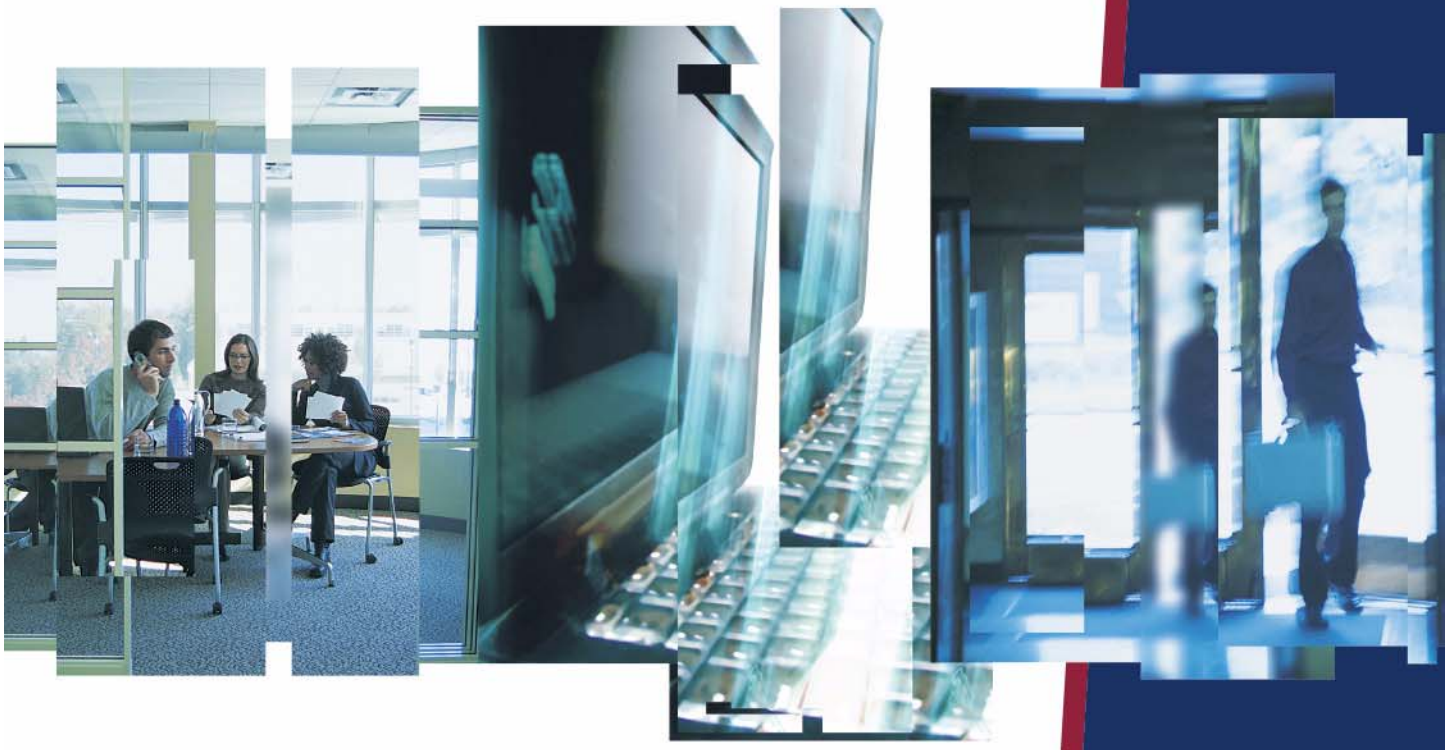
Telelogic

Statemate®

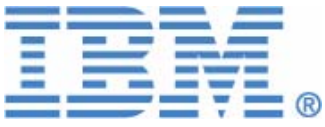
CG Builder API Reference Guide



IBM®

Statemate®

CG Builder API Reference Guide



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

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

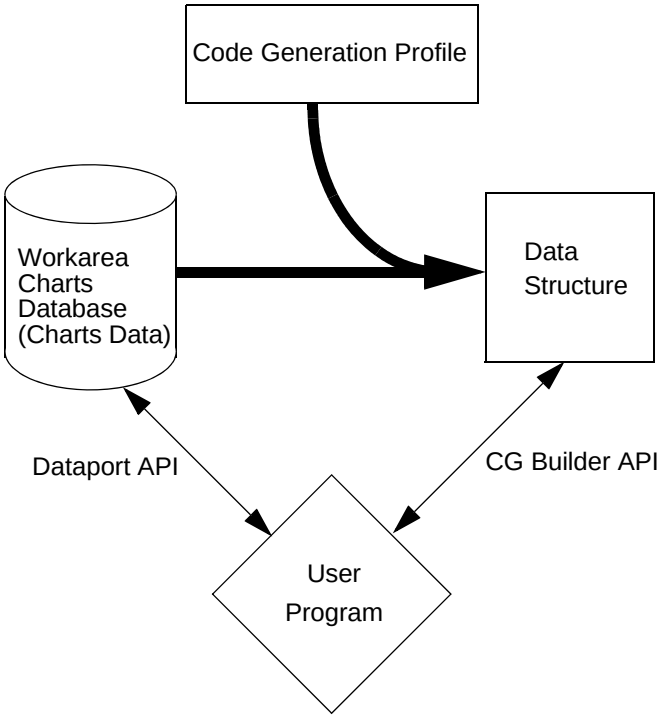
Contents

Introduction to the CG Builder API

CG Builder API contains functions that allow you to extract information from the Statemate database. With this information, the CG Builder API enables advanced users to build Statemate-based programs such as a customized code generator or model analyzer.

CG Builder API and Dataport

CG Builder API is similar to Dataport in that you can use it to write programs in C that extract information from a Statemate workarea and process the information as you choose.



Tool Differences

The tools differ in that

- Dataport is designed to extract model data.
- CG Builder API is designed to produce programs.

Dataport queries concentrate on extracting model information that is well suited for creating *documentation* such as descriptions and the text with which transitions are labeled.

The CG Builder API is a library of routines that extracts information for the purpose of generating code. It is also useful in creating a program to analyze the model in ways that are not provided by the Statemate tool set.

Features of the CG Builder API Library

CG Builder API functions have a C language interface that can be called from C language programs, as well as from programs written in another language that can call C functions.

CG Builder API has the following features:

- Supplies a Dataport-like library that provides access to the data structures used in the analyzer and code generator tools.
- Provides access to the mathematical structures and parsed expressions within transition labels and definitions.
- Targets programs that generate code or semantically analyze Statemate models.
- Controls the scope of analysis of the model through the use of code generation profiles. The module structure in the profile is supported.

Functions and Utilities

CG Builder API provides a set of functions and utilities that allow you to access all the Statemate objects and related data-structures. The file `api_types.h` includes definitions of all CG Builder API data-structures.

Lists (`stmm_list`) maintain the information on data structures in memory. These lists are similar to those provided in Dataport or in the Properties Browser. CG Builder API provides utilities to navigate through these lists and extract information from them.

Working with the CG Builder API

CG Builder API requires a defined scope in order to provide access to the model information. This scope is defined in terms of a workarea and an existing C or Ada Code Generation Profile.

Within the profile (scope), the preliminary division is that of modules. These are the modules you created as part of the act of defining the profile. Associated with each module are all the basic building-blocks of the database: charts, states, activities, data-items, and so forth. You can extract this information from the database by including calls to various CG Builder API functions in your program. For information on creating modules, refer to Defining Code Modules in the *Code Generation Reference Manual*.

Initialize

Initialize CG Builder API with the following function call:

```
stmm_open_session(project,workarea,profile,with_dataport)
```

This call accepts a project, workarea, a C or Ada code generation profile name, and the parameter `with_dataport`. When you invoke this call, CG Builder API reads the workarea and profile and builds a data structure in memory that other CG Builder API functions can understand and access. The form that is in memory already understands the model in the same way that the simulator or the code generator does.

Note

This eliminates the need for you to parse Statemate internal data schema and enables you to work at a ‘modeling’ level.

License Issues

CG Builder API and Dataport require separate licenses. If you have licenses for both, you can use the CG Builder API to access Dataport routines. It is the programs that you write using the API that use the licenses. The licenses are taken and released when the `stmm_open_session` and `stmm_close_session` are made.

If a program accesses both CG Builder API and Dataport calls, it requires both a CG Builder API and a Dataport license (and a Kernel license) to execute.

In order to use functions from the CG Builder API and Dataport in the same program, call the initialization and termination routines with the `with_dataport` parameter:

```
#define with_dataport 1

stmm_open_session(project, workarea, profile,  with_dataport);

stmm_close_session(with_dataport);
```

The program must also include the Dataport `include` file `dataport.h`.

Compiling, Linking, and Running CG Builder for Solaris System

After writing your program, compile it with the following command:

```
cc -c -g myprog.c -I<include directories> -o myprog.o
```

After writing and compiling your program, link it with the CG Builder API Library image:

```
cc -o myprog myprog.o \
-L<path for libanaport.so> -lanaport \
-lsocket -lm
```

Run your program.

Compiling, Linking, and Running CG Builder for Windows System

Use this Makefile to compile and link your application on a Windows system:

```
PROGRAM= myprog.exe
DLL= anaport.dll
DLIB = anaport.lib
HDR= <include files path>
SRCS= myprog.c

CFLAGS= /Zi /MD /W3 /DWINNT /DDLL_LINK /I$(HDR)
LIBS= kernel32.lib

all: $(PROGRAM) $(DLL)

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

Run your program.

Element Type Abbreviations

Statemate groups elements into several categories. The following table lists these elements along with their two-character abbreviation:

Element Types

Element	Abbreviation
Activity	ac
Action	an
Chart	ch
Compound transition/data-flow/control-flow/ signal/bus	cm
Condition	co
Connector	cn
Data_item	di
Enumerated_value	en
Event	ev
Field	fd
Information_flow	if
Life-line	ll
Message	msg
Order Insignificant	ord_insig
Reference Sequence Design	ref_sd
Scenario	scen
State	st
Subroutine	sb
Subroutine_data	sd

Element	Abbreviation
Textual elements (including an, co, di, en, ev, fd, if, sd, and ut)	tx
Time Constraint	tc
User_defined_type*	ut
* The CG Builder API uses **ut** for user_defined_type, and Dataport uses **dt** for user_defined_type. The CG Builder API also uses **dt**, but for the data-item sub-type (integer, real, and so forth).	

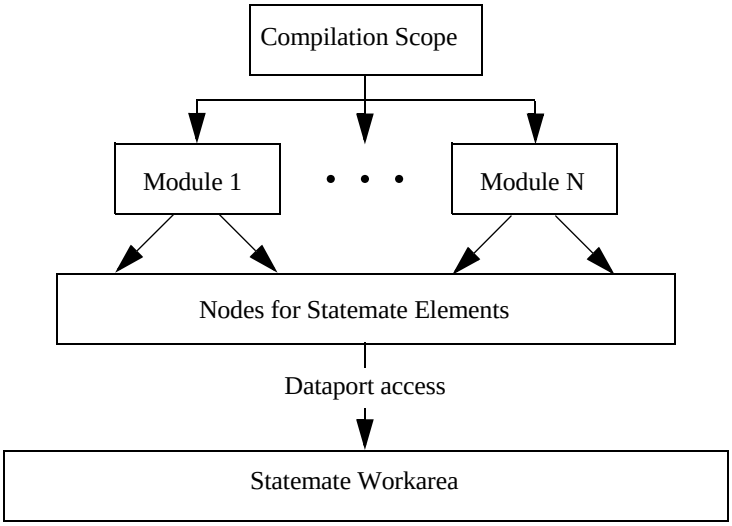
Types and Structures

This section explains the structure that the CG Builder API uses for general and special types of elements.

Data-Structure

When you call the `stmm_open_session()` function, the API builds a data-structure based on the code-generation profile. Most of the information you use from the API is contained directly in this data-structure.

You can access the data structure through a set of modules which you built in the compilation profile. These modules provide entry points for accessing the model information. The model information is structured as a cross-linked set of nodes, one node for each element in the model.



Scope of the Data Structure

This structure of reference only allows the program to access elements that are ‘in-scope’ for the code-generation profile. This is useful when you are generating code, because you never ‘find’ an element that you should not be generating code for using the CG API.

Note

It is possible to access non-relevant information through the Dataport calls if you are using them.

Element Nodes

The nodes for the elements are record structures that contain the kind of information you might find in the Properties definitions for the element. They are all based on the `stmm_el_node` structure (except context variables).

Note

All structures (for example, `stmm_el_node`) also have an associated type definition (for example, `stmm_el_nodep`) that is a pointer to the structure.

The following node types are defined:

- **stmm_el_node** - Base for all element nodes
- **stmm_tx_node** - All textual elements (an, co, di, en, ev, fd, if, sd, ut)
- **stmm_ch_node** - Chart
- **stmm_ac_node** - Activity
- **stmm_st_node** - State
- **stmm_sb_node** - Subroutine
- **stmm_cn_node** - Connector
- **stmm_cm_node** - Compound node for transition, data-flow, control-flow, module-flow, signal, or bus.

In addition, there are some other definitions that have nodes, even if they are not normally considered to be elements in a Statemate workarea.

- **stmm_module_node** - For the modules in the compilation profile.
- **stmm_cv_node** - For context variables
- **stmm_rpn_node** - For definition fields and other expressions.
- **one_limit_node** - For upper/lower ranges of arrays, integers, and so forth.

Typical Code Generator Program Structure

A typical code generator produces several output files containing different kinds of information. The order in which you generate the output files does not need to be the same as the order in which the compiler processes them.

Most programming languages separate the declaration of data-types and global variables from the code bodies or functions that become assembly language instructions. It may be easiest to structure your code generator program into phases:

1. **Phase 1** - Generate the data-declarations and data-types.
2. **Phase 2** - Build an internal data-structure that your generated code follows, probably based on Statemate's activity and state structure.
3. **Phase 3** - Run through the internal data-structure you just built, and generate code from it, using information from the element nodes.

In Phase 1, you may need to consider the following:

- Dependencies between data-types and data declarations affect the order in which you generate the code.
- For compound data-types, you may need to generate code to handle them. It may be necessary to store annotations about this so that the code is generated later.
- If there are special requirements for communication between different modules in the generated program, the analysis for the interface between modules needs to be done.

The structure of Phase 2 is recursive. The program contains subroutines for reading a node, and recursively following the links through the model structure to handle connected nodes. You need to decide whether to use the chart structure in your generated code, or whether to ignore it. If you choose to ignore chart structure, you need to handle off-page chart references.

Phase 3 is also a recursive program, in this case, around your own data-structure. In this phase, the body of the executable code is generated. The majority of the work here is in correctly translating expressions and the state transition relationships.

General Structure for Statemate Elements

A Statemate element is represented in the following general structure:

```
stmm_el_node
```

Note

The one exception to this rule is context variables (`stmm_cv_node`), which has its own structure.

Some of the fields in this structure are irrelevant for some types. In those cases, they have a default value.

id	stmm_id	always has a value	all elements
stmm_type	stmm_id_type	always has a value	all elements
name	string	NULL	tx/fn/ac/st/ch
code_gen_name	string	NULL	tx/ac/st
unique_name	string	NULL	tx/fn/ac/st
attributes*	stmm_list	NULL	tx/ac/st/ch
short_des	string	NULL	tx/ac/st/ch
synonym	string	NULL	tx/ac/st
cbk_status	stmm_cbk_status	stmm_cbk_missing	di/co/ev/ac/st
callback_binding	stmm_rpn_list	NULL	di/co/ev/ac/st
user_data**	genptr	NULL	all elements

***** The attributes field is an `stmm_list of stmm_attribute` structures. Each `stmm_attribute` contains two strings - attribute name and attribute value.

****** The `user_data` field is a generic pointer for your use.

General Structure for Textual Elements

The structure for textual elements `(stmm_tx_node)` is an extension of the general structure for Statemate elements `(stmm_el_node)`.

Textual elements include the following types with their abbreviation:

Textual Element	Abbreviation
actions	an
conditions	co
data-items	di
enumerated-values	en
events	ev
information-flows	if
fields	fd
subroutine-datas	sd
user-types	ut

Some of the fields in this structure are irrelevant for some types. In those cases, they have a default value.

id	stmm_id	always has a value	all textual types
stmm_type	stmm_id_type	always has a value	all textual types
name	string	always has a value	all textual types
code_gen_name	string	NULL	all textual types
unique_name	string	NULL	all textual types
attributes	stmm_list	NULL	all textual types
short_des	string	NULL	all textual types
synonym	string	NULL	all textual types
cbk_status	stmm_cbk_status	stmm_cbk_missing	di/co/ev/ac/st
callback_binding	stmm_rpn_list	NULL	di/co/ev/ac/st
user_data	genptr	NULL	all textual types
type	stmm_tx_type	always has a value	all textual types
data_structure	stmm_structure	stmm_missing	di/ut/fd/ev/co/sd
array_limits *	stmm_array_limits	NULL	di/ut/fd/ev/co/sd
data_type	stmm_data_type	stmm_dt_missing	di/ut/fd/sd

dt_limits **	stmm_dt_limits	NULL	di/ut/fd/co
user_defined_type	stmm_id	0	di/ut/fd/sd
ut_ref_by +	stmm_refered_by	ref_by_missing	di/ut/fd/sd
definition	stmm_rpn_list	NULL	di/ut/ev/co/an/if
fields	stmm_ids_list	NULL	di/ut
enum_type	stmm_id	0	en
ordinal	int	0	en
param_mode	stmm_gp_mode	‘ ’	sd

* The `array_limits` (of type `stmm_array_limits`) field is used in case the element is an array. This is a structure of two `one_limit_node` structures—left index and right index of the array.

The one_limit_node structure is:

stmm_refered_by refered_by;

stmm_id id;

int number;

If the `refered_by` field is

- **ref_by_literal** - Then the number field is that literal.
- **ref_by_name or ref_by_syn** - Then the `id` field is the id of that name or synonym.

** If the `stmm_type is stmm_data_item` and the `data_type is stmm_dt_user_def`, use the `ut_ref_by field`.

This field determines whether the data-item was defined by referring to the user-defined-type's name or synonym.

+ RPN Nodes and Lists

RPN nodes are used in any place where an element has an action language expression. These elements have a field of type `stmm_rpn_list`. These are special lists that contain RPN nodes (`stmm_rpn_node`), and store data in Reverse-Polish-Notation format.

RPN Node Structure

The following shows the `stmm_rpn_node` structure:

stmm_rpn_node_type	rpn_node_type	operator/identifier/constant/ context-var/field/ subroutine-local/ subroutine-parameter
stmm_id	id	id, when type is an identifier/ sb-local/sb-param
stmm_operator_type	operator_type	operator, when type is operator
stmm_constant_val	val	constant literal value, when type is constant
string	name	name of field or context_variable
stmm_cv_nodep	cv_info	context variable info, when type is context-var
genptr	user_data	for user’s data

The following are represented in RPN lists:

- **Definition** of data-items/conditions/events/actions/user-defined types/subroutines
- **Consists of** in information-flows
- **Label** of data-flow/bus/signal/transition/control-flow
- **Mini-spec** in activity
- **Static reactions** in state
- Combinational assignments
- **Actual bindings** in matrix of bindings in instance box
- **Callback bindings** in textuals and boxes

Examples of RPNs

Static Reactions

In static reactions of a state, the expression:

```
entering/E;;
entering/F;;
exiting/G
is represented by the list:
op_st_react
-> op_st_react_binary
-> op_st_react_binary
-> exp_trig_action_op
-> op_entering
-> id (E)
-> exp_trig_action_op
-> op_entering
-> id (F)
-> exp_trig_action_op
-> op_exiting
-> id (G)
```

There are some special cases, where an empty `id` is needed in the RPN representation:

Action Expressions

In Definition of action, or anywhere an action expression is allowed, the expression:

```
if (C) then
A;
end if
```

is represented by the list:

```
op_if_c
->id (C)
->id (A)
->id (Empty Id) for the missing "else" action
```

Transition Labels

In label of transition, the expression:

```
EV
```

is represented by the list:

```
op_trig_action
->id (EV)
->id (Empty Id) for the missing action
```

Box Hierarchy in Charts

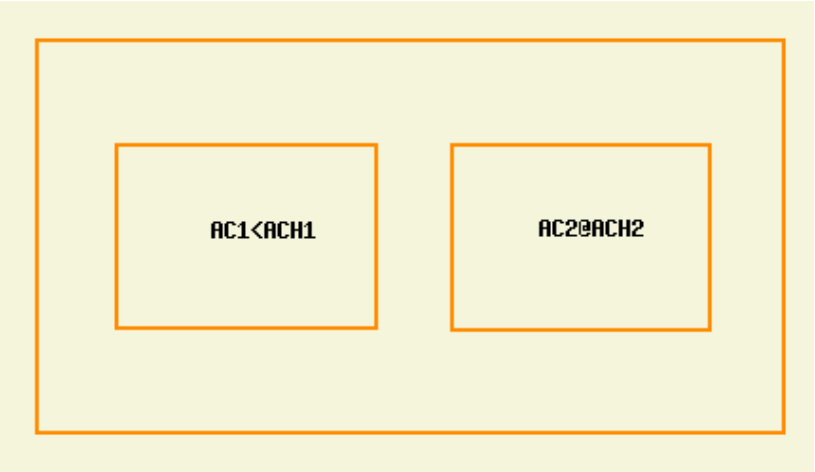
```
Every chart has one Diagram Box, which is the parent of the top level box[es]
of the chart. This Diagram Box is assigned a special type, called
stmm_st_diagram.
```

In the structure of an off-page chart instance-box, the `sons_ list` field includes the ID of the off-page chart's diagram box.

If the instance box is of a generic chart, the `sons_list` is empty, and the `definition_ch` field holds the id of the generic chart.

If a control activity is an off-page chart instance-activity, the `sons_list` field includes the id of the off-page Statechart's diagram box.

For example:



The `sons_list` field of activity AC2 include the id of the diagram-box of chart ACH2.

The `sons_list` of activity AC1 is empty, and its `definition_ch` field holds the id of chart ACH1.

Context Variable Structure

Context variables are not like other Statemate elements; they do not have IDs. They are identified by name and are recognized only in context of the action they belong to.

Context variables do not have a type definition, but Statemate tries to derive their type according to their usage. For instance, if you write:

```
$X:=STR
```

the type for `X` is derived from `STR`.

When there is no type-derivation, the default values for `stmm_cv_node` fields are:

```
stmm_type            stmm_data_item
data_structure       stmm_single
array_limits         NULL
data_type            stmm_dt_integer
dt_limits            NULL
user_defined_type    0
```

Compound Transition Structure - Sources and Targets

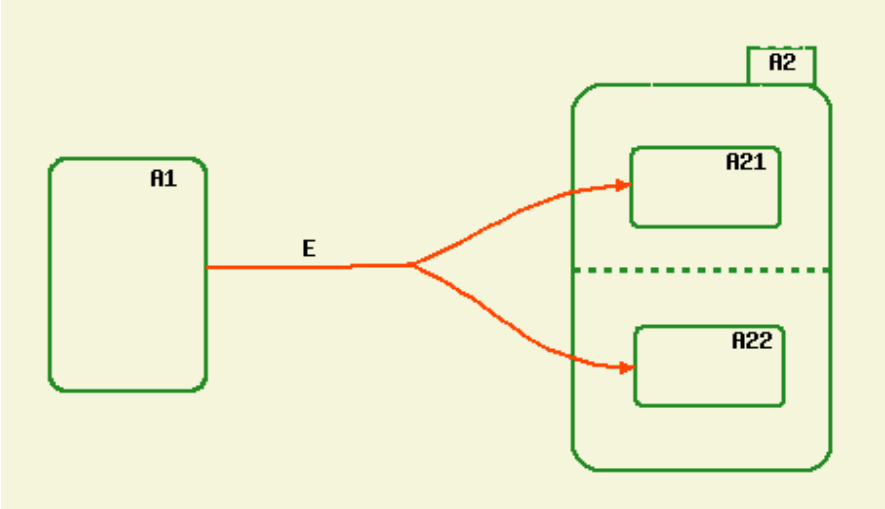
Compound transition structures (`stmm_cm_node`) have source and target fields and `main_source` and `main_target` fields.

Note

In an *and-state*, the sources and targets may be multiple states.

The `main_source` and `main_target` fields are the highest state that includes all the sources and targets, and are lower than the Least Common Ancestor (LCA).

For example:



For the compound arrow labeled E, the targets are A21 and A22, and the main target is A2.

Reset All Elements

When using the operator `reset_all_elements`, the following data is added to the mode:

- The operator `stmm_op_reset_all` appears in the expressions that use `reset_all_elements`.
- An action called `RESET_ALL_ELEMENTS_ACTION` in a dummy scope “0”.
- `RESET_ALL_ELEMENTS_ACTION` or an alternative way must be executed to support `reset_all_elements`.
- The `RESET_ALL_ELEMENTS_ACTION` does not contain:
  - Elements of generics
  - Parameters of subroutine.

When using the operator `reset_element`, note the following:

- The `reset_element` will be translated to an assignment, or a set of assignments, to the element’s default value.
- A basic type will have a single assignment while a complex type, like array or record, will have a set of assignments.
- The assignment(s) expression will be put in the original expression, replacing the `reset_element` operator. Doing that, no `reset_element` operators will be present in the expressions.

When using the operator `reset_all_elements`, now the following:

- An action called `RESET_ALL_ELEMENTS_ACTION` is added in a new (dummy) scope ‘0’. The definition of this Action includes resetting of the elements in the scope.
- In order to support this new operator, you need to:
  - Recognize the operator `reset_all_elements`.
  - Add support to this operator by executing `RESET_ALL_ELEMENTS_ACTION` or by an alternative way.

Note

`RESET_ALL_ELEMENTS_ACTION` does not contain elements of generics (parameters and local data).

Flowcharts as Subroutine Implementation

Flowcharts as subroutine implementation are presented into the anaport like procedural Statecharts. Note the following:

- Each Flowchart creates a separate scope that contains all the boxes and arrows that are defined in the Flowchart.
- The type of the chart in this scope is: `stmm_ch_flowchart`.
- Flowchart boxes are handled with the States APIs.
- Flowchart arrows are handled with Transitions APIs.
- The `enum type: stmm_st_type` (in api_types.h) was extended to include additional types that represents Flowchart boxes types:

```
typedef enum {
stmm_st_or, /* Or State */
stmm_st_and, /* And State */
stmm_st_reference,
stmm_st_diagram,
stmm_st_action_box,
stmm_st_decision_box,
stmm_st_switch_box
} stmm_st_type;
```

CG Builder API Utilities

This section lists the types of CG Builder API utilities, how to use them, and their purpose. The types of utilities are:

- General Utilities
- Single-element Functions
- Query Functions
- List Package Utilities
- RPN Package Utilities

General Utilities

This section lists the following utilities:

- stmm_avoid_rpn_conversions
- stmm_build_reset_all_action()
- stmm_close_session()
- stmm_free()
- stmm_get_scp_version_header()
- stmm_open_session()
- stmm_set_build_all
- stmm_set_build_basic_arrows
- stmm_set_include_enum_init
- stmm_set_regard_in_sim_expressions()

stmm_avoid_rpn_conversions

Usage:

```
stmm_avoid_rpn_conversions();
```

Purpose:

Saves the original rpn of an element, as it appears in the model, in addition to the processed/ modified anaport rpn.

Returned Type:

```
void
```

stmm_build_reset_all_action()

Usage:

```
stmm_build_reset_all_action(boolean);
```

Purpose:

Creates the reset-all related objects and adds them into the anaport database.

The API returns ‘0’ if the creation of the reset-all objects was not done (for example, if there is no usage of the operation ra! in the model) and ‘1’ if the creation of the reset-all objects is successful.

stmm_close_session()

Usage:

```
boolean = stmm_close_session(with_datport)
```

Purpose:

Closes a session.

stmm_free()

Usage:

```
stmm_free(void *pointer);
```

Purpose:

Use to free memory allocated by the anaport, such as strings returned by anaport APIs.

stmm_get_scp_version_header()

Usage:

```
stmm_get_scp_version_header()
```

Purpose:

Returns a string with the version of the database. This string is required to be the first line of any SCP file, to identify which version this SCP file refers to.

Returned Type:

string

stmm_open_session()

Usage:

```
boolean = stmm_open_session (project,workarea,profile,with_datport)
```

Purpose:

Opens a session, invokes the connection to the database, and builds all the Statemate data structures.

Note the following:

- If the profile is NULL, the modules are the separate clusters of the database (not implemented yet).
- `with_datport` is a boolean parameter. When set to true, it initializes Dataport.

stmm_set_build_all

Usage:

```
stmm_set_build_all();
```

Purpose:

Builds data for all elements in the workarea, and not only for the elements in the profile's scope.

Returned Type:

```
void
```

stmm_set_build_basic_arrows

Usage:

```
stmm_set_build_basic_arrows();
```

Purpose:

Builds data for each basic arrow, in addition to data of the complete compound transitions/ flowlines.

Returned Type:

```
void
```

stmm_set_include_enum_init

Usage:

```
stmm_set_include_enum_init();
```

Purpose:

Includes explicit values for enum values definitions, in case the enum definition contain explicit definition.

Returned Type:

```
void
```

stmm_set_regard_in_sim_expressions()

Usage:

```
stmm_set_regard_in_sim_expressions(boolean set_val);
```

Purpose:

When called with input parameter of `true`, the Anaport will include the `in_sim` expressions in its database.

Single-element Functions

This section lists the following utilities:

stmm_get_activity_parameters()

Usage:

```
stmm_id  ac_id;

stmm_ids_list *inputs,*outputs;
```

Usage:

```
stmmGetActivityParameters(ac_id,&inputs,&outputs);
```

Purpose:

Returns the input and output lists of elements to the activity `ac_id`.

Returned Type:

void

stmm_get_chart_id

Usage:

```
chart_id = stmm_get_chart_id(id);
```

Purpose:

Retrieve id of the chart of a specific element (the id of the chart in which this element is defined).

Returned Type:

```
int
```

stmm_get_el()

Usage:

```
element_rec = stmm_get_el(element_id)
```

Purpose:

Retrieves an element record. Element records are defined below.

Returned Type:

```
stmm_el_node *;
```

stmm_get_id_type()

Usage:

```
stmm_type = stmm_get_id_type(id);
```

Purpose:

Returns the Statemate type of the identifier.

Returned Type:

stmm_id_type

stmm_get_module_info()

Usage:

```
module_p = stmm_get_module_info(module_id)
```

Purpose:

Retrieves the structure of a module.

stmm_get_short_id

Usage:

```
short_id = stmm_get_short_id(id);
```

Purpose:

Retrieves the short id of a specific element (the id of the element inside the element's chart).

Returned Type:

```
int
```

stmm_which_chart

Usage:

```
chart_id = stmm_which_chart(element_id)
```

Purpose:

Gets the chart, (chart id), to which the element belongs.

Returned Type:

```
stmm_id
```

Query Functions

This section lists the following functions:

stmm_get_all_cms

Usage:

```
list_of_ids = stmm_get_all_cms;
```

Purpose:

Retrieves compound transitions / flows in the profile.

Returned Type:

```
stmm_ids_list
```

stmm_get_all_cns

Usage:

```
list_of_ids = stmm_get_all_cns();
```

Purpose:

Retrieves all connectors in the profile.

Returned Type:

```
stmm_ids_list
```

stmm_get_bindings

Usage:

```
list_of_stmm_bind_nodes = stmm_get_bindings();
```

Purpose:

Retrieves all panel bindings in the profile,

Returned Type:

`stmm_list`

stmm_get_panels

Usage:

```
list_of_stmm_panel_nodes = stmm_get_panels();
```

Purpose:

Retrieves all panels in the profile.

Returned Type:

`stmm_list`

stmm_get_procedural_modules()

Usage:

```
list_of_ids = stmm_get_procedural_modules()
```

Purpose:

Returns a list of the procedural scopes in the profile (procedural statecharts and flowcharts). `stmm_get_procedural_modules` complements `stmm_get_scope_modules`, which returns all the scopes except for the procedural ones.

stmm_get_scope_modules()

Usage:

```
list_of_ids = stmm_get_scope_modules()
```

Purpose:

Retrieves all modules in the scope/model. Modules are user-defined modules, entities, generics, and stubs.

stmm_get_xx()

Usage:

```
list_of_ids = stmm_get_xx(module_id)
```

`xx` stands for the element type. Refer to the **Element Type Abbreviations** table for a list of the element types and their abbreviations.

Purpose:

Retrieves all elements of a certain type in a certain module.

List Package Utilities

The Anaport API list package supports 8 bytes of data and 4 bytes of data. All list APIs that refer to a list value should use the following APIs:

- API for handling stmm_id lists (8 bytes data)
- API for handling pointers (4 bytes data)

The following table shows the list APIs that supports 8 bytes of data (API for `stmm_id Data`) and the list APIs that supports 4 byte data (API for pointer Data).

API for stmm_id Data	API for Pointer Data
`stmm_list_add_id_element`	`stmm_list_add_ptr_element`
`stmm_list_find_value_id`	`stmm_list_find_value_ptr`
`stmm_list_get_elm_value_id`	`stmm_list_get_elm_value_ptr`

This section lists the following utilities:

- **stmm_list_add_id_element()**
- **stmm_list_add_ptr_element()**
- **stmm_list_copy()**
- **stmm_list_create()**
- **stmm_list_delete_element()**
- **stmm_list_destroy()**
- **stmm_list_find_value_id()**
- **stmm_list_find_value_ptr()**
- **stmm_list_first_element()**
- **stmm_list_get_elm_value_id()**
- **stmm_list_get_elm_value_ptr()**
- **stmm_list_get_ith_element()**
- **stmm_list_last_element()**
- **stmm_list_length()**
- **stmm_list_next_element()**
- **stmm_list_previous_element()**
- **stmm_list_purge()**

stmm_list_add_id_element()

Usage:

```
success = stmm_list_add_id_element(list,val);
```

Purpose:

Adds an element containing an ID (val) to the list.

Returned Type:

```
boolean
```

stmm_list_add_ptr_element()

Usage:

```
success = stmm_list_add_ptr_element(list,val);
```

Purpose:

Adds an element containing a pointer (val) to the list.

Returned Type:

```
boolean
```

stmm_list_copy()

Usage:

```
list2 = stmm_list_copy(list1)
```

Purpose:

Copies a list. The values in the new list must have the same structure pointers.

Returned Type:

```
stmm_list
```

stmm_list_create()

Usage:

list = stmm_list_create();

Purpose:

Creates a new `stmm_list`.

Returned Type:

`stmm_list`

stmm_list_delete_element()

Usage:

```
stmm_list_delete_element(list,list_elm);
```

Purpose:

Deletes the element `list_elm` from the list.

Returned Type:

void

stmm_list_destroy()

Usage:

```
stmm_list_destroy(list);
```

Purpose:

Frees the memory of a list without freeing the structures in the list.

Returned Type:

void

stmm_list_find_value_id()

Usage:

```
list_elm = stmm_list_find_value_id(list,val);
```

Purpose:

Sees if there is an element in the list whose value is `id`, and if so, gets it.

Returned Type:

```
stmm_list_elm
```

stmm_list_find_value_ptr()

Usage:

```
list_elm = stmm_list_find_value_ptr(list,val);
```

Purpose:

Sees if there is an element in the list whose value is `ptr`, and if so, gets it.

Returned Type:

```
stmm_list_elm
```

stmm_list_first_element()

Usage:

```
list_elm = stmm_list_first_element(list);
```

Purpose:

Gets the first element in a list.

Returned Type:

```
stmm_list_elm
```

stmm_list_get_elm_value_id()

Usage:

```
val = (stmm_id)stmm_list_get_value_id (list_elm);
```

Purpose:

To get the `stmm_id` value inside `list_elm`.

Returned Type:

```
stmm_id
```

stmm_list_get_elm_value_ptr()

Usage:

```
val = (gen_ptr)stmm_list_get_value_ptr (list_elm);
```

Purpose:

To get the `ptr` value inside `list_elm`.

Returned Type:

```
gen_ptr
```

stmm_list_get_ith_element()

Usage:

```
list_elm = stmm_list_get_ith_element (stmm_list,i);
```

Purpose:

Gets the i'th element in the list. The first element’s index is 0.

Returned Type:

```
stmm_list_elm
```

stmm_list_last_element()

Usage:

```
list_elm = stmm_list_last_element(list);
```

Purpose:

Gets the last element in a list.

Returned Type:

```
stmm_list_elm
```

stmm_list_length()

Usage:

```
num = stmm_list_length(list);
```

Purpose:

Gets the number of elements in a list.

Returned Type:

integer

stmm_list_next_element()

Usage:

```
list_elm = stmm_list_next_element(list,list_elm);
```

Purpose:

Gets the next element in a list.

Returned Type:

```
stmm_list_elm
```

stmm_list_previous_element()

Usage:

```
list_elm = stmm_list_previous_element (list,list_elm);
```

Purpose:

Gets the previous element in a list.

Returned Type:

```
stmm_list_elm
```

stmm_list_purge()

Usage:

```
stmm_list_purge(list);
```

Purpose:

Frees the memory of a list including the structures in the list.

Returned Type:

void

RPN Package Utilities

This section lists the following utilities:

get_ith_operand()

Usage:

```
operand_elm = get_ith_operand (rpn_list,curr_list_elm,i);
```

Purpose:

Gets the ith operand of the current operator.

Returned Type:

```
stmm_list_elm
```

get_number_of_operands()

Usage:

```
num = get_number_of_operands (operator_type);
```

Purpose:

Gets the number of operands for a specific operator.

Returned Type:

integer

stmm_get_cm_exp_used_affected()

Usage:

```
stmm_get_cm_exp_used_affected(stmm_cm_exp *cm_exp, stmm_ids_list *used,
stmm_ids_list *affected, stmm_ids_list *used_affected)
```

Purpose:

Calculates used and affected elements of compound transitions.

Returned Type:

- `cm_exp`—Input - The field: expression in the structure: stmm_cm_node.
- `used`—Output - lists of id's used.
- `affected`—Output - lists of id's affected.
- `used_affected`—Output - lists of id's used and affected.

stmm_get_used_affected()

Usage:

```
stmm_list expressions; -- Input parameter, list of stmm_rpn_lists

stmm_list *used,*affected,*locals; -- Output parameters, lists of stmm_ids.
```

Usage:

```
stmm_get_used_affected(expressions,&used,&affected, &locals);
```

Purpose:

Returns the lists of elements that are used, affected, and used-affected in all the expressions given in *expressions*.

Note

Elements that are both used and affected appear **only** in the locals list and not in all three lists.

Returned Type:

void

Example:

In the following expression:

```
A(I):=J+K;I:=I+1;
```

- The used elements list is J,K.
- The affected elements list is A.
- The locals elements list is I.

stmm_rpn_skip_and_copy_exp()

Usage:

```
next_exp := stmm_rpn_skip_and_copy_exp (rpn_list,&list_elm);
```

Purpose:

Copies the expression beginning with `list_elm` to a new `stmm_rpn_list`, and to advance `list_elm` to the next expression in the `rpn_list`.

The new `stmm_rpn_list` is the returned value.

Returned Type:

```
stmm_rpn_list
```

stmm_rpn_skip_exp()

Usage:

```
next_list_elm = stmm_rpn_skip_exp (rpn_list,curr_list_elm)
```

Purpose:

Gets the next expression in the `rpn_list`.

Returned Type:

```
stmm_list_elm
```

Example:

The label **`EV[CO]/X:=14`** is represented by the following `rpn_list`:

```
'/' '[]' 'EV' 'CO' ':=' 'X' '14'
```

To skip to the action part:

- first_element = stmm_list_first_element (rpn_list);
  (`first_element` points to the `'/'` operator.)
- next_element = stmm_list_next_element rpn_list,first_element);
  (`next_element` points to the `'[]'` operator.)
- action_element = stmm_rpn_skip_exp (rpn_list,next_element);
  (`action_element` points to the `':='` operator.)

stmm_rpn_to_string()

Usage:

```
str = stmm_rpn_to_string(rpn_list)
```

Purpose:

Gets the string expression in the Statemate language represented in `rpn_list`.

Returned Type:

string

RPN and List Function Usage Example

The `get_ith_operand` writes:

a. stmm_list_elm

b. get_ith_operand(rpn_list,curr_list_elm,i)

c. stmm_rpn_list rpn_list;

d. stmm_list_elm curr_list_elm;

e. int i;

f. {

g. stmm_list_elm ith_operand;

h. int j;

i. stmm_rpn_nodep rpn_nodep;

j.

k. rpn_nodep = (stmm_rpn_nodep)stmm_list_get_elm_value (curr_list_elm);

l. if(rpn_nodep->rpn_node_type != stmm_operator)

m. return ZNIL;

n. if(get_number_of_operands(rpn_nodep->operator_type) < i)

o. return ZNIL;

p. ith_operand =stmm_list_next_element

q. ((stmm_list)rpn_list,curr_list_elm);

r. for (j=1;j<i;j++)

s. ith_operand = stmm_rpn_skip_exp

t. (rpn_list,ith_operand);

u. return ith_operand;

}

Sample Program

This section presents a detailed C program that shows how to use the CG Builder API to perform a number of function calls. The program is designed to extract information about activities from the specification database. The information extracted is both textual (for example, activity name, synonym, short description, and so forth.) and graphical (the Cartesian coordinates of the activity's box).

Sample program `example.c` is delivered as part of the Anaport package.

```
#include  <stdio.h>

#include  <duinc/uad_def.h>
#include  <aainc/api_types.h>


/* All these conversion-tables are just for getting the type-name as string -
in order to print it. */


typedef struct stmm_type_conversion {

  stmm_id_type   stmm_type;

  string         stmm_type_name;

} stmm_type_conversion;


stmm_type_conversion stmm_type_conv_table[] = {

  {stmm_chart,"stmm_chart"},

  {stmm_state,"stmm_state"},

  {stmm_connector,"stmm_connector"},

  {stmm_compound_ar,"stmm_compound_ar"},

  {stmm_data_item,"stmm_data_item"},

  {stmm_condition,"stmm_condition"},

  {stmm_event,"stmm_event"},

  {stmm_info_flow,"stmm_info_flow"},

  {stmm_action,"stmm_action"},

  {stmm_activity,"stmm_activity"},

  {stmm_module,"stmm_module"},
```

```
  {stmm_function,"stmm_subroutine"},
  {stmm_user_type,"stmm_user_type"},
  {stmm_field,"stmm_field"},
  {stmm_enumerated_value,"stmm_enumerated_value"},
  {stmm_subroutine_data,"stmm_subroutine data"},
  {stmm_error,"stmm_error"}
};

typedef struct inst_type_conversion {
  stmm_inst_type inst_type;
  string         inst_type_name;
} inst_type_conversion;

inst_type_conversion inst_type_conv_table[] = {
  {stmm_inst_offpage,"stmm_inst_offpage"},
  {stmm_inst_generic,"stmm_inst_generic"},
  {stmm_inst_missing,"stmm_inst_missing"}
};

typedef struct duration_conversion {
  stmm_ac_duration duration;
  string           duration_name;
} duration_conversion;

duration_conversion duration_conv_table[] = {
  {stmm_ac_within,"within"},
  {stmm_ac_throughout,"throughout"}
};
typedef struct ac_type_conversion {
  stmm_ac_type ac_type;
  string       ac_type_name;
} ac_type_conversion;
```

```
ac_type_conversion ac_type_conv_table[] = {
  {stmm_ac_internal,"stmm_ac_internal"},
  {stmm_ac_control,"stmm_ac_control"},
  {stmm_ac_reference,"stmm_ac_reference"},
  {stmm_ac_data_store,"stmm_ac_data_store"},
  {stmm_ac_external,"stmm_ac_external"},
  {stmm_ac_environment,"stmm_ac_environment"},
  {stmm_ac_diagram,"stmm_ac_diagram"}
};

typedef struct term_type_conversion {
  stmm_ac_term term_type;
  string       term_type_name;
} term_type_conversion;

term_type_conversion term_type_conv_table[] = {
  {stmm_ac_missing,"stmm_ac_missing"},
  {stmm_ac_self_term,"stmm_ac_self_term"},
  {stmm_ac_controlled_term,"stmm_ac_controlled_term"},
  {stmm_ac_procedure_like,"stmm_ac_procedure_like"}
};

typedef struct chart_conversion {
  stmm_ch_type  ch_type;
  string        ch_type_name;
}chart_conversion;

chart_conversion ch_type_conv_table[] = {
  {stmm_ch_statechart,"stmm_ch_statechart"},
  {stmm_ch_activitychart,"stmm_ch_activitychart"},
  {stmm_ch_modulechart,"stmm_ch_modulechart"},
  {stmm_ch_blockdiagram,"stmm_ch_blockdiagram"},
  {stmm_ch_gds,"stmm_ch_gds"}
};
```

```
typedef struct ref_by_conversion {
  stmm_refered_by    refered_by;
  string             ref_by_name;
} ref_by_conversion;

ref_by_conversion ref_by_conv_table[] = {
  {ref_by_name,"ref_by_name"},
  {ref_by_syn,"ref_by_syn"},
  {ref_by_literal,"ref_by_literal"},
  {ref_by_missing,"ref_by_missing"}
};

typedef struct cm_type_conversion {
  stmm_cm_type cm_type;
  string       cm_type_name;
} cm_type_conversion;

cm_type_conversion cm_type_conv_table[] = {
  {stmm_cm_transition,"stmm_cm_transition"},
  {stmm_cm_a_flow,"stmm_cm_a_flow"},
  {stmm_cm_b_flow,"stmm_cm_b_flow"},
  {stmm_cm_m_flow,"stmm_cm_m_flow"}
};

typedef struct tx_type_conversion {
  stmm_tx_type      tx_type;
  string            tx_type_name;
}tx_type_conversion;

tx_type_conversion tx_type_conv_table[] = {
  {stmm_reference,"stmm_reference"},
  {stmm_variable,"stmm_variable"},
  {stmm_compound,"stmm_compound"},
  {stmm_constant,"stmm_constant"},
  {stmm_alias,"stmm_alias"},
  {stmm_defined,"stmm_defined"},
```

```
  {stmm_local_var,"stmm_local_var"},
  {stmm_parameter,"stmm_parameter"}
};

typedef struct sb_type_conversion {
  stmm_sb_type      sb_type;
  string            sb_type_name;
}sb_type_conversion;

sb_type_conversion sb_type_conv_table[] = {
  {stmm_procedure,"stmm_procedure"},
  {stmm_function,"stmm_function"},
  {stmm_task,"stmm_task"},
  {stmm_stm_func,"stmm_stm_func"},
  {stmm_undef,"stmm_undef"}
};

typedef struct state_conversion {
  stmm_st_type  state_type;
  string        state_type_name;
}state_conversion;

state_conversion state_type_conv_table[] = {
  {stmm_st_or,"stmm_st_or"},
  {stmm_st_and,"stmm_st_and"},
  {stmm_st_reference,"stmm_st_reference"},
  {stmm_st_diagram,"stmm_st_diagram"}
};

typedef struct gp_type_conversion {
  stmm_gp_type  gp_type;
  string        type_name;
}gp_type_conversion;
```

```
gp_type_conversion gp_type_conv_tab[] = {
  {stmm_gp_data_item,"stmm_gp_data_item"},
  {stmm_gp_event,"stmm_gp_event"},
  {stmm_gp_condition,"stmm_gp_condition"},
  {stmm_gp_textual,"stmm_gp_textual"},
  {stmm_gp_activity,"stmm_gp_activity"}
};

typedef struct gp_mode_conversion {
  stmm_gp_mode gp_mode;
  string       mode_name;
}gp_mode_conversion;

gp_mode_conversion gp_mode_conv_tab[] = {
  {stmm_gp_mode_input,"stmm_gp_mode_input"},
  {stmm_gp_mode_output,"stmm_gp_mode_output"},
  {stmm_gp_mode_inout,"stmm_gp_mode_inout"},
  {stmm_gp_mode_constant,"stmm_gp_mode_constant"}
};

typedef struct struct_conversion {
  stmm_structure data_struct;
  string         struct_name;
}struct_conversion;

struct_conversion struct_conv_table[] = {
  {stmm_missing,"stmm_missing"},
  {stmm_single,"stmm_single"},
  {stmm_array,"stmm_array"},
  {stmm_queue,"stmm_queue"}
};

typedef struct data_type_conversion {
  stmm_data_type    data_type;
  string            data_type_name;
}data_type_conversion;
```

```
data_type_conversion data_type_conv_table[] = {
  {stmm_dt_missing,"stmm_dt_missing"},
  {stmm_dt_integer,"stmm_dt_integer"},
  {stmm_dt_real,"stmm_dt_real"},
  {stmm_dt_string,"stmm_dt_string"},
  {stmm_dt_bit,"stmm_dt_bit"},
  {stmm_dt_bit_array,"stmm_dt_bit_array"},
  {stmm_dt_record,"stmm_dt_record"},
  {stmm_dt_union,"stmm_dt_union"},
  {stmm_dt_user_def,"stmm_dt_user_def"},
  {stmm_dt_enum_type,"stmm_dt_enum_type"},
  {stmm_dt_condition,"stmm_dt_condition"},
  {stmm_dt_event,"stmm_dt_event"}
};

string
id_to_name(el_id)
     stmm_id el_id;
{
  stmm_el_nodep stmm_el = NULL;
  string str = (string)malloc(100);

  if (el_id == 0) {
    sprintf(str,"NULL ID");
    return(str);
  }
  /* extract element structure */
  stmm_el = stmm_get_el(el_id);
  if (!stmm_el) {
    sprintf(str,"Nonexistent element, id: %d",el_id);
    return(str);
  }
```

```
  /* check if name is not NULL */
  if (stmm_el->name && strcmp(stmm_el->name,"")) {
    strcpy(str,stmm_el->name);
    return(str);
  }
  /* check if code_gen_name is no NULL */
  else if(stmm_el->code_gen_name &&
     strcmp(stmm_el->code_gen_name,"")) {
    strcpy(str,stmm_el->code_gen_name);
    return(str);
  }
  sprintf(str,"No Name, id: %d",el_id);
  return(str);
}

void
ids_to_names(fp, ids)
     FILE *fp;
     stmm_list ids;
{
  stmm_id id;
  stmm_list_elm p;
  string name;

  stmm_for_every_element_in_list(ids,p) {

    /* get element value - id    */
    id = (int)stmm_list_get_elm_value(p);
    name = id_to_name(id);
    fprintf(fp,"    %s\n",name);
    free(name);
  }
}
```

```
void
print_rpn(fp,rpn_lst)
     FILE *fp;
     stmm_rpn_list rpn_lst;
{
  stmm_ids_list expressions,used,affected,used_affected;
  stmm_list_elm p;
  stmm_rpn_nodep nodep;
  string name;

  /* print rpn list */
  fprintf(fp,stmm_rpn_to_string(rpn_lst));
  fprintf(fp,"\n");
  stmm_for_every_element_in_list(rpn_lst,p) {
    nodep = stmm_list_get_elm_value(p);
    if (nodep->rpn_node_type == stmm_identifier) {
      if (!(stmm_get_el(nodep->id))) {
   name = id_to_name(nodep->id);
   fprintf(fp,"    %s\n",name);
   free(name);
      }
    }
  }
  used = affected = used_affected = NULL;
  expressions = stmm_list_create();
  stmm_list_add_element(expressions,rpn_lst);
  stmm_get_used_affected(expressions,&used,&affected,&used_affected);
  if (used) {
    fprintf(fp,"Used elements:\n");
    ids_to_names(fp, used, 0);
  }
  if (affected) {
    fprintf(fp,"Affected elements:\n");
    ids_to_names(fp, affected, 0);
  }
```

```
  if (used_affected) {
    fprintf(fp,"UsedAffected elements:\n");
    ids_to_names(fp, used_affected, 0);
  }
}

void
print_ids_list(fp,ids_list)
     FILE *fp;
     stmm_ids_list ids_list;
{
  stmm_list_elm p;
  stmm_id id;

  stmm_for_every_element_in_list(ids_list,p) {

    /* get element value - id    */
    id = (int)stmm_list_get_elm_value(p);

    fprintf(fp,"   %d \n",id);
  }
}

void
print_one_limit(fp,index)
     FILE *fp;
     one_limit_nodep index;
{
  string name;

  if(index->refered_by == ref_by_missing)
    fprintf(fp,"Limit is missing\n");

  /* if index is refered by literal - the literal is in 'number' */
  else if(index->refered_by == ref_by_literal)
    fprintf(fp,"Refered by Literal: %d\n",index->number);
```

```
  /* if index is refered by name or synonym -
     'id' holds the refered element's id       */
  else if((index->refered_by == ref_by_name)||
      (index->refered_by == ref_by_syn)){
    name = id_to_name(index->id);
    fprintf(fp,"Refered by name/syn %s\n",name);
    free(name);
  }
}

void
print_array_limits(fp,array_limits)
     FILE *fp;
     stmm_array_limits *array_limits;
{
  fprintf(fp,"Array Limits: \n");

  /* print left index */
  fprintf(fp,"     lindex: ");
  print_one_limit(fp,array_limits->lindex);

  /* print right index */
  fprintf(fp,"     rindex: ");
  print_one_limit(fp,array_limits->rindex);
}

void
print_integer_limits(fp,dt_limits)
     FILE *fp;
     stmm_dt_limits *dt_limits;
{
  fprintf(fp,"Integer Limits: \n");
```

```
  /* #Bits */
  fprintf(fp,"        Bits_num:
%d\n",dt_limits->integer_limits->bits_num);

  /* Min value */
  fprintf(fp,"        Min_val: ");
  print_one_limit(fp,dt_limits->integer_limits->min_val);

  /* Max value */
  fprintf(fp,"        Max_val: ");
  print_one_limit(fp,dt_limits->integer_limits->max_val);
}

void
print_names_of_ids(fp,ids,title1,title2)
     FILE *fp;
     stmm_ids_list ids;
     string title1,title2;
{
  stmm_list_elm p;
  stmm_id id;
  string name;

  fprintf(fp,"%title1: \n");
  if(ids){

    /* go through the list of id pointers */
    stmm_for_every_element_in_list(ids,p){

      /* get element value -id */
      id = (int)stmm_list_get_elm_value(p);
      name = id_to_name(id);
      fprintf(fp,"     %s : %s\n",title2,name);
      free(name);
    }
  }
```

```
  else
    fprintf(fp,"  No %s\n",title1);
}

void
print_string_limits(fp,dt_limits)
     FILE *fp;
     stmm_dt_limits  *dt_limits;
{
  fprintf(fp,"String Limits: \n");

  /* stmm_string_limits is typedef of one_limit_node */
  print_one_limit(fp,dt_limits->string_limits);
}

void
print_attributes(fp,attributes)
     FILE *fp;
     stmm_list attributes;
{
  stmm_attribute_p attr_p;
  stmm_list_elm p;

  if(attributes)

    /* if attributes list is not empty */
    if(stmm_list_length(attributes)){

      fprintf(fp,"Attributes:\n");

      /* go through the list of attribute structure pointers */
      stmm_for_every_element_in_list(attributes,p) {

   /* extract the attribute structure */
   attr_p = (stmm_attribute_p)stmm_list_get_elm_value(p);
```

```
    fprintf(fp,"   Attribute Name: %s   Value: %s\n",
           attr_p->name,attr_p->value);
      }
     }
}

void
print_instance_params(fp,instance_params)
     FILE *fp;
     stmm_list instance_params;
{
  stmm_inst_prm *inst_prm;
  stmm_list_elm p;

  if(instance_params)

    /* if list of instance_params is not empty */
    if(stmm_list_length(instance_params)){

      fprintf(fp,"Instance Parameters: ");

      /* go through the list of stmm_inst_prm pointers */
      stmm_for_every_element_in_list(instance_params,p){

   /* extract the stmm_inst_prm structure */
   inst_prm = stmm_list_get_elm_value(p);

   /* the actual_prm is an stmm_rpn_list */
   print_rpn(fp,inst_prm->actual_prm);
      }
      fprintf(fp,"\n\n");
     }
}
```

```
void
print_common(fp,el_rec)
     FILE *fp;
     stmm_el_nodep el_rec;
{
 /* This function prints the fields of stmm_el_node - common to all elements
*/


  /* name */
  if(el_rec->name)
    fprintf(fp,"Name: %s\n",el_rec->name);


  /* code_gen_name */
  if(el_rec->code_gen_name)
    fprintf(fp,"Codegen-name: %s\n",el_rec->code_gen_name);


  /* unique_name */
  if(el_rec->unique_name)
    fprintf(fp,"Unique_name: %s\n",el_rec->unique_name);


  /* id */
  fprintf(fp,"Id: %d\n",el_rec->id);


  /* stmm_type */
  fprintf(fp,"Stmm_type: %s\n",
       stmm_type_conv_table[el_rec->stmm_type].stmm_type_name);


  /* short_des */
  if(el_rec->short_des)
    if(strcmp("",el_rec->short_des))
      fprintf(fp,"Short Description: %s\n",el_rec->short_des);


  /* synonym */
  if(el_rec->synonym)
    if(strcmp("",el_rec->synonym))
      fprintf(fp,"Synonym: %s\n",el_rec->synonym);
```

```
  /* attributes */
  print_attributes(fp,el_rec->attributes);

  /* callback bindings */
  if (el_rec->callback_binding){
    fprintf(fp,"Callback Bindings: ");
    print_rpn(fp,el_rec->callback_binding);
  }
}

void
print_textual(fp,ids)
     FILE *fp;
     genptr ids;
{
  stmm_id id;
  stmm_tx_nodep txt_rec;
  stmm_list_elm p;

 /* This function prints stmm_tx_node fields - common to all textual elements
*/

  /* go through the list of id pointers */
  stmm_for_every_element_in_list(ids,p) {

    /* extract the element value - id */
    id = (int)stmm_list_get_elm_value(p);

    /* retrieve the element's structure */
    txt_rec = (stmm_tx_nodep)stmm_get_el(id);

    /* print common fields (stmm_el_node) */
    print_common(fp,(stmm_el_nodep)txt_rec);
```

```
/* type */
fprintf(fp,"Stmm_tx_type: %s\n",
    tx_type_conv_table[txt_rec->type].tx_type_name);

/* data_structure */
if(txt_rec->data_structure)
  fprintf(fp,"Structure: %s\n",
      struct_conv_table[txt_rec->data_structure].struct_name);

/* array_limits */
if(txt_rec->data_structure == stmm_array)
  print_array_limits(fp,txt_rec->array_limits);

/* data_type */
if(txt_rec->data_type)
  fprintf(fp,"Data Type: %s\n",
      data_type_conv_table[txt_rec->data_type].data_type_name);

/* dt_limits */
if(txt_rec->data_type == stmm_dt_bit_array)
  print_array_limits(fp,txt_rec->dt_limits->bit_array_limits);
if(txt_rec->data_type == stmm_dt_integer)
  print_integer_limits(fp,txt_rec->dt_limits);
if(txt_rec->data_type == stmm_dt_string)
  print_string_limits(fp,txt_rec->dt_limits);

/* ut_ref_by */
if(txt_rec->data_type == stmm_dt_user_def){
  if(txt_rec->ut_ref_by == ref_by_name)
fprintf(fp,"UDT Refered by Name\n");
  if(txt_rec->ut_ref_by == ref_by_syn)
fprintf(fp,"UDT Refered by Synonym\n");
```

```
        /* user_defined_type */
        fprintf(fp,"User-Defined-Type :
                   %d\n",txt_rec->user_defined_type);
      }

      /* fields */
      if((txt_rec->data_type == stmm_dt_record) ||
         (txt_rec->data_type == stmm_dt_union))
        print_names_of_ids(fp,txt_rec->fields,"Fields","field");

      /* definition */
      if(txt_rec->definition){
        fprintf(fp,"Definition: ");
        print_rpn(fp,txt_rec->definition);
      }
      fprintf(fp,"\n");
    }
}

void
print_compound(fp,ids)
     FILE *fp;
     genptr ids;
{
  stmm_id id;
  stmm_cm_nodep cm_rec;
  stmm_list_elm p,p1;
  stmm_cm_exp *labels;
  string name;

  /* Print fields of compound - stmm_cm_node */

  /* go through list of compound-id's pointers */
  stmm_for_every_element_in_list(ids,p) {
```

```
/* extract the element value - id */
id = (int)stmm_list_get_elm_value(p);

/* retrieve the compound's structure */
cm_rec = (stmm_cm_nodep)stmm_get_el(id);

/* print common fields (stmm_el_node) */
print_common(fp,(stmm_el_nodep)cm_rec);

/* type */
fprintf(fp,"Type: %s\n",
    cm_type_conv_table[cm_rec->type].cm_type_name);

/* triggers & actions */
if(cm_rec->type == stmm_cm_transition){
  labels = cm_rec->expression;
  if(labels->transition_labels.triggers){
fprintf(fp,"Triggers:");
print_rpn(fp,labels->transition_labels.triggers);
  }
  if(labels->transition_labels.actions) {
fprintf(fp,"Actions:");
stmm_for_every_element_in_list(labels->transition_labels.actions,p1) {
  print_rpn(fp,stmm_list_get_elm_value(p1));
}
  }
 }

/* lca */
fprintf(fp,"LCA id : %d\n",cm_rec->lca);

/* sources */
fprintf(fp,"Sources: \n");
ids_to_names(fp,cm_rec->sources);
```

```
    /* targets */
    fprintf(fp,"Targets: \n");
    ids_to_names(fp,cm_rec->targets);

    /* main_source */
    name = id_to_name(cm_rec->main_source);
    fprintf(fp,"Main Source : %s \n",name);
    free(name);

    /* main_target */
    name = id_to_name(cm_rec->main_target);
    fprintf(fp,"Main Target : %s\n",name);
    free(name);

    fprintf(fp,"\n");
  }
}

void
print_activity(fp,ids)
     FILE *fp;
     genptr ids;
{
  stmm_id id;
  stmm_ac_nodep ac_rec;
  stmm_list_elm p;
  string name;
```

```
/* Print fields of activity - stmm_ac_node */

/* go through list of activity-id's pointers */
stmm_for_every_element_in_list(ids,p) {

  /* extract the element value - id */
  id = (int)stmm_list_get_elm_value(p);

  fprintf(fp,"\n");

  /* retrieve the activity's structure */
  ac_rec = (stmm_ac_nodep)stmm_get_el(id);

  /* print common fields (stmm_el_node) */
  print_common(fp,(stmm_el_nodep)ac_rec);

  /* type */
  fprintf(fp,"Type: %s\n",
     ac_type_conv_table[ac_rec->type].ac_type_name);

  /* mini_spec */
  if(ac_rec->mini_spec){
    fprintf(fp,"Mini-Spec: ");
    print_rpn(fp,ac_rec->mini_spec);
  }

  /* combin_assigns */
  if(ac_rec->combin_assigns){
    fprintf(fp,"Combin_Assigns: ");
    print_rpn(fp,ac_rec->combin_assigns);
  }

  /* instance_type */
  fprintf(fp,"Instance type: %s\n",
     inst_type_conv_table[ac_rec->instance_type].inst_type_name);
```

```
/* termination_type */
fprintf(fp,"Termination type: %s\n",
   term_type_conv_table[ac_rec->termination_type].term_type_name);

/* instance_params */
print_instance_params(fp,ac_rec->instance_params);

/* parent */
name = id_to_name(ac_rec->parent);
fprintf(fp,"Parent : %s\n",name);
free(name);

/* definition_ch */
if(ac_rec->definition_ch) {
  name = id_to_name(ac_rec->definition_ch);
  fprintf(fp,"Definition Chart : %s\n",name);
  free(name);
}

/* entering_fl */
if(stmm_list_length(ac_rec->entering_fl)){
  fprintf(fp,"Flow-lines entering activity:\n");
  print_ids_list(fp,ac_rec->entering_fl);
}

/* exiting_fl */
if(stmm_list_length(ac_rec->exiting_fl)){
  fprintf(fp,"Flow-lines exiting activity:\n");
  print_ids_list(fp,ac_rec->exiting_fl);
}

/* sons_list */
if (ac_rec->sons_list){
  fprintf(fp,"Sons List:\n");
  ids_to_names(fp,ac_rec->sons_list);
}
```

```
    /* user code bindings */
    if (ac_rec->user_code_binding){
      fprintf(fp,"User Code Binding: ");
      print_rpn(fp,ac_rec->user_code_binding);
    }
  }
  fprintf(fp,"\n");
}

void
print_state(fp,ids)
     FILE *fp;
     genptr ids;
{
  stmm_id id;
  stmm_st_nodep st_rec;
  stmm_list_elm p,p1;
  stmm_ac_in_s *act;
  string name;

  /* Print fields of state - stmm_st_node */

  /* go through list of state's-id's pointers */
  stmm_for_every_element_in_list(ids,p) {

    /* extract the  element's value - id */
    id = (int)stmm_list_get_elm_value(p);

    /* retrieve the state's structure */
    st_rec = (stmm_st_nodep)stmm_get_el(id);

    /* print common fields (stmm_el_node) */
    print_common(fp,(stmm_el_nodep)st_rec);
```

```
/* type */
fprintf(fp,"State Type: %s\n",
    state_type_conv_table[st_rec->type].state_type_name);

/* static_reactions */
if(st_rec->static_reactions){
  fprintf(fp,"Static Reactions: ");
  print_rpn(fp,st_rec->static_reactions);
}

/* acs_in_state */
if(st_rec->acs_in_state){
  fprintf(fp,"Activities in state:\n");
  stmm_for_every_element_in_list(st_rec->acs_in_state,p1){
act = stmm_list_get_elm_value(p1);
name = id_to_name(act->activity_id);
fprintf(fp,"    %s",name);
free(name);
fprintf(fp,"
%s\n",duration_conv_table[act->duration].duration_name);
  }
}

/* combin_assigns */
if(st_rec->combin_assigns){
  fprintf(fp,"Combin_Assigns: ");
  print_rpn(fp,st_rec->combin_assigns);
}

/* instance_type */
fprintf(fp,"Instance type: %s\n",
    inst_type_conv_table[st_rec->instance_type].inst_type_name);

/* instance_params */
print_instance_params(fp,st_rec->instance_params);
```

```
    /* parent */
    name = id_to_name(st_rec->parent);
    fprintf(fp,"Parent State : %s\n",name);
    free(name);

    /* definition_ch */
    if(st_rec->instance_type != stmm_inst_missing) {
      name = id_to_name(st_rec->definition_ch);
      fprintf(fp,"Definition Chart : %s\n",name);
      free(name);
    }

    /* exiting_tr */
    if(stmm_list_length(st_rec->exiting_tr)){
      fprintf(fp,"Transitions exiting state:\n");
      print_ids_list(fp,st_rec->exiting_tr);
    }

    /* entering_tr */
    if(stmm_list_length(st_rec->entering_tr)){
      fprintf(fp,"Transitions entering state:\n");
      print_ids_list(fp,st_rec->entering_tr);
    }


    /* sons_list */
    if(st_rec->sons_list){
      fprintf(fp,"Sons List:\n");
      ids_to_names(fp,st_rec->sons_list);
    }

    fprintf(fp,"\n");
  }
  fprintf(fp,"\n");
}
```

```
void
print_subroutines(fp,ids)
     FILE *fp;
     genptr ids;
{
  stmm_id id;
  stmm_sb_nodep sb_rec;
  stmm_list_elm p,p1;
  string name;

  /* Print fields of function - stmm_sb_node */

  /* go through list of subroutine-id's pointers */
  stmm_for_every_element_in_list(ids,p) {

    /* extract the element value - id */
    id = (int)stmm_list_get_elm_value(p);

    /* retrieve the function's structure */
    sb_rec = (stmm_sb_nodep)stmm_get_el(id);

    /* print common fields (stmm_el_node) */
    print_common(fp,(stmm_el_nodep)sb_rec);

    /* type */
    fprintf(fp,"Stmm_sb_type: %s\n",
       sb_type_conv_table[sb_rec->type].sb_type_name);

    /* return_type */
    if(sb_rec->return_type)
      fprintf(fp,"Return Type: %s\n",
         data_type_conv_table[sb_rec->return_type].data_type_name);
```

```
 /* ut_ref_by */
 if(sb_rec->return_type == stmm_dt_user_def){
   if(sb_rec->ut_ref_by == ref_by_name)
fprintf(fp,"UDT Refered by Name\n");
   if(sb_rec->ut_ref_by == ref_by_syn)
fprintf(fp,"UDT Refered by Synonym\n");

   /* user_defined_type */
   fprintf(fp,"User-Defined-Type :
       %d\n",sb_rec->user_defined_type);
 }


 /* parameters */
 if (sb_rec->parameters) {
   fprintf(fp,"\nParameters :\n");
   print_textual(fp,sb_rec->parameters);
 }

 /* local variables */
 if (sb_rec->action_impl &&
sb_rec->action_impl->local_variables) {
   fprintf(fp,"\nAction Language implementation: \n");
   fprintf(fp,"Local Variables :\n");
   print_textual(fp,sb_rec->action_impl->local_variables);
   /*print_names_of_ids(fp,sb_rec->,"Local-Vars","local-var");*/
 }
 /* definition */
 if(sb_rec->action_impl && sb_rec->action_impl->code){
   fprintf(fp,"Subroutine Action Language Code: \n");
   print_rpn(fp,sb_rec->action_impl->code);
 }
```

```
     if (sb_rec->graphical_impl) {
       fprintf(fp,"\nGraphical Language implementation: \n");
       if (sb_rec->graphical_impl->local_variables) {
    fprintf(fp,"Local Variables :\n");
    print_textual(fp,sb_rec->graphical_impl->local_variables);
       }
       if (sb_rec->graphical_impl->procedural_sch_id) {
    name = id_to_name(sb_rec->graphical_impl->procedural_sch_id);
    fprintf(fp,"Procedural Statechart : %s\n",name);
    free(name);
       }
     }

     fprintf(fp,"\n");
  }
}

void
print_connector(fp,ids)
      FILE *fp;
      genptr ids;
{
  stmm_id id;
  stmm_cn_nodep cn_rec;
  stmm_list_elm p;
  string name;

  /* Print fields of connector - stmm_cn_node */

  /* go through list of connector-id's pointers */
  stmm_for_every_element_in_list(ids,p) {

    /* extract the element value - id */
    id = (int)stmm_list_get_elm_value(p);

    fprintf(fp,"CONNECTOR %d\n",id);
```

```
/* retrieve the connector's structure */
cn_rec = (stmm_cn_nodep)stmm_get_el(id);

/* stmm_type */
fprintf(fp,"Stmm_type: %s\n",
    stmm_type_conv_table[cn_rec->stmm_type].stmm_type_name);

/* type */
fprintf(fp,"Connector Type: ");
switch(cn_rec->type) {
case stmm_cn_history:
  fprintf(fp,"stmm_cn_history\n");
  break;
case stmm_cn_deep_history:
  fprintf(fp,"stmm_cn_deep_history\n");
  break;
case stmm_cn_termination:
  fprintf(fp,"stmm_cn_termination\n");
  break;
case stmm_cn_default:
  fprintf(fp,"stmm_cn_default\n");
  break;
}

/* in_cts */
if(stmm_list_length(cn_rec->in_cts)){
  fprintf(fp,"Entering compound transitions:\n");
  print_ids_list(fp,cn_rec->in_cts);
}

/* out_cts */
if(stmm_list_length(cn_rec->out_cts)){
  fprintf(fp,"Exiting compound transitions:\n");
  print_ids_list(fp,cn_rec->out_cts);
}
```

```
    /* parent_box */
    name = id_to_name(cn_rec->parent_box);
    fprintf(fp,"Parent box : %s\n",name);
    free(name);

    fprintf(fp,"\n");
  }
}

void
print_charts(fp,ids)
     FILE *fp;
     genptr ids;
{
  stmm_list_elm p,p1;
  stmm_ch_nodep ch_rec;
  stmm_id id;
  stmm_formal_prm_p formal_prm;

  /* Print fields of chart - stmm_ch_node */

  /* go through list of chart-id's pointers */
  stmm_for_every_element_in_list(ids,p) {

    /* extract the element value - id */
    id = (int)stmm_list_get_elm_value(p);

    /* retrieve the chart's structure */
    ch_rec = (stmm_ch_nodep)stmm_get_el(id);

    /* print common fields (stmm_el_node) */
    print_common(fp,(stmm_el_nodep)ch_rec);
```

```
    /* type */
    fprintf(fp,"Chart type: %s\n",
        ch_type_conv_table[ch_rec->type].ch_type_name);

    /* formal_prms */
    if(ch_rec->formal_prms){
      fprintf(fp,"Formal parameters:\n");

      /* go through list of formal_param's pointers */
      stmm_for_every_element_in_list(ch_rec->formal_prms,p1) {

   /* retrieve the formal-parameter's structure */
   formal_prm = (stmm_formal_prm_p)stmm_list_get_elm_value(p1);

   /* print formal-parameter's fields */
   fprintf(fp,"   id: %d  type: %s  mode: %s\n",formal_prm->id,
          gp_type_conv_tab[formal_prm->type].type_name,
          gp_mode_conv_tab[formal_prm->mode].mode_name);
      }
    }

    /* time units */
    if (ch_rec->time_unit) {
      fprintf(fp,"Time unit: %d %s\n",ch_rec->time_unit->value,
          ch_rec->time_unit->unit);
    }

    fprintf(fp,"\n");
  }
}
```

```
void
dump_activities()
{
  char chart[100];
  stm_id  ch_id,ac_id,fl_id;
  int  status;
  stm_list list1,list2,list3,list4,list5;
  stm_element_name name;
  stm_element_type type;
  FILE *fp1;

  fp1 = fopen("/tmp/dataport.dump","w");
  list1 = stm_list_create(end_of_list, &status);
  list2 = stm_list_create(end_of_list, &status);
  list3 = stm_list_create(end_of_list, &status);
  list4 = stm_list_create(end_of_list, &status);
  list5 = stm_list_create(end_of_list, &status);
  printf("Please Enter Root Chart: ");
  scanf("%s",chart);
  while(strcmp(chart,"000")){
    ch_id = stm_r_ch(chart,&status);
    if(!status)
      list1 = stm_list_add_element(list1,ch_id,&status);
    else
      printf("No such chart. Try again");
    printf("Please Enter Root Chart: ");
    scanf("%s",chart);
  }
  list2 = stm_r_ch_descendants_of_ch(list1,&status);
  list3 = stm_list_union(list1,list2,&status);
  list2 = stm_r_ch_referenced_all_by_ch(list3,&status);
  list1 = stm_list_union(list3,list2,&status);
  list2 = stm_r_ac_defined_in_ch(list1,&status);
  list3 = stm_r_ac_instance_of_ch(list1,&status);
  list4 = stm_r_bl_defined_in_ch(list1,&status);
  list5 = stm_r_bl_instance_of_ch(list1,&status);
```

```
list1 = stm_list_union(list3,list2,&status);
list2 = stm_list_union(list1,list4,&status);
list1 = stm_list_union(list2,list5,&status);
list_destroy(list3);
list3 = stm_list_create(end_of_list, &status);
for(ac_id = (stm_id)stm_list_first_element(list1,&status);
    status == stm_success;
    ac_id = (stm_id)stm_list_next_element(list1,&status)){
  name = stm_r_uniquename(ac_id,&status);
  if(!status)
    fprintf(fp1,"\nActivity/Block : %s id: %d\n",name,ac_id);
  else
    fprintf(fp1,"\nActivity/Block id: %d\n",ac_id);
  list3 = stm_list_add_element(list3,ac_id,&status);
  type = stm_r_element_type(ac_id,&status);
  if(type == stm_activity)
    list2 = stm_r_af_from_source_ac(list3,&status);
  else
    list2 = stm_r_bf_from_source_bl(list3,&status);
  if(stm_list_length(list2)){
    fprintf(fp1,"Exiting flowlines:\n");
    for(fl_id = (stm_id)stm_list_first_element(list2,&status);
   status == stm_success;
   fl_id = (stm_id)stm_list_next_element(list2,&status)){
 fprintf(fp1,"   \t\t%d\n",fl_id);
    }
  }
  if(type == stm_activity)
    list2 = stm_r_af_to_target_ac(list3,&status);
  else
    list2 = stm_r_bf_to_target_bl(list3,&status);
  if(stmm_list_length(list2)){
    fprintf(fp1,"Entering flowlines:\n");
    for(fl_id = (stm_id)stm_list_first_element(list2,&status);
   status == stm_success;
   fl_id = (stm_id)stm_list_next_element(list2,&status)){
```

```
    fprintf(fp1,"   \t\t%d\n",fl_id);
      }
    }
    list3 = stm_list_delete_element(list3,ac_id,&status);
  }
  fclose(fp1);
}

main()
{
  FILE *fp;
  char  workarea[100], profile[100],project[100];
  stmm_list modules,ids;
  int scope_id,i=0,with_dataport=0;
  stmm_list_elm p;
  stmm_module_nodep scope_info;

  printf("Please Enter Project: ");
  scanf("%s",project);
  printf("Please Enter Workarea: ");
  scanf("%s",workarea);
  printf("Please Enter Profile-name: ");
  scanf("%s",profile);

  /* initialize and build the database */
  if(!stmm_open_session(project,workarea,profile,with_dataport)){
    printf("Error/s in database - exiting program\n");
    exit (-1);
  }

  /* open output file */
  if(with_dataport)
    dump_activities();
  fp = fopen("/tmp/ana_data.dump","w");
```

```
/* retrieve all modules in the scope */
modules = stmm_get_scope_modules();

/* go through list of module pointers */
stmm_for_every_element_in_list(modules,p) {

  /* extract the element value - scope id */
  scope_id = (int)stmm_list_get_elm_value(p);
  scope_info = stmm_get_module_info(scope_id);

  i++;

  fprintf(fp,"~~~~~~~~~~~");
  fprintf(fp,"\nMODULE %s\n",scope_info->name);
  fprintf(fp,"~~~~~~~~~~~\n");

  /* retrieve all charts in the module */
  ids = stmm_get_ch(scope_id);
  fprintf(fp,"\n***********");
  fprintf(fp,"\nCharts : \n");
  fprintf(fp,"***********\n");
  print_charts(fp,ids);

  /* retrieve all activities in the module */
  ids = stmm_get_ac(scope_id);
  if(stmm_list_length(ids)){
    fprintf(fp,"\n**************");
    fprintf(fp,"\nActivities : \n");
    fprintf(fp,"**************\n");
    print_activity(fp,ids);
  }
```

```
/* retrieve all states in the module */
ids = stmm_get_st(scope_id);
if(stmm_list_length(ids)){
  fprintf(fp,"\n**********");
  fprintf(fp,"\nStates : \n");
  fprintf(fp,"**********\n");
  print_state(fp,ids);
}

/* retrieve all data-items in the module */
ids = stmm_get_di(scope_id);
if(stmm_list_length(ids)){
  fprintf(fp,"\n**************");
  fprintf(fp,"\nDataItems : \n");
  fprintf(fp,"**************\n");
  print_textual(fp,ids);
}

/* retrieve all events in the module */
ids = stmm_get_ev(scope_id);
if(stmm_list_length(ids)){
  fprintf(fp,"\n**********");
  fprintf(fp,"\nEvents : \n");
  fprintf(fp,"**********\n");
  print_textual(fp,ids);
}

/* retrieve all conditions in the module */
ids = stmm_get_co(scope_id);
if(stmm_list_length(ids)){
  fprintf(fp,"\n**************");
  fprintf(fp,"\nConditions : \n");
  fprintf(fp,"**************\n");
  print_textual(fp,ids);
}
```

```
/* retrieve all user-types in the module */
ids = stmm_get_ut(scope_id);
if(stmm_list_length(ids)){
  fprintf(fp,"\n**************");
  fprintf(fp,"\nUser-Types : \n");
  fprintf(fp,"**************\n");
  print_textual(fp,ids);
}

/* retrieve all actions in the module */
ids = stmm_get_an(scope_id);
if(stmm_list_length(ids)){
  fprintf(fp,"\n***********");
  fprintf(fp,"\nActions : \n");
  fprintf(fp,"***********\n");
  print_textual(fp,ids);
}

/* retrieve all connectors in the module */
ids = stmm_get_cn(scope_id);
if(stmm_list_length(ids)){
  fprintf(fp,"\n**************");
  fprintf(fp,"\nConnectors : \n");
  fprintf(fp,"**************\n");
  print_connector(fp,ids);
}

/* retrieve all information-flows in the module */
ids = stmm_get_if(scope_id);
if(stmm_list_length(ids)){
  fprintf(fp,"\n********************");
  fprintf(fp,"\nInformation-flows : \n");
  fprintf(fp,"********************\n");
  print_textual(fp,ids);
}
```

```
/* retrieve all compound_arrows in the module */
ids = stmm_get_cm(scope_id);
if(stmm_list_length(ids)){
  fprintf(fp,"\n********************");
  fprintf(fp,"\nCompound-Arrows : \n");
  fprintf(fp,"********************\n");
  print_compound(fp,ids);
}

/* retrieve all functions in the module */
ids = stmm_get_sb(scope_id);
if(stmm_list_length(ids)){
  fprintf(fp,"\n**************");
  fprintf(fp,"\nSubroutines : \n");
  fprintf(fp,"**************\n");
  print_subroutines(fp,ids);
}

/* retrieve all fields in the module */
ids = stmm_get_fd(scope_id);
if(stmm_list_length(ids)){
  fprintf(fp,"\n**************");
  fprintf(fp,"\nFields : \n");
  fprintf(fp,"**************\n");
  print_textual(fp,ids);
}
```

```
    /* retrieve all enumerated values in the module */
    ids = stmm_get_en(scope_id);
    if(stmm_list_length(ids)){
      fprintf(fp,"\n******************");
      fprintf(fp,"\nEnumerated-Values :");
      fprintf(fp,"\n******************\n");
      print_textual(fp,ids);
    }

  }

  /* close the session */
  stmm_close_session(with_dataport);

  fclose(fp);
}
```

CG Builder API Library Data Types

This section defines the CG Builder API data types in the `api_types.h` file, which you must include in your C program.`string_limit_node`.

The `api_types.h` file is delivered with the Anaport package.

```
/* Interface File for the Code-Gen Api version 1.1 */

#ifdef DLL_LINK
#if !defined(__ANAPORT_H__)/* { */
#define __ANAPORT_H__

#if defined(__cplusplus)
extern "C" {
#endif

#define PASTE(_x,_y) _x##_y
#if defined(BLD_DLL)
#define FUNC_NAME(_x) PASTE(wrap_,_x)
#else
#define FUNC_NAME(_x) _x
#endif

#if defined(__cplusplus)
}
#endif

#endif /* __ANAPORT_H__ *//* } */
#endif DLL_LINK
```

```
#ifndef BASICS_H
typedef char *string;
#define ARGS(c) ()
typedef void *genptr;
typedef int  boolean;
#endif

/* Lists structures */

typedef genptr stmm_ids_list;
typedef genptr stmm_rpn_list;

#if defined (STM4NT) || defined (DLL_LINK)
typedef __int64 stmm_id;
#else
typedef long long stmm_id;
#endif

typedef genptr stmm_list;
typedef genptr stmm_list_elm;

/* All element types that exist in the API data base */

typedef enum {
  stmm_chart,
  stmm_state,
  stmm_connector,
  stmm_compound_ar,
  stmm_data_item,
  stmm_condition,
  stmm_event,
  stmm_info_flow,
  stmm_action,
  stmm_activity,
  stmm_module,
  stmm_subroutine,
```

```
  stmm_user_type,
  stmm_field,
  stmm_enumerated_value,
  stmm_subroutine_data,
  stmm_error,
  stmm_lifeline,
  stmm_message,
  stmm_timing_constraint,
  stmm_order_insignificant,
  stmm_referenced_sd,
  stmm_scenario
} stmm_id_type;



/* Type of Textuals */


typedef enum {    /* Description:                              relavant for: */
  stmm_reference, /* Not explicitly defined by DDE             all           */
  stmm_variable,  /* Variable element (empty definition)       di/co/ev/an
*/
  stmm_compound,  /* Defined by a compound expression          di/co/ev/an
*/
  stmm_constant,  /* Defined by a constant literal/element/exp di,co
*/
  stmm_alias,     /* Alias to another element/component        di            */
  stmm_defined,   /* Explicitly defined by DDE                 if,en,ut,fd   */
  stmm_local_var, /* Subroutine's local variable               sd            */
  stmm_parameter  /* Subroutine's parameter                    sd            */
} stmm_tx_type;


typedef enum {    /* Description:                              relavant for: */
  stmm_procedure, /* Subroutine defined as procedure           sb            */
  stmm_function,  /* Subroutine defined as function            sb            */
  stmm_task,      /* Subroutine defined as task                sb            */
  stmm_stm_func,  /* Statemate function, e.g. SIN,COS,MAX      sb            */
  stmm_undef      /* Subroutine that wasn't defined by DDE     sb            */
} stmm_sb_type;
```

```
/* Data Type for Data-Items/User-Type/Field. legal values are: */

typedef enum {
  stmm_dt_missing,
  stmm_dt_integer,
  stmm_dt_real,
  stmm_dt_string,
  stmm_dt_bit,
  stmm_dt_bit_array,
  stmm_dt_record,
  stmm_dt_union,
  stmm_dt_user_def,
  stmm_dt_enum_type, /* Relavant only for User-Def Type                    */
  stmm_dt_condition, /* Relavant for User-Def Type,Field or Subroutine-Data
*/
  stmm_dt_event,     /* Relavant only for Subroutine-Data                  */
  stmm_dt_void,
  stmm_dt_state,     /* Relevant only for stmm_bind_node                   */
  stmm_dt_activity   /* Relevant only for stmm_bind_node                   */
} stmm_data_type;


/* Types of activities. Legal values are: */
typedef enum {
  stmm_ac_internal,             /* Internal activity  */
  stmm_ac_control,              /* Control activity   */
  stmm_ac_reference,            /* Reference activity */
  stmm_ac_data_store,           /* Data-Store         */
  stmm_ac_external,             /* External           */
  stmm_ac_environment,          /* Environment        */
  stmm_ac_diagram,
  stmm_ac_lifeline,             /* Sequence Diagram Lifeline */
  stmm_ac_ext_lifeline,         /* Sequence Diagram External Lifeline */
```

```
  stmm_ac_referenced_sd,          /* Referenced Sequence Diagram  */
  stmm_ac_router,                 /* Router */
  stmm_ac_external_router        /* External-Router */
} stmm_ac_type;

/* Types of States. Legal values are: */
typedef enum {
  stmm_st_or,               /* Or State  */
  stmm_st_and,              /* And State */
  stmm_st_reference,
  stmm_st_diagram,
  stmm_st_action_box,
  stmm_st_decision_box,
  stmm_st_switch_box
} stmm_st_type;

typedef enum {
  stmm_inst_offpage,
  stmm_inst_generic,
  stmm_inst_component,
  stmm_inst_continuous_vsm,
  stmm_inst_missing
} stmm_inst_type;

/* Types of Charts. Legal values are :   */

typedef enum {
  stmm_ch_statechart,
  stmm_ch_activitychart,
  stmm_ch_modulechart,
  stmm_ch_blockdiagram ,
  stmm_ch_gds,
  stmm_ch_continchart,
  stmm_ch_flowchart,
  stmm_ch_sequence_diagram
} stmm_ch_type;
```

```
/* Termination Type of an Activity. Legal values are: */
typedef enum {
  stmm_ac_missing,                  /* No termination specified */
  stmm_ac_self_term,                /* Self termination         */
  stmm_ac_controlled_term,          /* Controlled Termination   */
  stmm_ac_procedure_like            /* Procedure like           */
} stmm_ac_term;

typedef enum {
  stmm_gp_data_item,
  stmm_gp_event,
  stmm_gp_condition,
  stmm_gp_textual,
  stmm_gp_activity
} stmm_gp_type;

typedef enum {
  stmm_gp_mode_input,
  stmm_gp_mode_output,
  stmm_gp_mode_inout,
  stmm_gp_mode_constant
} stmm_gp_mode;

typedef enum {
  stmm_cn_history,
  stmm_cn_deep_history,
  stmm_cn_termination,
  stmm_cn_default
} stmm_cn_type;

typedef enum {
  stmm_identifier,
  stmm_operator,
  stmm_context_var,
  stmm_field_name,
```

```
  stmm_integer,
  stmm_real,
  stmm_string,
  stmm_boolean,
  stmm_sb_local,
  stmm_sb_parameter,
  stmm_missing_node_type,
  stmm_empty_id
} stmm_rpn_node_type;

typedef enum {
  stmm_op_trig_action,        /* "/"        "EVENT/ACTION"                */
  stmm_op_ev_co,              /* "[]"       "[C]"                         */
  stmm_op_action_sep,         /* ";"        "ACTION1;ACTION2"             */
  stmm_op_if_c,               /* "if"       "if C then ACTION end if"     */
  stmm_op_when,               /* "when"     "when E then ACTION end when" */
  stmm_op_and_ev,             /* "and"      "E1 and E2"                   */
  stmm_op_or_ev,              /* "or"       "E1 or E2"                    */
  stmm_op_not_ev,              /* "not"      "not E"                 */
  stmm_op_unary_plus,          /* "+"        "+EXP1"     */
  stmm_op_unary_minus,        /* "-"        "-EXP1"                       */
  stmm_op_plus,                /* "+"        "EXP1 + EXP2"      */
  stmm_op_minus,               /* "-"        "EXP1 - EXP2"      */
  stmm_op_multiply,            /* "*"        "EXP1 * EXP2"     */
  stmm_op_divide,              /* "/"        "EXP1 / EXP2"     */
  stmm_op_power,               /* "**"       "EXP1 ** EXP2"     */
  stmm_op_binary,             /* ","        "A,B"                         */
  stmm_op_entered,            /* "en"       "en(STATE1)"                  */
  stmm_op_exited,             /* "ex"       "ex(STATE1)"                  */
  stmm_op_started,            /* "st"       "st(ACTIVITY1)"               */
  stmm_op_started_ac,         /* "st"       "st" in the activity itself   */
  stmm_op_stopped,            /* "sp"       "sp(ACTIVITY1)"               */
  stmm_op_read,               /* "rd"       "rd(DATA_ITEM1)"              */
  stmm_op_written,            /* "wr"       "wr(DATA_ITEM1)"              */
  stmm_op_became_true,         /* "tr"       "tr(C1)"                     */
  stmm_op_became_false,        /* "fs"       "fs(C1)"                     */
```

```
stmm_op_changed,            /* "ch"       "ch(DI1)"                    */
stmm_op_timeout,            /* "tm"       "tm(EV1,10)"                 */
stmm_op_delay,             /* "dy"       "dy(10[,ev])"                */
stmm_op_in,                 /* "in"       "in(ST1)"                    */
stmm_op_active,             /* "ac"       "ac(AC1)"                    */
stmm_op_hanging,            /* "hg"       "hg(AC1)"                    */
stmm_op_rel_eq,             /* "="        "DI1 = DI2"                  */
stmm_op_rel_neq,            /* "/="       "DI1 /= DI2"                 */
stmm_op_rel_gt,             /* ">"        "DI1 > DI2"                  */
stmm_op_rel_lt,             /* "<"        "DI1 < DI2"                  */
stmm_op_rel_geq,            /* ">="       "DI1 >= DI2"                 */
stmm_op_rel_leq,            /* "<="       "DI1 <= DI2"                 */
stmm_op_assignment,         /* ":="       "DI1 := DI2"                 */
stmm_op_schedule,           /* "sc!"      "sc(ACTION1,10)"             */
stmm_op_history_clear,      /* "hc!"      "hc!(ST1)"                   */
stmm_op_deep_clear,         /* "dc!"      "dc!(ST1)"                   */
stmm_op_start,              /* "st!"      "st!(AC1)"                   */
stmm_op_stop,               /* "sp!"      "sp!(AC1)"                   */
stmm_op_stop_ac,           /* "sp!"      "sp!" in the activity itself */
stmm_op_suspend,            /* "sd!"      "sd!(AC1)"                   */
stmm_op_resume,             /* "rs!"      "rs!(AC1)"                   */
stmm_op_make_true,          /* "rt!"      "tr!(CO1)"                   */
stmm_op_make_false,         /* "fs!"      "fs!(CO1)"                   */
stmm_op_get,                /* "gt!"      "gt!(Res)"                   */
stmm_op_op_release,         /* "rl!"      "rl!(Res)"                   */
stmm_op_read_data,          /* "rd!"      "rd!(DI1)"                   */
stmm_op_write_data,         /* "wr!"      "wr!(DI1)"                   */
stmm_op_send,              /* "sn!"      "sn!(mess)"                  */
stmm_op_receive,           /* "rc!"      "rc!(mess)"                  */
stmm_op_entering,           /* "ns"       "ns" in the state itself     */
stmm_op_exiting,            /* "xs"       "xs" in the state itself     */
stmm_op_and_co,             /* "and",     "CO1 and CO2"                */
stmm_op_or_co,              /* "or"       "CO1 or CO2"                 */
stmm_op_not_co,             /* "not"      "not CO"                     */
stmm_op_if_consists_of,     /* ""         first op in definition of    */
                           /*                      information flow    */
```

```
stmm_op_to,                 /* "to"        "1 to 10"                    */
stmm_op_downto,             /* "downto"    "10 downto 1"                */
stmm_op_any_ev,             /* "any"       "any(ARR_EV)"                */
stmm_op_all_ev,             /* "all"       "all(ARR_EV)"                */
stmm_op_any_co,             /* "any"       "any(ARR_CO)"                */
stmm_op_all_co,             /* "all"       "all(ARR_CO)"                */
stmm_op_or_di,              /* "or"        "EXP1 or EXP2"               */
stmm_op_and_di,             /* "and"       "EXP1 and EXP2"              */
stmm_op_not_di,             /* "not"       "not EXP1"                   */
stmm_op_nor,                /* "nor"       "EXP1 nor EXP2"              */
stmm_op_nand,               /* "nand"      "EXP1 nand EXP2"             */
stmm_op_xor,                /* "xor"       "EXP1 xor EXP2"              */
stmm_op_nxor,               /* "nxor"      "EXP1 nxor EXP2"             */
stmm_op_or_dico,            /* "or"        "EXP1 or EXP2"               */
stmm_op_and_dico,           /* "and"       "EXP1 and EXP2"              */
stmm_op_not_dico,           /* "not"       "not EXP1"                   */
stmm_op_if_binary_consists,  /* ";" separator op between if components */
stmm_op_st_react,           /* "*"     first op in static reactions list  */
stmm_op_parenth,            /* "()"        "(EXP)"                      */
stmm_op_st_react_binary,    /* ";;"separator op between static reactions */
stmm_op_while,              /* "while"     "while C loop AN end loop"     */
stmm_op_for,                /* "for"  "for $I in K to L loop AN end loop" */
stmm_op_break,              /* "break"     "break" in the loop section"   */
stmm_op_array_aggregate_di,  /* "{}"        "{1,2,3}"                    */
stmm_op_array_aggregate_co,  /* "{}"        "{true,false,true}"          */
stmm_op_array_aggregate_dico,/* "{}"        "{1,2,3}"                    */
stmm_op_times_di,           /* "*"         "{1,2,10*3}                  */
stmm_op_to_end_di,          /* "*"         "{1,2,*3}"                   */
stmm_op_times_co,           /* "*"         "{true,false,4*true}"        */
stmm_op_to_end_co,          /* "*"         "{true,false,*true}"         */
stmm_op_times_dico,         /* "*"         "{1,2,10*3}                  */
stmm_op_to_end_dico,        /* "*"         "{1,2,*3}"                   */
stmm_op_aggregate_binary_di, /* ","        "1,2"                        */
stmm_op_aggregate_binary_co, /* ","        "true,false"                 */
stmm_op_aggregate_binary_dico,/* ","        "1,2"                        */
stmm_op_length_of,          /* "length_of" "length_of(DI1)"             */
```

```
stmm_op_rindex,              /* "rindex"    "rindex(DI1)"                  */
stmm_op_lindex,              /* "lindex"    "lindex(DI1)"                  */
stmm_op_rel_compare,         /* "=="        "EXP1 == EXP2"                 */
stmm_op_cl_assignment,       /* ":=" "CE:=A" in combinational assignment" */
stmm_op_cl_when,             /* "when"      "CE:=A when C else B"          */
stmm_op_cl_sep,              /* ";"         "separator op between cas"     */
stmm_op_concatenate,         /* "&"         "EXP1 & EXP2"                  */
stmm_op_field,               /* "."         "RECORD.FIELD"                 */
stmm_op_array,               /* "()"        "ARRAY(INDEX)"                 */
stmm_op_array_slice,         /* "(..)"      "ARRAY(INDEX1..INDEX2)"        */
stmm_op_array_ev,            /* "()"        "EV_ARRAY(INDEX)"              */
stmm_op_array_slice_ev,    /* "(..)" "EV_ARRAY(INDEX1..INDEX2)"     */
stmm_op_func_call,           /* "()"        "FUNC(OPERAND)"                */
stmm_op_q_put,               /* "put!"      "put!(DICO_QUEUE,ITEM)"        */
stmm_op_q_uput,              /* "uput!"     "uput!(DICO_QUEUE,ITEM)"       */
stmm_op_q_get,              /* "get!"    "get!(DICO_QUEUE,ITEM,[STATUS])" */
stmm_op_q_flush,             /* "fl!"       "fl!(DICO_QUEUE)"              */
stmm_op_q_length,            /* "q_length"  "q_length(DICO_QUEUE)"         */
stmm_op_q_peak,             /* "peek!"   "peek!(DICO_QUEUE,ITEM,[STATUS]" */
stmm_op_enum_def,            /* "{}"      "{SUNDAY,MONDAY}" in enum def    */
stmm_op_enum_sep,            /* ","       "{SUNDAY,MONDAY,TUESDAY}"        */
stmm_op_enum_first,          /* "enum_first"   "enum_first(DAYS)"          */
stmm_op_enum_last,           /* "enum_last"    "enum_last(DAYS)"           */
stmm_op_enum_succ,           /* "enum_succ"    "enum_succ(SUNDAY)"         */
stmm_op_enum_pred,           /* "enum_pred"    "enum_pred(MONDAY)"         */
stmm_op_enum_ordinal,       /* "enum_ordinal" "enum_ordinal(SUNDAY)"      */
stmm_op_enum_value,          /* "enum_value"   "enum_value(DAYS,0)"        */
stmm_op_enum_image,          /* "enum_image"   "enum_image(SUNDAY)"        */
stmm_op_array_aggregate_en, /* "{}"          "{SUNDAY,MONDAY,TUESDAY}"    */
stmm_op_aggregate_binary_en, /* ","          "{SUNDAY,MONDAY,TUESDAY}"    */
stmm_op_times_en,            /* "*"           "{SUNDAY,MONDAY,3*TUESDAY}" */
stmm_op_to_end_en,           /* "*"           "{SUNDAY,MONDAY,*TUESDAY}"  */
stmm_op_func_or_array_ev,    /* "()"          "E/FUNC_OR_EV(1)"           */
stmm_op_return,              /* "return"       "return(1)"                */
stmm_op_return_from_tt,      /* used in truth-table rpn's                  */
```

```
stmm_op_yes,
stmm_op_no,
stmm_op_fl_else,
stmm_op_switch_exp,
stmm_op_switch_case,
stmm_op_inline,              /*for Verbatim                                  */
stmm_op_forloop_sep,        /* "$;" used internally in for-loops rpns     */
stmm_op_unary_plusplus,      /* X++                                        */
stmm_op_unary_minusminus,    /* X--                                        */
stmm_op_unary_tt_neq,
stmm_op_switch_c,
stmm_op_case_c,
stmm_exp_case_sep,
stmm_op_default,
stmm_op_case_ada,
stmm_op_when_ada,
stmm_op_others_ada,
stmm_op_choice_sep_ada,
stmm_op_break_sw,
stmm_op_unary_tt_eq,
stmm_op_unary_tt_lt,
stmm_op_unary_tt_gt,
stmm_op_unary_tt_geq,
stmm_op_unary_tt_leq,
stmm_op_tmin,
stmm_op_tmax,
stmm_op_addition_assignment,
stmm_op_subtraction_assignment,
stmm_op_multiplication_assignment,
stmm_op_division_assignment,
stmm_op_not_avalible,
stmm_op_tr_else,            /* else on transitions, mini-specs, reactions */
stmm_op_enum_init,
stmm_op_in_sim,
stmm_op_reset_element,
stmm_op_reset_all,
```

```
  stmm_op_last
}stmm_operator_type;

/* Structure for the Code-Gen profile's Modules */
typedef struct stmm_module_node {
  stmm_id        id;   /* with which to call all the queries on this module */
  string          name; /* given by the user, in the Code-Gen profile */
}stmm_module_node, *stmm_module_nodep;

/* Data structure for Data-Item/User-Type/Field/Ev/Co. legal values are: */
typedef enum {
  stmm_missing,
  stmm_single,
  stmm_array,
  stmm_queue
} stmm_structure;

/* indicates weather the limit was given as an element name, */
/* an element synoym or a literal                            */

typedef enum stmm_refered_by{
  ref_by_name,
  ref_by_syn,
  ref_by_literal,
  ref_by_missing
} stmm_refered_by;

/* Structures for a Literal/Identifier given as a limit for  */
/* an Array,Bit-array,Integer or String                      */

typedef struct one_limit_node {
  stmm_refered_by refered_by;
  stmm_id         id;
  int             number;
} one_limit_node, *one_limit_nodep;
```

```
/* Structures for a Literal/Identifier given as a limit for real */

typedef struct real_limit_node {
  stmm_refered_by refered_by;
  stmm_id         id;
  double          number;
} real_limit_node, *real_limit_nodep;

/* Structures for a Literal/Identifier given as a limit for string */

typedef struct string_limit_node {
  stmm_refered_by refered_by;
  stmm_id         id;
  string          str;
} string_limit_node, *string_limit_nodep;

/* Structures for a Literal/Identifier given as a limit for bit-array */

typedef struct ba_limit_node {
  stmm_refered_by refered_by;
  stmm_id         id;
  stmm_rpn_list   ba_val;
} ba_limit_node, *ba_limit_nodep;

/* Strucure for the indices of an element of an array structure */

typedef struct stmm_array_limits {
  one_limit_nodep lindex;
  one_limit_nodep rindex;
} stmm_array_limits;
```

```
/* Structure for the limits of an integer element */

typedef struct stmm_integer_limits {
  stmm_id        bits_num;
  one_limit_nodep min_val;
  one_limit_nodep max_val;
  one_limit_nodep initial_val;
} stmm_integer_limits;

/* Structure for the limits of a real element */

typedef struct stmm_real_limits {
  real_limit_nodep min_val;
  real_limit_nodep max_val;
  real_limit_nodep initial_val;
} stmm_real_limits;

/* Structure for the limits of a string element */

typedef struct stmm_string_limits {
  one_limit_nodep    length;
  string_limit_nodep initial_val;
} stmm_string_limits;

/* Structure for the limits of a bit-array element */

typedef struct stmm_ba_limits{
  stmm_array_limits *array_limits;
  ba_limit_nodep    initial_val;
} stmm_ba_limits;
```

```
/* Structure for the limits of a bit element */

typedef struct stmm_bit_limits {
  one_limit_nodep initial_val;
} stmm_bit_limits;

/* Structure for the limits of a condition element */

typedef struct stmm_condition_limits {
  one_limit_nodep initial_val;
} stmm_condition_limits;

/* Structure for the limits of a enumerated type */

typedef struct stmm_enum_limits {
  one_limit_nodep initial_val;
} stmm_enum_limits;

/* Structure for all limits needed for the definition of  */
/* an element data-type                                   */

typedef union stmm_dt_limits {
  stmm_integer_limits   *integer_limits;
  stmm_string_limits    *string_limits;
  stmm_ba_limits        *bit_array_limits;
  stmm_real_limits      *real_limits;
  stmm_bit_limits       *bit_limits;
  stmm_condition_limits *condition_limits;
  stmm_enum_limits      *enum_limits;
} stmm_dt_limits;
```

```
/* Structure for one attribute of an element */

typedef struct stmm_attribute {
  string       name;
  string       value;
} stmm_attribute, *stmm_attribute_p;

/* Structure for Instance Parameter Bindings */

typedef struct stmm_inst_prm {
  stmm_id           formal_prm;
  stmm_rpn_list     actual_prm;
  stmm_id_type      type;        /* Data-Item/Condition/Event/Activity */
} stmm_inst_prm;

typedef struct stmm_formal_prm {
  stmm_id         id;
  stmm_gp_type    type;
  stmm_gp_mode    mode;
} stmm_formal_prm, *stmm_formal_prm_p;

typedef union stmm_cm_exp {
  struct {
    stmm_rpn_list   triggers;
    stmm_list       actions;      /* list of stmm_rpn_list of all actions */
  } transition_labels;            /* in case of compound transition */
  stmm_list flow_list ;           /* any other compound arrow       */
} stmm_cm_exp;

typedef union stmm_constant_val {
  double   r_value;
  int      i_value;
  string   s_value;
  boolean  b_value;
} stmm_constant_val;
```

```
typedef enum {
  stmm_ac_within,
  stmm_ac_throughout
} stmm_ac_duration;

typedef enum {
  stmm_cm_transition,  /* Compound Transition in statechart */
  stmm_cm_a_flow,      /* Data-Flow/Control-Flow in activity chart */
  stmm_cm_b_flow,      /* Signal/Bus in block diagram */
  stmm_cm_m_flow       /* Flow-Line in module chart */
} stmm_cm_type;

typedef  enum {
  stmm_ac_regular,      /* Regular activity */
  stmm_ac_procedure,    /* Activity marked as procedure by the profile */
  stmm_ac_task          /* Activity marked as taks by the profile */
} stmm_ac_impl;

typedef enum {
  stmm_ucb_disable,     /* User Code Bindings Disabled */
  stmm_ucb_enable,      /* User Code Bindings Enabled  */
  stmm_ucb_missing
} stmm_ucb_status;

typedef struct stmm_ac_in_s {
  stmm_id          activity_id;
  stmm_ac_duration duration;
} stmm_ac_in_s;

typedef struct stmm_time_unit {
  string unit;
  int value;
} stmm_time_unit, *stmm_time_unitp;
```

```
typedef enum {
  stmm_cbk_missing,    /* Default value              */
  stmm_cbk_enable,     /* Enable callbacks bindings  */
  stmm_cbk_disable     /* Disable callbacks bindings */
} stmm_cbk_status;

typedef struct stmm_global_var {
  stmm_id        id;                /* Id of the global variable di/co/ev */
  stmm_refered_by refered_by;        /* Refered by name or synonym        */
  stmm_gp_mode    mode;              /* In/Out/InOut                      */
} stmm_global_var, *stmm_global_varp;

/* Structure for the Subroutine's code[s] */

typedef struct stmm_user_code {
  stmm_list kr_c_code;   /* list of strings */
  stmm_list ansi_c_code; /* list of strings */
  stmm_list ada_code;    /* list of strings */
  stmm_list external_tool_code;
} stmm_user_code, *stmm_user_codep;

/* Subroutine's selected implementation */

typedef enum {
  stmm_action_lang,      /* Statemate Action Language               */
  stmm_graphic_proc,     /* Procedural Statechart                   */
  stmm_truth_table,      /* Truth-table                             */
  stmm_kr_c_code,        /* Kernighan & Ritchie C Code              */
  stmm_ansi_c_code,      /* Ansi C Code                             */
  stmm_ada_code,         /* Ada Code                                */
  stmm_vhdl_code,        /* VHDL Code                               */
  stmm_verilog_code,     /* Verilog Code                            */
  stmm_best_match,       /* Best match according to existing Code[s] */
  stmm_none,
  stmm_lookup_code,
  stmm_external_tool_code
```

```
} stmm_def_language;

typedef struct stmm_action_impl {
  stmm_ids_list local_variables; /* List of local variables ids */
  stmm_rpn_list code;
  stmm_rpn_list orig_code;
} stmm_action_impl, *stmm_action_implp;

typedef struct stmm_graph_impl {
  stmm_ids_list local_variables;      /* List of local variables ids       */
  stmm_id       procedural_sch_id;    /* id of the connected procedural sch */
  stmm_id       procedural_scope_id;  /* id of the scope containing all the */
                                      /* states/arrows/connectors.. of proc */
} stmm_graph_impl, *stmm_graph_implp;

/* Action & Activity selected implementation */

typedef enum stmm_act_impl {
  stmm_mini_spec_imp,
  stmm_subroutine_bind_imp,
  stmm_truth_table_imp,
  stmm_definition_imp,
  stmm_no_imp,
  stmm_best_match_imp
} stmm_act_impl;
```

```
typedef int stmm_cell_type;
#define stmm_rpn_cell             0x01
#define stmm_empty_cell           0x02
#define stmm_dont_care_cell       0x04
#define stmm_generate_ev          0x08
#define stmm_not_generate_ev      0x10
#define stmm_not_factorized_cell  0x0000
#define stmm_same_as_up_cell      0x0100
#define stmm_same_as_down_cell    0x0200
#define stmm_same_as_up_down_cell (stmm_same_as_up_cell |
stmm_same_as_down_cell)

#define stmm_is_rpn_cell(X)                 ((X) & stmm_rpn_cell)
#define stmm_is_empty_cell(X)               ((X) & stmm_empty_cell)
#define stmm_is_dont_care_cell(X)           ((X) & stmm_dont_care_cell)
#define stmm_is_generate_ev_cell(X)         ((X) & stmm_generate_ev)
#define stmm_is_not_generate_ev_cell(X)     ((X) & stmm_not_generate_ev)
#define stmm_is_factorized_cell(X)          ((X) & 0xFF00)
#define stmm_is_same_as_up_cell(X)          ((X) & stmm_same_as_up_cell)
#define stmm_is_same_as_down_cell(X)        ((X) & stmm_same_as_down_cell)
#define stmm_is_same_as_up_and_down_cell(X) (((X) & stmm_same_as_up_cell) &&
((X) & stmm_same_as_down_cell))

typedef struct stmm_tt_cell {
  stmm_cell_type cell_type;
  stmm_rpn_list  cell_rpn;
  stmm_rpn_list  orig_cell_rpn;
}stmm_tt_cell, *stmm_tt_cell_p;

typedef stmm_tt_cell_p* stmm_tt_row_data;
typedef stmm_tt_row_data* stmm_tt_cells_data;
```

```
typedef struct stmm_truth_table_node
{
  int          number_of_rows;
  int          number_of_columns;
  int          num_in;
  int          num_out;
  boolean      action_exists;
  boolean      default_exists;
  stmm_tt_cell ***cells;
}stmm_truth_table_node, *stmm_truth_table_nodep;

typedef struct stmm_el_node
{
  stmm_id           id;                /* id of element          */
  stmm_id_type      stmm_type;         /* di/co/ev/etc.          */
  string            name;              /* Statemate name         */
  string            code_gen_name;     /* Name in generated code */
  string            unique_name;       /* Statemate unique name  */
  stmm_list         attributes;        /* List of stmm_attribute */
  string            short_des;         /* Short description      */
  string            synonym;           /* Synonym                */
  stmm_cbk_status   cbk_status;        /* Enable/Disable cbk     */
  stmm_rpn_list     callback_binding;  /* Callbacks bindings     */
  genptr            user_data;         /* For user data.         */
} stmm_el_node, *stmm_el_nodep;

/* Textuals (di/ut/co/ev/an/if/fd/en/sd) */
```

```
typedef struct stmm_tx_node
{
  stmm_id           id;                /* id of element                      */
  stmm_id_type      stmm_type;         /* di/co/ev/udt/fd/if                 */
  string            name;              /* Statemate name                     */
  string            code_gen_name;     /* Name in generated code             */
  string            unique_name;       /* Statemate unique name              */
  stmm_list         attributes;        /* List of stmm_attribute             */
  string            short_des;         /* Short description                  */
  string            synonym;           /* Synonym                            */
  stmm_cbk_status   cbk_status;        /* Enable/Disable Callbacks bindings */
  stmm_rpn_list     callback_binding;  /* Callbacks bindings                 */
  genptr            user_data;         /* For user data.                     */
  stmm_tx_type      type;              /* variable/compound/constant         */
                                 /*  /alias/reference                  */
  stmm_structure    data_structure;    /* Single/Array/Queue                 */
  stmm_array_limits *array_limits;     /* Array limits in case of array      */
  stmm_data_type    data_type;         /* Int/Real/Str/Bit/Bit-Array         */
                                       /* /Record/Union/User-Type            */
  stmm_dt_limits    *dt_limits;       /* Limits for Int/Bit-array/String   */
  stmm_id           user_defined_type; /* For di defined as UDT.            */
  stmm_refered_by   ut_ref_by;         /* User-Def-Type refered by Name/Syn */
  stmm_act_impl     selected_impl;     /* Selected implementation            */
                                       /* (for action)                       */
  stmm_rpn_list     definition;        /* Definition of data-item.           */
  stmm_rpn_list     orig_definition;   /* Definition of data-item.           */
  stmm_rpn_list     truth_table;       /* Truth-table rpn (of action)        */
  stmm_truth_table_nodep truth_table_rec;/* Truth-table data                  */
  stmm_ids_list     fields;            /* Fields of record and unions.       */
  stmm_id           enum_type;         /* Enumerated Type of the Enum Value */
  int               ordinal;           /* Ordinal of an Enum Value           */
  stmm_gp_mode      param_mode;        /* In/Out/InOut for Subroutine-Data  */
} stmm_tx_node, *stmm_tx_nodep;
```

```
/* Subroutines */


typedef struct stmm_sb_node
{
  stmm_id           id;               /* Subroutine id                      */
  stmm_id_type      stmm_type;        /* always stmm_sb_type                */
  string            name;             /* Statemate name                     */
  string            code_gen_name;    /* filler                             */
  string            unique_name;      /* Statemate unique name              */
  stmm_list         attributes;       /* List of stmm_attribute             */
  string            short_des;        /* Short description                  */
  string            synonym;          /* Synonym                            */
  stmm_cbk_status   cbk_status;       /* filler                             */
  stmm_rpn_list     callback_binding; /* filler                             */
  genptr            user_data;        /* For user_data.                     */
  stmm_sb_type      type;             /* Function/Procedure/Task            */
  stmm_data_type  return_type;      /* Int/Real/Str/Bit/Bit-Array/User-Type*/
  stmm_id          user_defined_type;/* For the returned User-Type         */
  stmm_refered_by   ut_ref_by;       /* User-Def-Type refered by Name/Syn   */
  stmm_ids_list     parameters;      /*list of parameters (subroutine datas)*/
  stmm_list         global_variables; /* list of stmm_global_varp           */
  stmm_def_language default_language   /* Action-Lang/Procedural/C/Ada      */
  stmm_action_implp action_impl;      /* Action Language implementation     */
  stmm_user_codep   user_code;        /* Subroutines's Code in C/Ansi-C/Ada */
  stmm_graph_implp  graphical_impl;   /* Graphical Language implementation */
  stmm_truth_table_nodep truth_table_rec; /* Truth-table data               */
  stmm_action_implp truth_table_impl; /* Truth-table Language implementation
*/
  stmm_action_implp lookup_table_impl; /* Lookup-table Language
implementation */
} stmm_sb_node,  *stmm_sb_nodep;
```

```
/* Connector */

typedef struct stmm_cn_node
{
  stmm_id           id;                /* Connector's id                  */
  stmm_id_type      stmm_type;         /* di/co/ev/etc.                   */
  string            name;              /* filler                          */
  string            code_gen_name;     /* filler                          */
  string            unique_name;       /* filler                          */
  stmm_list         attributes;        /* filler                          */
  string            short_des;         /* filler                          */
  string            synonym;           /* filler                          */
  stmm_cbk_status   cbk_status;        /* filler                          */
  stmm_rpn_list     callback_binding;  /* filler                          */
  genptr            user_data;         /* For user_data.                  */
  stmm_cn_type      type;             /* History/Deep history/Termination */
  stmm_ids_list     in_cts;           /* Entering compound transitions    */
  stmm_ids_list     out_cts;          /* Exiting compound transitions     */
  stmm_id           parent_box;        /* Parent box (state)              */
} stmm_cn_node, *stmm_cn_nodep;

/* Activity */

typedef struct stmm_ac_node
{
  stmm_id          id;                  /* id of element                 */
  stmm_id_type     stmm_type;           /* di/co/ev/etc.                 */
  string           name;                /* Statemate name                */
  string           code_gen_name;       /* Name in generated code        */
  string           unique_name;         /* Statemate unique name         */
  stmm_list        attributes;          /* List of stmm_attributes       */
  string           short_des;           /* Short description             */
  string           synonym;             /* Synonym                       */
  stmm_cbk_status  cbk_status;          /* Enable/Disable Cbk bindings   */
  stmm_rpn_list    callback_binding;    /* Callbacks bindings            */
  genptr           user_data;           /* For user data.                */
```

```
  stmm_ac_type     type;                /* internal/reference/control/
                      data-store/external/environment */
  stmm_id          parent;               /* Id of parent.                  */
  stmm_id          definition_ch;        /* Chart id for instance activity */
  stmm_inst_type   instance_type;        /* type of box instance/regular   */
  stmm_list        instance_params;      /* List of stmm_inst_prm          */
  stmm_ac_term     termination_type;     /* Termination type               */
  stmm_act_impl    selected_impl;        /* Selected implementation        */
  stmm_rpn_list    mini_spec;            /* Activity mini-spec             */
  stmm_rpn_list    orig_mini_spec;       /* Original Activity mini-spec    */
  stmm_truth_table_nodep truth_table_rec; /* Truth-table data             */
  stmm_rpn_list    truth_table;          /* Activity truth-table           */
  stmm_ids_list    exiting_fl;           /* Flow-lines exiting activity    */
  stmm_ids_list    entering_fl;          /* Flow-lines entering activity   */
  stmm_rpn_list    combin_assigns;       /* Combinational assignments.     */
  stmm_ids_list    sons_list;            /* List of sons                   */
  stmm_ac_impl     implementation_type;  /* regular/procedure/task         */
  stmm_ucb_status  user_code_status;    /* UserCode Bindings Enable/Disable*/
  stmm_rpn_list    user_code_binding;    /* User Code Bindings             */
  string           library_name;         /* For instance of component      */
  string           component_name;       /* For instance of component      */
} stmm_ac_node, *stmm_ac_nodep;

/* State */

typedef struct stmm_st_node
{
  stmm_id         id;                /* id of element                     */
  stmm_id_type    stmm_type;         /* di/co/ev/etc.                     */
  string          name;              /* Statemate name                    */
  string          code_gen_name;     /* Name in generated code            */
  string          unique_name;       /* Statemate unique name             */
  stmm_list       attributes;       /* List of attributes (stmm_attribute) */
  string          short_des;         /* Short description                 */
  string          synonym;           /* Synonym                           */
  stmm_cbk_status cbk_status;        /* Enable/Disable Callbacks bindings */
```

```
  stmm_rpn_list   callback_binding;  /* Callbacks bindings                */
  genptr          user_data;         /* For user_data.                    */
  stmm_st_type    type;              /* And/Or/Reference                  */
  stmm_id         parent;            /* Id of parent.                     */
  stmm_id         definition_ch;     /* Chart id for instance activity    */
  stmm_inst_type  instance_type;     /* type of box instance/regular      */
  stmm_list       instance_params;   /* Instance parameters (stmm_inst_prm) */
  stmm_list       acs_in_state;     /* Within/Throughout activities in state*/
                                     /* list of stmm_ac_in_s              */
 stmm_rpn_list   static_reactions;  /* State's static reactions          */
 stmm_rpn_list   orig_static_reactions;
                                    /* State's original static reactions  */
  stmm_ids_list   exiting_tr;       /* Transitions exiting state         */
  stmm_ids_list   entering_tr;      /* Transitions entering state        */
  stmm_rpn_list   combin_assigns;    /* Combinational assignments.       */
  stmm_ids_list   sons_list;         /* List of sons                     */
} stmm_st_node, *stmm_st_nodep;


/* Chart */

typedef struct stmm_ch_node
{
  stmm_id          id;               /* id of element                    */
  stmm_id_type     stmm_type;        /* di/co/ev/etc.                    */
  string           name;             /* Statemate name                   */
  string           code_gen_name;    /* filler                           */
  string           unique_name;      /* filler                           */
  stmm_list       attributes;       /* List of attributes (stmm_attribute) */
  string           short_des;        /* Short description                */
  string           synonym;          /* filler                           */
  stmm_cbk_status  cbk_status;       /* filler                           */
  stmm_rpn_list    callback_binding; /* filler                           */
  genptr           user_data;        /* For user_data.                   */
  stmm_ch_type     type;        /* (Statechart/Activity-chart/
                        Module-chart/GDS/BD)                 */
```

```
  stmm_list        formal_prms;      /* Formal parameters for a generic
                                        list of stmm_formal_prm              */
  stmm_time_unitp  time_unit;       /* units and value of delayed time     */
} stmm_ch_node, *stmm_ch_nodep;

/* Compound */

typedef struct stmm_cm_node
{
  stmm_id          id;               /* id of element                      */
  stmm_id_type     stmm_type;        /* di/co/ev/etc.                      */
  string           name;             /* filler                             */
  string           code_gen_name;    /* filler                             */
  string           unique_name;      /* filler                             */
  stmm_list        attributes;       /* filler                             */
  string           short_des;        /* filler                             */
  string           synonym;          /* filler                             */
  stmm_cbk_status  cbk_status;       /* filler                             */
  stmm_rpn_list    callback_binding; /* filler                             */
  genptr           user_data;        /* For user_data.                     */
  stmm_cm_type     type;              /* (Transition/Data-flow/
                          Control/M-flow-line/Signal/Bus)    */
  stmm_id          lca;               /* Id of Least Common Ancestor of all
                                         sources and targets.              */
  stmm_cm_exp      *expression;      /* Flowing elements/Triggers-Actions */
  stmm_ids_list    sources;          /* Sources.                           */
  stmm_ids_list    targets;          /* Targets.                           */
  stmm_id          main_source;       /* The highest state that contains the
                                        source[s], but is lower than LCA.  */
  stmm_id          main_target;       /* The highest state that contains the
                                        target[s], but is lower than LCA.  */
  stmm_list        contained_list;   /* List of arrow id's contained in    */
                                     /* this compound                      */
  stmm_rpn_list    orig_rpn;         /* Original rpn                       */
}stmm_cm_node, *stmm_cm_nodep;
```

```
/* structure for context varaible */

typedef struct stmm_cv_node
{
  string           name;              /* Statemate name                    */
  stmm_id_type     stmm_type;         /* di/co                             */
  stmm_structure   data_structure;    /* Single/Array/Queue                */
  stmm_array_limits *array_limits;    /* Array limits in case of array     */
  stmm_data_type   data_type;         /* Int/Real/Str/Bit/Bit-Array        */
  stmm_dt_limits   *dt_limits;        /* Limits for Int/Bit-array/String   */
  stmm_id          user_defined_type; /* User-Def derived from assignments */
} stmm_cv_node, *stmm_cv_nodep;

/* structure for one item of the rpn (Reverse Polish Notation)
   of an expression                                              */

typedef struct stmm_rpn_node {
  stmm_rpn_node_type  rpn_node_type;  /* operator/id/constant/context-var  */
  stmm_id             id;             /* id when type is an identifier     */
  stmm_operator_type  operator_type;  /* operator when operator            */
  stmm_constant_val   val;            /* union, for constant value.        */
  genptr              user_data;      /* For user_data.                    */
  string              name;           /* name of field or context_variable */
  stmm_cv_nodep       cv_info;
} stmm_rpn_node, *stmm_rpn_nodep;

typedef struct stmm_panel_node {
  string     panel_name;
  string     display_name;
} stmm_panel_node, *stmm_panel_nodep;
```

```
typedef struct stmm_bind_node {
  stmm_id             id;
  string              name;
  stmm_gp_mode        mode;
  stmm_data_type      type;
} stmm_bind_node, *stmm_bind_nodep;

/* Type of Arrows. */
typedef enum {
  stmm_ar_timing_constraint,    /* SD Timing Constraint     */
  stmm_ar_order_insignificant,  /* SD Order Insignificant   */
  stmm_ar_message,              /* SD message               */
  stmm_ar_scenario              /* SD Scenario (partition)  */
} stmm_arrow_type;

typedef struct stmm_scen_node
{
  stmm_id         id;               /* id of element                        */
  stmm_id_type    stmm_type;        /* di/co/ev/etc.                        */
  string          name;             /* Statemate name                       */
  string          code_gen_name;    /* Name in generated code               */
  string          unique_name;      /* Statemate unique name                */
  stmm_list       attributes;       /* List of attributes (stmm_attribute)  */
  string          short_des;        /* Short description                    */
  string          synonym;          /* Synonym                              */
  stmm_cbk_status cbk_status;       /* Enable/Disable Callbacks bindings    */
  stmm_rpn_list   callback_binding; /* Callbacks bindings                   */
  genptr          user_data;        /* For user_data.                       */
  stmm_arrow_type    type;          /*                                      */

  stmm_ids_list     messages;       /* List of Messages in the Scenario     */
} stmm_scen_node, *stmm_scen_nodep;
```

```
typedef struct stmm_msg_node
{
  stmm_id         id;               /* id of element                          */
  stmm_id_type    stmm_type;        /* di/co/ev/etc.                          */
  string          name;             /* Statemate name                         */
  string          code_gen_name;    /* Name in generated code                 */
  string          unique_name;      /* Statemate unique name                  */
  stmm_list       attributes;       /* List of attributes (stmm_attribute)    */
  string          short_des;        /* Short description                      */
  string          synonym;          /* Synonym                                */
  stmm_cbk_status cbk_status;       /* Enable/Disable Callbacks bindings      */
  stmm_rpn_list   callback_binding; /* Callbacks bindings                     */
  genptr          user_data;        /* For user_data.                         */
  stmm_arrow_type   type;           /*                                        */

  stmm_id         from_lifeline;    /* Origin Lifeline                        */
  stmm_id         to_lifeline;      /* Target Lifeline                        */
  stmm_rpn_list   message_rpns;     /* Message rpn's                          */
  stmm_id         prev_element;     /* Previous Message or Referenced SD
within Scenario */
  stmm_id         next_element;     /* Next Message or Referenced SD within
Scenario     */
  stmm_id        my_scenario;    /* The Scenario which include the Message */
  stmm_id         my_ord_insig;     /* The Order-Insignificant which include
the Message */
  stmm_id         my_tc;            /* The Timing-Constraint which include
the Message   */
} stmm_msg_node, *stmm_msg_nodep;

typedef struct stmm_tc_node
{
  stmm_id         id;               /* id of element                          */
  stmm_id_type    stmm_type;        /* di/co/ev/etc.                          */
  string          name;             /* Statemate name                         */
  string          code_gen_name;    /* Name in generated code                 */
  string          unique_name;      /* Statemate unique name                  */
  stmm_list       attributes;       /* List of attributes (stmm_attribute)    */
  string          short_des;        /* Short description                      */
```

```
  string          synonym;           /* Synonym                              */
  stmm_cbk_status cbk_status;        /* Enable/Disable Callbacks bindings    */
  stmm_rpn_list   callback_binding;  /* Callbacks bindings                   */
  genptr          user_data;         /* For user_data.                       */
  stmm_arrow_type    type;           /*                                      */

  string          note;             /* Timing Constraint label              */
  stmm_id         from_msg;          /* From Messsage                        */
  stmm_id         to_msg;            /* To Message                           */
} stmm_tc_node, *stmm_tc_nodep;

typedef struct stmm_ord_insig_node
{
  stmm_id         id;                /* id of element                        */
  stmm_id_type    stmm_type;         /* di/co/ev/etc.                        */
  string          name;              /* Statemate name                       */
  string          code_gen_name;     /* Name in generated code               */
  string          unique_name;       /* Statemate unique name                */
  stmm_list       attributes;        /* List of attributes (stmm_attribute)  */
  string          short_des;         /* Short description                    */
  string          synonym;           /* Synonym                              */
  stmm_cbk_status cbk_status;        /* Enable/Disable Callbacks bindings    */
  stmm_rpn_list   callback_binding;  /* Callbacks bindings                   */
  genptr          user_data;         /* For user_data.                       */
  stmm_arrow_type    type;           /*                                      */

  stmm_list      msg_within;        /* messages in Order Insignificant      */
} stmm_ord_insig_node, *stmm_ord_insig_nodep;
```

```
typedef struct stmm_ref_sd_node
{
  stmm_id         id;               /* id of element                        */
  stmm_id_type    stmm_type;        /* di/co/ev/etc.                        */
  string          name;             /* Statemate name                       */
  string          code_gen_name;    /* Name in generated code               */
  string          unique_name;      /* Statemate unique name                */
  stmm_list       attributes;       /* List of attributes (stmm_attribute)  */
  string          short_des;        /* Short description                    */
  string          synonym;          /* Synonym                              */
  stmm_cbk_status cbk_status;       /* Enable/Disable Callbacks bindings    */
  stmm_rpn_list   callback_binding; /* Callbacks bindings                   */
  genptr          user_data;        /* For user_data.                       */
  stmm_ac_type    type;             /*                                      */

  stmm_id         prev_element;     /* Previous Messsage or Referenced SD   */
  stmm_id         next_element;     /* Next Message or Referenced SD        */
} stmm_ref_sd_node, *stmm_ref_sd_nodep;

typedef struct stmm_ll_node
{
  stmm_id         id;               /* id of element                        */
  stmm_id_type    stmm_type;        /* di/co/ev/etc.                        */
  string          name;             /* Statemate name                       */
  string          code_gen_name;    /* Name in generated code               */
  string          unique_name;      /* Statemate unique name                */
  stmm_list       attributes;       /* List of attributes (stmm_attribute)  */
  string          short_des;        /* Short description                    */
  string          synonym;          /* Synonym                              */
  stmm_cbk_status cbk_status;       /* Enable/Disable Callbacks bindings    */
  stmm_rpn_list   callback_binding; /* Callbacks bindings                   */
  genptr          user_data;        /* For user_data.                       */
  stmm_ac_type    type;             /*                                      */

  stmm_id         my_orig_id;       /* The original Lifeline id (rather than
                                       the Activity it was resolved to      */
```

```
} stmm_ll_node, *stmm_ll_nodep;

#ifdef DLL_LINK
stmm_el_nodep __stdcall FUNC_NAME(stmm_get_el)(stmm_id id);
#else
extern   stmm_el_nodep stmm_get_el
ARGS((stmm_id id));
#endif

#ifdef DLL_LINK
stmm_id_type __stdcall FUNC_NAME(stmm_get_id_type)(stmm_id id);
#else
extern  stmm_id_type stmm_get_id_type
ARGS((stmm_id id));
#endif

#ifdef DLL_LINK
stmm_id __stdcall FUNC_NAME(stmm_which_chart)(stmm_id id);
#else
extern  stmm_id stmm_which_chart
ARGS((stmm_id id));
#endif

#ifdef DLL_LINK
boolean __stdcall FUNC_NAME(stmm_open_session)(string project,string
workarea,
                        string profile,
                        boolean with_dataport);
#else
extern  boolean stmm_open_session
ARGS((string  project,
      string  workarea,
      string  profile,
      boolean with_dataport));
#endif
```

```
#ifdef DLL_LINK
boolean __stdcall FUNC_NAME(stmm_close_session)(boolean with_dataport);
#else
extern  boolean stmm_close_session
ARGS((boolean with_dataport));
#endif

#ifdef DLL_LINK
boolean __stdcall FUNC_NAME(stmm_build_reset_all_action)(void);
#else
extern   boolean stmm_build_reset_all_action
ARGS((void));
#endif

#ifdef DLL_LINK
stmm_list __stdcall FUNC_NAME(stmm_get_scope_modules)(void);
#else
extern   stmm_list stmm_get_scope_modules
ARGS((void));
#endif

#ifdef DLL_LINK
stmm_list __stdcall FUNC_NAME(stmm_get_procedural_modules)(void);
#else
extern   stmm_list stmm_get_procedural_modules
ARGS((void));
#endif

#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_di)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_di
ARGS((stmm_id scope_id));
#endif
```

```
#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_ev)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_ev
ARGS((stmm_id scope_id));
#endif

#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_co)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_co
ARGS((stmm_id scope_id));
#endif

#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_ut)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_ut
ARGS((stmm_id scope_id));
#endif

#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_an)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_an
ARGS((stmm_id scope_id));
#endif

#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_ac)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_ac
ARGS((stmm_id scope_id));
#endif
```

```
#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_md)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_md
ARGS((stmm_id scope_id));
#endif

#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_st)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_st
ARGS((stmm_id scope_id));
#endif

#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_ch)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_ch
ARGS((stmm_id scope_id));
#endif

#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_cn)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_cn
ARGS((stmm_id scope_id));
#endif

#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_if)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_if
ARGS((stmm_id scope_id));
#endif
```

```
#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_cm)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_cm
ARGS((stmm_id scope_id));
#endif

#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_sb)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_sb
ARGS((stmm_id scope_id));
#endif

#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_fd)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_fd
ARGS((stmm_id scope_id));
#endif

#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_en)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_en
ARGS((stmm_id scope_id));
#endif

#ifdef DLL_LINK
stmm_list __stdcall FUNC_NAME(stmm_get_panels)(void);
#else
extern   stmm_list         stmm_get_panels
ARGS((void));
#endif
```

```
#ifdef DLL_LINK
stmm_list __stdcall FUNC_NAME(stmm_get_bindings)(void);
#else
extern   stmm_list         stmm_get_bindings
ARGS((void));
#endif

#ifdef DLL_LINK
stmm_list __stdcall FUNC_NAME(stmm_get_all_cms)(void);
#else
extern   stmm_list         stmm_get_all_cms
ARGS((void));
#endif

#ifdef DLL_LINK
stmm_list __stdcall FUNC_NAME(stmm_get_all_cns)(void);
#else
extern   stmm_list         stmm_get_all_cns
ARGS((void));
#endif
/* List utilities */


/* Backward Competability */
#ifdef STM_LIST_OLD_INTERFACE
#define stmm_list_add_element   stmm_list_add_id_element
#define stmm_list_find_value    stmm_list_find_value_id
#define stmm_list_get_elm_value stmm_list_get_elm_value_id
#endif

#define stmm_for_every_element_in_list(l,p) \
for(p=stmm_list_first_element(l);p;p=stmm_list_next_element(l,p))
```

```
#ifdef DLL_LINK
stmm_list __stdcall FUNC_NAME(stmm_list_create)(void);
#else
extern  stmm_list stmm_list_create
ARGS((void));
#endif

#ifdef DLL_LINK
int __stdcall FUNC_NAME(stmm_list_length)(stmm_list  list);
#else
extern  int stmm_list_length
ARGS((stmm_list  list));
#endif

#ifdef DLL_LINK
boolean __stdcall FUNC_NAME(stmm_list_destroy)(stmm_list  list);
#else
extern  boolean stmm_list_destroy
ARGS((stmm_list  list));
#endif

#ifdef DLL_LINK
boolean __stdcall FUNC_NAME(stmm_list_purge)(stmm_list  list);
#else
extern  boolean stmm_list_purge
ARGS((stmm_list  list));
#endif

#ifdef DLL_LINK
stmm_list_elm __stdcall FUNC_NAME(stmm_list_first_element)(stmm_list  list);
#else
extern  stmm_list_elm stmm_list_first_element
ARGS((stmm_list  list));
#endif
```

```
#ifdef DLL_LINK
stmm_list_elm __stdcall FUNC_NAME(stmm_list_last_element)(stmm_list  list);
#else
extern  stmm_list_elm stmm_list_last_element
ARGS((stmm_list  list));
#endif

#ifdef DLL_LINK
stmm_list_elm __stdcall FUNC_NAME(stmm_list_next_element)(stmm_list list,
      stmm_list_elm  list_elm);
#else
extern  stmm_list_elm stmm_list_next_element
ARGS((stmm_list      list,
      stmm_list_elm  list_elm));
#endif

#ifdef DLL_LINK
stmm_list_elm __stdcall FUNC_NAME(stmm_list_previous_element)(stmm_list list,
      stmm_list_elm  list_elm);
#else
extern  stmm_list_elm  stmm_list_previous_element
ARGS((stmm_list      list,
      stmm_list_elm  list_elm));
#endif

#ifdef DLL_LINK
boolean __stdcall FUNC_NAME(stmm_list_add_id_element)(stmm_list list,stmm_id
val);
#else
extern  boolean stmm_list_add_id_element
ARGS((stmm_list  list,
      stmm_id    val));
#endif
```

```
#ifdef DLL_LINK
boolean __stdcall FUNC_NAME(stmm_list_add_ptr_element)(stmm_list list,genptr
val);
#else
extern  boolean stmm_list_add_ptr_element
ARGS((stmm_list  list,
      genptr    val));
#endif

#ifdef DLL_LINK
boolean __stdcall FUNC_NAME(stmm_list_delete_element)(stmm_list list,
      stmm_list_elm  list_elm);
#else
extern  boolean stmm_list_delete_element
ARGS((stmm_list  list,
      stmm_list_elm  list_elm));
#endif

#ifdef DLL_LINK
stmm_list_elm __stdcall FUNC_NAME(stmm_list_find_value_id)(stmm_list list,
                 stmm_id id_val);
#else
extern  stmm_list_elm stmm_list_find_value_id
ARGS((stmm_list  list,
      stmm_id id_val));
#endif

#ifdef DLL_LINK
stmm_list_elm __stdcall FUNC_NAME(stmm_list_find_value_ptr)(stmm_list list,
                 genptr ptr_val);
#else
extern  stmm_list_elm stmm_list_find_value_ptr
ARGS((stmm_list  list,
      genptr ptr_val));
#endif
```

```
#ifdef DLL_LINK
stmm_list __stdcall FUNC_NAME(stmm_list_copy)(stmm_list  list1);
#else
extern  stmm_list stmm_list_copy
ARGS((stmm_list  list1));
#endif


#ifdef DLL_LINK
stmm_id __stdcall FUNC_NAME(stmm_list_get_elm_value_id)(stmm_list_elm
list_elm);
#else
extern  stmm_id stmm_list_get_elm_value_id
ARGS((stmm_list_elm  list_elm));
#endif


#ifdef DLL_LINK
genptr __stdcall FUNC_NAME(stmm_list_get_elm_value_ptr)(stmm_list_elm
list_elm);
#else
extern  genptr stmm_list_get_elm_value_ptr
ARGS((stmm_list_elm  list_elm));
#endif


#ifdef DLL_LINK
stmm_list_elm __stdcall FUNC_NAME(stmm_list_get_ith_element)(stmm_list list,
                      int i);
#else
extern  stmm_list_elm stmm_list_get_ith_element
ARGS((stmm_list  list,
      int       i));
#endif


/* Api expressions utilities */
```

```
#ifdef DLL_LINK
stmm_list_elm __stdcall FUNC_NAME(stmm_rpn_skip_exp)
     (stmm_rpn_list  stmm_rpn_lst,
      stmm_list_elm  curr_list_elm);
#else
extern   stmm_list_elm stmm_rpn_skip_exp
ARGS((stmm_rpn_list  stmm_rpn_lst,
      stmm_list_elm  curr_list_elm));
#endif

#ifdef DLL_LINK
stmm_rpn_list __stdcall FUNC_NAME(stmm_rpn_skip_and_copy_exp)
     (stmm_rpn_list   rpn_list,
      stmm_list_elm   *curr_list_elm);
#else
extern  stmm_rpn_list  stmm_rpn_skip_and_copy_exp
ARGS((stmm_rpn_list   rpn_list,
      stmm_list_elm   *curr_list_elm));
#endif

#ifdef DLL_LINK
string __stdcall FUNC_NAME(stmm_rpn_to_string)(stmm_rpn_list   rpn_list);
#else
extern  string stmm_rpn_to_string
ARGS((stmm_rpn_list   rpn_list));
#endif

#ifdef DLL_LINK
int __stdcall FUNC_NAME(get_number_of_operands)
     (stmm_operator_type stmm_operator);
#else
extern  int get_number_of_operands
ARGS((stmm_operator_type  stmm_operator));
#endif
```

```
#ifdef DLL_LINK
stmm_list_elm __stdcall FUNC_NAME(get_ith_operand)
     (stmm_rpn_list  rpn_list,
      stmm_list_elm  curr_list_elm,
      int            i);
#else
extern  stmm_list_elm get_ith_operand
ARGS((stmm_rpn_list  rpn_list,
      stmm_list_elm  curr_list_elm,
      int           i));
#endif

#ifdef DLL_LINK
void __stdcall FUNC_NAME(stmm_get_used_affected)(stmm_list expressions,
                 stmm_list *used,
                 stmm_list *affected,
                 stmm_list *used_affected);
#else
extern void stmm_get_used_affected
ARGS((stmm_list expressions,
      stmm_list *used,
      stmm_list *affected,
      stmm_list *used_affected));
#endif

#ifdef DLL_LINK
void __stdcall FUNC_NAME(stmm_get_cm_exp_used_affected)(stmm_cm_exp *cm_exp,
                         stmm_list *used,
                stmm_list *affected,
                stmm_list *used_affected);
```

```
#else
extern void stmm_get_cm_exp_used_affected
ARGS((stmm_cm_exp *cm_exp,
      stmm_list *used,
      stmm_list *affected,
      stmm_list *used_affected));
#endif

#ifdef DLL_LINK
void __stdcall FUNC_NAME(stmm_get_activity_parameters)(stmm_id ac_id,
                        stmm_ids_list *inputs,
                        stmm_ids_list *outputs);
#else
extern  void stmm_get_activity_parameters
ARGS((stmm_id ac_id,
      stmm_ids_list *inputs,
      stmm_ids_list *outputs));
#endif

#ifdef DLL_LINK
void __stdcall FUNC_NAME(stmm_free)(void *pointer);
#else
extern  void stmm_free
ARGS((void *pointer));
#endif

#ifdef DLL_LINK
stmm_module_nodep __stdcall FUNC_NAME(stmm_get_module_info)(stmm_id
scope_id);
#else
extern stmm_module_nodep stmm_get_module_info
ARGS((stmm_id scope_id));
#endif
```

```
#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_scen)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_scen
ARGS((stmm_id scope_id));
#endif

#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_ref_sd)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_ref_sd
ARGS((stmm_id scope_id));
#endif

#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_ord_insig)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_ord_insig
ARGS((stmm_id scope_id));
#endif

#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_tc)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_tc
ARGS((stmm_id scope_id));
#endif

#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_msg)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_msg
ARGS((stmm_id scope_id));
#endif
```

```
#ifdef DLL_LINK
stmm_ids_list __stdcall FUNC_NAME(stmm_get_ll)(stmm_id scope_id);
#else
extern   stmm_ids_list     stmm_get_ll
ARGS((stmm_id scope_id));
#endif

#ifdef DLL_LINK
string __stdcall FUNC_NAME(stmm_get_scp_version_header)();
#else
extern   string    stmm_get_scp_version_header
ARGS(());
#endif

#ifdef DLL_LINK
void __stdcall FUNC_NAME(stmm_set_include_enum_init)(boolean set_val);
#else
extern   void     stmm_set_include_enum_init
ARGS((boolean set_val));
#endif

#ifdef DLL_LINK
void __stdcall FUNC_NAME(stmm_set_build_basic_arrows)(boolean set_val);
#else
extern   void     stmm_set_build_basic_arrows
ARGS((boolean set_val));
#endif

#ifdef DLL_LINK
void __stdcall FUNC_NAME(stmm_avoid_rpn_conversions)(boolean set_val);
#else
extern   void     stmm_avoid_rpn_conversions
ARGS((boolean set_val));
#endif
```

```
#ifdef DLL_LINK
void __stdcall FUNC_NAME(stmm_set_build_all)(boolean set_val);
#else
extern   void     stmm_set_build_all
ARGS((boolean set_val));
#endif


#ifdef DLL_LINK
void __stdcall FUNC_NAME(stmm_set_regard_in_sim_expressions)(boolean
set_val);
#else
extern   void     stmm_set_regard_in_sim_expressions
ARGS((boolean set_val));
#endif


#ifdef DLL_LINK
int __stdcall FUNC_NAME(stmm_get_short_id)(stmm_id id);
#else
extern   int     stmm_get_short_id
ARGS((stmm_id id));
#endif


#ifdef DLL_LINK
int __stdcall FUNC_NAME(stmm_get_chart_id)(stmm_id id);
#else
extern   int     stmm_get_chart_id
ARGS((stmm_id id));
#endif
```

Index

S

T

W