number of the Rational Statemate chart to be plotted.
`plot_file`	In	`stm_filename`	The name of the file destination to which the plot is written. The operating system path name conventions are followed. You can specify a full path name to any directory for which you have write access. If you specify a simple file name, the plot is written to your workarea. If you do not specify a value (‘’), the plot is included as part of the output segment file.
`width`	In	`double`	The maximum possible width of the plot (in inches).
`height`	In	`double`	The maximum possible length of the plot (in inches). If you specify a plot size (width and height parameters) that is larger than the paper size defined for the specific printer, the plot simply uses the maximum allowable height and width defined for that printer.
`with_labels`	In	`stm_boolean`	Determines whether labels are plotted (TRUE) or not (FALSE).
`with_names`	In	`stm_boolean`	Determines whether names are plotted (TRUE) or not (FALSE).

Argument	Input/ Output	Type	Description
with_notes	In	stm_boolean	Determines whether notes are plotted (TRUE) or not (FALSE).
with_hyperlink	In	stm_boolean	Specifies whether to generate hyperlinks for lifelines and referenced sequence diagrams (TRUE).
plot_type	In	char*	Specifies the plot type. This can be a supported formatting language if the plot is to be handled by a formatting processing system that has its own graphics language. To configure a new plotter or printer, select **Utilities > Output Devices** from the main Rational Statemate window. Plots created using Word format in the Output Device dialog box are HPGL files. To import these files into Word, rename them as .HGL or .PLT files.
date_position	In	stm_plt_position	The date position. This is an integer parameter of type stm_plt_position that indicates where to place the plot date. The possible values are as follows: • stm_plt_none - The date is not included. • stm_plt_top - The date is placed at the top of the plot. • stm_plt_bottom - The date is placed at the bottom of the plot.
title_position	In	stm_plt_position	The title position. This is an integer parameter of type stm_plt_position that indicates where to place the plot title. The possible values are as follows: • stm_plt_none - The title is not included. • stm_plt_top – The title is placed at the top of the plot. • stm_plt_bottom - The title is placed at the bottom of the plot.
title	In	char*	Specifies the title to be printed with the plot.
do_rotate	In	stm_boolean	Determines whether the orientation of the plot is landscape (TRUE) or portrait (FALSE).

Argument	Input/ Output	Type	Description
with_file_header	In	stm_boolean	Indicates whether a header is added to the file (TRUE). Use this option if you do not want the plot as part of the document (FALSE).
actual_height	Out	double	Specifies the actual height (in inches) of the plotted output.
with_breakpages	In	stm_boolean	Specifies whether to break the SD across multiple pages (TRUE).
pages_in_x	Out	int	Specifies how many pages the tool attempted to break the SD into along the x-axis. Note that if pages_in_x==0 and pages_in_y==0, the tool calculates a break pages scheme and assigns these variables so they can be read by the user after the call.
pages_in_y	Out	int	Specifies how many pages the tool attempted to break the SD into along the y-axis.
page_index_in_x	In	int	Plots the *i*th page in the x-axis.
page_index_in_y	In	int	Plots the *i*th page in the y-axis.

Status Codes

- stm_success
- stm_can_not_open_file
- stm_id_out_of_range
- stm_not_enough_memory
- stm_id_not_found
- stm_empty_chart
- stm_unknown_plotter
- stm_plot_failure
- stm_unresolved
- stm_illegal_parameter

Notes

Function parameters are as follows:

```
boolean with_hyperlink (IN) /* generate hyperlinks */
boolean with_breakpages (IN) /* enable break pages */
integer pages_in_x (OUT) /* try to break to # of pages in x axis */
integer pages_in_y (OUT) /* try to break to # of pages in y axis */
```

Note: If `pages_in_x == 0` and `pages_in_y==0`, the tool calculates a break pages scheme and assigns these variables so they can be read by the user after the call.

```
integer page_index_in_x (IN) /*plot the ith page in x axis */
integer page_index_in_y (IN) /*plot the ith page in y axis */
```

Call the function `STM_PLOT_SET_DATA()` before plotting a sequence diagram using `STM_PLOT_WITH_BREAK`. Call the function `STM_PLOT_RESET_DATA()` after finishing the sequence diagram multiple pages plot.

stm_plot_with_headerline

Prints a sequence diagram with the names of lifelines on every page.

Function Type

```
int
```

Syntax

```
stm_plot_with_headerline (id, plot_file, width, height, with_labels,
with_names, with_notes, with_hyperlink, plot_type, title_position, title,
do_rotate, with_file_header,actual_y, with_breakpages, pages_in_x,
pages_in_y, page_index_in_x, page_index_in_y, with_hyperlink_exp,
with_headerline, headerline_y,)
```

Arguments

Argument	Input/ Output	Type	Description
`id`	In	`stm_id`	The ID number of the Rational Statemate chart to be plotted.
`plot_file`	In	`stm_filename`	The name of the file destination to which the plot is written. The operating system path name conventions are followed. You can specify a full path name to any directory for which you have write access. If you specify a simple file name, the plot is written to your workarea.
`width`	In	`double`	The maximum possible width of the plot (in inches).
`height`	In	`double`	The maximum possible length of the plot (in inches). If you specify a plot size (width and height parameters) that is larger than the paper size defined for the specific printer, the plot simply uses the maximum allowable height and width defined for that printer.
`with_labels`	In	`stm_boolean`	Determines whether labels are plotted (TRUE) or not (FALSE).
`with_names`	In	`stm_boolean`	Determines whether names are plotted (TRUE) or not (FALSE).
`with_notes`	In	`stm_boolean`	Determines whether notes are plotted (TRUE) or not (FALSE).
`with_hyperlink`	In	`stm_boolean`	Specifies whether to generate hyperlinks for lifelines and referenced sequence diagrams (TRUE).

Argument	Input/ Output	Type	Description
plot_type	In	char*	Specifies the plot type. This can be a supported formatting language if the plot is to be handled by a formatting processing system that has its own graphics language. To configure a new plotter or printer, select **Utilities > Output Devices** from the main Rational Statemate window. Plots created using Word format in the Output Device dialog box are HPGL files. To import these files into Word, rename them as .HGL or .PLT files.
date_position	In	stm_plt_ position	The date position. This is an integer parameter of type stm_plt_position that indicates where to place the plot date. The possible values are as follows: • stm_plt_none - The date is not included. • stm_plt_top - The date is placed at the top of the plot. • stm_plt_bottom - The date is placed at the bottom of the plot.
title_position	In	stm_plt_ position	The title position. This is an integer parameter of type stm_plt_position that indicates where to place the plot title. The possible values are as follows: • stm_plt_none - The title is not included. • stm_plt_top - The title is placed at the top of the plot. • stm_plt_bottom - The title is placed at the bottom of the plot.
title	In	char*	Specifies the title to be printed with the plot.
do_rotate	In	stm_boolean	Determines whether the orientation of the plot is landscape (TRUE) or portrait (FALSE).
with_file_header	In	stm_boolean	Indicates whether a header is added to the file (TRUE). Use this option if you do not want the plot as part of the document (FALSE).
actual_height	Out	double	Specifies the actual height (in inches) of the plotted output.

Argument	Input/ Output	Type	Description
with_breakpages	In	stm_boolean	Specifies whether to break the SD across multiple pages (TRUE).
pages_in_x	Out	int	Specifies how many pages the tool attempted to break the SD into along the x-axis. Note that if pages_in_x==0 and pages_in_y==0, the tool calculates a break pages scheme and assigns these variables so they can be read by the user after the call.
pages_in_y	Out	int	Specifies how many pages the tool attempted to break the SD into along the y-axis.
page_index_in_x	In	int	Plots the *i*th page in the x-axis.
page_index_in_y	In	int	Plots the *i*th page in the y-axis.
with_hyperlink_exp	In	stm_boolean	Specifies whether to generate hyperlinks for message labels (TRUE).
with_headerline	In	stm_boolean	Specifies whether to print the names of the lifelines on every page (TRUE).
headerline_y	In	double	Defines the vertical coordinate on the page of the headerline. This is usually 1.0.

Status Codes

- stm_success
- stm_can_not_open_file
- stm_id_out_of_range
- stm_not_enough_memory
- stm_id_not_found
- stm_empty_chart
- stm_unknown_plotter
- stm_plot_failure
- stm_unresolved
- stm_illegal_parameter

stm_r_global_interface_report

Function Type

`stm_list`

Description

Returns the global interface report for the elements in the input list (box_lst).

Syntax

```
stm_r_global_interface_report (box_lst, sort_by_elm, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`box_list`	in	`stm_list`	List of elements.
`sort_by_elm`	in	`stm_boolean`	The way the global interface report is.
`status`	out	`int`	The function status code.

Status Codes

- `stm_id_out_of_range`
- `stm_success`

stm_r_local_interface_report

Function Type

```
stm_list
```

Description

Returns the local interface report for the elements in the input list (box_lst).

Syntax

```
stm_r_local_interface_report (box_lst, &status)
```

Arguments

Argument	Input/ Output	Type	Description
box_lst	in	stm_list	List of elements.
status	out	int	The function status code.

Status Codes

- stm_id_out_of_range
- stm_success

stm_run_simulation_profile

Function Type

```
stm_Boolean
```

Description

Sends a message to Rational Statemate to open and execute a Simulation profile by the name passed as a parameter.

Syntax

```
stm_run_simulation_profile (string profile_name, int* status)
```

stm_save

Function Type

None

Description

Saves a chart (or any other configuration item file) from the current workarea to an external file. It is one of the four utility functions (`stm_load`, `stm_save`, `stm_unload`, and `stm_unload_all`) that provide an interface between the workarea of the Rational Statemate user and external files.

You must work in the `automatic_transaction` mode when using this function by specifying `automatic_transaction` as the third argument (`trans_mode`) of the `stm_init_uad` function. Your program should contain the following call:

```
int success, status;
...
success = stm_init_uad ("MY_PROJECT",
    "/local/my_work_area", automatic_transaction,
    &status);
if (!success)
...
```

Syntax

```
stm_save (file_name, item_name, message, &status)
```

Arguments

Argument	Input/Output	Type	Description
`file_name`	In	`char *`	The full path name for the file. Any name in any directory can be specified for `file_name`. In charts, the chart is converted into an ASCII format and written to the specified file. The specified file should not exist before calling this function. Note that no description or lock files are created by this function.

Argument	Input/Output	Type	Description
item_name	In	char *	The item name and type. The possible values are as follows: • ach - Activity-charts • cgenset - Compilation profiles • chk_mdl_set - Check Model profiles • cnf - Simulation status files • config - Configuration files • dgl - Documentor templates • dic - Global definition sets • dyn_set - Simulation analysis profiles • inc - Documentor include files • mch - Module-charts • pnl - Panels of the Prototyper • req - Requirement files • sch - For statecharts • scp - Simulation SCL files
message	Out	char *	A buffer that holds the error message, if an error occurs. This buffer can hold 127 characters.
status	Out	int	The function status code.

Status Codes

- stm_success
- stm_error_in_save_operation
- stm_chart_not_in_database
- stm_file_not_in_work_area
- stm_cannot_copy_file

Example

The following call saves a statechart in the workarea named SYSTEM_CHART to an external file named saved_chart in the /tmp directory:

```
int  status;
char mess[128];
.
.
stm_save ("system_chart.sch", "/tmp/saved_chart",
   message, &status);
if (status != stm_success)
.
.
```

stm_select_id

Function Type

```
stm_boolean
```

Description

Enables selection of an element by its ID in Statemate Graphical Editor.

> **Note:** The function works only for real elements. Generated IDs, such as for flow-lines are not legal. You can pass as an argument an ID of the basic arrow which constructs the flow-line.

Syntax

```
stm_select_id (id, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`id`	In	`stm_id`	The element ID
`status`	Out	`int`	Function status code

Status Codes

- `stm_success`
- `stm_error_in_open_socket_to_statemate`
- `stm_id_not_found`

Example

To select an activity by the name of A1 in , use the following statements:

```
stm_id ac_id;
int             status;

ac_id = stm_r_ac ("A1", &status);
stm_select_id(id, &status);
```

stm_start_transaction

Function Type

```
void
```

Description

Enables transaction operations on the database. If you are retrieving information from a database that has changed since the last transaction opening (the last start), it is important to do a commit followed by another start before calling a new function. This sequence establishes access to database changes because it refreshes the database image in memory.

> **Note:** This function is relevant only in `self_transaction` mode. In the `automatic_transaction` mode, the `start` and `commit` functions are performed automatically.

Syntax

```
stm_start_transaction ()
```

Example

To enable transaction operations on the database, use the following statements:

```
stm_start_transaction();
stm_r...                        -- a retrieval function
            .
            .
```

stm_start_transaction_rw

Function Type

```
void
```

Description

Enables read/write transaction operations on the database.

Syntax

```
stm_start_transaction_rw ()
```

stm_trigger_of_reaction

Function Type

```
stm_expression
```

Description

Returns the trigger part of a reaction (label of transition or static reaction). The syntax of the reaction is `trigger/action`.

Note

- The reaction is achieved by the following single-element functions:
  - stm_r_st_reactions
  - stm_r_tr_labels
- The function returns an empty string when the trigger is missing.

Syntax

```
stm_trigger_of_reaction (reaction, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`reaction`	In	`char *`	The reaction to decompose
`status`	Out	`int`	The function status code

Status Codes

```
stm_success
```

Example

To list all events that have influence on `S1`, which has several static reactions, use the following statements:

```
    stm_id            st_id;
    int               status;
    stm_list          reactions;
    stm_expression  rct;

st_id = stm_r_st ("S1", status);
reactions = stm_r_st_reactions (st_id, status);
printf ("\n Triggers of reaction is S1:");
 for (rct= (string)
      stm_list_first_element (reaction, &status);
      status == stm_success;
      rct = (string)
      stm_list_next_element (reaction, &status))
      printf ("\n %S", stm_trigger_of_reaction (rct,
         status));
```

stm_uad_attribute

Function Type

```
int
```

Description

Writes the predefined attribute report to the specified output file. The output contains commands for the specified word processor.

Syntax

```
stm_uad_attribute (elist, attrs, attr_title, file_name,wp, append,
with_header, p_width, pheight)
```

Arguments

Argument	Input/ Output	Type	Description
elist	In	stm_list	Lists the Rational Statemate elements for which the report is produced.
attrs	In	stm_list	Lists the names of the attributes for which the report should be generated. If this list is empty, the report retrieves all the attributes for each element.
attr_title	In	stm_attr_name	Specifies the attribute whose value precedes the element name in the report.
file_name	In	stm_filename	The name of the output file.
wp	In	char*	The target word processor.
append	In	stm_boolean	Determines whether the new information is appended to the output file, if it already exists.
with_header	In	stm_boolean	Specifies whether to include set-up commands to the word processor (true).
p_width	In	int	Specifies the page width.
p_height	In	int	Specifies the page length.

Status Codes

- stm_success
- stm_unknown_plotter
- stm_can_not_open_file

stm_uad_dictionary

Function Type

```
int
```

Description

Writes the predefined dictionary report to the specified output file. The output contains commands for the designated word processor.

Syntax

```
stm_uad_dictionary (elist, ldes, attr, attr_title,file_name, wp, append,
with_header, p_width, pheight)
```

Arguments

Argument	Input/ Output	Type	Description
elist	In	stm_list	Lists the Rational Statemate elements for which the report is produced.
ldes	In	stm_boolean	If this is true, the long description of each element is included in the report.
attr	In	stm_boolean	If this is true, include the element's attributes in the report.
attr_title	In	stm_attr_name	Specifies the attribute whose value precedes the element name in the report.
file_name	In	stm_filename	The name of the output file.
wp	In	char*	The target word processor.
append	In	stm_boolean	Determines whether the new information is appended to the output file, if it already exists.
with_header	In	stm_boolean	Specifies whether to include set-up commands to the word processor (true).
p_width	In	int	Specifies the page width.
p_height	In	int	Specifies the page length.

Status Codes

- stm_success
- stm_unknown_plotter
- stm_can_not_open_file

stm_uad_interface

Function Type

```
int
```

Description

Writes the predefined attribute report to the specified output file. The output contains commands for the specified word processor.

Syntax

```
stm_uad_attribute (elist, attrs, attr_title, file_name,wp, append,
with_header, p_width, pheight)
```

Arguments

Argument	Input/ Output	Type	Description
elist	In	stm_list	Lists the Rational Statemate elements for which the report is produced.
attrs	In	stm_list	Lists the names of the attributes for which the report should be generated. If this list is empty, the report retrieves all the attributes for each element.
attr_title	In	stm_attr_name	Specifies the attribute whose value precedes the element name in the report.
file_name	In	stm_filename	The name of the output file.
wp	In	char*	The target word processor.
append	In	stm_boolean	Determines whether the new information is appended to the output file, if it already exists.
with_header	In	stm_boolean	Include set-up commands to the word processor.
p_width	In	int	Specifies the page width.
p_height	In	int	Specifies the page length.

Status Codes

- stm_success
- stm_unknown_plotter
- stm_can_not_open_file

stm_uad_list

Function Type

```
int
```

Description

Writes the predefined list report to the specified output file. The output contains commands for the designated word processor.

Syntax

```
stm_uad_list (elist, file_name, wp, append, with_header,p_width, pheight)
```

Arguments

Argument	Input/ Output	Type	Description
elist	In	stm_list	The list of elements for which the report is produced.
file_name	In	stm_filename	The name of the output file.
wp	In	char*	The target word processor.
append	In	stm_boolean	Determines whether the new information is appended to the output file, if it already exists.
with_header	In	stm_boolean	Specifies whether to include set-up commands to the word processor (true).
p_width	In	int	Specifies the page width.
p_height	In	int	Specifies the page length.

Status Codes

- stm_success
- stm_unknown_plotter
- stm_can_not_open_file

stm_uad_n2

Function Type

```
int
```

Description

Writes the predefined N2-chart report to the specified output file. The output contains commands for the designated word processor.

Syntax

```
stm_uad_n2 (elist, names, level, env, chart, dis, ftype, file_name, wp,
append, with_header, p_width, pheight)
```

Arguments

Argument	Input/ Output	Type	Description
`elist`	In	`stm_list`	A list expression, which must be of the type `list of modules` or `list of activities`, that specifies the elements in the diagonal.
`names`	In	`char`	Specifies whether to display the names (`N`) or synonyms (`S`) of the elements that appear on the diagonal of the matrix.
`level`	In	`char`	Specifies whether the sub-box (`B`) or parent box (`P`) is placed on the diagonal of the matrix.
`env`	In	`stm_boolean`	If this is true, the environment is added to the matrix.
`chart`	In	`char`	Indicates whether activity-chart (`A`) or module-chart (`M`) arrows are taken into account when the report is generated.
`dis`	In	`char`	Specifies the kind of information to appear in the report. The possible values are as follows: • I - Flow labels • P - Parent information items • B - Basic information items
`ftype`	In	`char`	Specifies the kind of information flows to show in the report. The possible values are as follows • `D` - Data-flows • `C` - Control-flows • `B` - Both data- and control-flows

Argument	Input/ Output	Type	Description
file_name	In	stm_filename	The name of the output file.
wp	In	char*	The target word processor.
append	In	stm_boolean	Determines whether the new information is appended to the output file, if it already exists.
with_header	In	stm_boolean	Specifies whether to include set-up commands to the word processor (true).
p_width	In	int	Specifies the page width.
p_height	In	int	Specifies the page length.

Status Codes

- stm_success
- stm_unknown_plotter
- stm_can_not_open_file

stm_uad_protocol

Function Type

```
int
```

Description

Writes the predefined protocol report to the specified output file. The output contains commands for the specified word processor.

Syntax

```
stm_uad_protocol (elist, attr_title, file_name, wp,append, with_header,
p_width, pheight)
```

Arguments

Argument	Input/ Output	Type	Description
elist	In	stm_list	Lists the Rational Statemate elements for which the report is produced.
attr_title	In	stm_attr_name	Specifies the attribute whose value precedes the element name in the report.
file_name	In	stm_filename	The name of the output file.
wp	In	char*	The target word processor.
append	In	stm_boolean	Determines whether the new information is appended to the output file, if it already exists.
with_header	In	stm_boolean	Specifies whether to include set-up commands to the word processor (true).
p_width	In	int	Specifies the page width.
p_height	In	int	Specifies the page length.

Status Codes

- stm_success
- stm_unknown_plotter
- stm_can_not_open_file

stm_uad_resolution

Function Type

```
int
```

Description

Writes the predefined resolution report to the specified output file. The output contains commands for the specified word processor.

Syntax

```
stm_uad_resolution (clist, type, file_name, wp, append,with_header, p_width,
pheight)
```

Arguments

Argument	Input/ Output	Type	Description
clist	In	stm_list	Lists the Rational Statemate charts for which the report is produced.
type	In	stm_attr_name	Specifies the type of element to include in the report. The possible values are as follows: • stm_textual • stm_graphical • stm_mixed • stm_state • stm_module • stm_activity • stm_data_store
file_name	In	stm_filename	The name of the output file.
wp	In	char*	The target word processor.
append	In	stm_boolean	Determines whether the new information is appended to the output file, if it already exists.
with_header	In	stm_boolean	Specifies whether to include set-up commands to the word processor (true).
p_width	In	int	Specifies the page width.
p_height	In	int	Specifies the page length.

Status Codes

- stm_success
- stm_unknown_plotter
- stm_can_not_open_file

stm_uad_state_interface

Function Type

```
int
```

Description

Writes the predefined state interface report to the specified output file. The output contains commands for the specified word processor.

Syntax

```
stm_uad_state_interface (elist, file_name, wp, append,with_header, p_width,
pheight)
```

Arguments

Argument	Input/ Output	Type	Description
elist	In	stm_list	Lists the Rational Statemate elements for which the report is produced.
file_name	In	stm_filename	The name of the output file.
wp	In	char*	The target word processor.
append	In	stm_boolean	Determines whether the new information is appended to the output file, if it already exists.
with_header	In	stm_boolean	Specifies whether to include set-up commands to the word processor (true).
p_width	In	int	Specifies the page width.
p_height	In	int	Specifies the page length.

Status Codes

- stm_success
- stm_unknown_plotter
- stm_can_not_open_file

stm_uad_structure

Function Type

```
int
```

Description

Writes the predefined structure report to the specified output file. The output contains commands for the specified word processor.

Syntax

```
stm_uad_structure (elist, width, file_name, wp, append,with_header, p_width,
pheight)
```

Arguments

Argument	Input/ Output	Type	Description
elist	In	stm_list	Lists the Rational Statemate elements for which the report is produced.
width	In	int	The report width, in inches.
file_name	In	stm_filename	The name of the output file.
wp	In	char*	The target word processor.
append	In	stm_boolean	Determines whether the new information is appended to the output file, if it already exists.
with_header	In	stm_boolean	Specifies whether to include set-up commands to the word processor (true).
p_width	In	int	Specifies the page width.
p_height	In	int	Specifies the page length.

Status Codes

- stm_success
- stm_unknown_plotter
- stm_can_not_open_file

stm_uad_tree

Function Type

```
int
```

Description

Writes the predefined tree report to the specified output file. The output contains commands for the specified word processor.

Syntax

```
stm_uad_tree (elist, depth, file_name, wp, append,with_header, p_width,
pheight)
```

Arguments

Argument	Input/ Output	Type	Description
elist	In	stm_list	Lists the Rational Statemate elements for which the report is produced.
depth	In	int	Specifies the hierarchical level to which the report should be generated. To include all levels, use the value 99.
file_name	In	stm_filename	The name of the output file.
wp	In	char*	The target word processor.
append	In	stm_boolean	Determines whether the new information is appended to the output file, if it already exists.
with_header	In	stm_boolean	Specifies whether to include set-up commands to the word processor (true).
p_width	In	int	Specifies the page width.
p_height	In	int	Specifies the page length.

Status Codes

- stm_success
- stm_unknown_plotter
- stm_can_not_open_file

stm_unload

Function Type

```
void
```

Description

Unloads (deletes from the current workarea) a chart or any other configuration item file. It is one of the four utility functions (`stm_load`, `stm_save`, `stm_unload`, and `stm_unload_all`) that provide an interface between the workarea of the Rational Statemate user and external files.

You must work in the `automatic_transaction` mode when using this function by specifying `automatic_transaction` as the third argument (trans_mode) of the `stm_init_uad` function. Your program should contain lines similar to the following:

```
int success, status;
.
.
success = stm_init_uad ("MY_PROJECT",
    "/local/my_work_area", automatic_transaction,
    &status);
if (!success)
.
.
```

Syntax

```
stm_unload (item_name, enforce, message, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`item_name`	In	`char *`	The item name and type. The possible values are as follows: • `ach` - Activity-charts • `cgenset` - Compilation profiles • `chk_mdl_set` - Check Model profiles • `cnf` - Simulation status files • `config` - Configuration files • `dgl` - Documentor templates • `dic` - Global definition sets • `dyn_set` - Simulation analysis profiles • `inc` - Documentor include files • `mch` - Module-charts • `req` - Requirement files • `sch` - Statecharts • `scp` - Simulation SCL files
`enforce`	In	`stm_boolean`	If this is true, it enforces the load—even in cases when the new or modified item with the same name already exists in the workarea. If this is false, the load operation fails, in these cases, with the corresponding status code and error message.
`message`	Out	`char *`	A buffer that holds the error message, if an error occurs. This buffer can hold 127 characters.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_chart_not_in_database`
- `stm_file_not_in_work_area`
- `stm_chart_is_active`
- `stm_not_unloaded_modified`
- `stm_not_unloaded_new`

Example

To unload (delete from the current workarea) an activity-chart named `bad_chart`, use the following statements:

```
#define DONT_ENFORCE 0
int   status;
char  mess[128];
.
.
stm_unload ("bad_chart.ach", DONT_ENFORCE, message,
    &status);
if (status != stm_success)
.
.
```

To avoid losing information, do not enforce the operation in case this chart is new or modified.

stm_unload_all

Function Type

```
void
```

Description

Unloads all charts from the current workarea and clears all database fields. It is one of the four utility functions (`stm_load`, `stm_save`, `stm_unload`, and `stm_unload_all`) that provide an interface between the workarea of the Rational Statemate user and external files.

This function is *not* equivalent to calling the `stm_unload` function for each chart in the database. The differences between these two functions are as follows:

- `stm_unload` takes a fixed amount of time regardless of the number of charts in the database. It does not perform an unload of individual charts, but rather cleans all database data. Usually this function cleans the database much faster than by unloading individual charts one by one.
- `stm_unload_all` clears the internal ID counter of charts in the database, whereas the `stm_unload` function does not. This counter starts from 0 when the workarea database is initially created and is incremented each time a chart is created or loaded into the database. It is not decremented when charts are unloaded (deleted). When this counter reaches the value of 1023, no more charts can be loaded or created in the database. There are two ways to reset this counter: calling the `stm_unload_all` function from a program, or interactively via the **Delete charts from the Workarea** option of Rational Statemate with the **Delete all** and **Without confirmation** flags set.

  Note: When using this function, you must work in the automatic transaction mode by specifying `automatic_transaction` as the third argument (`trans_mode`) of the `stm_init_uad` function. Your program should contain lines similar to the following:

```
int success, status;
...
success = stm_init_uad("MY_PROJECT",
         "/local/my_work_area",
             automatic_transaction, &status);
if (!success)
 ....
```

Syntax

```
stm_unload_all (message, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`message`	Out	`char *`	A buffer that holds the error message, if an error occurs. This buffer can hold 127 characters.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_chart_is_active`

Project Management

This section describes special project management functions. For each function, the following information is provided:

- Description
- Syntax
- Arguments
- Status codes

The following table lists the project management functions.

Function	Description
stm_r_pm_member_workareas	Returns the workareas of the specified user.
stm_r_pm_operator_projects	Returns a list of all the projects in which the specified user is a member.
stm_r_pm_project_databank	Returns the databank name of the specified project in the project management database.
stm_r_pm_project_manager	Returns the manager of the specified project in the project management database.
stm_r_pm_project_members	Returns a list of all the members of the specified project in the project management database.
stm_r_pm_projects	Returns a list of all the projects in the project management database.

stm_r_pm_member_workareas

Function Type

`stm_list`

Description

Returns the workareas of the specified user.

Syntax

```
stm_r_pm_member_workareas (o_name, p_name, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`o_name`	In	`char *`	The name of the user.
`p_name`	In	`char *`	The name of the project.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nonexistent_project`
- `stm_not_member_of_project`

stm_r_pm_operator_projects

Function Type

```
stm_list
```

Description

Returns a list of all the projects in which the specified user is a member.

Syntax

```
stm_r_pm_operator_projects (oname, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`o_name`	In	`char *`	The name of the user.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_no_projects`

stm_r_pm_project_databank

Function Type

```
char *
```

Description

Returns the databank name of the specified project in the project management database.

Syntax

```
stm_r_pm_project_databank (pname, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`pname`	In	`char *`	The name of the project.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nonexistent_project`

stm_r_pm_project_manager

Function Type

```
char *
```

Description

- Returns the manager of the specified project in the project management database.

Syntax

```
stm_r_pm_project_manager (pname, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`pname`	In	`char`	The name of the project.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nonexistent_project`

stm_r_pm_project_members

Function Type

```
stm_list
```

Description

Returns a list of all the members of the specified project in the project management database.

Syntax

```
stm_r_pm_project_members (pname, &status)
```

Arguments

Argument	Input/ Output	Type	Description
`pname`	In	`char *`	The name of the project.
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_nonexistent_project`

stm_r_pm_projects

Function Type

```
stm_list
```

Description

Returns a list of all the projects in the project management database.

Syntax

```
stm_r_pm_projects (&status)
```

Arguments

Argument	Input/ Output	Type	Description
`status`	Out	`int`	The function status code.

Status Codes

- `stm_success`
- `stm_no_projects`

Data Types

The data types are defined in the `dataport.h` file, which you must include in your C program.

The file is located in your Statmate installation path under the include directory.

Function Status Codes

Dataport functions return only one output parameter, the function status code. This code reports whether the function call was successfully completed. If the function call fails, the status code indicates the problem. This status code can be used to pinpoint run-time errors in your program.

For example, assume the following call appears in your program:

```
state_id = stm_r_st ("%", &status);
```

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

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

Status codes have three severity levels:

- **S** for success
- **W** for warning
- **E** for error

When a warning or error status is returned, attempts to execute statements using the return value of the function can produce erroneous or unexpected results. Therefore, you should check the return status codes to ensure that your function call is successful before using the returned values.

The following table lists the status codes and their severity levels.

Code	Status Name	Severity Level
–4	`stm_no_stm_root` **UNIX**: The `STM_ROOT` environment variable is not defined. **VMS**: The `STM$ROOT` or `STM$PM` logical name does not exist.	E
–3	`stm_obsolete_function` Irrelevant function for the current version.	E
–2	`stm_missing_elements_in_list` Input elements do not exist in the database.	W
–1	`stm_list_type_mismatch` Incorrect element type used in the query.	E
0	`stm_success` The function call was successful.	S
1	`stm_id_out_of_range` The specified ID is not valid for this element type.	E
2	`stm_id_not_found` An element with the specified ID does not exist.	E
3	`stm_illegal_name` The specified name is not legal.	E
4	`stm_name_not_found` The specified name does not exist.	E
5	`stm_name_not_unique` There is more than one element with the specified name, so a specific path name is required.	E
6	`stm_missing_name` The specified element has no name.	W
7	`stm_missing_synonym` The specified element has no synonym.	W
8	`stm_missing_short_description` The specified element has no short description.	W
9	`stm_missing_long_description` The specified element has no long description.	W

10	stm_attribute_name_not_found The specified element has no attribute name.	W
11	stm_starting_keyword_not_found The long description of the specified element does not contain the given starting keyword.	W
12	stm_ending_keyword_not_found The long description of the specified element does not contain the given ending keyword.	W
13	stm_primitive_element The element is primitive.	W
14	stm_can_not_open_file The operating system cannot open the file with the specified name.	E
15	stm_illegal_address The pointer address is illegal.	E
16	stm_not_an_and_state This state is not supposed to contain and-lines.	W
17	stm_no_and_lines_in_and_state This and-state is missing and-lines.	E
18	stm_missing_graphic_data Graphic data is missing from the element.	E
19	stm_nil_list There is no input list.	E
20	stm_list_element_does_not_exist The specified element does not exist.	W
21	stm_cannot_compare_lists The lists cannot be compared because the list types are different, or the lists are not initialized.	E
22	stm_elements_without_name The list cannot be sorted because its elements have no names.	E
23	stm_elements_without_synonym The list cannot be sorted because its elements have no synonyms.	E
24	stm_elements_not_hierarchical The list cannot be sorted because it is not hierarchical.	E
25	stm_illegal_extract_type You cannot extract this element type.	E

26	stm_no_such_list No such list exists	E
27	stm_not_diagram_connector There is no value in a connector that is not a diagram connector.	E
28	stm_implicit_element The element is defined implicitly—it is an internally defined entity.	E
29	stm_missing_label The element has no label.	W
30	stm_unknown_plotter The plotter type is unknown.	E
31	stm_unresolved The element is unresolved.	W
32	stm_elements_without_attributes The list cannot be sorted because its elements have no attributes.	E
33	stm_not_instance The element is not an instance.	E
34	stm_no_updated_pmdb The workarea database is not updated to the current version.	E
35	stm_no_updated_projdb The installation database is not updated to the current version.	E
36	stm_no_legal_operator The user is not authorized as a Rational Statemate operator.	E
37	stm_deadlock Deadlock situation.	E
38	stm_not_member_of_project The user is not a member of the specified project.	E
39	stm_nonexistent_project The specified project does not exist.	E
40	stm_not_enough_memory The plot cannot be produced because there is not enough memory.	E
41	stm_empty_chart The plot file cannot be produced because the chart is empty.	E

42	stm_plot_failure The plot file was not produced because of a system error.	E
43	stm_no_file_of_licensed_host The file containing the name of the licensed host does not exist.	E
44	stm_empty_file_of_licensed_host The file containing the name of the licensed host is empty.	E
45	stm_cannot_chdir_to_work_area Could not change directory to the workarea.	E
46	stm_cannot_write_to_file No space is left on device for writing a file.	E
47	stm_illegal_parameter An illegal parameter value was supplied.	E
48	stm_illegal_parameter_mode Illegal parameter mode.	E
49	stm_illegal_parameter_name Illegal parameter name.	E
50	stm_null_string The input string is null.	E
51	stm_illegal_len The length value is illegal.	E
52	stm_illegal_index The index value is illegal.	E
53	stm_cannot_read_file Cannot read from a file that was not opened.	E
54	stm_end_of_file Reached the end-of-file.	E
55	stm_not_a_parameter The specified ID is not a parameter.	E
56	stm_param_not_compatible The actual and formal parameters are not compatible.	W
57	stm_error_in_file There is an error in the requirement file.	E

58	stm_missing_field A field is missing in the requirement record.	W
59	stm_missing_user_type The specified element has no user-defined type.	E
60	stm_illegal_attribute_name The attribute name is illegal.	E
61	stm_illegal_attribute _value The attribute value is too long.	E
62	stm_duplicate_attribute _pair The specified attribute name/value pair already exists.	E
63	stm_not_in_rw_transaction Attempt to modify the database when not in a read/write transaction.	E
64	stm_missing_of_enum_type The specified element has no enumerated type associated with its array type definition.	W
65	stm_missing_user_code The specified element has no user code.	W
66	stm_missing_subroutine_params The specified element has no subroutine parameters.	W
67	stm_missing_local_data The specified element has no local data.	W
68	stm_missing_global_data The specified element has no global data.	W
69	stm_no_connected_chart The specified element is not connected to a chart.	W
70	stm_attribute_cannot_be_deleted The specified element's attribute cannot be deleted.	E
71	stm_missing_cbk_binding The specified element has no callback binding.	W
72	stm_missing_subroutine_binding The specified element has no subroutine binding.	W
73	stm_missing_statemate_action_lang The specified element has no action language.	W

74	stm_no_projects There are no projects in the project management database.	W
75	stm_member_has_no_wa	E
76	stm_missing_external_link Specific element has no long description.	W
77	stm_not_chart_id	E
78	stm_message_not_found	E
79	stm_not_referenced_sd	E
80	stm_not_timing_constraint	E
81	stm_not_order_insignificant	E
82	stm_missing_note Element has no note.	W
83	stm_missing_description_file Element does not have an external description file defined.	W
84	stm_not_boundry_box	E
85	stm_not_use_case	E
86	stm_not_actor	E
87	stm_not_partition	E
88	stm_not_sequence_diagram	E
89	stm_not_activity	E
90	stm_invalid_use_case_scen_num	E
91	stm_missing_extention_point_definition	W
92	stm_missing_timing_constraint_note	W
93	stm_no_use_case_scen_attr_defined	W
94	stm_use_case_scen_attr_val_not_defined	W
95	stm_illegal_chart	E
96	stm_info_flow_component_exists	E
97	stm_missing_info_flow_component	W

98	stm_generic_chart_not_in_database	E
99	stm_cannot_delete_parent_of_control_activity	E
100	stm_hyperlinked_expression_not_implemented_for_ plotter	W
101	stm_illegal_param_min_val	E
102	stm_illegal_param_max_val	E
103	stm_illegal_param_ba_lindex	E
104	stm_illegal_param_ba_rindex	E
105	stm_illegal_param_user_type	E
106	stm_illegal_param_enum_type	E
107	stm_illegal_param_type	E
108	stm_illegal_param_structure_type	E
109	stm_illegal_local_var_structure_type	E
110	stm_illegal_short_description_length	E
111	stm_illegal_local_var_min_val	E
112	stm_illegal_local_var_max_val	E
113	stm_illegal_local_var_ba_lindex	E
114	stm_illegal_local_var_ba_rindex	E
115	stm_illegal_local_var_user_type	E
116	stm_implementation_missing	E
117	stm_implementation_exists	E
118	stm_missing_subroutine	E
119	stm_illegal_global_var_mode	E
120	stm_illegal_global_var_name	E
121	stm_illegal_expression_n_chart There is an illegal expression in the loaded chart.	E
122	stm_error_in_chart There is an error in the loaded chart.	E

123	stm_cannot_open_chart_file Cannot open the chart file to be loaded.	E
124	stm_exceeded_max_id_number There are more than 1023 IDs in the workarea.	E
125	stm_chart_not_in_database Cannot find a chart in the database to be saved or unloaded.	E
126	stm_file_not_in_work_area Cannot find a file in the workarea to be saved or unloaded.	E
127	stm_cannot_copy_file Cannot copy a file during a save or load operation.	E
128	stm_cannot_create_file Cannot create an auxiliary file during a load to the workarea.	E
129	stm_illegal_version An illegal version was specified for the load operation.	E
130	stm_file_not_found Cannot find a source file in the load operation.	E
131	stm_not_loaded_because_modified A modified version of loaded chart or file exists in the workarea.	E
132	stm_not_loaded_because_new A new version of the loaded chart or file exists in the workarea.	E
133	stm_not_unloaded_modified The chart or file to be unloaded is modified.	E
134	stm_not_unloaded_new The chart or file to be unloaded is new.	E
135	stm_chart_is_active The chart to be unloaded is currently being edited by a graphics editor.	E
136	stm_error_in_save_operation There was a write to disk error during the save operation.	E
137	stm_illegal_load_mode An illegal mode was specified for the load operation.	E
138	stm_not_loaded_because_type A chart with the same name, but of another type, exists in the workarea.	E

139	stm_illegal_type An illegal type of configuration item was specified.	E
140	stm_illegal_parameters An illegal parameter to the load function was specified.	E
141	stm_illegal_bindings There is an error in the loaded chart file.	E
142	stm_too_long_line There is a line too long in the loaded chart file.	E
143	stm_instance_type_conflict There is an instance type conflict in the loaded chart file.	E
144	stm_usage_conflict There is a usage conflict in the loaded chart file.	E
145	stm_unrecognized_format The loaded chart file contains an unrecognized conflict.	E
146	stm_double_chart_parameters There is an error in the loaded chart file.	E
147	stm_double_chart_bindings There is an error in the loaded chart file.	E
148	stm_no_bindings	W
149	stm_missing_truth_table	E
150	stm_truth_table_invalid_row	E
151	stm_component_interface_changed	E
152	stm_cannot_load_component	E
153	stm_cannot_open_new_wa	E
154	stm_element_exists	E
156	stm_coordinates_out_of_range	E
157	stm_illegal_coordinates	E
158	stm_illegal_local_var_enum_type	E
159	stm_illegal_local_var_type	E
160	stm_illegal_local_var_name	E

161	stm_truth_table_invalid_column	E
162	stm_invalid_truth_table_cell	E
163	stm_truth_table_convert_failed	E
164	stm_conflicting_array_indices_types	E
165	stm_invalid_sd_scope	W
166	stm_sd_scope_not_defined	W
167	stm_use_all_public_gds	E
168	stm_error_in_backup	E
169	stm_not_message	E
170	stm_cannot_delete_file	E
171	stm_plot_illegal_option_key	E
172	stm_plot_illegal_option_val	E
173	stm_no_local_vars_in_selected_impelentation	E
174	stm_illegal_hyperlink_format	E
175	stm_illegal_font_name	E
176	stm_illegal_factor_value	E
177	stm_invalid_key	E
178	stm_no_legal_wa_operator The user is not authorized as the workarea operator.	E
179	#define stm_error_in_open_socket_to_statemate	E

Index

F

G

H

I

L

M

N

P

Q

R

S

T

U

W