For further information, comments and suggestions please contact:

Stan BUDKOWSKI
INT - Institut National des Télécommunications
Software-Networks Department
9, rue Charles Fourier, 91011 EVRY, FRANCE
Tel: +33 (1) 60 76 47 20
Fax: +33 (1) 60 76 47 11
E_mail: stan@int-evry.fr

Note : The following persons participated in the Edb development:

The information in this document is subject to change without notice and should not be construed as a commitment by INT.

INT takes no responsibility for any errors or omissions that may exist in this document or for damages resulting from the use of the information herein.
# TABLE OF CONTENTS

1. INTRODUCTION ................................................................................. 5
   1.1 New functionalities................................................................. 6

2. Edb INSTALLATION AND INVOCATION ........................................ 7
   2.1 Edb invocation options......................................................... 8
   2.2 Files created during the simulation session ....................... 9

3. AN OVERALL VIEW AT THE SIMULATION SESSION ...................... 10

4. COMMAND LANGUAGE .................................................................. 12
   4.1 Access to objects................................................................. 12
   4.2 Commands and command composition .................................. 19
      4.2.1 Commands............................................................... 19
      4.2.2 Command composition.............................................. 20
   4.3 General-session commands............................................... 22
   4.4 Navigation commands....................................................... 24
   4.5 Display and file-display commands..................................... 24
      4.5.1 Display............................................................... 24
      4.5.2 File-display............................................................. 30
   4.6 Modify commands............................................................... 30
      4.6.1 General modify commands......................................... 30
      4.6.3 Timer modify_commands........................................... 32
   4.7 Observers and observer_commands....................................... 33
   4.8 Macro command and macro-call ......................................... 36
   4.9 Select commands............................................................... 37
      4.9.1 Select executable-systems command............................ 38
      4.9.2 Select executable-module command............................ 40
      4.9.3 Select executable-transitions command....................... 41
         4.9.3.1 Full mode......................................................... 41
         4.9.3.2 Simplified mode................................................. 43
   4.10 Special environment for performance evaluation study ............ 44
      4.10.1 Predefined functions............................................... 44
      4.10.2 Edb commands....................................................... 51

REFERENCES ....................................................................................... 53

APPENDIX 1: SYNTAX OF THE COMMAND LANGUAGE ....................... 55
   1. NOTATION AND REMARKS ................................................. 55
   2. GRAMMAR OF THE COMMAND LANGUAGE ......................... 55

APPENDIX 2: PREDEFINED FUNCTIONS ........................................ 63
   1. NOTATION.............................................................................. 63
   2. LIST OF PREDEFINED FUNCTIONS ...................................... 63

APPENDIX 3: ABBREVIATIONS ......................................................... 97
   1. ABBREVIATIONS OF Edb SIMPLE COMMANDS ..................... 97
   2. ABBREVIATIONS OF PREDEFINED FUNCTIONS ..................... 98
   3. MODIFIED PREDEFINED FUNCTIONS (wrt version 4.0)........... 100
   4. NEWLY DEFINED PREDEFINED FUNCTIONS ......................... 101

APPENDIX 4: SUMMARY OF PREDEFINED FUNCTION APPLICATIONS ... 103
this page is intentionally left empty
1. INTRODUCTION

Edb is the symbolic and interactive simulator/debugger for the ISO standardised formal description technique (FDT) named Estelle (ISO International Standard 9074).

Estelle is a formal description technique (language) for specifying distributed, concurrent information processing systems with a particular application in mind, namely that of communication protocols and services of the layers of Open Systems Interconnection (OSI) architecture defined by ISO.

Estelle can be briefly described as a technique that is based on an extended state transition model, i.e., a model of a nondeterministic communicating automaton extended by the addition of Pascal language. More precisely, Estelle may be viewed as a set of extensions to ISO standardised Pascal (ISO International Standard 7185), level 0, which models a specified system as a hierarchical structure of communicating automata which:

- may run in parallel, and
- may communicate by exchanging messages and by sharing, in a restricted way, some variables.

Edb is based directly on the Semantic Model defined in the ISO Estelle standard (ISO 9074) [1]. It executes the simulation algorithm described in [3] with some modifications in particular for interactive simulation. In accordance with the "transition atomicity" principle of the Estelle Semantic Model, the unit of execution in Edb is a simple transition. All sorts of nondeterminism inherent to an Estelle description are resolved during the simulation by a random selection if processed without any user intervention. Special commands are offered to control some of these nondeterministic choices in an interactive manner.

The purpose of Edb is to help the user in discovering and processing errors that occur during the execution of an Estelle specification. The user may control, observe and trace the execution of the specification by means of the simulation commands.

The inputs to the debugger are: the Intermediate Form and several C-files generated by the Estelle compiler [4]. The data representation in the debugger is defined in [5].

Edb has been developed by BULL S.A based on the functional specification [2,7,8] carried out within the Esprit SEDOS (No.410) project. Several extensions of Edb have been realised by the INT. They have been partially funded by the COPERNICUS COP#62 project and contracts with Centre National d'Etudes des Télécommunications (CNET) of France Telecom (performance evaluation extensions). Edb is written in C and runs on HP9000/700, SUN 4 and PC computers under corresponding UNIX systems (HP-UX, SunOs4.1.4 SOLARIS 2.5, and LINUX).

This document describes the Edb functionalities as included in the actual version. These functionalities extend and modify those realised in the previous versions (see Appendix 3 Section 3 for the list of modified predefined functions and Section 4 for newly defined predefined functions and Section 4.10 for the special Edb commands introduced to enable performance evaluation studies).

Edb offers the powerful means (the macro-commands, observers) to describe a simulation scenario (including the specification of anomalies to be detected) and it does not require of the user any particular knowledge of how the simulation is carried out. The user can fully concentrate on properties he wants to verify or detect in a direct reference to the simulated Estelle specification and the Estelle semantics.
There are three major and interrelated fields of applications of the Estelle simulator/debugger:

1. The analysis and validation of "global" properties of the simulated specification by means of its execution in a simulated environment. According to the Estelle semantic model, the unit of execution is a transition. This means that the user may observe the overall effects of executing transitions with respect to communication between the system components (module instances). In this sense the simulation concentrates on "global" system behaviour and provides the means to detect and eliminate "logical" design errors (e.g., deadlocks, undesired sequences of transitions, etc.).

2. The indication of run-time errors and warning situations while executing a transition or evaluating a transition's enabling condition (the errors or warning situations are caused, for instance, by an uninitialised variable, division by zero, lost output, etc.). This classical debugging facility complements that of (1) in that it detects "local", with respect to a transition, errors that can influence the global behaviour.

3. The simulation and debugging, in the sense of (1) and (2), of Estelle specifications extended with "time constraints" defined by the user, i.e., with explicit specification of the average speed of a module instance or a subsystem (e.g., average execution time of transitions). Those time constraints may reflect the known execution speeds of components of a real computer architecture in which a specification is to be implemented. That way implementation dependent problems related to performance may be studied in a simulated environment. The idea and details of that approach are given in [2].

By default there are no time-constraints (i.e., the execution time of each transition is assumed "infinitely small" or, as it is usual to say, equals zero). Thus, no execution speed is taken into account and all potential time relationships are represented. In this default case only relative values of "time-outs" (delays specified in transitions) are considered. This agrees with the intention of the Estelle semantics that treats time relationships as implementation dependent.

The document is organised as follows. Section 2 provides information on Edb installation and invocation. Section 3 gives an overview of a simulation session. Section 4 is an overview of the command language (see also Appendix 1). Appendix 1 gives the complete syntax of the command language. Appendix 2 gives the information on predefined functions. Appendix 3 gives abbreviations of Edb simple commands and of predefined functions. Appendix 4 gives the summary of application of predefined functions.

1.1 New functionalities

From Edb version (4.2) some new predefined functions have been added providing access to information concerning interactions inputed/outputed by the last fired transition and timers running/stopped, for use in expressions and for displaying. These function names are:

$\text{is\_enabled\_delay\_trans} \quad $\text{last\_trans\_output\_inter\_name}$
$\text{last\_trans\_input\_inter\_name} \quad $\text{last\_trans\_output\_inter\_param}$
$\text{last\_trans\_input\_inter\_param} \quad $\text{last\_trans\_output\_inter\_point}$
$\text{last\_trans\_input\_inter\_point} \quad $\text{last\_trans\_outputs\_number}$

The by default value of the function "$\text{transition\_weight}$" was changed from 0.0 to 1.0 (the 0.0 value eliminates transition from the fireable set).

Several Edb scripts are automatically executed directly after specification initialisation (see Section 3) to create initial simulation environment.
The simplified mode has been added when select command is executed (see Section 4.9.3.1).

Please note that invoking Edb from the X11-graphic interface XEDT tremendously augment the convivability of the user interface (see XEDT User reference Manual). It also permits to on-line generation of Message Sequence Charts (new functionality offered from version 4.3)

2. Edb INSTALLATION AND INVOCATION

Edb is a part of the ESTELLE DEVELOPMENT TOOLSET (EDT) package which includes also the Estelle Compiler (Translator and C code Generator), the documentation (table) generator and the Universal (Ug).

With respect to the previous version the user has to verify that he has the necessary environment to access to the Gnuplot and Ghostview tools (used to plot and view histograms and curves). This is necessary ONLY if the Edb tool is used WITHOUT the graphic interface (XEDT).

To make an Estelle specification ready for simulation, it is necessary:

- To have a complete specification. All occurrences of undefined elements (e.g., "...", "any TYPE" and "external" constructs) must be replaced by concrete types, constants and bodies, respectively (also see the Remark on the following page).

- To name the file containing the specification to be simulated with the suffix ".stl" (i.e., the name must be <source_file_name>.stl for any identifier "<source_file_name>"n1).

The Edb simulator/debugger can be directly invoked from the UNIX-shell environment by the single command:

    edb [options] <source_file_name>.stl

where 'options' is a sequence (empty included) of the elementary options. The user is informed by the system if a not admissible combination of elementary options were used. A detailed description of the elementary options is given below.

The edb command first invokes the Estelle Compiler (by default, with options -d, and -D, and optionally with the options additionally given within edb invocation, see Section 2.1 below). With the -d option the Compiler generates the C-code with extensions required by the Edb. The option -D enables "run-time error detection capability" during the simulation process.

If no errors are found during compilation (the Intermediate Form file -<source_file_name>.if and several C-source files are generated), then the C-compiler is automatically invoked to process all these C-files and to produce the <source_file_name>.o object file. This object file is then linked to the Edb simulation kernel to obtain a run-time image of "<source_file_name>" (<source_file_name>.edb) with which (and with <source_file_name>.if and <source_file_name>.stl) the simulation session may begin. The session automatically starts by executing the initialisation-part of the simulated specification (and of all its descendent modules).

When <source_file_name>.edb (and <source_file_name>.if and <source_file_name>.stl) file exists, then Edb invocation

**************************************************************************************************

1Please note that the "<source_file_name>" identifier is NO MORE limited to 10 characters. The "<source_file_name>," may correspond to the absolute path name.
edb [-n] [-T]<source_file_name>.stl

executes only the specification initialisation (the compilation phases are bypassed). Please note that in such a case the generator and translator options are not allowed. If such options are used the compilation phases will not be bypassed.

**Remark:** The file <source_file_name>.edb can also be generated in the following way. First the Ec compiler is invoked with the options -d, -D and -M (other options may be also included like -W). Second, the command make -f <source_file_name>.mk is to be issued ("<source_file_name>.mk is the file generated by Ec). Note that when the Estelle Compiler Ec is invoked with options M and d the generated makefile 'spec.mk' contains information on all the libraries necessary to generate the executable code for the Edb simulator/debugger. This way of generating the executable file <source_file_name>.edb is useful when some options of the C-generator not permitted when Edb is invoked must be used (e.g., -W, -u, -R) to process an uncomplete specification.

*It is strongly suggested to the user to invoke Edb with a specification, which already and successfully passed the compilation (translation). The reader can find in the Estelle-to-C Reference Manual how to invoke the Estelle Compiler with the Translator option.*

### 2.1 Edb invocation options

The options authorised when Edb is invoked are composed of a subset of translator options, a subset of generator options and some specific simulator/debugger options. The -e, -m, -c -N and -I<dir> translator options and the -q, -r, -w and -Ô generator options are only authorised (see Estelle-to-C User Reference Manual for details).

The following specific options can be used:

- **link option:**
  
  **-L<library>**  
  With this option the user may indicate the name of the library in which the bodies of procedures and functions, declared within the Estelle specification as "primitive", are stored. The -L option may be used several times (it is currently limited to 10 times)

- **simulator/debugger options:**

  **-n**  
  This option deals with the screen management to control scrolling: without the -n option Edb stops displaying after a complete screen was written (like the "pg" command of UNIX) waiting for the user to depress the <RETURN> key. The default length of the screen is 24 lines, but the user may set it differently through the LINES exported shell variable. With the -n option screen management is disabled. This is useful for running simulations with a predefined user scenario (automatic testing).

  **-y**  
  With this option the files spec_name.trstat_<nb>.ps, spec_name.qstat_<nb>.ps, spec_name.fscurve_<nb>.ps, spec_name.tmcurve_<nb>.ps will not be generated and corresponding statistics will be not plotted (using ghostview).

  **-T<file_name>**  
  With this option the file <file_name> is created. It contains the trace of all Edb commands issued by the user during the simulation session and the trace of executed transitions (in the form "exec"(instance-number, transition-number))
2.2 Files created during the simulation session

The following files and temporary directories are created:

- `<source_file_name>.edb` and `<source_file_name>.if` files mentioned above,

- the following files when during simulation session the corresponding commands `display $transition_statistics`, `display $queue_statistics`, `fscurve`, `tmcurve`, `throughput` or `resptime` have been executed:

  ```
  spec_name.trstat_<nb>.gr,
  spec_name.qstat_<nb>.gr,
  spec_name.fscurve_<nb>.gr,
  spec_name.fscurve<nb>,
  spec_name.tmcurve_<nb>.gr
  spec_name.tmcurve<nb>
  spec_name.throughput_<nb>,
  spec_name.throughput<nb>,
  spec_name.resptime_<nb>,
  spec_name.resptime<nb>,
  ```

- the following files only when Edb is NOT invoked with -y option:

  ```
  spec_name.trstat_<nb>.gr.epx,
  spec_name.qstat_<nb>.gr.ps,
  spec_name.fscurve_<nb>.gr.ps,
  spec_name.tmcurve_<nb>.gr.ps
  ```

- "<source_file_name>.cpp" file, when Edb was invoked with the -I option, which contains the Estelle text after the inclusion of all files defined within `<source_file_name>.stl` file by means of `#include "file_name"` directives,

- user named `<file_name>` when Edb was invoked with the -T option,

- "edbdir<PID>" temporary directory, which contains all C files generated by the C code generator (automatically removed when `<source_file_name>.edb` file is created).
3. AN OVERALL VIEW AT THE SIMULATION SESSION

A successful Edb invocation returns one of the following edb prompts:

```
edb>
edb-DEADLOCK>
```

The prompt without the "deadlock" message means that the specification initialisation (i.e., the execution of the initialisation-part of the specification) ended in a state from which the simulation may continue to execute transitions. If the "deadlock" message appears, then a deadlock situation has been detected. In any case, and not only after the initialisation, an edb prompt indicates that the simulation stopped (paused) and the user may issue commands.

Several simple commands (see Section 4.2) are offered to the user to display objects of the simulated specification or to control the simulation process. A small language over simple commands is also defined to allow the user to compose simple commands. The user may also define macro commands to prepare his simulation environment. A macro command may be defined either interactively or can be written in advance, stored in a file (so-called Edb simulation script) to be loaded any time during the simulation session and then used interactively. There is a simulation script furnished with the Edb simulator stored in a file $ESTEL/edb/.edbrc, which is automatically called (executed) directly after the specification initialisation. This simulation script defines several scratchpad variables and can not be modified by the user. The user may create his own simulation scripts $HOME/.edbrc and ./spec_name.rc which are also automatically executed (if they exist) directly after execution of $ESTEL/edb/.edbrc (the sequence of execution is the following: $ESTEL/edb/.edbrc, $HOME/edbrc (if exist) and ./spec_name.rc (if exist)). Please note that only ./spec_name.rc simulation script is in relation with the simulated specification spec_name.stl.

The user may conduct the simulation in many different ways. The simplest way is to simulate "step-by-step". One computation step in the simulation (one simulation step) corresponds to the execution of one transition. A transition to be executed may be chosen either randomly (by the simulation motor) among those fireable at a given moment or the user may display all fireable transitions and select one of them interactively. Instead of selecting a transition explicitly, the user also has the possibility of selecting only a subsystem or a module to which belongs a transition to be executed (see select commands in Section 4.10). The number of steps of uninterrupted simulation greater then 1 may be also chosen. In such a case the user may only select the first transition to be executed, the others will be chosen randomly. The user may also define, by means of the command language, the simulation script (simulation scenario) to be followed (see, for example the macro defined in Sec.4:9). The simulation scenario may also include so-called "observers" (see Section 4.8) that have the power of describing an overall simulation situation and act with respect to it. They can, for example "break" the simulation when a specified situation (e.g., an anomaly to be detected) has been attained.

The continue command is used to run the simulation for the defined number of steps (transitions). The user may, however, also set a time limit (in seconds) of uninterrupted simulation after which the simulation will stop even if the requested number of fired transitions has not been attained.

The simulation follows Estelle semantics exactly. One of the principles of this semantics is the atomicity of transitions. For this reason one computation step in the simulation (one simulation step) corresponds to the execution of one transition and the user cannot observe values during a transition execution. He can only interact in between two transition executions. An uninterrupted simulation continues to execute transitions one by one verifying after each step the user-defined simulation conditions (if any) and realising
corresponding, also user defined, actions (if any). The above conditions and actions are defined by the user within "observers".

The uninterrupted simulation continues unless a break command was executed within an "observer", a deadlock situation occurs or a run-time error in a transition execution (or in a transition's firing condition) is detected. When a run-time error in a transition's firing condition was detected this condition is assumed false. The simulation does not immediately stop. It halts after completing one simulation step. The error(s) is(are) signalised only when all other firing conditions (if any) are evaluated in a given situation. When a run-time error occurs in a transition's execution, the edb "jumps" to the end of the executed transition and gives control to the user with an appropriate error message being displayed.

The user may conduct the simulation with or without additional "time constraints". Their initial values can be specified by so-called "system-management-times" (min, max), and/or so-called "execution-times" (min, max). Their actual values change dynamically during simulation.

The system_management_time_min (respectively-max) for a subsystem may be interpreted as the minimal (respectively maximal) time during which the subsystem can still recognise the changes in its local context. These changes may be cased by the reception of interactions from other subsystems or by modifications introduced by the user during the simulation session by means of modify commands.

The execution-time_min (respectively-max) of a transition may be interpreted as the minimal (respectively-maximal) time during which the transition will terminate.

By default all these time-constraints interval (real) values are equal (0.0, 0.0) but the user, by means of modify commands, may change the min and max values, delimiting these intervals, to any real number. It means that in the default case the time does not progress when transitions are fired (it does not take time either to evaluate their fireness or to execute them). In general when the specification of the simulated system contains delayed transitions, the current min and max "time-out's" of those transitions that are enabled at a given moment are counted-down by the amount of time elapsed after a transition is fired. In the default case when there is no time-constraints the time is automatically moved to any value within the interval in which at least one of the enabled delayed transitions may fire and this value is subtracted from the current value of "time-out's of enabled delayed transitions.

The user may, however, issue at any time a command advance_time ("at" in short) to advance time by any real value. Observe that it is very useful when the user desires to force some delayed transition to fire even if there are some non delayed transitions ready to fire and all time-constraints interval values (min - max of the system_management_time and execution-time) are set to (0.0, 0.0).

Besides the transition execution time and system management time intervals the user may also use other predefined functions to dynamically annotate the simulated specification. In particular he may set a particular density distribution (uniform, exponential, geometric, Poisson) to generate adequate random values within the intervals. The user may also associate execution probabilities for nondeterministic transitions. All these new features permit the user to collect (and display in form of histograms or curves) some measures (statistics of executed transitions, statistics of queue contents, throughputs, response time etc.) during a simulation session. For more details please consult Section 4.10.

An on-line help command ("help", "h" or "?"") may be issued any time the user has control (i.e., when edb> prompt is displayed). This command provides the list of all Edb commands. The help command followed by a given command name provides information about the command. Some commands use as arguments so-called predefined functions. To get information about these functions, the command "d $pf" may be issued.
A UNIX interrupt signal "INTR" (usually caused by "CTRL C" or by the "break" key on the keyboard) stops the simulation after completion of the current simulation step. The control is returned to the user unless a specific action within an observer has been associated with the condition $\text{is\_interrupt}$.

A UNIX "quit" signal (frequently caused by "CTRL \") kills the Edb process. Its use is recommended when an infinite loop occurred within a transition execution. It is to be noted, however, that a UNIX "quit" signal will generate a core file, which may be of significant size. It is recommended to remove it from your file system.

4. COMMAND LANGUAGE

The syntactic notation used in command definitions is the following:

- " " and " " encloses a terminal symbol;
- [ ] encloses an optional item (which may occur zero or one time);
- { } encloses an item which may occur zero or more times;
- | denotes an "or" in the alternatives.

4.1. Access to objects

The simulator/debugger commands manipulate objects that are of different natures. Their characteristics and the way they are accessed are a basis to understand the commands. Typically, the access is either by an Estelle source name or, in case such a name does not exist, by a predefined function.

Two categories of objects may be distinguished:

- internal Edb objects, and;
- objects related to an Estelle specification.

An object of the second category can be accessed either by an Estelle source name or, in case such a name does not exist (e.g., a queue contents), by a predefined function. The current values of these objects can be displayed and modified/compared.

An object of the first category is always accessed by a predefined function call, except for user defined variables (scratch-pad variables). The values of these objects can be displayed and most of them modified/compared.

The user can access the following internal Edb objects:

(1) Module instance context:

A data structure called a module instance context. This data structure is associated with each module instance within the hierarchical tree structure of module instances existing at a given moment of the simulation session. A unique number refers to a module instance’s context. The numbers are assigned in order of instance creations (that way the instance of the specification module always has the number 1 assigned). Any module instance context of the hierarchical tree structure of instances is accessible by a
so-called "working pointer" pointing to a module instance of this structure. The working pointer is set by the Edb to the root of the tree (i.e., to the specification module identified by the number 1) every time the user gets control after executing a transition. The user may then use the navigation commands to modify the working pointer that is to place it at the required module instance context (to allow access to the instance's objects). The value of the working pointer itself (i.e., the unique number of the module instance context to which it is currently pointing) is accessed by the "$wp" predefined function call.

(2) Some "flags" and "counters":

-"firing_steps" value is accessed by the predefined function $firing_steps (or $fs, for short); the value determines the number of transitions to be fired without giving control to the user; the value is set to 1 (step-by-step simulation), when the user (re)starts the simulation session, and it can be modified by the user command (modify command); its last set value can be displayed (display command) and compared with another value any time the user gets control;

-"current-firing-step" counter value is accessed by the predefined function $current_firing_step ($cfs); the counter increments, modulo current value of "firing-step" flag ($fs), every time a transition has been fired; its value is 0, when the simulation session (re)starts or when $fs is modified, and the current value can be displayed (display command) or compared with another value any time the user gets control;

-"total-firing-step" counter value is accessed by the predefined function $total_firing_steps ($tfs); the counter increments by 1 every time a transition has been executed; its value is 0, when the simulation session (re)starts, and the current value can be displayed (display command) or compared with another value any time the user gets control;

-"real_time" counter value is accessed by the predefined function $real_time_limit ($stime). The value determines the time limit (in seconds) after which the simulation stops without any user intervention. Its value is "infinity", i.e. the simulation time is unlimited, when the simulation session (re)starts. Its last set value can be displayed (display command) or compared with another value any time the user gets control;

-"simulation_time" counter value is accessed by the predefined function $simulation_time_limit ($sint). The value determines the time limit (in same units as delay values transition execution and system management time) after which the simulation stops without any user intervention. Its value is "infinity", i.e. the simulation time is unlimited, when the simulation session (re)starts. Its last set value can be displayed (display command) or compared with another value any time the user gets control;

-"$total_simulation_time" counter value is accessed by the predefined function $total_simulation_time ($tsint); the counter increments:
  - by the amount corresponding to the dynamic execution time of a transition every time it has been executed,
  - by the amount corresponding to the dynamic system management time of a system module each time it selected a transition to be offered to execute,
- by the amount corresponding to the dynamic delay time of a delayed transition each time it has been executed when no other transition was fireable.

The value of the counter is 0, when the simulation session (re)starts, and the current value can be displayed (display command) or compared with another value any time the user gets control;

"run-time error" flag value is accessed by the predefined function $is_error. The value of the flag is initially false and it is automatically set to true when a run-time error occurs. The "continue" and "restart" commands reset the flag to false;

"deadlock" flag value is accessed by the predefined function $is_deadlock. The value of the flag is initially false and it is automatically set to true when a deadlock occurs (the deadlock situation is also signalised within edb prompt);

"break" flag value is accessed by the predefined function $is_break. The value of the flag is initially false and it is automatically set to true when a break command is executed within an observer. The "continue" and "restart" commands reset the flag to false;

"system-management-time-max" (real) value is accessed by the predefined function $sm_time_max ($smmax). The value determines the maximum time during which the given subsystem can still recognise the changes in its local context due to the reception of interactions from other subsystems. The value is 0.0, when the simulation session (re)starts, and the current value can be displayed (display command) and modified by the user command (modify command);

"system-management-time-min" (real) value is accessed by the predefined function $sm_time_min ($smmin). The value determines the minimum time during which the given subsystem can still recognise the changes in its local context due to the reception of interactions from other subsystems. The value is 0.0, when the simulation session (re)starts, and the current value can be displayed (display command) and modified by the user command (modify command);

"execution-time-max" (real) value is accessed by the predefined function $exec_time_max ($etmax). The value determines the maximum execution time of transitions of the given module instance. The value is 0.0, when the simulation session (re)starts, and the current value can be displayed (display command) and modified by the user command (modify command);

"execution-time-min" (real) value is accessed by the predefined function $exec_time_min ($etmin). The value determines the minimum execution time of the identified transition within the given module instance. The value is 0.0, when the simulation session (re)starts, and the current value can be displayed (display command) and modified by the user command (modify command).

(3) The "seed" of the random number generator

The "seed" value (any integer greater than 0) of the random number generator within Edb is accessed by the predefined function $random ($rd). Its initial value is 1987 and its current value can be modified by an assignment (modify
command) or it can be displayed \((display\ \text{command})\). When the simulation session is restarted \((restart\ \text{command})\) the last seed value is kept.

(4) The scratch pad variables

The user may define some special variables \((\text{scratch pad variables})\) during the simulation session for his own use. The number of scratch pad variables the user can create is limited to 200 (but can be augmented on user request). The scratch pad variables can be created and initialised/modified by assignments \((\text{modify}\ \text{command})\). When the simulation session is restarted \((\text{restart}\ \text{command})\) their last values are kept. They disappear when the simulation session ends \((\text{command}\ \text{quit})\). The values of these variables are accessed by their identifiers which always begin with "#" and whose length is limited to 25 characters. Their type is determined by the first assignment in which they appear on the left-hand-side. Only integer, real, boolean or character variables may be defined in the scratch pad. The following predefined functions may be assigned to the scratch pad variables:

\[
\begin{align*}
$\text{down}$& \\
$\text{up}$& \\
$\text{left}$& \\
$\text{right}$& \\
$\text{working\_pointer}\,(\text{wp})$& \\
$\text{last\_instance\_nb}\,(\text{#lins})$& \\
$\text{highest\_instance\_nb}\,(\text{#hin})$& \\
$\text{last\_transition\_id}\,(\text{#ltrid})$& \\
$\text{least\_fired\_transition}\,(\text{#lfr})$& \\
$\text{most\_fired\_transition}\,(\text{#mfr})$& \\
$\text{highest\_trans\_nb}\,(\text{#htn})$& \\
\end{align*}
\]

For example:

\[
\text{#myown} := \text{wp}
\]

signifies that the \text{#myown} variable is a reference to a module instance context currently pointed to by the working pointer ("wp").

\[
\text{#nb} := \text{true}
\]

signifies that the \text{#nb} variable is created (or modified) and its value is equal "true"

The arithmetic operation "+", ",-", "*" and "/" can be executed on scratch pad variables of the real or integer type. For the first three operations both arguments should be either of real or integer type. For "/" operation the type of arguments may be mixed.

For example:

\[
\begin{align*}
\text{#myown} & := 5.0 \\
\text{#yours} & := 3.0 \\
\text{#res} & := \text{#myown} \ast \text{#yours} \\
\text{display}\ \text{#res} & \\
\text{#res} & = 15.0
\end{align*}
\]
(5) The name of the file containing the currently simulated specification name

The name of the file containing the currently simulated specification is accessed by the predefined function $spec_name (or $spec).

The following objects related to the Estelle specification are accessible to the user:

(1) The values of some named objects coming from the Estelle specification. The values of these objects are accessible by the Estelle specification source name relative to a given module instance context. The values that are directly accessible to the user by source Estelle names, for a given module instance, are:

- the values of the parameters of the module instance;
- the values of the exported variables of the module instance;
- the values of the variables of the module instance (i.e., of Pascal variables of the "var" section of the module declaration-part); note that the local variables of transitions and those of procedures and functions are not accessible because control may not be given to user during the execution of a transition;
- the values of the module-variables of the module instance (i.e., of the variables of the "modvar" section of the module declaration-part). A value of a module variable (if defined) is the reference to the children module instance to which the variable is currently assigned;
- information about given internal and external interaction points of the module instance.

(2) The values of some objects coming from the Estelle specification that do not have a name. The values of these objects are accessible by the predefined functions relative to a given module instance context. The values that are accessible to the user are (the list below is not exhaustive):

- the name of the current control state of the given module instance is accessed by the predefined function $state ($st);
- information concerning all the interaction points of the given module instance is accessed by the predefined function $interaction_point ($ip);
- information concerning the content and the size of the queue(s) associated with the interaction point(s) of the given module instance are accessed by the predefined functions $queue ($q), $queue_interaction ($qi), $queue_size ($qs).
- information concerning the subtree hierarchy of the module instances whose root is currently pointed to by $wp is accessed by the predefined function $hierarchy ($h);
- information concerning subsystems currently able to move (to terminate a transition that is "in execution" or to choose a new,
nonempty, set of transitions to be executed) is accessed by the predefined function call $executable systems ($es);

- information concerning modules currently able to move (to terminate a transition that is "in execution" or to start a new transition to be executed) is accessed by the predefined function call $executable modules ($em);

- information concerning transitions currently able to terminate or to start is accessed by the predefined function call $executable transitions ($et).

To summarise, an access to an object in the edb commands is defined as follows (this is a simplified version; the complete syntax of the command language is given in Appendix 1):

\[
access = \begin{cases} 
  sp-access, \\
  ip_access, \\
  modvar-access, \\
  var-access, \\
  implicit-access 
\end{cases}
\]

sp-access = "#"scratch-pad-id

ip_access = \[
\begin{array}{l}
  \{instance_access \"->\"\} ip-id \\
  \{instance_access \"->\"\} ip-id[\"index\",\"index\"] \\
\end{array}
\]

modvar-access = \[
\begin{array}{l}
  \{ instance_access \"->\"\} modvar-id \\
  \{instance_access \"->\"\} modvar-id[\"index\",\"index\"] \\
\end{array}
\]

var-access = \[
\begin{array}{l}
  \{ instance_access \"->\"\} pascal-var-access \\
\end{array}
\]

implicit-access = function-call

instance_access = unsigned-integer \ sp-access

trans-access = \[
\begin{array}{l}
  \{instance_access \"->\"\} trans-number \\
\end{array}
\]

index = unsigned-constant \ sp-access

function-call = "$"function-id \ (\"parameter-list\")\"

parameter-list = parameter \ (\"parameter-list\")

parameter = access \ instance_access \ trans-access

For scratch-pad variables and function calls the "instance_access" is not necessary (hence omitted), and for all other cases either it is explicitly specified or, if it is omitted, it is assumed to be that of "$wp$".

Note that "pascal-var-access" means an access, in the sense of Pascal (with some limitations, see Appendix 1), to a variable, module parameter or exported variable within the scope of the module whose instance is explicitly indicated or it is that given by "$wp$".
Note that transitions within Edb are referenced (trans-access) by their associated numbers (and optionally by their names). The transitions are consecutively numbered within each module body definition. The first (textual order) defined transition has the number 0, the second - number 1 etc.

The situation is more complicated when an any-clause, stateset or statelist (the last two within from_clause) are used in a transition definition. The following example illustrates the any-clause case.

```
trans
  from S1 to S2
  any n: 1..2; k: 3..4 do
    when p[n].m
      name shorthand_any:
      begin
        variable := k {transitions numbered 0 (n=1, k=3),
                        1 (n=2, k=3),
                        2 (n=1, k=4) and 3 (n=2, k=4)}
      end;
```

since it is a shorthand for the following nested transition (composed of 4 elementary transitions):

```
trans
  from S1 to S2
  when p[1].m
    begin variable := 3 {transition numbered 0 (n=1, k=3)} end;

  when p[2].m
    begin variable := 3 {transition numbered 1 (n=2, k=3)} end;

  when p[1].m
    begin variable := 4 {transition numbered 2 (n=1, k=4)} end;

  when p[2].m
    begin variable := 4 {transition numbered 3 (n=2, k=4)} end;
```

In case of the stateset or statelist used within from_clause in the definition of a transition there are in fact as many transitions as there are states defined within the stateset or statelist and they are numbered following the order of the occurrence of a given state within the stateset or stateslist. For example:

```
trans
  from open wait
to same
  begin
  end;
```

is shorthand for the following nested transition (composed of 2 elementary transitions):
trans

from open
to same
begin
end;

from wait
to same
begin
end;

consecutively numbered (n et n+1)

4.2. Commands and command composition

4.2.1 Commands

The Edb commands are grouped in categories of commands as follows:

- General-session commands;
- Navigation commands;
- Display commands;
- Modify commands;
- Observer commands;
- Macro command and macro call;
- Collect commands;
- Select commands.

(There are also two commands "exit" and "break". The former used within a "do-loop" (see below), the latter within an observer action (see Sec. 4.8)).

The above command categories are summarised before we define each command separately.

The **general-session commands** include: the *help* commands that give the user the syntax and function of simple commands, the *restart* command, which serves to reinitialise the simulation, the *quit* command to end the simulation session, and the UNIX command to allow the user to suspend the simulation, use Unix commands, and then return to the simulation at the point of suspension. There are also commands globally limiting the duration of a simulation or the number of execution steps of simulation.

**Navigation commands** provide the means to move along the tree of module instances. They modify the working pointer ($wp$).

**Display commands** provide the means of displaying a tree of module instances, variables of module instances, user (scratch-pad) variables, a part of source Estelle text, last executed transition and its outputs, lists of transitions, modules or systems that may execute next, etc. The displayed information may also be stored in a file; i.e., the information is "displayed to a file" and not on the screen (file-display commands).

**Modify commands** may be used to modify only Pascal variables of a given module instance, control states, parameters of an interaction in a FIFO queue, user (scratch-pad) variables, and a few other simulation parameters.
Observer commands allow the insertion (and removal) of an "observer" which serves to describe a global simulation context the user wants to observe or detect and an associated action to be taken. Observers that use display and "file-display" commands in their actions, constitute a powerful means for tracing, during simulation, the evolution of chosen specification objects. A small language over simple commands described below is used to define the observers' actions. An observer action can, however, use only a restricted set of simple commands; see Sec.4.8.

A macro command serves to declare a sequence of commands as a macro that can then be called for execution either directly by the user from the terminal or in an observer's action.

Collect commands serves to collect data to predict the performance of the implementation generated from the simulated Estelle specification.

Select commands offer the user the possibility of interactively selecting and controlling the future sequence of execution steps with various degrees of precision (selecting a particular system, module or transition to execute next).

4.2.2 Command composition

A small language over simple commands described below is used either interactively or it serves to define an automatically executed simulation scenario. The language is also used to define the observers' actions. The collected syntax of the command language can be found in Appendix 1.

\[
\text{simple-command} = \text{advance_time} \mid \text{break} \mid \text{chain_observer} \mid \text{change_wp} \mid \text{continue} \mid \text{delete_from_queue} \mid \text{delete_observer} \mid \text{disable_observer} \mid \text{display} \mid \text{display_text} \mid \text{enable_observer} \mid \text{exit} \mid \text{file_display} \mid \text{help} \mid \text{insert_in_queue} \mid \text{macro} \mid \text{print} \mid \text{quit} \mid \text{restart} \mid \text{select} \mid \text{set_observer} \mid \text{unix} \\
:= \{\text{See modify commands (Sec. 4.6.1)}\} \mid \text{fscurve} \mid \text{tmcurve} \mid \text{throughput} \mid \text{resptime}
\]

Semantics: The semantics of each simple command is given in the following sections. It should be stressed that execution of a simple command is considered terminated even if a run-time error in the command specification (e.g., the command uses non-existing names)
was detected. The error is signalised but it does not prevent the following command from being executed. In the automatic mode of execution it means that these errors are ignored.

The "exit" command is to be used within a "do-command". Used outside of a "do-command" it has no effect.

The "break" command is to be used within an observer action. Used outside of such an action it has no effect.

\[
\text{command} = \begin{cases} 
\text{simple-command} \\
\text{if-command} \\
\text{do-command}
\end{cases}
\]

Commands may be composed into sequences.

\[
\text{command-sequence} = \text{command} \left[ ";" \text{command-sequence} \right]
\]

**Semantics:** A command sequence is executed one by one from left to right. A command sequence terminates **only if** all simple commands of the sequence were executed.

\[
\text{if-command} = "if" \text{ boolean-expression} \\
\quad "\text{then}" \text{ command-sequence} \\
\quad \left[ "\text{else}" \text{ command-sequence} \right] \\
\quad "fi"
\]

**Semantics:** Boolean expressions must evaluate to a boolean value. The command executes in the usual way. Its execution terminates when the command-sequence that follows the "then" or the "else" part terminated or when boolean expression does not evaluate to a boolean value.

\[
\text{do-command} = "do" \text{ command-sequence } "od"
\]

**Semantics:** The command repeats execution of the command-sequence until an "exit" command is reached which terminates the whole do-command.

**Remark1:** The single line comments starting with the % character and ending with the end of line may be used. The % character may be put also just after the \\ (anti-slash) character.

**Remark2:** Each command (or a sequence of commands) issued by the user has to be confirmed by "return" to be taken into account. To continue writing a command (or a sequence of commands) on the following line, the anti-slash character should precede the "return" \\ character. Just after the \\ (anti-slash) character and before "return" a comment starting with % character may be written. **No space or tab is allowed between \\ (anti-slash) character and "return".**

4.3. General-session commands
"help" [command-name]
"h" [command-name] | "?" [command-name]

The "help" command gives the user the list of the edb commands if the command-name is not specified and it gives the syntax of the command and its function, otherwise.

............... "$real_time_limit" ":=" expression

("expression" must evaluate to a positive integer).

The command sets the real times limit in seconds (it has nothing to do with the time e.g., of simulated delays) after which the simulation stops without any user intervention. The execution may stop earlier when a "break" command or a deadlock is achieved, a run-time error occurs, or another limit (see "$fs" below) has been attained. By default the value is assumed "infinity", i.e., the simulation goes, in principle, forever. The timer is consulted in each simulation step. If the value is changed by the user the previous timer is cancelled and the command "continue" restarts the simulation with the new time limit. If the limit has been achieved and is not changed the command "continue" restarts the simulation with the same limit and the timer is as previously cancelled. If the simulation stopped earlier for one of the reasons mentioned above, then the command "continue" restarts it with the same limit and it does not cancel the counter (see the interpretation of the command "continue" below). Thus the command gives the user the possibility to interact with the debugger in intervals measured in seconds, e.g., every 5 seconds.

............... "$firing_steps" ":=" expression | "$fs" ":=" expression

("expression" must evaluate to a positive integer).

The command sets the interval of uninterrupted simulation in number of executed transitions. By default the value is assumed 1, i.e., the simulation will stop after each transition execution (step by step simulation). An associated "current-firing-step" counter ("$cfs") sums up the transitions executed without interruption to stop the simulation when the counter achieves the assigned value. If this occurs, the counter is reset to 0 and the command "continue" repeats the same interval (see the interpretation of the command "continue" below). If the value of "$fs" is changed by the user, the command "continue" restarts the simulation with the new limit and the counter "$cfs" is reset to 0. If the simulation stopped earlier for some other reason (see "$time" above), then the command "continue" restarts it with the same limit and it does not cancel the counter (see the interpretation of the command "continue" below). Thus the command gives the user the possibility to interact with the debugger in intervals measured in number of executed transitions, e.g., if the number is 10, then the interaction is every 10 fired transitions. If no such limit is desired the value can be assigned "infinity" ("$infinity").

Remark: Since the values of the two simulation intervals above may be changed (by an assignment), these commands also belong to the modify category of commands (see Section 4.6).

............... "continue" | "c"

The command "continue" (re)starts the simulation from the point it had been stopped for the number of steps determined by the actual value of $fs or $time. In other words the uninterrupted simulation will continue until the simulation achieved one of the two limits
described above ("$time" or "$fs"). The simulation may stop, however, before one of the limits has been achieved. It may happen because the command "break" in an observer (see Section 4.8) has been executed, a run-time error has occurred, or the simulation has been deadlocked.

When the simulation stops it is indicated which one of the three situations occurred (break, error or deadlock). In the interactive mode of simulation it is up to the user how to treat such a termination. To treat properly the situations in an automatic mode, the following three predefined testing functions (for predefined functions see Appendix 2) may be used:

"$is_break"
"$is_error"
"$is_deadlock"

Remark: The command continue resets the working pointer to 1, and the values of the above testing functions, to false.

"restart" | "rs"

The "restart" command reinitialises the Edb environment as when beginning a new session but without quitting the session. The values which are not reinitialised are: the "seed" of the random number generator (in this way the simulator may choose another execution path when restarted), the scratch-pad variables, and statistics of the number of fired transitions ($transition_statistics or $global_trans_statistics ($gtrstat). The defined "macro's" and "observers" continue to exist.

"quit" | "q"

The "quit" command causes the "end" of the Edb simulation session.

"unix" [shell_command]
"!" [shell_command]

The "unix" command gives the user the access to the Unix shell. When the shell completes the user command(s), the command "CTRL-D" or "exit" gives over control again to the simulator/debugger and restores its screen. The shell_command must be delimited by two double-quotes.

4.4. Navigation commands

"change_wp" [ argument ] | "cwp" [ argument ]

argument = instance-number | sp-access | implicit-access

Remark: An "argument" in the form of "implicit-access" is restricted to the following predefined functions:"$up"("$u"), "$down" ("$d"), "$left" ("$l"), "$right" ("$r"), $last_inst_nb ($lins) and $highest_inst_nb ($hin).

When "change_wp" is without argument, the working pointer ("$wp") is reset to 1, i.e., to the specification module instance context.
When "change_wp" has an "instance-number" as "argument", the working pointer ("$wp") is set to the given module instance context; if the value does not correspond to an existing module instance, then an error message is issued.

When "change_wp" has a scratch-pad variable reference as "argument", the working pointer ("$wp") is set to the module instance context whose number equals the value of this variable; if the value does not correspond to an existing module instance, then an error message is issued.

When "change_wp" has one of the following predefined functions: "$up", "$down", "$left" and "$right" ("$u", "$d", "$l" and "$r") as an argument, the working pointer ("$wp") is moved from its current position (in the tree of module instances contexts) to:

- its parent module instance context ("cwp $up");
- its first child module instance context ("cwp $down");
- its right sibling module instance context ("cwp $right");
- its left sibling module instance context ("cwp $left").

Note that the sibling instances are dealt with in a circular manner (hence, the right and left sibling instances of an instance always exist).

In the first two cases ("$up" and "$down") an error message is issued if the parent or the children instances do not exist, respectively.

### 4.5. Display and file-display commands

#### 4.5.1. Display

The command

```
display_text [ line ]["," [line ]]
dtx[ line ]["," [line ]]
```

\[
\text{line} = \text{unsigned-integer}
\]

displays the text of the simulated Estelle specification (source_file_name.stl file) between given line-numbers. If the first line-number is omitted, the beginning of the file is taken as the first argument. If the second line-number is omitted, the end of the file is taken as the second argument. If the command is issued without arguments, the Edb gives control to the "vi" or "emacs" editor with the **source_file_name.stl** file. The position of the cursor is the first character of the text, unless the "dtx" command is entered just after a run-time error was detected or after the command display $last_transition_id (d $ltrid). In those cases the cursor of the used editor will point to the concerned part of the Estelle specification text. The editor invoked ("vi" or "emacs") depends on the value of the EDITOR user environment variable (see Section 3 in the 'General Information' manual). By default (when the EDITOR variable is not defined) the "vi" editor is invoked.

**************

The command
"display" access | "d" access

displays information concerning the object(s) defined by "access" (see Sec.4.1. for the "access" definition).

The display command may also use as its argument a character string (delimited by two double-quotes).

\[ display \langle \text{character\_string} \rangle \]

Such display command is equivalent to the \[ print \langle \text{character\_string} \rangle \] command.

**Remark.** It is strongly suggested not to use the \[ print \] command since it will be removed from the set of admissible Edb commands in the next version of Edb.

The objects may be accessed either by their names (an Estelle source name or scratch pad variable name) or, in case such a name does not exist, by a predefined function.

Consider first named objects. The display commands have, in such a case, the following form:

\[ "display" \ [ instance\_access \ "->" ] object-name \]
\[ "d" \ [ instance\_access \ "->" ] object-name \]

-If the accessed object is an interaction point, then the information about its current attachment and/or connection (if any) is displayed. For example, the command \( d \, 3 \, -> \, P \) serves to display information about the interaction point named \( P \) (declared within a module-header of the simulated specification) of the module instance numbered \( 3 \):

-If the accessed object is a variable (module parameter, exported variable, module variable, and scratch pad variable) then its current value is displayed.

In the two cases above, if "instance_access" is omitted, then it is assumed to be the module instance identified by the current value of the "working- pointer" ("$wp").

************

Consider now objects whose names do not exist within an Estelle specification, e.g., queue. Interesting information about these objects is accessible by means of predefined functions. The list of predefined functions and their descriptions is given in Appendix 2 (note that not all of them are admissible for the display commands - see Appendix 4). Below are some typical examples of their use.

The command

\[ "display" \ "$queue" \ ["("instance\_access")"] \ |
\[ "d" \ "$queue"\ ["("instance\_access")"] \]

where \( instance\_access = \text{unsigned-integer} \mid \"#\"sp\text{-id} \)

displays information concerning the current contents of the queues associated with all interaction points of the module instance identified by "instance_access". If "instance_access" is omitted, then it is assumed to be the module instance identified by the current value of the "working- pointer" ("$wp").
The command

```
"display" "$queue"("[ instance_access " ->"] ip-id ")"
"d" "$queue"("[ instance_access " ->"] ip-id ")"
```

where "ip-id" is a name of an interaction point

displays information concerning the current contents of the queue associated with the interaction point identified by "ip-id". If "instance_access" is omitted, then it is assumed to be the module instance identified by the current value of the "working-pointer" ("$wp").

The command

```
"display" "$queue_interaction"("[instance_access"->"]ip-id"," queue-position")"
"d" "$qi"("[instance_access"->"]ip-id"," queue-position")"
```

displays the name of the interaction on a given queue-position of the queue associated with the interaction point defined by "ip-id". If "instance_access" (within "ip_access") is omitted, then it is assumed to be the module instance identified by the current value of the "working-pointer" ("$wp"). For example, the command `display $queue_interaction (4->ip-id, 3)` displays the name of the interaction in the 3rd position of the queue associated with the interaction point defined by "ip-id" within the 4th instance.

**Remark:** There are other queue-related predefined functions (see Appendix 2).

The command

```
"display" "$interaction_point"["("instance_access")"]
"d" "$ip"["("instance_access ")]"
```

displays information concerning all the interaction points of the instance identified by "instance-number". If "instance-number " is omitted the module instance is that given by "$wp". Please note that the information about a given interaction point can be displayed by means of display [ "("instance_access")" " -" ] "ip-id" command.

**Remark:** There are other interaction points related predefined functions (see Appendix 2).

The command

```
"display" "$state" [ "("instance_access ")]"
"d" "$st" [ "("instance_access ")]"
```

displays the current control (major) state of the module instance identified by "instance-number". If "instance-number" is omitted the module instance is that given by "$wp". For example, the command `display $state(8)` displays the current control state of the module instance whose number is 8.

The command

```
"display" "$hierarchy" [ "("instance_access")"]
"d" "$h" [ "("instance_access")"]
```

displays the current hierarchical tree structure whose root is the module instance identified by "instance_access" (or given by "$wp" if "instance_access" is omitted).

The command

```
"display" "$predefined_functions" |
"d" "$pf"
```

displays the list of all predefined functions.

Another interesting group of display commands is that which provides information concerning the last transition fired and the Estelle statements issued by the last transition fired (outputs, initialisations, releases, attachments, detachments, connections, and disconnections). If they are applied within an "observer" (see Section 4.8), they enable tracing (on the screen or in a file) information concerning last transition fired. Below there are some examples of the display commands. The description of all predefined functions used in commands of that group may be found in Appendix 2.

The command

```
"display" "$last_transition_id" |
"d" "$ltrid"
```

displays information concerning the identification of the last fired transition (transition-number and transition-id, if such exists).

The command

```
"display" "$last_instance_nb" |
"d" "$lins"
```

displays information concerning the identification of the module (module-header-id, module-body-id) whose instance fired the transition).

The command

```
"display" "$last_transition_input" |
"d" "$ltri"
```

displays information concerning the identification of the last fired transition input (the identification of the interaction point and interaction identification corresponding to the "when" clause (if it exists) of the transition). To access separately different type of information concerning he last fired transition input some other function may be used, namely: $last_trans_input_inter_point, $last_trans_input_inter_name and $last_trans_input_inter_param(INTER_PARAMETER)

The command

```
"display" "$last_transition_state" |
"d" "$ltrst"
```
displays information concerning the identification of the last fired transition states (the control states -if exist- corresponding to the "from" and "to" clauses of the transition).

The command

```
"display" "$last_transition_outputs" |
"d" "$ltro"
```

displays information concerning the outputs of the last fired transition. The information displayed consists, for each executed output-statement, of:

- the identifier of the interaction and of the interaction point through which this interaction was sent; if the interaction has parameters their values are also displayed;

- the identification of the instance (instance-number, module-header-id, module-body-id) and of its interaction point to which the interaction was sent; if the output was "lost" (i.e., it was not received by an instance), then an appropriate warning message is issued.

To access separately different type of information concerning he last fired transition outputs some other function may be used, namely:

- \$last_trans_outputs_number,
- \$last_trans_output_inter_point(SCRATCH_INT_ACCESS),
- \$last_trans_output_inter_name(SCRATCH_INT_ACCESS),
- \$last_trans_output_inter_param(SCRATCH_INT_ACCESS, INTER_PARAMETER)

The command

```
"display" "$last_transition_connect" |
"d" "$ltrc"
```

displays information concerning the connections established by the last executed transition. The information displayed is the following:

- the identification of the transition which executed the connections and of the instance which fired the transition (instance-number, module-header-id, module-body-id and transition-number and transition-id, if such exists);

- for each executed connect-statement, the identification of the interaction points being connected.

Another interesting group of display commands is that which provides information concerning some cumulative information collected during the simulation session (e.g., number of times transitions have been fired - see the note at the end of §4.1-Access to objects - for more information how transitions are numbered within Edb). Since these commands are performance related they are described in Section 4.9

Another interesting group of display commands is that which makes it possible to obtain information concerning the (sub)systems, module instances or transitions that may execute in the next step of simulation. Note that, select commands also exist (see Section 4.10), which offers the user the possibility of selecting one of the systems, modules or transitions to be effectively executed in the next step of simulation.
The command

```
"display" "$executable_systems" | "d" "$es"
```
displays the identification of the (sub)systems (subtree of module instances whose root is attributed "systemprocess" or "systemactivity") that may execute in the next step of simulation.

The command

```
"display" "$executable_modules" | "d" "$em"
```
displays the identification of the systems with their module instances which may execute in the next step of simulation.

The command

```
"display" "$executable_transitions" [ "(""instance_access ")" ] | "d" "$et" [ "(""instance_access ")" ]
```
displays - in a structured manner - all transitions, which may be currently executed in all module instances within a sub-tree of module instances whose root is given by instance_access (or given by the current value of $wp, if the instance_access is omitted). The transitions are grouped into "synchronisation-groups" to indicate with what other transitions (if any) a given transition is to be executed in parallel. Only in the case of system attributed "systemprocess" a synchronisation-group may contain more than one transition. Note that a transition is identified by the unique number of the module instance to which it belong (e.g., 3, 4 and 5) and by its unique number relative to the module instance. Note that the transition name, if such a name exists (e.g., send_A, receive_B and receive_C) may be not unique and thus the transition name is a complementary information. For example, a transition of module instance 5 may be identified by (5->2) or, if it has a name e.g., T1 within a specification, by (5->2(T1)).

More information about the structure in which executable transitions are display is given in Section 4.10 where the select commands are explained.

### 4.5.2. File-display

All information that may be displayed (by the command “display access”) can also be registered (displayed) in a file "file-name". The command

```
"file_display"  file-name access | "fd" file-name access
```
serves this purpose. The file-display commands used within the actions of "observers" (see Section 4.7) registers in a file a trace of values "observable" between execution of two consecutive transitions or certain elements "in execution".

### 4.6. Modify commands
Modify commands which attempt to modify Estelle specification-related objects can be executed only when a (sub)system to which these objects belong is in a so-called "system-management-phase" i.e. it does not have any transition "in execution". In other words, it may only "start" -or select - a new set of transitions to be executed next. An attempt to execute these commands in other situations gives a warning message.

It should be stressed that the execution of a modify-command that modifies a source Estelle related object (e.g., a Pascal variable, a state or a queue content), or delay-timers, ends with a re-evaluation of transitions' firing conditions since it may happen that the modification enables or disables some transitions. For the same reasons, any previous (to the modification) and not executed selection (see select commands) are ignored.

4.6.1 General modify commands

```
var-access "=" expression
sp-access "=" expression
implicit-access "=" expression
```

The modify commands serve to modify accessible Pascal variables of the simulated specification, the user scratch-pad variables and some objects accessed by predefined functions. If "instance_access" (within "var-access" - see Section 4.1, or within an access used in "expression") is omitted, then the module instance context given by "$wp" is considered.

The assignment may concern only those expressions whose type is compatible with the left-hand-side of the assignment.

For example, the assignment

```
3->F := 5
```

sets to 5 the integer variable F within the scope of the module instance 3, the assignment

```
#my_own := true
```

sets to true the user scratch-pad variable #my_own, and the assignment

```
$state(2) := IDLE
```

sets to IDLE the control state of the instance 2.

The "implicit-access" on the left-hand-side of the assignment is reduced to the following predefined functions (or their abbreviations - see Appendix 3):

* "$real_time_limit" - to modify the time limit (in seconds) of a simulation;
* "simulation_time_limit" - to modify the time limit (in units of simulation time) of a simulation;
* "$firing_steps" - to modify the number of transitions to be fired;
* "$random" - to modify the "seed" of the random generator;
*.$transition_weight "(" trans_access ")" - to modify the weight of the indicated transition;

Remark: In all the three cases "expression" must evaluate to a positive integer.

* $exec_time_min ["(" trans_access ")"] - to modify minimum transition execution time,
* $exec_time_max ["(" trans_access ")"] - to modify maximum transition execution time,
* $sm_time_min ["(" instance_access ")"] - to modify minimum system management time,
* $sm_time_max ["(" instance_access ")"] to modify maximum system management time,
* $smmean "(" instance_access ")" - to modify the mean value of the time density distribution assigned to the system module instance (module attributed by either systemactivity or systemprocess) indicated by "instance_access"
* $etmean "(" trans_access ")" - to modify the mean value of the time density distribution assigned to the transition indicated by "trans_access",

Remark: In all four cases "expression" must evaluate to a positive real number.

* $state ["(" instance_number ")"] - to modify the control (major) state of an instance, e.g., $st(2) := IDLE, where IDLE is a state identifier in the scope of the module instance 2);
* $queue "(" ip_access, queue-position, interaction-parameter ")" - to modify the value of the parameter of an interaction at the given position in the queue associated with the interaction point given by "ip_access",
* $smdensity ["(" instance_access ")"] - to modify the density distribution assigned to the system module instance (module attributed by either systemactivity or systemprocess) indicated by "instance_access"; the right hand side must be the character "u" (for uniform), "e" (for exponential), "g" (for geometric), "p" (for Poisson)
* $etdensity "(" trans_access ")" - to modify the density distribution assigned to the transition indicated by "trans_access"; the right hand side must be the character "u" (for uniform), "e" (for exponential), "g" (for geometric), "p" (for Poisson).

4.6.2 Queue modify commands

The following two modify commands are related to the contents of queues. The command

"delete_from_queue" "$queue "(" ip_access, queue-position")"

deletes the interaction (and its parameters) on the "queue-position" of the queue associated to the interaction point given by "ip_access". For example, the command delete_from_queue $queue(5->ip,1) deletes the element (interaction and its parameters) in
the 1st position of the queue associated with the interaction point given by "ip" within the module instance whose number is 5.

The command

"insert_in_queue" "$queue" "("ip_access, queue-position")"

inserts the last deleted interaction (and its parameters) in the "queue-position" of the queue of the interaction point given by "ip_access". For example, the command insert_in_queue $queue (5->ip, 3) inserts the last deleted interaction (and its parameters) in the 3rd position of the queue associated with the interaction point given by "ip" within the module instance whose number is 5.

4.6.3 Timer modify_commands
The semantic model of Estelle retains the hypothesis that execution times of transitions are unknown. This knowledge is considered implementation dependent.

The semantic model of Estelle is, however, dependent on a time process. This process is assumed to exist, but it is voluntarily specified by a few constraints to be satisfied in every implementation. The main assumption is that the time progresses as the computation does. Nothing is said, however, when and how fast the time progresses.

When the time progresses the delay-timers (the current min and max "time-out's" whose initial values are specified in the delay-clauses of transitions) decrease accordingly. Recall, that the delay-timers of each delayed transition are observed to decide whether the transition can or cannot be fired.

There exist in Edb different possibilities to simulate the time process. One possibility is to specify the execution time of transitions and/or system management time for subsystems (by means of modify_commands, mentioned in Section 4.6.1, whose left hand side is

"$exec_time_min",
"$exec_time_max",

"$sm_time_min",
"$sm_time_max".

In that case the time automatically advances along with fired transitions (execution time) and/or management (system management time) of the subsystems. When neither the execution times of transitions nor the system management times for subsystems are specified (by default, they are both set to zero, i.e., to 0.0) the time does not advance during the simulation until the situation is reached in which the only enabled transitions are delayed transitions. In such a case, the time is automatically moved to a value within the interval in which the least delayed transition may fire.

The user is offered another possibility of advancing time, in addition to that described above, namely he can issue a special modify_command named advance_time:

"advance_time" "("real_value")"
"at" "("real_value")"

The result is that the time advances by the user specified 'real value'. The value may be also given indirectly by a scratch-pad variable, which has to be of the real type.

The advance_time command may be issued at any moment that is allowed for issuing modify_commands.
4.7. Observers and observer_commands

The commands of that category make it possible to describe certain situations the user wants to observe or to detect and, at the same time, they allow for defining associated actions to be taken. The set_observer command introduces a so-called "observer" and returns to the user a "unique" number identifying the observer. The number may be used subsequently as the reference in particular commands (which we list below) concerning observers. The number of observers the user can create is limited to 200. The "length" of each observer is limited to 2000 tokens. Both limits can be augmented on user request.

"set_observer" IDENTIFIER "{"command-sequence "}" | 
"so" IDENTIFIER "{"command-sequence "}" | 
"enable_observer" observer-id | 
"eo" observer-number | 
"disable_observer" observer-id | 
"dso" observer-id | 
"delete_observer" observer-id | 
"dlo" observer-id | 
"chain_observer" "{observer-id","observer-id","observer-id"}" | 
"co" "{observer-id","observer-id","observer-id"}" |

observer-id = unsigned-integer | identifier

After each computation step (i.e., between the execution of two consecutive transitions), the Edb executes command-sequence (verify conditions and executes actions) defined in all "enabled" observers whose identifications (numbers or names) are listed in the sequence specified by the command chain_observer and in the order given by this command. If the sequence of execution is not given (chain_observer command is not present), the observers are executed in the increasing order of their numbers as if all of them were consecutively listed by a chain_observer command. Note, that the observer numbers/names may appear more than once within chain_observer command. The number of observer numbers is limited to 1000 but can be augmented on user request. It should be stressed that each observer is computed with the working pointer set to 1 (the specification module instance), i.e., their executions are independent with respect to this value and dependent only on user scratch-pad variables and "enable" and "disable" commands.

Observers that are introduced by the set_observer command are by default enabled. Observers deleted are no longer accessible.

The following edb commands are not permitted within a set_observer command:

- quit command;
- select commands;
- continue command;
- restart command
- set_observer command;
- delete_observer command.

The break ("b") simple command can be used within a sequence of commands of an observer (break has no effect when used outside of observers). It stops the execution of the Edb and gives control to the user in the sense explained below. Note that all other admissible Edb commands do not break the execution neither explicitly nor implicitly.

2. Please consult Remark 2 in Section 4.2 (page 21)
If among commands of the executed 'observer' there is a break command, then it is not effective until the execution of all observers completed. This means that the control is given to the user at the end of the observers' evaluation if there is at least one "break" in their executed actions. A prompt message indicates the reference to the observation point(s) that caused the break.

Example:

```plaintext
so  break_0{ if (\$(state(3) = OPEN) and (\$state(2) = CLOSED)) then break fi } 

so  breake_1 { if (\$queue_size(5->U) > 4) or ((\$qi(5->U,1) <> ACK) and (\$state(5) = IDLE)) then break fi }

c0 { 1, break_0 }
```

Note that just after a so (set_observer) command has been issued and confirmed by "return" Edb returns to the user a "unique" number (name) identifying the observer (starting with the number 0). Note also that the order of executing the observers is set by the command co (chain_observer).

******

It was mentioned earlier that observers containing display commands in their actions might serve to trace objects. A few simple examples illustrate this possibility.

Examples:

```plaintext
so  last_tr_id{ d \$last_transition_id }
```

This observer displays on the screen (until disabled or deleted) the sequence of all transition identifiers fired during the simulation while

```plaintext
so { file_display myfile \$last_transition_id }
```

registers the same sequence on the file named "myfile".

Assume now, that the user created the scratch-pad variable #my initialised to false

```plaintext
#my := false;
```

and introduced the following observer

```plaintext
set_observer{ if (\$st(3) = IDLE) and (\$st(2) = WAIT) then #my := true fi;\n    if (\$st(3) = OPEN) and (\$st(2) = CLOSED) then #my := false; d $tro fi;\n    if #my = true then d $tro fi;\n }
```

******

The observer traces (on the screen) all outputs between two situations described by the if-conditions on states.

The following is an example of an observer (it is assumed that there are 4 module instances but only instances 2, 3 and 4 have transitions), which stops the simulation when all transitions in all module instances have been executed at least once. In such a case the message "all transitions have been executed at least once" will be printed as well as the number of times each transition has been fired (d$transition_statistics).

```plaintext
% An observer which stops ('break') the simulation when all transitions in all
```
% modules within the simulated specification have been fired at least once.
% When the simulation stops, the following message is displayed
% 'all_transitions_haveBeen_executed_at_least_once'
% and the statistics of the number of times the transitions has been fired
% (d$transition_statistics) is also displayed

set_observer STOP \     
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
Example: The following is an example of a macro. It defines a simulation scenario, which consists in executing 5 times 200 (randomly chosen) transitions each time restarting the simulation to try to execute a different sequence of transitions.

If during simulation there was neither a deadlock nor run-time error and simulation was not stopped by a break command within an observer, the message "successful end" will be printed.

If simulation was stopped by a break command within an observer, the message "BREAK" will be printed. Otherwise, the message "non_successful_end" followed by the message(s) "DEADLOCK" and/or "RUN TIME ERROR" will be printed as well as the cycle number (#nb) and the current_firing_step number within the cycle.

% simulation scenario which consists in executing 5 times 200 (randomly chosen) 
% transitions each time restarting the simulation

macro RUN
/
dl nb:=0;
   do if #nb=5
      then display #nb;
      display "successful end";
      exit
   else $firing_steps := 200;
      continue;
   if $total_firing_steps = 200
      then display #nb;
      restart;
      #nb:=#nb+1
   else if $is_break
      then display "BREAK"
      else display "non_successful end";
      fi;
   display #nb;
   display $current_firing_step;
   if $is_deadlock then display "DEADLOCK"
   if $is_error then display "RUN TIME ERROR"
   exit
   fi
 od
}

The following edb command is not permitted within a macro command:
- select commands;

The following edb commands are not recommended within a macro command:
- quit command;
- macro command

4.9. Select commands
This is an important group of commands related to information helping in interactive and "guided" simulations. These commands indicate the systems, modules or transitions that may execute in the next step of simulation and offer the user the possibility of making his own choice.

If the select commands above are repeated, then only the last user response before a continue command is taken into account. Any previous selection is discarded. In other words, if the last select command ended with a user selection, then this selection is valid, otherwise (i.e., the user made no selection in this last select command) the simulation continues selecting randomly.

The identification of systems, modules or transitions includes, in general, one of the following attributes:

"can start first";
"can terminate first";
"can't start first";
"can't terminate first".

The last two attributes may appear only when the user has specified so-called "time-constraints" during the simulation session. These constraints specify the execution times (max and min) of transitions within the given module instances and the system-management-times (max and min) of the given (sub)systems. Remember that a subsystem is a subtree of module instances whose root is attributed "systemactivity" or "systemprocess".

Each subsystem progresses by consecutively executing two phases:

1) the system-management-phase, which once terminated initiates the "start" of execution of transitions chosen by the subsystem to be executed, and

2) transition-execution-phase, which ends when all those transitions are terminated.

The system-management-time_min (respectively-max) for a subsystem may be interpreted as the minimal (respectively-maximal) time during which the subsystem can still recognise the changes in its local context due to the reception of interactions from other subsystems or modifications introduced by the user during the simulation session by means of modify commands.

The execution-time_min (respectively-max) of a transition may be interpreted as the minimal (respectively-maximal) time during which the transition will terminate.

The initial values of system-management-times (min and max) for the given subsystem and of execution-times (min and max) of transitions within the given module instances can be specified by the user by means of assignments whose left-hand-side are the following predefined functions: $sm\_time\_max$, $sm\_time\_min$, $exec\_time\_max$, $exec\_time\_min$ (with an optional parameter "trans_access") and whose right-hand-side is any positive real value. By default all these initial values are zero (0.0).

The current values of system-management-times (min and max) and of execution-times (min and max) change dynamically during simulation taking into account the time actually spent to execute. Each time a subsystem enters its system-management-phase (ends its transition-execution-phase) it reinitialises the system-management-time (min and max) values. Each time a subsystem enters its transition-execution-phase (ends its system-management-phase) it reinitialises the values of execution time (min and max) of transitions selected for execution.

The attribute "can start first" (or "can't start first") means that the associated subsystem is in the "system-management-phase" and it can (or respectively can't) complete this phase before
another subsystem complete an event (either a transition execution or a system-management-phase).

The attribute "can terminate first" (or "can't terminate first") means that the associated subsystem is in a phase of executing transition(s) and can (or respectively can't) terminate a transition before another transition terminates either in the same subsystem, if this subsystem is a "systemprocess", or before another subsystem completes an event.

In the case of attributes "can't ..." information about the (sub)system, module or transition is displayed only for information as the user cannot choose such items. They are marked by "#" while the user can choose only those marked by an integer. The aim of displaying such information is to inform the user that with differently chosen time-constraints (and always in the default case) these items could appear with "can ..." attributes and thus could be chosen.

We will now explain the effects of the select commands starting with default case.

### 4.9.1 Select executable-systems command

The command

```
"select" "$executable_systems" | "s" "$ses"
```

displays the identification of the systems that may execute an event. The user may select a system to be executed next end exit or he may simply exit by typing "return" not preceded by a selection. If a system was selected and the command `continue` follows, then the first transition executed will be one chosen randomly from those able to fire in the system.

**Example**: When the user typed the following command

```
s $es
```

and confirmed it by "return", the following information might be displayed:

**1: system 2 (systemprocess) can start first**

**2: system 5 (systemactivity) can start first**

*Type your choice and "return" or "return" to exit:*

Note, that systems numbered 2 and 5 correspond to the unique numbers of the module instances being roots of the systems in the tree-like instance hierarchy.

The user may select either system 2, by typing 1, or system 5 by typing 2. The choice has to be confirmed by subsequently typing "return". Once selected and confirmed the system will start (conceptually) a transition chosen randomly from those able to fire within the selected system. He can also type "return" without previously selecting a system provoking no selection.

Assume that the user selected system 5 and confirmed his choice. In such a case the new following information will be automatically displayed:

**1: system 2 (systemprocess) can start first**

**2: system 5 (systemactivity) can terminate first**

*Type your choice and "return" or "return" to exit:*
At this moment the user may select either to terminate the execution of the selected (and started) transition, by typing 2, or select to start a transition within the system 2 by typing 1. In the second case both systems have started the execution of selected transitions in parallel. Assume that the latter was the user's choice. In such a case the following information will be automatically displayed:

1: system 2 (systemprocess) can terminate first
2: system 5 (systemactivity) can terminate first

Type your choice and "return" or "return" to exit:

Now the user has a choice to either terminate the execution of the selected transition within system 2 or system 5. Once he made his choice the s$es command is terminated, the edb> prompt will appear and the user can introduce the next command. If the command continue follows, then Edb will effectively terminate the execution of the selected transition. The user can, however, introduce once more the s$es command, if he changed his mind and wants now to terminate the execution of the selected transition within the second system or even not decide whatsoever (typing "return" not preceded by a selection). In the last case when the command continue follows Edb will make a random choice.

Let us now examine the different effects of the s$es command when the user has specified in advance "time-constraints" in the following way:

\[ \begin{align*}
    \text{sm\_time\_max}(2) & := 2.0, \quad \text{sm\_time\_min}(2) := 1.0 \\
    \text{sm\_time\_max}(5) & := 5.0, \quad \text{sm\_time\_min}(5) := 3.0
\end{align*} \]

Example : When the user typed the following command

s $es

and confirmed it by "return", the following information will be now displayed:

1: system 2 (systemprocess) can start first
*: system 5 (systemactivity) can't start first

Type your choice and "return" or "return" to exit:

The system 5 can't start first now since

\[ \text{sm\_time\_min}(5) > \text{sm\_time\_max}(2) \]

and the only choice is to select system 2.

4.9.2 Select executable-module command

The command

"select" "$executable\_modules" |
"s" "$em"
displays the identification of the systems with their module instances which may execute. The user may select a module instance, and hence a system, to be executed next end exit or he may simply exit by typing "return" not preceded by a selection. If a module instance was selected and the command continue follows, then the first transition executed will be one chosen randomly from those able to fire in the module instance.

**Example** Assume that the user issued the following command

\[ s \text{"sem"} \]

and confirmed it by "return". The following information might be displayed:

\[
\text{system 2 (systemprocess)}
\]

1: module instance 3 (process) can start first  
2: module instance 4 (activity) can start first

\[
\text{system 5 (systemactivity)}
\]

3: module instance 5 (systemactivity) can start first

*Type your choice and "return or "return" to exit :*

The user may now select among module instances belonging to different systems. Referring to the previous example when he used the command \(s\$es\) he now obtains more information since now he can differentiate between module instances 3 and 4 both belonging to the system 2. The further steps of the process of selection is similar to that explained for \(s\$es\) command.

**4.9.3 Select executable-transitions command**

The command

\[
"\text{select}" \quad \"\text{executable_transitions}\{\text{"instance_access"}\}\}
\]

\[
"\text{s}\" \quad \"\text{set}\{\text{"instance_access"}\}\}
\]

displays all transitions, which may be currently executed in all module instances within a sub-tree of module instances whose root is given by instance_access (or given by the current value of \$wp, if the instance_access is omitted).

There are two modes called full and simplified to display and execute the transitions. The mode may be selected by setting a predefined function \$simplified_mode (\$smd - in short) value to false or true, respectively.

**4.9.3.1 Full mode**

The transitions, which may be executed, are shown on per system module (called also subsystems i.e., modules attributed systemprocess or systemactivity) basis. Within each system module transitions are grouped into "synchronization-groups" which contain transitions to be executed in parallel. Please note that only in the case of a systemprocess system module a synchronization-group may contain more than one transition since only in this case transitions of child modules may be executed in synchronous parallel mode. Within this mode the transitions to be executed are selected in two steps. In the first step selected transitions within a synchronization-group start their execution, and in the second step the selected transition terminates its execution. This two step selection process is necessary to give the possibility to execute transitions belonging to different system modules in asynchronous parallel mode. Each selection has to be confirmed by pressing the "return"
key. If the user press the "return" key not preceded by a selection then it exits the select
command. To fire the selected transitions the user has to issue the continue command.

**Example:** When the user typed the following command (assume that he issued this
command instead of s$em command since he wants to have more information)

\[ s \ $et \]

and confirmed it by "return", the following information might be displayed:

\[
\begin{align*}
\text{system 2 (systemprocess)} \\
\text{synchronisation-group} \\
\hspace{1em} 1: \quad \{(3 \to 5 (send_A)) \} \text{ can start first} \\
\hspace{1em} (4 \to 7 (receive_C)) \\
\text{synchronisation-group} \\
\hspace{1em} 2: \quad \{(3 \to 5 (send_A)) \} \text{ can start first} \\
\hspace{1em} (4 \to 6 (receive_B)) \\
\text{system 5 (systemactivity)} \\
\text{synchronisation-group} \\
\hspace{1em} 3: \quad \{(5 \to 2) \} \text{ can start first}
\end{align*}
\]

*Type your choice and "return" or "return" to exit:*

Note that a transition is identified by the unique number of the module instance to which it
belongs (e.g., 3, 4 and 5) and by its unique number relative to the module instance (e.g., 2 in
case of the system 5) and optionally by its name, if such a name exists (e.g., send_A,
receive_B and receive_C). Please consult the note at the end of §4.1 (Access to objects) for
more information how transitions are referenced within Edb.

The user may now select among the sets of transitions belonging to different module
instances, which, in turn, belong to different systems. Referring to the previous examples
for which the commands $ses, and $sem were used, the user now obtains much more
information. He knows that each of module instances 3 and 5 can only execute one transition
(send_A and 2, respectively) while module instance 4 can execute either transition receive_B
or receive_C. He knows also (synchronization-groups) the transitions to be executed
synchronously in parallel.

Assume now that the user choice is 2 followed by "return". The following information will
then be automatically displayed:

\[
\begin{align*}
\text{system 2 (systemprocess)} \\
\text{synchronisation-group} \\
\hspace{1em} 1: \quad (3 \to 5 (send_A)) \text{ can terminate first} \\
\hspace{1em} 2: \quad (4 \to 6 (receive_B)) \text{ can terminate first}
\end{align*}
\]

\[
\begin{align*}
\text{system 5 (systemactivity)} \\
\text{synchronisation-group} \\
\hspace{1em} 3: \quad \{(5 \to 2) \} \text{ can start first}
\end{align*}
\]

*Type your choice and "return" or "return" to exit:*

The information shown above for default case may be different in case the user has
previously set "time-constraints" related to the execution time of transitions within module
instances 3 and 4. Assume that the following modify commands were introduced before the user issued the s$et command:

\[
\begin{align*}
&\text{$exec\_time\_max(4\to6):=4.0$;} & &\text{\ $exec\_time\_min(4\to6):=3.0$;} \\
&\text{$exec\_time\_max(3\to5):=6.3$;} & &\text{\ $exec\_time\_min(3\to5):=5.0$}
\end{align*}
\]

In such a case the following information will be displayed instead of that shown above:

\[
\begin{align*}
\text{system 2 (systemprocess)}
&\text{ synchronisation-group} \\
&\text{*:} \quad (3 \to 5 (send\_A)) \hspace{1cm} \text{can't terminate first} \\
&\text{1:} \quad (4 \to 6 (receive\_B)) \hspace{1cm} \text{can terminate first}
\end{align*}
\]

\[
\begin{align*}
\text{system 5 (systemactivity)}
&\text{ synchronisation-group} \\
&\text{3:} \quad \{ (5 \to 2) \hspace{1cm} \text{can start first}
\end{align*}
\]

\text{Type your choice and "return" or "return" to exit :}

The transition "send\_A" of module instance 3 can't terminate first. This is due to the execution time-constraints that were introduced. Since

\[
\text{$exec\_time\_max(3\to5):>exec\_time\_max(4\to6)$}
\]

the transition "receive\_B" of module instance 4 will always terminate before the transition "send\_A" of module instance 3 can terminate.

The further process of selection is similar to that explained for s$es command.

\section*{4.9.3.2 Simplified mode}

When the ‘simplified’ mode has been selected by the user the executable transitions are displayed in a flat manner (they are not grouped neither w.r.t sub-systems nor w.r.t "synchronisation-groups"). In contrary to the full mode the selected transition terminates, if possible, at once. As a consequence there is no more possible to emulate parallelism between sub-systems. This is due to the fact that the user cannot any more start transitions within more than one subsystem before terminating one of them. This simplified mode is much "user friendly" but restricts the possible execution paths.

Let us follow the same example. Instead of the following information displayed for full-mode

\[
\begin{align*}
\text{system 2 (systemprocess)}
&\text{ synchronisation-group} \\
&\text{1:} \quad \{(3 \to 5 (send\_A)) \hspace{1cm} \text{can start first} \\
&\text{ synchronisation-group} \\
&\text{2:} \quad \{(3 \to 5 (send\_A)) \hspace{1cm} \text{can start first} \\
\text{system 5 (systemactivity)}
&\text{ synchronisation-group} \\
&\text{3:} \quad \{ (5 \to 2) \hspace{1cm} \text{can start first}
\end{align*}
\]

\text{Type your choice and "return" or "return" to exit :}
the following will be displayed for simplified-mode:

1: \( (3 \rightarrow 5 \text{ (send}_A) ) \)  
2. \( (4 \rightarrow 7 \text{ (receive}_C) ) \)  
3. \( (4 \rightarrow 6 \text{ (receive}_B) ) \)  
4: \( (5 \rightarrow 2) \)

*Type your choice and "return" or "return" to exit:*

Assume that the user choice is 1 (followed by 'return'. In such a case the transition

\[ 3 \rightarrow 5 \text{ (send}_A) \]

will start and terminate (without the necessity to issue the `continue` command).

Please note that this transition has to be executed in parallel ("synchronization-groups") with either

\[ 4 \rightarrow 7 \text{ (receive}_C) \text{ or } (4 \rightarrow 6 \text{ (receive}_B) \]

This choice will be done (randomly) by the simulator. Let us assume that the simulator choice was \( (4 \rightarrow 6 \text{ (receive}_B) \). The following will be displayed next when the user will issue the select command:

1. \( (4 \rightarrow 6 \text{ (receive}_B) ) \)
2: \( (5 \rightarrow 2) \)

### 4.10 Special environment for performance evaluation study

To conduct performance prediction some quantitative features (e.g., time evolution, statistical parameterisation of nondeterminism) have to be introduced. To this end either Estelle should be extended or Estelle model should be annotated. This second approach has been adopted in EDT [13].

#### 4.10.1 Predefined functions

The following predefined functions have been defined for allowing performance evaluation study during simulation.

First category of predefined function is related to annotate the Estele model by allowing the user to set the execution time interval for selected transitions (by default their execution time is 0.0) and the system management time for selected system modules (by default their system management time is 0.0). To this category belongs also functions allowing the user to choose the distribution function (uniform by default)-used when a concrete time value will be selected in a given interval, and to set a mean value parameter for a selected distribution function.

The predefined function

```
"$exec_time_min" \ "(" \trans_access\")" |  
"$et_min" \ "(" \trans_access\")"
```
accesses the minimum execution time assigned to transitions of the module instance indicated by "the " trans_access ". If the "instance_access" within the " trans_access" is omitted the function accesses the minimum transition execution time of the instance pointed by the current value of $wp. The "$exec_time_min" for a given transition should be not greater than "$exec_time_max". The initial, by default, value of the "$exec_time_min" is 0.0. After each Edb restart command the "$exec_time_min" is reinitialised.

This predefined function may be used as an argument of the display and file_display commands and as the left-hand-side of assignments (the right-hand-side expression must evaluate to a real value) to modify the minimum execution time of transitions within the given module instance. For example, to set to 3.8 the minimum execution time of a transition numbered 3 within the module instance numbered 4 the following modify command may be issued: $exec_time_max(4->3) := 3.8 . The modification of the "$exec_time_min", for a given transition, will be not immediately accepted if proposed at the moment the transition is executing (started but not yet terminated) and will be not accepted at all if the proposed value is greater than the current value of the "$exec_time_max".

NOTE: In the previous version of Edb this function had as argument the "instance_access"

The predefined function

"$exec_time_max" [(" trans_access ")] |
"$setmax" [(" trans_access ")] |

accesses the maximum execution time assigned to transitions of the module instance indicated by "the " trans_access ". If the "instance_access" within the " trans_access" is omitted the function accesses the maximum transition execution time of the instance pointed by the current value of $wp. The "$exec_time_max" for a given transition should be not smaller than "$exec_time_min". The initial, by default, value of the "$exec_time_max" is 0.0. After each Edb restart command the "$exec_time_max" is reinitialised.

This predefined function may be used as an argument of the display and file_display commands and as the left-hand-side of assignments (the right-hand-side expression must evaluate to a real value) to modify the maximum execution time of transitions within the given module instance. For example, to set to 5.7 the maximum execution time of a transition numbered 3 within the module instance numbered 4 the following modify command may be issued: $exec_time_max(4->3) := 5.7 . The modification of the "$exec_time_max", for a given transition, will be not immediately accepted if proposed at the moment the transition is executing (started but not yet terminated) and will be not accepted at all if the proposed value is smaller than the current value of the "$exec_time_min".

NOTE: In the previous version of Edb this function had as argument the instance_access

The predefined function

"$sm_time_max" [ [ (" instance_access ")] ]
"$smmax" [ [ (" instance_access ")] ]

accesses the maximum system management time (real value) of the subsystem indicated by "instance_access" (it has to be a number corresponding to a module instance attributed "systemactivity" or "systemprocess", otherwise an error message is displayed). If the "instance_access" is omitted the function accesses the maximum system management time of the instance pointed by the current value of $wp.

This predefined function may be used as an argument of the display and file_display commands and as the left-hand-side of assignments (the right-hand-side expression must
evaluate to a real value) to modify the maximum system management time of a given module instance. For example, to set to 7.7 the maximum system management time of the system module instance numbered 4 the following modify command may be issued: $sm_time_max(4) := 7.7$. The modification of the "$sm_time_max", for a given system module instance, will be not immediately accepted if proposed at the moment a transition is executing (started but not yet terminated) within this system module instance and will be not accepted at all if the proposed value is smaller than the current value of the "$sm_time_min".

The predefined function

 "$sm_time_min" [ "(" instance_access ")" ]
 "$smmin" [ "(" instance_access ")" ]

accesses the minimum system management time (real value) of the subsystem indicated by "instance_access" (it has to be a number corresponding to a module instance attributed "systemactivity" or "systemprocess", otherwise an error message is displayed). If the "instance_access" is omitted the function accesses the minimum system management time of the instance pointed by the current value of $wp$.

This predefined function may be used as an argument of the display and file_display commands and as the left-hand-side of assignments (the right-hand-side expression must evaluate to a real value) to modify the minimum system management time of a given module instance.. For example, to set to 7.7 the minimum system management time of the system module instance numbered 4 the following modify command may be issued: $sm_time_min(4) := 7.7$. The modification of the "$sm_time_min", for a given system module instance, will be not immediately accepted if proposed at the moment a transition is executing (started but not yet terminated) within this system module instance and will be not accepted at all if the proposed value is bigger than the current value of the "$sm_time_max".

The predefined function

 "$setdensity" "("trans_access")"

accesses the time distribution assigned to the transition indicated by "trans_access" (for its execution time interval). If the "instance_access" within the " trans_access" is omitted the function accesses the time distribution assigned to the transition within the instance pointed by the current value of $wp$. The initial, by default, distribution is uniform. After each restart Edb command the distribution is reinitialised.

This predefined function may be used as an argument of the display and file_display commands and as the left-hand-side of assignments (the right-hand-side expression must evaluate to "e" (Exponential), "g" (Geometric), "p" (Poisson), or "u" (Uniform)).

For example, to set the density of transition numbered 3 within the module instance numbered 2 to the Exponential distribution, the following command is to be issued:

 $setdensity(2->3) := e.$

The predefined function

 "$smdensity" "(" instance_access")"

accesses the time distribution assigned to the system module instance (module attributed by either systemactivity or systemprocess) indicated by "instance_access" (for its system management time interval). If the "instance_access" is omitted the function accesses the time
distribution assigned to the system module instance pointed by the current value of $wp. The initial, by default, distribution is uniform. After each \texttt{restart} Edb command the distribution is reinitialised.

This predefined function may be used as an argument of the \texttt{display} and \texttt{file_display} commands and as the left-hand-side of assignments (the right-hand-side expression must evaluate to "e" (Exponential), "g" (Geometric), "p" (Poisson), or "u" (Uniform)).

For example, to set the density of the module instance numbered 2 to the Exponential distribution, the following command is to be issued:

\begin{verbatim}
$smdensity(2) := e.
\end{verbatim}

The predefined function

\begin{verbatim}
"setmean" "("trans_access")"
\end{verbatim}

accesses the mean of the time distribution assigned to the transition indicated by "trans_access". If the "instance_access" within the "trans_access" is omitted the function accesses mean of the time distribution assigned to the transition within the instances pointed by the current value of $wp.

This predefined function may be used as an argument of the \texttt{display} and \texttt{file_display} commands and as the left-hand-side of assignments (the right-hand-side expression must evaluate to a real value).

The predefined function

\begin{verbatim}
"$smmean" ["("instance_access")"]
\end{verbatim}

accesses the mean of the time distribution assigned to the system module instance (module attributed by either \texttt{systemactivity} or \texttt{systemprocess}) indicated by "instance_access". If the "instance_access" is omitted the function accesses the mean of the time distribution assigned to the system module instance pointed by the current value of $wp.

This predefined function may be used as an argument of the \texttt{display} and \texttt{file_display} commands and as the left-hand-side of assignments (the right-hand-side expression must evaluate to a real value).

**********

Second category of predefined function is related to annotate the Estele model by allowing the user to set the probability of execution of nondeterministic transitions:

The predefined function

\begin{verbatim}
"$transition_weight"  "("trans_access")" | 
"trweight"  "("trans_access")"
\end{verbatim}

accesses the weight assigned to the transition indicated by "trans_access". This predefined function may be used as an argument of the \texttt{display} and \texttt{file_display} commands and as the left-hand-side of assignments (the right-hand-side expression must evaluate to a real value).

The weight value is used by the Edb transition selection process to calculate the probability assigned to nondeterministic transitions fireable at a given moment. By default the weights are equal 1.0. The transition with the highest probability will be offered.
When the weights equal 0.0 is assigned to a transition it will eliminate it from the fireable transition set regardless the values of weights associated to any other transition (it will happens also in case there is no other nondeterministic transitions fireable at a given moment).

To understand the way probabilities are calculated let us take the following example. Let us assume that the instance whose number is 3 has the behaviour described by the following automata. In the state IDLE there are potentially 3 non-deterministic transitions numbered 0, 1 and 2. Potentially, since there fireability is conditioned by CONDITION_i (i=0,1,2) and in given moment not necessarily all conditions are satisfied.

![Automata Diagram](image)

Let us assume that the user issued the following commands:

```plaintext
$transition_weight (3->0) := 1.0
$transition_weight (3->2) := 2.0
$transition_weight (3->2) := 3.0
```

If in a given moment only transitions 1 and 2 have their conditions satisfied then the probabilities assigned dynamically to the transition 1 and 2 will be equal to:

\[
P(t1) = \frac{2.0}{2.0 + 3.0} \\
P(t2) = \frac{3.0}{2.0 + 3.0}
\]

**************

Third category of predefined function is related to the simulation time.

The predefined function

```
"$sim_time_limit"
"$sint"
```

returns the current simulation time limit. Initially, by default, the simulation time limit value is "infinity", which means that no simulation time limit is defined. After each restart Edb command the simulation time limit is reinitialised.

The assignment in which "$sim_time_limit" appears at the left-hand-side of an assignment (the right-hand-side expression must evaluate to a real value) sets the simulation time limit after which the simulation stops without any user intervention (the execution may stop earlier when a "break" command within an observer has been executed or a deadlock is achieved or
a run-time error occurs or another limit is attained, see "$firing_steps" and $real_time_limit). By default the value is assumed to be "infinity", i.e., the simulation goes on, in principle, forever. The simulation time limit is consulted in each simulation step. If the "$sim_time_limit" value is modified by the user the previous value is cancelled and the command "continue" restarts the simulation with the new simulation time limit. If the limit has been achieved and is not modified by the user the command "continue" restarts the simulation with the same limit. If the simulation stopped earlier for one of the reasons mentioned above, then the command "continue" forces the simulation to go on for the rest of the limit. Thus the user has the possibility to interact with the debugger in intervals measured in simulation time units. The simulation time may advance if the execution time of some transitions has been set to non-zero values (see $etmax and $etmin), if the system management time of some system module instance has been set to non-zero values (see $smmax and $smmin), and if there are some dalay_transitions (time-outs) defined.

This predefined function may be also used as an argument of the display and file_display commands.

The predefined function

"$total_simulation_time"
"$tsimt"

returns the current simulation time elapsed from the beginning of the session. The simulation time advance when the execution time of executed transitions has been set to non-zero values (see $etmax and $etmin), when the system management time of some system module instance set to non-zero values (see $smmax and $smmin) elapsed, and when some delays (time-outs) expired. The value of this function is reinitialised after each 'restart' Edb command.

This predefined function may be used as an argument of the display and file_display commands and within comparisons (=, <=, >=) with a real value.

***************

Forth category of predefined function is related to collect some information during the simulation run. This category includes calculating some statistics.

The predefined function

"$deadtime" [ "("trans_access")" ]

returns the dead time of the transition indicated by "trans_access" which has to be a delay transition (a transition whose definition contains a delay clause). The dead time associated to a delay transition corresponds to the amount of time a system is idle before executing that delay transition.

This predefined function may be used as an argument of the display and file_display commands.

The predefined function

"$global_trans_statistics"
"$gtrstat"
Semantics: The function accesses, for each module body defined within the simulated specification, the list of its transitions and gives for each transition the number of times it has been fired.

This predefined function may be used as an argument of the `display` and `file_display` commands.

**Note 1:** In the previous version of Edb this function was called `$transition_statistics`. The `$transition_statistics` function has now a different definition.

**Note 2:** One difference with `$transition_statistics` (instance_access) consist in having statistics "per module's bodies", i.e., if there is more then one instance of a given module (module_header, module_body) then the number of times a transition is fired is a global measure for transitions of all instances. Another difference consists in the fact that the statistics is not reinitialised after execution of the 'restart' Edb command.

The predefined function

```
"$queue_statistics" [ "("instance_access")" ]
"$qstat" 
```

returns the list of interaction points of the module instance indicated by "instance_access" and gives for each interaction point statistics of the message traffic vehiculed through it. The statistics displayed are related to the queues associated to all interaction points within the given instance. The statistics concerns the maximum and mean number of messages stored in each queue and the number of interactions outputted through each interaction point.

This predefined function may be used as an argument of the `display` and `file_display` commands. The execution of the commands

```
display $queue_statistics
fd file_name $queue_statistics
```

creates the file

```
specname.qstat_<instance_access>.gr
```

(where the `specname` corresponds to the currently simulated specification name) which can be used by the `plot` Edb command to plot the statistics.

The predefined function

```
"$trans_nb_of_use" 
"$trans_nb_of_use" 
```

returns the number of times the transition indicated by "trans-access" has been fired. The `$trans_nb_of_use` function is reinitialised to 0 after each `restart` Edb command.

This predefined function may be used as an argument of the `display` and `file_display` commands, within boolean expression (comparisons with an integer) and as right-hand-side of assignments.

**EXAMPLES:**

```
if $trans_nb_of_use(3->5) <> 0 then ... fi
```
tests whether the transition numbered 5 of module instance 3 has been fired more than once,

if $\text{trans\_nb\_of\_use}(3->$least\_fired\_transition(3)) \neq 0 then … fi

tests whether the least fired transition of module instance 3 has been fired more then once, or
in other words, whether each transition of the module instance 3 has been fired at least once.

The predefined function

"$\text{transition\_statistics}" ["(\"instance\_access\")"]
"$\text{trstat}" ["(\"instance\_access\")"]

returns the list of transitions of the module instance indicated by "instance\_access" and gives
for each transition the number of times it has been fired (see also "$\text{global\_trans\_statistics}"").

This predefined function may be used as an argument of the display and file\_display commands. The execution of the commands

\begin{verbatim}
display $\text{transition\_statistics} (INSTANCE\_ACCESS)
fd file\_name $\text{transition\_statistics} (INSTANCE\_ACCESS)
\end{verbatim}

creates the file

\text{specname.trstat}_{<\text{INSTANCE\_ACCESS}>}.\text{gr}

(where the specname corresponds to the currently simulated specification name) which can
be used by the plot Edb command to plot the statistics.

\section*{4.10.2 Edb commands}

Five commands have been defined to help the user to collect data to predict the performance
of the implementation generated from the simulated Estelle specification.

The command

\begin{verbatim}
"\text{fscurve}" "(" xvar, yvar, firing\_step ")"
\end{verbatim}

generates in the current directory an Edb command\_script named specname.fscurve\_nb\_ in
which an observer allows collecting the xvar and yvar information each firing\_step\ (integer corresponding to a number of fired transitions). The xvar and yvar could be any
object identifier (returning a single integer or real value) that could be accessed by Edb
(Pascal variables, scratch pad variables, some predefined functions listed below, …) put
between double quotes. The observer will collect data in a file named specname.fscurve_{<\text{nb}>}.\text{gr}. The statistics collected by the observer may be plotted by
means of the plot command.

The following predefined functions may be used as xvar and yvar arguments:

\begin{verbatim}
"$\text{current\_firing\_step}"
"$\text{deadtime}"
"$\text{deadtime}" "(" TRANS\_ACCESS ")"
"$\text{queue\_size}" "(" IP\_ACCESS ")"
"$\text{total\_firing\_steps}"
"$\text{total\_sim\_time}"
"$\text{trans\_nb\_of\_use}" "(" TRANS\_ACCESS ")"
\end{verbatim}
The \texttt{fscurve (<xvar>, <yvar>, <firing_step>)} command can be issued several times during the simulation session to allow the user to collect several (different) xvar and yvar with given periodicity. Each time this command is issued it will generate a different command\_script \texttt{(specname.fscurve1, specname.fscurve2, rspecname.fscurve3, ... ).} The same applies to naming the scratch pad variables defined within the command\_script, and the results files \texttt{(specname.fscurve\_1.gr, specname.fscurve\_2.gr, specname.fscurve\_3.gr, ... ).}

The command

\texttt{"tmcurve""(" <xvar>, <yvar>, <simulation_time_step> ")"}

Semantics: The command generates in the current directory an Edb command\_script named \texttt{specname.tmcurve\_<nb>} in which an observer allows collecting the xvar and yvar information each \texttt{simulation_time_step} (real value corresponding to the simulation time measured by \texttt{$total\_simulation\_time$}) units of simulation time. The xvar and yvar could be any object identifier (returning an integer or real value) that could be accessed by Edb (Pascal variables, scratch pad variables, some predefined functions listed below, ...) put between double quotes. The observer will collect data in a file named \texttt{specname.tmcurve\_<nb>.gr}. The statistics collected by the observer may be plotted by means of the plot command.

The following predefined functions may be used as xvar and yvar arguments:

\begin{itemize}
  \item \texttt{"$current\_firing\_step"}
  \item \texttt{"$deadline"}
  \item \texttt{"$deadline""(" TRANS\_ACCESS ")"}
  \item \texttt{"$queue\_size""(" IP\_ACCESS ")"}
  \item \texttt{"$total\_firing\_steps"}
  \item \texttt{"$total\_sim\_time"}
  \item \texttt{"$trans\_nb\_of\_use""(" TRANS\_ACCESS ")"}
\end{itemize}

The \texttt{tmcurve (<xvar>, <yvar>, <simulation_time_step>)} command can be issued several times during the simulation session to allow the user to collect several (different) xvar and yvar with given periodicity. Each time this command is issued it will generate a different command\_script \texttt{(specname.tmcurve1, specname.tmcurve2, rspecname.tmcurve3, ... ).} The same applies to naming the scratch pad variables defined within the command\_script, and the results files \texttt{(specname.tmcurve\_1.gr, specname.tmcurve\_2.gr, specname.tmcurve\_3.gr, ... ).}

**Remark:** The only difference between fscurve and tmcurve is that the xvar and yvar information are collected in the first case each time a given number of transitions have been fired, and in the second case each time a given period of simulation time have elapsed.

The command

\texttt{"resptime"("<trans\_access1\_restricted>, <trans\_access2\_restricted>, <real\_value>" )"}

\begin{verbatim}
  trans_access1_restricted = instance_access_restricted ">" trans_id_restricted
  trans_access2_restricted = instance_access_restricted ">" trans_id_restricted
  instance_access_restricted = integer
  trans_id_restricted = integer
\end{verbatim}

generates in the current directory an Edb command\_script named \texttt{specname.resptime<nb>} in which an observer allows computing the mean response time between the firing of the transition indicated by the \texttt{<trans\_access1>} and of the transition indicated by the \texttt{<trans\_access2>}. The \texttt{<real\_value>} argument specifies the error percentage allowed in computing the mean resptime with a 95% confidence interval. This observer will stop
(break) the simulation once the calculation of the mean value of the response time terminated and will store the mean response time in a file named specname.resptime_<nb>.

Please note that the syntax of the <trans_access> is restricted to its most simple form, that is integer -> integer, where the first integer represents an existing module instance, and the second integer represents an existing transition number within the module instance.

The resptime (<trans_access>, <trans_access>, <real_value>) command can be issued several times during the simulation session to allow the user to evaluate throughput of several transitions. Each time this command is issued it will generate a different command_script (specname.resptime1, specname.resptime2, specname.resptime3, ... ). The same applies to naming the scratch pad variables defined within the command_script, and the results files (specname.resptime_1, specname.resptime_2, specname.resptime_3, ... ).

The command

"throughput" "(" <trans_access_restricted>,<real_value>)"

trans_access_restricted = instance_access_restricted "->" trans_id_restricted
instance_access_restricted = integer
trans_id_restricted = integer

Semantics: The command generates in the current directory an Edb command_script named specname.throughput<nb> in which an observer allows collecting the mean throughput of the transition referenced by <trans_access>. The <real_value> argument specifies the error percentage allowed in computing the mean throughput with a 95% confidence interval. This observer stop (break) the simulation once the calculation of the mean value of the throughput terminated and will store the mean throughput in a file named specname.throughput_<nb>.

Please note that the syntax of the <trans_access> is restricted to its most simple form, that is integer -> integer, where the first integer represents an existing module instance, and the second integer represents an existing transition number within the module instance.

The throughput (<trans_access>, <real_value>) command can be issued several times during the simulation session to allow the user to evaluate throughput of several transitions. Each time this command is issued it will generate a different command_script (specname.throughput1, specname.throughput2, throughput3, ... ). The same applies to naming the scratch pad variables defined within the command_script, and the results files (specname.throughput_1, specname.throughput_2, specname.throughput_3, ... ).

The command

"plot" file_name

plots the statistics contained in the file indicated by "file_name". The "file_name" may be one of the following files:

- specname.qstat_<nb>.gr
- specname.trstat_<nb>.gr
- specname.fscurve_<nb>.gr
- specname.tmcurve_<nb>.gr

where 'specname' is the base_name of the Estelle specification (specname.stl), and <nb> is an integer generated by Edb (incremented each time the command was issued).
REFERENCES

[5] E. Mariani, Data representation in the debugger
APPENDIX 1: SYNTAX OF THE COMMAND LANGUAGE

1. NOTATION AND REMARKS

The notation used below is the following:

- " and " enclose a terminal symbol;
- [ and ] enclose an optional item;
- | denotes an "or" in the alternatives
- < and > enclose a language elements recognized by a lexical analyzer.

Remark 1: See also the abbreviations of the language keywords in Section 1 of Appendix 3.

Remark 2: The distinction between upper-case and lower-case letters is significant for all identifiers except those used within simulated Estelle specification. To avoid syntax errors, it is strongly recommended to use upper-case letters for accessing Estelle specification identifiers when they coincide with key-words of the command language (e.g., break or b). The list of all key-words of the command language (abbreviations included) can be found in Section 1 of Appendix 3.

2. GRAMMAR OF THE COMMAND LANGUAGE

AXIOM = [COMMAND_LIST] <Carriage-return>
COMMAND_LIST = COMMAND [";" COMMAND_LIST]
COMMAND = HELP_COMMAND | QUIT_COMMAND | RESTART_COMMAND | UNIX_COMMAND | CONTINUE_COMMAND | CHANGE_WP_COMMAND | SELECT_COMMAND | PRINT_COMMAND | DISPLAYTEXT_COMMAND | DISPLAY_COMMAND | FILEDISPLAY_COMMAND | DELETE_COMMAND | INSERT_COMMAND | MODIFY_COMMAND | ADVANCE_TIME_COMMAND | SETOBSERVER_COMMAND | ENABLEOBSERVER_COMMAND | DISABLEOBSERVER_COMMAND | DELETOBSERVER_COMMAND | CHAINOBSERVER_COMMAND | MACRODEF_COMMAND
| MACROCALL_COMMAND |
| BREAK_COMMAND |
| EXIT_COMMAND |
| IF_COMMAND |
| DO_COMMAND |
| FSCURVE_COMMAND |
| TMCURVE_COMMAND |
| THROUGHPUT_COMMAND |
| RESPTIME_COMMAND |

HELP_COMMAND = "help" [OPT_HELP_TOKEN]

OPT_HELP_TOKEN = "help"
| "quit"
| "restart"
| "unix"
| "continue"
| "change_wp"
| "display_text"
| "display"
| "file_display"
| ":="
| "advance_time"
| "set_observer"
| "enable_observer"
| "disable_observer"
| "delete_observer"
| "chain_observer"
| "macro"
| "break"
| "exit"
| "insert_in_queue"
| "delete_from_queue"
| "print"
| "select"
| "if"
| "do"
| "fscurve"
| "tmcurve"
| "throughput"
| "resptime"

QUIT_COMMAND = "quit"
RESTART_COMMAND = "restart"
UNIX_COMMAND = "unix"
CONTINUE_COMMAND = "continue"
CHANGE_WP_COMMAND = "change_wp" [OPT_INSTANCE_ACCESS]
SELECT_COMMAND = "select" PREDEF_SEL
PRINT_COMMAND = "print" <string>
DISPLAYTEXT_COMMAND = "display_text" [OPT_INTEGER] [OPT_LINE2]
DISPLAY_COMMAND = "display" ACCESS
| "display" <string>
FILEDISPLAY_COMMAND = "file_display" <file_name> ACCESS

**************************************************

*) The "display" <string> command is the equivalent command. Please note that the command "print" <string> will be removed from the set of Edb command in the next version of Edb.

*************************************************************************

Last change: mai 29, 2000
© 1989 - 1990, BULL S.A
"file_display" <file_name> <string>

DELETE_COMMAND = "delete_from_queue" PREDEF_INS_DEL

INSERT_COMMAND = "insert_in_queue" PREDEF_INS_DEL

MODIFY_COMMAND = VARIABLE_ACCESS "=" EXPRESSION
| SCRATCHPAD "=" EXPRESSION
| PREDEF_LEFT_HAND_SIDE "=" EXPRESSION

ADVANCE_TIME_COMMAND = "advance_time" (" REAL")
| "advance_time" (" SCRATCHPAD ")

SETOBSERVER_COMMAND = "set_observer" IDENTIFIER "{ COMMAND_LIST "}"

ENABLEOBSERVER_COMMAND = "enable_observer" OBSERVER_ID

DISABLEOBSERVER_COMMAND = "disable_observer" OBSERVER_ID

DELETEOBSERVER_COMMAND = "delete_observer" OBSERVER_ID

CHAINOBSERVER_COMMAND = "chain_observer" "{ OBSERVER_ID_LIST "}"

MACRODEF_COMMAND = "macro" IDENTIFIER "{ COMMAND_LIST "}"

MACROCALL_COMMAND = IDENTIFIER ("")

BREAK_COMMAND = "break"

EXIT_COMMAND = "exit"

IF_COMMAND = "if" EXPRESSION
| "then" COMMAND_LIST
| ["else" COMMAND_LIST] "fi"

DO_COMMAND = "do" COMMAND_LIST "od"

FSCURVE_COMMAND = "fscurve" (""XVAR"", ""YVAR"", INTEGER )"

TSCURVE_COMMAND = "tscurve" (""XVAR"", ""YVAR"", REAL )"

THROUGHPUT_COMMAND = "throughput" (" TRANS_ACCESS, REAL ")

RESPTIME_COMMAND = "resptime" (" TRANS_ACCESS, TRANS_ACCESS, REAL ")

OPT_INTEGER = INTEGER
| VOID

OPT_LINE2 = "," OPT_INTEGER

BASIC_ELT = ORD_ELT
| ACCESS
| [" ELEMENT_LIST "]

ORD_ELT = INTEGER
| CHAR
| REAL
| "true"
| "false"

ACCESS = SCRATCH_INT_ACCESS
| VARIABLE_ACCESS
| IMPLICIT_ACCESS_CALL

VARIABLE_ACCESS = [INSTANCE_ACCESS "->"] V_ID

V_ID = IDENTIFIER
| ARRAY_ELEMENT_ACCESS  |
| RECORD_FIELD_ACCESS  |

ARRAY_ELEMENT_ACCESS = V_ID "[" EXPRESSION_LIST "]"

RECORD_FIELD_ACCESS = V_ID "." V_ID

ELEMENT_LIST = ELEMENT
+ ELEMENT "," ELEMENT_LIST

ELEMENT = ORD_ELT
+ VARIABLE_ACCESS

EXPRESSION_LIST = EXPRESSION
+ EXPRESSION "," EXPRESSION_LIST

EXPRESSION = TERM1
+ EXPRESSION REL_OP TERM1

REL_OP = "="
+ "<>"
+ "<"
+ "<="
+ ">
+ ">="

TERM1 = FACTOR
+ UN_OP FACTOR
+ TERM1 ADD_OP [UN_OP] FACTOR
+ TERM1 MUL_OP [UN_OP] FACTOR

UN_OP = "+"
+ "-"

ADD_OP = "+"
+ "-"
+ "or"

MUL_OP = "*"
+ "/
+ "mod"
+ "div"
+ "and"

FACTOR = "(" EXPRESSION ")"
+ "not" FACTOR
+ BASIC_ELJT

IMPLICIT_ACCESS_CALL = PREDEF
+ PREDEF_LEFT_HAND_SIDE
+ PREDEF_SEL
+ PREDEF_INS_DEL
+ PREDEF_FS_TS

PREDEF_SEL = PREDEF_SELECT

PREDEF_SELECT = "$executable_systems"
+ "$executable_modules"
+ "$executable_transitions"(" INSTANCE__ACCESS ")"

PREDEF_INS_DEL = PREDEF_INSERT_DELETE

PREDEF_INSERT_DELETE = "$queue" "," QUEUE_POSITION ")"

PREDEF_FS_TS = "$current_firing_step"
+ "$deadtime"
+ "$deadtime" "," TRANS_ACCESS ")"
+ "$queue_size" "," IP_ACCESS ")"
"$total_firing_steps"
"$total_sim_time"
"$trans_nb_of_use" ")( TRANS_ACCESS )"

PREDEF

"$all_ip_id"
"$all_ip_id" ")( INSTANCE_ACCESS )"
"$all_trans_id" ")( INSTANCE_ACCESS )"
"$all_var_id"
"$all_var_id" ")( INSTANCE_ACCESS )"
"$are_attached" "( IP_ACCESS ", IP_ACCESS )"
"$are_connected" "( IP_ACCESS ", IP_ACCESS )"
"$export_var_id"
"$export_var_id" "( INSTANCE_ACCESS )"
"$external_ip_id"
"$external_ip_id" "( INSTANCE_ACCESS )"
"$global_transition_statistics"
"$hierarchy"
"$hierarchy" "( INSTANCE_ACCESS )"
"$highest_trans_nb" "( INSTANCE_ACCESS )"
"$infinity"
"$internal_ip_id"
"$internal_ip_id" "( INSTANCE_ACCESS )"
"$interaction_point"
"$interaction_point" "( INSTANCE_ACCESS )"
"$is_break"
"$is_connected"
"$is_connected" "( IP_ACCESS )"
"$is_deadlock"
"$is_delay_transition" "( TRANS_ACCESS )"
"$is_down_attached"
"$is_down_attached" "( IP_ACCESS )"
"$is_enabled_delay_trans" "( TRANS_ACCESS )"
"$is_error"
"$is_input_transition" "( TRANS_ACCESS )"
"$is_interrupt"
"$is_spont_transition" "( TRANS_ACCESS )"
"$is_up_attached"
"$is_valid_expression" "( EXPRESSION)"

LAST_TRANS_ID
"$last_transition_attach"
"$last_transition_connect"
"$last_transition_delay"
"$last_transition_detach"
"$last_transition_disconnect"
"$last_transition_init"
"$last_transition_input"
"$last_trans_input_inter_name"
$last_trans_input_inter_params(INTEGER_PARAM)
$last_trans_input_inter_point
$last_transition_outputs
"$last_trans_output_inter_name(SCRATCH_INT_ACCESS)"
"$last_trans_output_inter_params(SCRATCH_INT_ACCESS, INTEGER_PARAM)"
"$last_trans_output_inter_point(SCRATCH_INT_ACCESS)"
"$last_trans_outputs_number"

"$last_transition_releases"
"$last_transition_state"
"$last_transition_terminates"
"$least_fired_transition"
"$least_fired_transition" "( INSTANCE_ACCESS )"
"$locally_firable_transitions"
"$locally_firable_transitions" "( INSTANCE_ACCESS )"
"$local_variable_id"

$local_variable_id" "( INSTANCE_ACCESS )"
$macro"
$module_param_id"
$module_param_id" "( INSTANCE_ACCESS )"
Appendix 1

Estelle Simulator/Debugger (Edb)

---

```
| "$module_var_id" | "$module_var_id" "(" INSTANCE_ACCESS ")" |
| $most_fired_transition | $most_fired_transition "(" INSTANCE_ACCESS ")" |
| MOVE_WP | MOVE_WP |
| "$observer" | "$observer" "(" OBSERVER_ID ")" |
| $spredfunc | $spredfunc |
| "$queue" | "$queue" "(" INSTANCE_IP_ACCESS ")" |
| "$queue_interaction" | "$queue_interaction" "(" IP_ACCESS "," QUEUE_POSITION ")" |
| $sscratch_pad | $sscratch_pad |
| "$spec_name" | "$spec_name" |
| "$stransition_statistics" | "$stransition_statistics" "(" INSTANCE_ACCESS ")" |

PREDEF_LEFT_HAND_SIDE = PREDEF_LHS

PREDEF_LHS = "$setdensity" "(" TRANS_ACCESS ")" |
| "$smdensity" | "$smdensity" "(" INSTANCE_ACCESS ")" |
| "$real_time_limit" | "$real_time_limit" |
| "$firing_steps" | "$firing_steps" |
| "$setmean" | "$setmean" "(" ITRANS_ACCESS ")" |
| "$smmean" | "$smmean" "(" INSTANCE_ACCESS ")" |
| $srandom | $srandom |
| "$transition_weight" | "$transition_weight" "(" TRANS_ACCESS ")" |
| "$sstate" | "$sstate" "(" INSTANCE_ACCESS ")" |
| "$squeue" | "$squeue" "(" IP_ACCESS "," QUEUE_POSITION "," INTER_PARAMETER ")" |
| "$sexec_time_max" | "$sexec_time_max" "(" TRANS_ACCESS ")" |
| "$sexec_time_min" | "$sexec_time_min" "(" TRANS_ACCESS ")" |
| "$sm_time_max" | "$sm_time_max" "(" INSTANCE_ACCESS ")" |
| "$sm_time_min" | "$sm_time_min" "(" INSTANCE_ACCESS ")" |
| "$sm_time_min" | "$sm_time_min" "(" INSTANCE_ACCESS ")" |

INSTANCE_TRANS_ACCESS = INSTANCE_ACCESS |
| TRANS_ACCESS |

TRANS_ACCESS = TRANS_ID |
| INSTANCE_ACCESS "->" TRANS_ID |

TRANS_ID = SCRATCH_INT_ACCESS |
| LAST_TRANS_ID |

LAST_TRANS_ID = "$last_transition_id" |

INSTANCE_IP_ACCESS = IP_ACCESS |
| INSTANCE_ACCESS |

IP_ACCESS = INSTANCE_ACCESS "->" IP_MV_ID |
| IP_MV_ID |

IP_MV_ID = IDENTIFIER |
| IDENTIFIER "(" EXPRESSION_LIST ")" |

OPT_INSTANCE_ACCESS = INSTANCE_ACCESS |
| VOID |

MOVE_WP | "$up" |
| "$down" |
| "$right" |
| "$left" |

---

Last change: mai 29, 2000 © 1989 - 1990, BULL S.A
"$highest_instance_nb"
"$working_pointer"

INSTANCE_ACCESS = SCRATCH_INT_ACCESS
| MOVE_WP

LAST_INST_ID = "$last_instance_nb"

QUEUE_POSITION = SCRATCH_INT_ACCESS

INTER_PARAMETER = V_ID

OBSERVER_ID = INTEGER_IDENTIFIER

INTEGER_IDENTIFIER = SCRATCH_INT_ACCESS
| IDENTIFIER

SCRATCH_INT_ACCESS = INTEGER
|SCRATCHPAD

OBSERVER_ID_LIST = OBSERVER_ID
| OBSERVER_ID "," OBSERVER_ID_LIST

INTEGER = <integer>

IDENTIFIER = <identifier>

SCRATCHPAD = 

CHAR = """"<character>"""

REAL = <integer> "." <integer>

XVAR = FS_TS_PARAM

YVAR = FS_TS_PARAM

FS_TS_PARAM = PREDEF_FS_TS
|SCRATCHPAD
|VARIABLE_ACCESS
APPENDIX 2: PREDEFINED FUNCTIONS

1. NOTATION

The notation used below is the following:

- " and " enclose a terminal symbol;
- [ and ] enclose an optional item;
- { and } enclose an item which may occur zero or more times;
- | denotes an "or" in the alternatives.

A call to a predefined function is defined as follows:

function-call = "$"function-id [ "("parameter-list ")"]
parameter-list = parameter [ ","parameter-list ]

2. LIST OF PREDEFINED FUNCTIONS

Syntax :  "$all_ip_id" ["("INSTANCE_ACCESS ")"] | 
          "$aii" ["("INSTANCE_ACCESS ")"]

Semantics: The function returns the identifiers of all interaction points (external and internal) declared within a given module instance. If the "instance_access" is omitted, then the module instance accessed is that given by "$wp$. See also "$external_ip_id" "("INSTANCE_ACCESS ")" and "$internal_ip_id" "("INSTANCE_ACCESS ")"

Application: Display and file-display commands.

Syntax :  "$all_trans_id" ["("INSTANCE_ACCESS ")"] | 
          "$ati" ["("INSTANCE_ACCESS ")"]

Semantics: The function returns the identifiers of all transitions declared within a given module instance and they "from" state. If the "instance_access" is omitted, then the module
instance accessed is that given by "$wp". The transition identifier consist of the transition number (index), transition name (if defined).

**Application:** Display and file-display commands.

**Syntax:**

```
"$all_var_id" ["("INSTANCE_ACCESS ")"] |
"$avi" ["("INSTANCE_ACCESS ")"]
```

**Semantics:** The function returns the identifiers of all variables (local, exported and module_parameter but not modvar) declared within a given module instance. If the "instance_access" is omitted, then the module instance accessed is that given by "$wp". See also "$local_var_id", $export_variable_id" and "$module_var_id".

**Application:** Display and file-display commands.

**Syntax:**

```
"$are_attached" "("IP_ACCESS ","IP_ACCESS")" |
"$are_att" "("IP_ACCESS ","IP_ACCESS")"
```

**Semantics:** The function returns "true" if the interaction points accessed by the two "ip_accesses" are attached one to another (the module instances to which the interaction points belong must be in a parent/children relationship). If the "instance_access" within one of the two "ip_accesses" is omitted, then the module instance of this interaction point is taken to be that of "$wp". Note that "instance_access" may be omitted only in one of the two "ip_accesses".

**Application:** Boolean expressions.

**Syntax:**

```
"$are_connected" "("IP_ACCESS ","IP_ACCESS")" |
"$are_conn" "("IP_ACCESS ","IP_ACCESS")"
```

**Semantics:** The function returns "true" if the interaction points accessed by the two "ip_accesses" are connected one to another (the module instances to which the interaction points belong must be in a sibling relationship). If the "instance_access" within one (or both) of the two "ip_accesses" is omitted, then the module instance of this interaction point is taken to be that of "$wp". Note that "instance_access" may be the same in both "ip_accesses", e.g., when it is omitted in both cases, since two different interaction points of the same module instance may be connected (both of them, in that case, must be either internal or external).

**Application:** Boolean expressions.
**Syntax:**
"$current_firing_step" | "$cfs"

**Semantics:** The function returns the current value of the "firing-step-counter" related to the current "firing-step" limit, i.e., the value is always between 0 and the last number assigned to "$firing_steps". The value is reset to 0 each time the "$firing_steps" value has been modified.

**Application:** Assignments (right-hand-side) and expressions.

---

**Syntax:**
"$deadtime" [ "("TRANS_ACCESS")"

**Semantics:** The function returns the dead time of the transition indicated by "trans_access" which has to be a delay transition (a transition whose definition contains a delay clause). The dead time associated to a delay transition corresponds to the amount of time a system is idle before executing that delay transition. If "trans_access" is omitted, the function returns the total dead time of all delay transitions defined within the simulated specification.

**Application:** The Edb commands display and file-display .

---

**Syntax:**
"$down" ["("INSTANCE_ACCESS ")"] | "$d" ["("INSTANCE_ACCESS ")"]

**Semantics:** The function returns the value of the unique number assigned to the first children instance of the given module instance. If the "instance_access" is omitted, then the module instance accessed is that given by "$wp" (currently pointed by the "working-pointer" - see in Sec. 4.1 the description of a module instance context data structure). If there is no children instance, then the function returns the value 0 that does not correspond to any possible number assigned to an instance (the smallest number is always 1 and is assigned to the main i.e., specification module).

**Application:** The cwp ("change_wp") navigation command, the display and file-display commands, the right-hand-side of assignments, within boolean expressions (comparisons with any integer) and as an argument of any predefined function having as argument the instance_access..

---

**Syntax:**
"$etdensity" "("TRANS_ACCESS")"

**Semantics:** The function accesses the time distribution assigned to the transition indicated by "trans_access" (for its execution time interval). If the "instance_access" within the "

© 1989 -1990, BULL S.A

Last change: mai 29, 2000
trans_access" is omitted the function accesses the time distribution assigned to the transition within the instance pointed by the current value of $wp. The initial, by default, distribution is uniform. After each restart Edb command the distribution is reinitialised.

**Application:** The commands display, file_display and in left-hand-side of assignments (the right-hand-side expression must evaluate to "e" (Exponential), "g" (Geometric), "p" (Poisson), or "u" (Uniform)). For example, to set the density of transition numbered 3 within the module instance numbered 2 to the Exponential distribution, the following command is to be issued:

\[
\text{\$setdensity}(2\rightarrow 3) := e.
\]

**Syntax:** "\$setmean" "("TRANS_ACCESS")"

**Semantics:** The function accesses the mean of the time distribution assigned to the transition indicated by "trans_access". If the "instance_access" within the "trans_access" is omitted the function accesses mean of the time distribution assigned to the transition within the instances pointed by the current value of $wp.

**Application:** The commands *display, file_display* and in left-hand-side of assignments (the right-hand-side expression must evaluate to a real value).

**Syntax:** "\$executable_modules" | "\$em"

**Semantics:** The function returns the identification of the systems with their module instances that may fire.

**Applications:** The commands *display, file_display* and *select*..

**Syntax:** "\$executable_systems" | "\$es"

**Semantics:** The function returns the identification of the systems that may fire.

**Applications:** The commands *display, file_display* and *select*..

**Syntax:** "\$executable_transitions" [ "("instance_access ")" ] | "\$et" [ "("instance_access ")" ]
**Semantics:** The function returns all transitions, which may be currently executed in all module instances within a sub-tree of module instances whose root is given by *instance_access* (or given by the current value of $wp$, if the *instance_access* is omitted). The transitions are grouped into "synchronisation-groups" to indicate with what other transitions (if any) a given transition is to be executed in parallel (only in the case of a systemprocess may a synchronisation-group contain more than one transition).

**Applications:** The commands *display*, *file_display* and *select*.

**Syntax:** 
```
"$exec_time_min"  "("  TRANS_ACCESS ")" ]
"setmin"  "("  TRANS_ACCESS ")"
```

**Semantics:** The function accesses the minimum execution time assigned to transitions of the module instance indicated by "the " trans_access ". If the "instance_access" within the "trans_access" is omitted the function accesses the minimum transition execution time of the instances pointed by the current value of $wp$. The "$exec_time_min" for a given transition should be not greater than "$exec_time_max". The initial, by default, value of the "$exec_time_min" is 0.0. After each Edb *restart* command the "$exec_time_min" is reinitialised.

**Application:** The commands display, file_display and in left-hand-side of assignments (the right-hand-side expression must evaluate to a real value) to modify the minimum execution time of transitions within the given module instance. For example, to set to 3.8 the minimum execution time of a transition numbered 3 within the module instance numbered 4 the following modify command may be issued: $exec_time_max(4->3) := 3.8.

The modification of the "$exec_time_min", for a given transition, will be not immediately accepted if proposed at the moment the transition is executing (started but not yet terminated) and will be not accepted at all if the proposed value is greater than the current value of the "$exec_time_max".

**Note:** In the previous version of Edb this function had as argument the "instance_access"

**Syntax:** 
```
"$exec_time_max"  "("  TRANS_ACCESS ")" ]
"setmax"  "("  TRANS_ACCESS ")"
```

**Semantics:** The function accesses the maximum execution time assigned to transitions of the module instance indicated by "the " trans_access ". If the "instance_access" within the "trans_access" is omitted the function accesses the maximum transition execution time of the instances pointed by the current value of $wp$. The "$exec_time_max" for a given transition should be not smaller than "$exec_time_min". The initial, by default, value of the "$exec_time_max" is 0.0. After each Edb *restart* command the "$exec_time_max" is reinitialised.

**Application:** The commands *display*, *file_display* and in left-hand-side of assignments (the right-hand-side expression must evaluate to a real value) to modify the maximum execution time of transitions within the given module instance. For example, to set to 5.7 the maximum execution time of a transition numbered 3 within the module instance numbered 4 the
following modify command may be issued: $exec_time_max(4->3) := 5.7. The modification of the "$exec_time_max", for a given transition, will be not immediately accepted if proposed at the moment the transition is executing (started but not yet terminated) and will be not accepted at all if the proposed value is smaller than the current value of the "$exec_time_min".

**Note:** In the previous version of Edb this function had as argument the "instance_access"

**Syntax:**
```
"$export_var_id" ["("INSTANCE_ACCESS ")"] | 
"$evi" ["("INSTANCE_ACCESS ")"]
```

**Semantics:** The function returns the identifiers of all exported variables declared within a given module instance. If the "instance_access" is omitted, then the module instance accessed is that given by "$wp". See also "$local_variable_id", and "$module_variable_id"

**Application:** Display and file-display commands.

**Syntax:**
```
"$external_ip_id" ["("INSTANCE_ACCESS ")"] | 
"$eii" ["("INSTANCE_ACCESS ")"]
```

**Semantics:** The function returns the identifiers of all external interaction points declared within a given module instance. If the "instance_access" is omitted, then the module instance accessed is that given by "$wp". See also "$internal_ip_id" and "$all_ip_id"

**Application:** Display and file-display commands.

**Syntax:**
```
$firing_steps" | 
"$fs"
```

**Semantics:** The function returns the current limit of simulation in number of fired transitions. By default the value is assumed 1, which means that the user gets control after each executed transition. The reset Edb command resets the current value always to 1.

**Application:** The limit may be set up if "$fs" appears at the left-hand-side of an assignment (modify commands). For example $fs := 5 sets the limit to 5, and thus if the user issues the continue command then the simulation will be stopped after executing 5 transitions. An associated "firing-step-counter" (see "$current_firing_step") sums up numbers of transitions executed to stop the simulation when the counter achieves the assigned value. If this happens the "firing-step-counter" is reset to 0 and the command "continue" repeats the same interval. If the value of "$fs" is changed by the user the counter "firing-step-counter" is reset to 0 and...
the command *continue* run the simulation with the new limit. If the simulation stopped earlier than the limit defined by the $fs value (run-time error occurred during simulation, command "break" executed within an "observer" or the simulation duration limit defined by "$real_time_limit" has been achieved), then the next *continue* command run the simulation with the unchanged limit. It gives the user the possibility to interact with the Edb in intervals measured in number of executed transitions, e.g., if the number is 10, then the user gets control every 10 fired transitions. If no such limit is desired the value can be assigned "infinity" ("$infinity").

**Syntax:**

```
"$global_trans_statistics"
"$gtrstat"
```

**Semantics:** The function accesses, for each module body defined within the simulated specification, the list of its transitions and gives for each transition the number of times it has been fired.

**Application:** The commands display and file_display.

**Note 1:** In the previous version of Edb this function was called $transition_statistics. The $transition_statistics function has now a different definition.

**Note 2:** One difference with newly defined $transition_statistics (instance_access) consists in having statistics "per module's bodies", i.e., if there is more then one instance of a given module (module_header, module_body) then the number of times a transition is fired is a global measure for transitions of all instances. Another difference consists in the fact that the statistics is not reinitialised after execution of the restart Edb command.

**Syntax:**

```
"$highest_instance_nb
"$hin"
```

**Semantics:** The function returns the highest instance number (integer) actually created.

**Application:** The cwp ("change_wp") navigation command, the commands display, file_display, within assignment (right-hand-side) and expressions and as an argument of any predefined function having as argument the instance_access.

**Syntax:**

```
"$highest_trans_nb ["("INSTANCE_ACCESS")"]
"$htn" ["("INSTANCE_ACCESS")"]
```

**Semantics:** The function returns the highest transition number (integer) assigned to transitions within the module instance indicated by the "instance_access". If the "instance_access" is omitted, then the module instance accessed is that given by "$wp".
**Appendix 2**

**Application:** The commands `display`, `file_display` and within assignment (right-hand-side) and expressions.

**Syntax:**
```
"$hierarchy" ["("INSTANCE_ACCESS")"] |
"$h" ["("INSTANCE_ACCESS")"]
```

**Semantics:** The function accesses information concerning the current hierarchical tree structure of module instances whose root is the module instance identified by "instance_access" (or given by "$wp" if "instance_access" is omitted).

**Application:** The commands `display` and `file_display`.

**Syntax:**
```
"$infinity" |
"$inf"
```

**Semantics:** The function value (which in fact is 0) has to indicate that a limit ("$real-time_limit" or "$firing_steps") is not bound (i.e., it is "infinity").

**Application:** On the right-hand-side of an assignment to "$real_time_limit" ("infinity" is its default value) and "$firing_steps" (which has 1 by default).

**Syntax:**
```
"$internal_ip_id" ["("INSTANCE_ACCESS ")"] |
"$iii" ["("INSTANCE_ACCESS ")"]
```

**Semantics:** The function returns the identifiers of all internal interaction points declared within a given module instance. If the "instance_access" is omitted, then the module instance accessed is that given by "$wp". See also "$external_ip_id" "("INSTANCE_ACCESS ")" and "$all_ip_id" "("INSTANCE_ACCESS ")"

**Application:** Display and file-display commands.

**Syntax:**
```
"$interaction_point" ["("INSTANCE_ACCESS")"] |
"$ip" ["("INSTANCE_ACCESS")"]
```

**Semantics:** The function accesses information concerning all the interaction points of the instance indicated by "instance_ip_access". If "instance_access" is omitted, then it is assumed to be that of "$wp". For every interaction point thus accessed, the information consists of:
- the name of the interaction point and identification of the module instance to which it belongs;
- the interaction point qualification (external or internal) and the class (common or individual) of its queue;
- if the interaction point is connected, then the information includes the name of the connected interaction point and identification of its module instance;
- if the interaction point is attached, then the information includes the name(s) of the attached interaction point(s) and identification of it’s (their) module instance(s).

**Application:** The commands *display* and *file_display*.

**Syntax:**
```
"$interaction_point" [ "("IP_ACCESS")"]
"$ip" [ "("IP_ACCESS")"]
```

**Semantics:** The function accesses information concerning the interaction point defined by the parameter "ip". If "instance_access" (within "ip_access") is omitted, then it is considered to be that of "$wp". If "ip_access" is omitted the information concerns all interaction points of the module instance given by "$wp". For every interaction point thus accessed, the information consists of:

- the name of the interaction point and identification of the module instance to which it belongs;
- the interaction point qualification (external or internal) and the class (common or individual) of its queue;
- if the interaction point is connected, then the information includes the name of the connected interaction point and identification of its module instance;
- if the interaction point is attached, then the information includes the name(s) of the attached interaction point(s) and identification of it’s (their) module instance(s).

**Application:** The commands *display* and *file_display*.

**Syntax:**
```
"$is_break"
```

**Semantics:** The function returns true if the simulation was stopped by a "break" command.

**Application:** The commands *display* and *file_display* and in boolean expressions of commands controlling automatically the overall simulation session (see example given for "$is_error" below).

**Syntax:**
```
"$is_connected" ["("ip_access")"]
"$is_conn" ["("ip_access")"]
```
Semantics: The function returns "true" if the interaction point accessed by "ip_access" is connected to an interaction point of a sibling module instance of the module instance determined by "instance_access" within "ip_access" or taken, by default, to be that of "$wp". If "ip_access" is omitted, then the function returns "true" if there is an interaction point "p" of the instance indicated by "$wp" that satisfies "$is_conn(p)" (i.e., if there is a sibling instance connected to the current instance).

Application: The commands display and file_display and in Boolean expressions.

Syntax: "$is_deadlock"

Semantics: The function returns true if the simulation was stopped by a deadlock.

Application: The commands display and file_display and in boolean expressions of commands controlling automatically the overall simulation session (see Example given for "$is_error" below).

Syntax: "$is_delay_transition" "("TRANS-ACCESS")" |
"$is_dtr" "("TRANS-ACCESS")"

Semantics: The function returns "true" if the transition indicated by "trans-access" (could also be "$last_transition_id", which means that the test concerns the last fired transition) is a delay transition, i.e., has a "delay" clause. If "instance_access" within "trans-access" is omitted (or for $last_transition_id) it is that given by "$wp".

Applications: The commands display and file_display and in boolean expressions.

Syntax: "$is_down_attached" ["("IP_ACCESS")"] |
"$is_datt" ["("IP_ACCESS")"]

Semantics: The function returns "true" if the interaction point accessed by "ip_access" is attached to an interaction point of a child module instance of the module instance determined by "instance_access" within "ip_access" or taken, by default, to be that of "$wp". If "ip_access" is omitted, then the function returns "true" if there is an interaction point "p" of the instance indicated by "$wp" that satisfies "$is_datt(p)" (i.e., if there is a child instance attached to the current instance).

Application: The commands display and file_display and in boolean expressions.
Syntax: "$is_enabled_delay_trans" "("TRANS_ACCESS")"
"$is_en_dtr" "("TRANS_ACCESS")"

Semantics: The function returns "true" if "trans_access" identifies a delay transition and this transition is currently enabled, i.e., the associated timer is running. Otherwise, it returns "false" (a warning is displayed if "trans_access" does not identify a delay transition). The predefined function "$last_transition_id" can be used as "trans_access", in which case the test concerns the last fired transition and the instance is that indicated by "$last_instance". In all other cases, if the "instance_access" within "trans_access" is omitted, the instance is that indicated by "$wp".

Application: Expressions, display command, right-hand-side of assignment.

Syntax: "$is_error"

Semantics: The function returns "true" if the simulation was stopped by a run-time error.

Application: The commands display and file_display and in boolean expressions of commands automatically controlling the overall simulation session (see Example below).

Example:

Assume the simulator is initialised and the following command is defined:

```
edb> $time := 3600;\nso { if (3->x) > 0 then break fi };\ndo if $is_break \n    then file_display breakfile $tfs; continue \n    else if $is_error \n        then file_display errorfile $last_transition_id; exit \n        else $is_deadlock \n            then file_display deadlockfile $tfs; exit \n            else continue \n        fi \n    fi \nfi \n```

The meaning of the above command sequence is the following. The simulation stops after 1 hour if no deadlock or run-time error occurred earlier. If the deadlock happened the total number of fired transitions ("$tfs") is noted on the file "deadlockfile" and the simulation halts. If a run-time error occurs, the information concerning the last transition executed is saved on the file "errorfile" and the simulation halts. If the simulation was stopped by a "break" command then it continues after saving the number of steps on the file "breakfile". Note that this sequence of commands may also form a macro-command (see below).
Syntax: "$is_input_transition" "("TRANS-ACCESS")" | "$is_intr" "("TRANS-ACCESS")"

Semantics: The function returns "true" if the transition indicated by "trans-access" (could be also "$last_transition_id", which means that the test concerns the last fired transition) is an input transition, i.e., has a "when" clause. If "instance_access" within "trans-access" is omitted (or for $last_transition_id) it is that given by "$wp".

Applications: The commands display and file_display and in boolean expressions.

Syntax: "$is_interrupt"

Semantics: The function returns "true" in response to a UNIX interruption "INTR" (provoked, for example, by "CTRL_C" or sometimes by the "break" key on the keyboard). Such an interruption breaks the simulation (after the simulation step is completed) and gives hand to the user provided no specific action has been specified within an observer. Such an action should be associated to the condition $is_interrupt.

Applications: The commands display and file_display and in boolean expressions.

Syntax: "$is_spont_transition" "("TRANS-ACCESS")" | "$is_sptr" "("TRANS-ACCESS")"

Semantics: The function returns "true" if the transition indicated by "trans-access" (could be also "$last_transition_id", which means that the test concerns the last fired transition) is a spontaneous transition, i.e., does not have a "when" clause. If "instance_access" within "trans-access" is omitted (or for $last_transition_id) it is that given by "$wp".

Applications: The commands display and file_display and in boolean expressions.

Syntax: "$is_up_attached" "("IP_ACCESS")" | "$is_uatt" "("IP_ACCESS")"

Semantics: The function returns "true" if the interaction point accessed by "ip_access" is attached to an interaction point of the parent module instance of the module instance determined by "instance_access" within "ip_access" or taken by default to be that of "$wp". If "ip_access" is omitted, then the function returns "true" if there is an interaction point "p" of the instance indicated by "$wp" that satisfies "$is_uatt(p)" (i.e., if the parent instance is attached to the current instance).
**Application:** The commands *display* and *file_display* and in boolean expressions.

**Syntax:**

```
"$is_valid_expression"  "("EXPRESSION")"
"$is_ve"  "("EXPRESSION")"
```

**Semantics:** The function returns "true" if the argument of this function is indeed an expression and if this expression is valid i.e., when all accessed objects within this expression are defined. Informally, an expression is an item which can follow an "if-command" or which can be used as a right-hand-side in an assignment. See the Appendix 4 for the predefined functions that can be used as expressions (category "E" in the table).

**Application:** The commands *display* and *file_display* and within IF-commands.

**Example:**

```
if $is_valid_expression (3->X) then display 3->X fi
```

will result in displaying the value of variable X from the instance 3 only if the instance 3 and the variable X within this instance both exist

```
if $is_valid_expression ($left(4)) then display $interaction_point(4) fi
```

will result in displaying the information on the interaction points of the module instance 4 only if such instance exists.

**Syntax:**

```
"$last_instance_nb"
"$lins"
```

**Semantics:** The function accesses the instance identification of the last fired transition.

**Application:** The *cwp* ("change_wp") navigation command, the commands *display* and *file_display*, within boolean expressions and as an argument of any predefined function having as argument the *instance_access*.

**Example:**

```
cwp $lins
```

change the module instance context to which the "working-pointer" is currently pointing

```
if $lins = 3 then ... fi
```

tests whether the last fired transition was one of the transitions of the module instance_3.
Syntax: "$last_transition_attach" | "$ltratt"

Semantics: The function accesses information concerning the attachments (if any) established by the last fired transition (for each executed attach-statement, the identification of the interaction points being attached).

Application: The commands display and file_display.

Syntax: "$last_transition_connect" | "$ltrc"

Semantics: The function accesses information concerning the connections (if any) established by the last fired transition (for each executed connect-statement, the identification of the interaction points being connected)

Application: The commands display and file_display.

Syntax: "$last_transition_delay" | "$ltrd"

Semantics: The function accesses information concerning the delay-values of the last fired transition (if it is not a delay-transition an appropriate message is displayed). The information consists of:

- the actual values of expressions E1 and E2 of the delay-clause before the transition was fired;

- the time advance after the transition was fired (calculated as sum of two components; the first component is the time value randomly chosen between the actual values of E1 and E2 and the second component is the time value randomly chosen between the minimum and maximum value of the execution time associated to this transition (by default these values are equal zero but they may be set by the user to the other values by means of predefined functions $exec_time_max and $exec_time_min).

Application: The commands display and file_display.

Syntax: "$last_transition_detach"
Semantics: The function accesses information concerning the detachments (if any) established by the last fired transition (for each executed detach-statement, the identification of the interaction points being detached).

Application: The commands display and file_display.

Syntax: "$ltrdet"

Semantics: The function accesses information concerning the disconnections (if any) established by the last fired transition (for each executed disconnect-statement, the identification of the interaction points being disconnected).

Application: The commands display and file_display.

Syntax: "$last_transition_disconnect" | "$ltrdc"

Semantics: The function accesses information concerning the initialisations (if any) established by the last fired transition (for each executed init-statement, the identification of the instances (instance-number, module-header-id, module-body-id) being initialised).

Application: The commands display and file_display.

Syntax: "$last_transition_init" | "$ltrinit"

Semantics: The function accesses the identification (instance-number, transition-number or transition-id) of the last fired transition. A transition-number (integer type) corresponds to the order the transition was textually declared within a body. The first declared transition has the number 0; when any-clause is used within a transition declaration a transition-number is assigned to each expanded transition (see the end of the Section 4.1).

Application: The commands display and file_display, within assignments (right-hand-side), boolean expressions, and as "trans-access".

Examples:
#var := $last_transition_id
assign to the scratch-pad variable "var" the last fired transition-number

if $last_transition_id = 3 then ... fi
tests whether the last fired transition-number is 3

if $is_input_transition($last_transition_id) then ... fi
tests whether the last fired transition is an input_transition.

**Syntax:**
"$last_transition_input" | "$ltri"

**Semantics:** The function accesses information concerning the input interaction removed from a queue when the last transition was fired. The information consists of:
- the identification of the interaction point to which the queue is associated;
- the interaction identification; if the interaction has parameters their values are also displayed.

**Application:** The commands *display* and *file_display*.

**Syntax:**
"$last_trans_input_inter_name" | "$ltri_inter"

**Semantics:** The function returns the name of the interaction input (removed from a queue) when the last transition was fired.

**Application:** Expressions with the operators "=" and "<>" and the *display* command.

**Note1:** The syntax for comparison expressions is, e.g., for equality:
"$last_trans_input_inter_name" "=" IDENTIFIER |
IDENTIFIER "=" "$last_trans_input_inter_name"

**Note2:** The function should be used after checking if any interaction was input, using "$is_input_transition($last_transition_id)" (see the example below). A comparison expression that calls the function in improper circumstances (e.g., no fired transition or no input interaction) has the value "false".

**Example:**

A typical application of this function is the detection with an observer that a particular interaction was consumed by the last fired transition (i.e., the occurrence of an "input event"). The following observer checks after each computation step if a DataReq interaction was input (by any module instance):

```plaintext
so detect_input_data_req \n  if ($is_intr($ltrid)) then\n    if ($ltri_inter = DataReq) then ... fi;\n```
Syntax: "$last_trans_input_inter_param" "("INTER_PARAMETER")" | "$ltri_inter_p" "("INTER_PARAMETER")"

Semantics: The function accesses the interaction input when the last transition was fired and returns the value of the parameter (or parameter component) indicated by "INTER_PARAMETER". It must be a simple, nonstructured type.

Application: Expressions, display command, right-hand-side of assignment.

Note: The function should be used after checking that an interaction of the desired type was input, using "$is_input_transition($last_transition_id)" and "$last_trans_input_inter_name" (see the example below). If the function is called in improper circumstances (e.g., no input interaction or wrong type), the left-hand-side of an assignment is not modified and the value of a comparison expression is "false".

Example:

The function can be used for selecting relevant "input events". The following observer checks after each computation step whether an interaction DataReq with a value of the parameter sdu.lng greater than 100 was input:

```plaintext
so detect_data_req_large \n  if ($is_intr($ltrid)) then\n    if ($ltri_inter = DataReq) then\n      \% a DataReq interaction was input
      if ($ltri_inter_p(sdu.lng) > 100) then ... fi;
    fi;
  fi;
```

Syntax: "$last_trans_input_inter_point" | "$ltri_ip"

Semantics: The function returns the identification of the interaction point ("ip_access" format) for which an interaction was removed from queue when the last transition was fired.

Application: Expressions with the operators "+" and ",<>" and the display command.

Note 1: The syntax for comparison expressions is, e.g., for equality:

```
"$last_trans_input_inter_point" "=" IP_ACCESS | IP_ACCESS "=" "$last_trans_input_inter_point"
```

The "instance_access" within the "ip_access" identifying the interaction point can be omitted. In this case, the interaction point is assumed to belong to the module instance which fired the last transition, i.e., that given by "$last_instance".
Note 2: The function should be used after checking if any interaction was input, using "$is_input_transition($last_transition_id)" (see the example below). A comparison expression that calls the function in improper circumstances (e.g., no fired transition or no input interaction) has the value "false".

Examples:

1). The last fired transition removed an interaction from the queue of the interaction point $TSAP[1]$ of the instance 3:

   edb> display $ltri_ip
   3->TSAP[1]

2). The following observer checks after each computation step whether an interaction $DataReq$ was input from the interaction point $TSAP[1]$ of the module instance 3:

   so detect_input_data_req \n   if ($lims=3) and ($is_intr($ltrid)) then\n     if ($ltri_inter = DataReq) and ($ltri_ip = TSAP[1]) then ... fi;\n   fi;\n   
Syntax: "$last_transition_outputs" | "$ltro"

Semantics: The function accesses information concerning the outputs of the last fired transition. For each executed output-statement:

- the identification of the interaction point by which an interaction was sent and the interaction identification; if the interaction has parameters their values are also displayed,

- the identification (instance-number) of the instance and its interaction point to which the interaction was sent.

Application: The commands display and file_display.

Syntax: "$last_trans_output_inter_name" | "$ltro_inter" "("SCRATCH_INT_ACCESS")" | "("SCRATCH_INT_ACCESS")"

Semantics: The function returns the name of an interaction output by the last fired transition. The parameter selects one of the output interactions and its value must be in the range 1.."$last_trans_outputs_number" (the value $n$ selects the $n$--th output interaction, as ordered by the sequential execution of the Estelle instructions of the last fired transition).
**Application:** Expressions with the operators "=" and "<>" and the *display* command.

**Note 1:** The syntax for comparison expressions is, e.g., for equality:

```
"$last_trans_output_inter_point" 
IDENTIFIER | 
IDENTIFIER 
"$last_trans_input_inter_name"
```

**Note 2:** The function should be used after checking if any interaction was output, using "$last_trans_outputs_number". A comparison expression that calls the function in improper circumstances (e.g., no fired transition or no output interaction) has the value "false".

**Example:**

A typical application of this function is the detection with an observer that a particular interaction was output by the last fired transition (i.e., the occurrence of an "output event"). The following observer detects that a *DataInd* interaction was output (by any module instance) and in such a case stops the simulation:

```
so stop_on_data_ind \
#outnb := $ltro_nb\n
if (#outnb >0) then
  #cnt := 1;
  do
    if ($ltro_inter(#cnt) = DataInd) then exit; fi;
    #cnt := #cnt+1;
    if (#cnt > #outnb) then exit; fi;
  od;
fi;
if (#cnt <= #outnb) then break; fi;
```

**Syntax:**

```
"$last_trans_output_inter_param" 
("SCRATCH_INT_ACCESS", "INTER_PARAMETER") | 
"$ltro_inter_p" 
("SCRATCH_INT_ACCESS", "INTER_PARAMETER")
```

**Semantics:** The function selects an interaction output by the last fired transition and returns the value of the parameter (or parameter component) specified by "INTER_PARAMETER". It must be of simple, nonstructured type. The desired output interaction is indicated by the function's first parameter, an integer value in the range 1.."$last_trans_outputs_number" (the value n selects the n-th output interaction, as ordered by the sequential execution of the Estelle instructions of the last fired transition).

**Application:** Expressions, *display* command, right-hand-side of assignment.

**Note:** The function should be used after checking with "$last_trans_outputs_number" and "$last_trans_output_inter_name" that the last fired transition output an interaction of the desired type. If the function is called incorrectly (e.g., no output interaction, type mismatch or wrong selector value), the left-hand-side of an assignment is not modified and the value of a comparison expression is "false".

**Example:**

© 1989 -1990, BULL S.A

Last change: mai 29, 2000
The function can be used for selecting relevant "output events". The following observer checks whether the last fired transition output a DataInd interaction with a value of the parameter sdu.lng equal to 100:

\[
\text{so detect_data_ind} \\
\text{#outnb} := \$ltro_nb; \\
\text{if } (#outnb > 0) \text{ then } \% \text{ outputs exist} \\
\text{#cnt} := 1; \\
\text{do} \\
\text{if } (\$ltro_inter(#cnt) = \text{DataInd}) \text{ then} \\
\% \text{ a DataReq interaction was output} \\
\text{if } (\$ltro_inter_p(#cnt, \text{sdu.lng}) = 100) \text{ then ... fi;} \\
\text{fi;} \\
\text{#cnt} := \#cnt + 1; \\
\text{if } (#cnt > #outnb) \text{ then exit; fi;} \\
\text{od;} \\
\text{fi;} \\
\]

**Syntax:**

"$last_trans_output_inter_point" ("SCRATCH_INT_ACCESS") | "$ltro_ip" ("SCRATCH_INT_ACCESS")

**Semantics:** The function returns the identification of an interaction point ("ip_access" format) through which an interaction was output by the last fired transition. The parameter selects one of the output interactions and its value must be in the range 1.."$last_trans_outputs_number" (the value n selects the n-th output interaction, as ordered by the sequential execution of the Estelle instructions of the fired transition).

**Application:** Expressions with the operators "=" and "<>" and the display command.

**Note 1:** The syntax for comparison expressions is, e.g., for equality:

"$last_trans_output_inter_point" "=" IP_ACCESS | IP_ACCESS "=" "$last_trans_output_inter_point"

The "instance_access" within the "ip_access" identifying the interaction point can be omitted. In this case, the interaction point is assumed to belong to the module instance which fired the last transition, i.e., that given by "$last_instance".

**Note 2:** The function should be used after checking if any interaction was output, using "$last_trans_outputs_number". A comparison expression that calls the function in improper circumstances (e.g., no fired transition, no output interaction, wrong selector value) has the value "false".

**Example:**

The following observer checks whether the last fired transition output an interaction DataInd through the interaction point TSAP[1] of the module instance 3:

\[
\text{so detect_data_ind} \\
\text{#outnb} := \$ltro_nb; \\
\text{if } (#outnb > 0) \text{ then } \% \text{ outputs exist} \\
\text{#cnt} := 1; \\
\]
do
    if ($ltro_ip(#cnt)=3->TSAP[1]) and ($ltro_inter(#cnt)=DataInd) then ... fi;
    #cnt := #cnt+1;
    if (#cnt > #outnb) then exit; fi;
    od;
fi;
}

Syntax: "$last_trans_outputs_number" | "$ltro_nb"

Semantics: The function returns the number of interactions output by the last fired transition.

Application: Expressions, display command, right-hand-side of assignment.

Syntax: "$last_transition_releases" | "$ltrrel"

Semantics: The function accesses information concerning the releases (if any) established by the last fired transition (for each executed release-statement, the identification of the instances (instance-number, module-header-id, module-body-id) being released).

Application: The commands display and file_display.

Syntax: "$last_transition_state" | "$ltrst"

Semantics: The function accesses the "from" and "to" state identifiers (if they exist) of the last fired transition.

Application: The commands display and file_display. When used within boolean expressions (equality comparisons) only "from" state is accessed.

Example:

if $ltrst = IDLE then ... fi

tests whether the "departure" control state of the last fired transition was that named "IDLE".
Syntax: "$last_transition_terminates" | "$ltrt"

Semantics: The function accesses information concerning the terminates (if any) established by the last fired transition (for each executed terminate-statement, the identification of the instances (instance-number, module-header-id, module-body-id) being terminated.

Application: The commands display and file_display.

Syntax: "$left" ["("INSTANCE_ACCESS ")] | "$l" ["("INSTANCE_ACCESS ")]''

Semantics: The function returns the value (integer) of the unique number assigned to the left sibling instance of the given module instance. If the "instance_access" is omitted, then the module instance accessed is that given by "$wp" (currently pointed by the "working-pointer" - see in Sec. 4.1 the description of a module instance context data structure). A sibling module instance of an existing module instance always exists since such instances are round-chained.

Application: The cwp ("change_wp") navigation command, the display and file-display commands, the right-hand-side of assignments, within boolean expressions (comparisons with any integer) and as an argument of any predefined function having as argument the instance_access.

Syntax: "$least_fired_transition " ["("INSTANCE_ACCESS")""]
"$lftr" ["("INSTANCE_ACCESS")"]

Semantics: The function accesses, information concerning the least fired transition of the module instance indicated by "instance_access", (or by the current value of $wp - if "instance access" is omitted). The information consists of the module instance number and of its header and body identifiers, of the transition number (or transition name - if such has been given within the simulated specification) and of the number of times this transition has been fired. If more than one such transition exists, then the function chooses one of them randomly.

The $least_fired_transition function returns -1 when there is no transition in the indicated module instance.

The $least_fired_transition function is reinitialised to 0 after each restart Edb command.
**Application:** The commands `display` and `file_display`, within boolean expressions (equality comparisons with transition number) and as right-hand-side of assignments.

**Example:**

```
if $\text{least_fired_transition}(3) \not= -1 \text{ then } \text{<Edb_command_sequence>} \text{ fi}
```

permits to execute the `<Edb_command_sequence>` only if there is at least one transition declared in the module instance 3.

**Syntax**:

```
"$local_var_id" ["("INSTANCE_ACCESS ")"] |
"$lvi" ["("INSTANCE_ACCESS ")"]
```

**Semantics:** The function returns the identifiers of all local variables declared within a given module instance. If the "instance_access" is omitted, then the module instance accessed is that given by "$wp". See also "$export_variable_id", and "$module_variable_id"

**Application:** Display and file-display commands.

**Syntax**:

```
"$\text{locally_firable_transitions}" ["("INSTANCE_ACCESS")"] |
"$\text{lft}" [ "("INSTANCE_ACCESS")"]
```

**Semantics:** The function returns information concerning all fireable transitions (the set can be empty) of the module instance indicated by "instance_access". If "instance_access" is omitted, then it is that given by "$wp". Note that, due to the parent/children priority, locally fireable transitions are not necessary included in "executable-transitions" (see "$et") for a given system. For each locally fireable transition the accessed information consists of:

- the module instance identification (instance-number, module-header-id, module-body-id), the transition identification (transition-number and transition-name -if such exists), the module attribute and :

  - the control (major) states (if they exist) corresponding to the "from" and "to" clauses of the transition;

  - the interaction point and interaction identification corresponding to the "when" clause (if exists) of the transition;

  - the bounds of the "delay" clause.

**Applications:** The commands `display` and `file_display`.

**Syntax:**  

```
"$\text{macro}"
```
**Semantics:** The function accesses the current list of user defined macros.

**Application:** The commands `display`, `file_display`.

**Syntax:**
```
"$most_fired_transition" ["("INSTANCE_ACCESS ")] | "$mftr"
""["("INSTANCE_ACCESS ")]"
```

**Semantics:** The function accesses, information concerning the most fired transition of the module instance indicated by "instance_access", (or by the current value of $wp - if "instance_access" is omitted). The information consists of the module instance number and of its header and body identifiers, of the transition number (or transition name - if such has been given within the simulated specification) and of the number of times this transition has been fired. If more than one such transition exists, then the function chooses one of them randomly.

The $most_fired_transition function returns -1 when there is no transition in the indicated module instance.

The $most_fired_transition function is reinitialised to 0 after each restart Edb command.

**Application:** The commands `display` and `file_display`, within boolean expressions (equality comparisons with transition number) and as right-hand-side of assignments.

**Syntax:**
```
"$module_param_id" ["("INSTANCE_ACCESS ")] | "$mpi"
""["("INSTANCE_ACCESS ")]"
```

**Semantics:** The function returns the identifiers of all module parameters declared within the header of a given module instance. If the "instance_access" is omitted, then the module instance accessed is that given by "$wp". See also "$export_var_id", and "$local_var_id"

**Application:** `Display` and `file-display` commands.

**Syntax:**
```
"$module_var_id" ["("INSTANCE_ACCESS ")] | "$mvi"
""["("INSTANCE_ACCESS ")]"
```

**Semantics:** The function returns the identifiers of all module variables declared (after the keyword `modvar`) within a given module instance. If the "instance_access" is omitted, then the module instance accessed is that given by "$wp". See also "$export_var_id", and "$local_var_id"

**Application:** `Display` and `file-display` commands.
Syntax: 
"$observer" [ "(" OBSERVER-NUMBER ")" ] |
"$ob" [ "(" OBSERVER-NUMBER ")" ]

Semantics: The function accesses the observer indicated by "observer-number". If "observer-number" is omitted, then numbers of all currently present observers are displayed together with their current status (enabled, disabled).

Application: The commands display and file_display.

Syntax: 
"$predefined_functions" |
"$pf"

Semantics: The function returns the list of all predefined functions.

Application: The commands display and file_display.

Syntax: 
"$queue"["("INSTANCE_ACCESS")"]

Semantics: The function provides access to information concerning the current contents of the queues associated to all interaction points of the module instance given by "instance_ip_access". If "instance_access" is omitted, then it is considered to be that of "$wp". For each queue, the information thus accessed consists of:

- the identification of the interaction point to which the queue belongs;
- the current size of the queue, i.e., the number of interactions actually in the queue;
- the queue content.

Application: The commands display and file_display.

Syntax: 
"$queue" [ "("IP_ACCESS")" ]

Semantics: The function provides access to information concerning the current contents of the queue of the interaction point defined by "ip_access". If "instance_access" (within "ip_access") is omitted, then it is considered to be that of "$wp". If "ip_access" is omitted, the information concerns queues of all interaction points of the module instance given by "$wp". For each queue, the information thus accessed consists of:
- the identification of the interaction point to which the queue belongs,
- the current size of the queue, i.e., the number of interactions in the queue;
- the queue content

**Application:** The commands *display* and *file_display*.

**Syntax:**
"$queue" "("IP_ACCESS"," QUEUE-POSITION")"

QUEUE-POSITION = \<integer\> | "#" \<identifier\>

**Semantics:** The function returns the current contents (interaction-id, parameters’ values) of the queue on a given "queue-position". It concerns the queue of the interaction point defined by "ip_access". If "instance_access" (within "ip_access") is omitted, then it is the queue of the interaction point given by "ip_access" of the module instance identified by the current value of "$wp". The top element of the queue has position 1.

**Application:** The commands *display* and *file_display*.

**Syntax:**
"$queue" "("IP_ACCESS"," QUEUE-POSITION"," INTER-PARAMETER")"

QUEUE-POSITION = \<integer\> | "#" \<identifier\>

**Semantics:** The function returns the current value of the parameter (or its component) of the interaction on a given "queue-position". It concerns the queue of the interaction point defined by "ip_access". If "instance_access" (within "ip_access") is omitted, then it is assumed that of "$wp". The top element of the queue has position 1.

**Application:** The commands: *display*, *file_display* and *modify*. It can be used as an access in an expression if the accessed value is of a simple type.

**Syntax:**
"$queue_interaction" "("IP_ACCESS"," QUEUE-POSITION")"

QUEUE-POSITION = \<integer\> | "#" \<identifier\>

**Semantics:** The function returns the name of the interaction on a given "queue-position".
**Application:** The commands *display* and *file_display*. Within the expression, the only admissible operations are "=" or "<>".

**Syntax:**

```
"$queue_size" "("IP_ACCESS")" | "$qs" "("IP_ACCESS")"
```

**Semantics:** The function returns the current size (or length, i.e., the number of interactions currently present in the queue) of the queue associated to the interaction point defined by "ip_access". If "instance_access" (within "ip_access") is omitted, then the information concerns the queue of the interaction point given by "ip_access" of the module instance identified by the current value of "$wp".

**Application:** The commands *display* and *file_display* and as an access in an expression.

**NOTE:** This function may be compared with an integer value to enable congestion detection. When congestion has been detected the user may also request to drop (using *delete_from_queue* Edb command) any new coming message until the situation becomes normal. These actions are usually specified within a user-defined observer executed after each transition execution.

**Warning:** The $qsize abbreviation disappeared in this version.

**Syntax:**

```
"$queue_statistics" [ "("INSTANCE_ACCESS")" ] | "$qstat" [ "("INSTANCE_ACCESS")" ]
```

**Semantics:** The function returns the list of interaction points of the module instance indicated by "instance_access" and gives for each interaction point statistics of the message traffic vehiculed through it. If the "instance_access" is omitted the statistics concern the module instance pointed by the current value of $wp. The statistics displayed are related to the queues associated to all interaction points within the given instance. The statistics concerns the maximum and mean number of messages stored in each queue and the number of interactions outputted through each interaction point.

**Application:** The commands display and *file_display*. The execution of the command

```
display $queue_statistics
```

creates the file

```
specname.qstat_<INSTANCE_ACCESS>.gr
```

(where the specname corresponds to the currently simulated specification name) which can be used by the *plot* Edb command to plot the statistics.

**Syntax:**

```
"$random" | "$rd"
```
Semantics: The function returns the current "seed" of the random function of the Edb.

Application: Assignments and expressions.

Syntax: "$right" ["("INSTANCE_ACCESS ")"] | "$r" ["("INSTANCE_ACCESS ")"]

Semantics: The function returns the value of the unique number assigned to the right sibling instance of the given module instance. If the "instance_access" is omitted, then the module instance accessed is that given by "$wp" (currently pointed by the "working-pointer" - see in Sec. 4.1 the description of a module instance context data structure). A sibling module instance of an existing module instance always exists since such instances are round-chained.

Application: The cwp ("change_wp") navigation command, the display and file-display commands, the right-hand-side of assignments, within boolean expressions (comparisons with any integer) and as an argument of any predefined function having as argument the instance_access..

Syntax: "$real-time_limit"
"$time"

Semantics: The function returns the current time limit of simulation in seconds. By default the value is assumed to be "infinity", which means that no time limit is defined. The reset Edb command resets the current value always to "infinity".

Application: The time limit may be set up if "$real-time_limit" ("$time") appears at the left-hand-side of an assignment (modify commands). The assignment sets the time limit in seconds (it has nothing to do with the time e.g., of simulated delays) after which the simulation stops without any user intervention (the execution may stop earlier when a "break" command within an observer has been executed or a deadlock is achieved or a runtime error occurs or another limit is attained, see "$fs"). By default the value is assumed to be "infinity", i.e., the simulation goes on, in principle, forever. The time limit is consulted in each simulation step. If the $time value is modified by the user the previous value is cancelled and the command "continue" restarts the simulation with the new time limit. If the limit has been achieved and is not modified by the user the command "continue" restarts the simulation with the same limit. If the simulation stopped earlier for one of the reasons mentioned above, then the command "continue" forces the simulation to go on for the rest of the limit. Thus the user has the possibility to interact with the debugger in intervals measured in seconds, e.g., every 5 seconds.

Syntax: "$scratch_pad" | "$scpd"
**Semantics:** The function returns the list of all user "scratch-pad" variables and their current values.

**Application:** The commands `display` and `file_display`.

---

**Syntax:**

```
"$smdensity" ["(" INSTANCE_ACCESS")"]
```

**Semantics:** The function accesses the time distribution assigned to the system module instance (module attributed by either `systemactivity` or `systemprocess`) indicated by "instance_access" (for its system management time interval). If the "instance_access" is omitted the function accesses the time distribution assigned to the system module instance pointed by the current value of $wp. The initial, by default, distribution is `uniform`. After each `restart` Edb command the distribution is reinitialised.

**Application:** The commands `display`, `file_display` and in left-hand-side of assignments (the right-hand-side expression must evaluate to "e" (Exponential), "g" (Geometric), "p" (Poisson), or "u" (Uniform)). For example, to set the density of the module instance numbered 2 to the Exponential distribution, the following command is to be issued: `$smdensity(2) := e.`

---

**Syntax:**

```
"$sim_time_limit"
"$sint
```

**Semantics:** The function returns the current simulation time limit. Initially, by default, the simulation time limit value is "infinity", which means that no simulation time limit is defined. After each restart Edb command the simulation time limit is reinitialised.

**Application:** The simulation time limit may be set up if "$sim_time_limit" appears at the left-hand-side of an assignment (modify commands). The assignment sets the simulation time limit after which the simulation stops without any user intervention (the execution may stop earlier when a "break" command within an observer has been executed or a deadlock is achieved or a run-time error occurs or another limit is attained, see "$firing_steps" and "$real_time_limit"). By default the value is assumed to be "infinity", i.e., the simulation goes on, in principle, forever. The simulation time limit is consulted in each simulation step. If the "$sim_time_limit" value is modified by the user the previous value is cancelled and the command `continue` restarts the simulation with the new simulation time limit. If the limit has been achieved and is not modified by the user the command "continue" restarts the simulation with the same limit. If the simulation stopped earlier for one of the reasons mentioned above, then the command "continue" forces the simulation to go on for the rest of the limit. Thus the user has the possibility to interact with the debugger in intervals measured in simulation time units. The simulation time advance when the execution time of some transitions has been set to non-zero values (see $setmax and $setmin), when the system management time of some system module instance has been set to non-zero values (see $smaxmax and $sminmin), and when some delays (time-outs) expired.
**Syntax:**  
"$smmean" ["(" INSTANCE_ACCESS ")")"]

**Semantics:** The function accesses the mean of the time distribution assigned to the `system module instance` (module attributed by either `systemactivity` or `systemprocess`) indicated by "instance_access". If the "instance_access" is omitted the function accesses the mean of the time distribution assigned to the `system module instance` pointed by the current value of $wp.

**Application:** The commands `display`, `file_display` and in left-hand-side of assignments (the right-hand-side expression must evaluate to a real value).

---

**Syntax:**  
"$sm_time_max" ["(" INSTANCE_ACCESS ")")"]  
"$smmax" ["(" INSTANCE_ACCESS ")")"]

**Semantics:** The function accesses the maximum system management time (real value) of the subsystem indicated by "instance_access" (it has to be a number corresponding to a module instance attributed "systemactivity" or "systemprocess", otherwise an error message is displayed). If the "instance_access" is omitted the function accesses the maximum system management time of the instance pointed by the current value of $wp.

**Application:** The commands `display`, `file_display` and in left-hand-side of assignment (the right-hand-side expression must evaluate to a real value) to modify the maximum system management time of a given module instance. For example, to set to 7.7 the maximum system management time of the system module instance numbered 4 the following modify command may be issued: $sm_time_max(4) := 7.7. The modification of the "$sm_time_max", for a given system module instance, will be not immediately accepted if proposed at the moment a transition is executing (started but not yet terminated) within this system module instance and will be not accepted at all if the proposed value is smaller than the current value of the "$sm_time_min".

**Note:** In the previous versions if the "instance_access" was omitted the function accessed the maximum system management time of all subsystems belonging to the subtree of instances whose root is currently pointed by $wp (provided that such subsystems exists).

---

**Syntax:**  
"$sm_time_min" ["(" INSTANCE_ACCESS ")")"]  
"$smmin" ["(" INSTANCE_ACCESS ")")"]

**Semantics:** The function accesses the minimum system management time (real value) of the subsystem indicated by "instance_access" (it has to be a number of a module instance attributed "systemactivity" or "systemprocess", otherwise an error message is displayed). If the "instance_access" is omitted the function accesses the minimum system management time of the instance pointed by the current value of $wp.

**Application:** The commands `display`, `file_display` and in left-hand-side of assignment (the right-hand-side expression must evaluate to a real value) to modify the minimum system management time of a given module instance. For example, to set to 7.7 the minimum system management time of the system module instance numbered 4 the following modify
command may be issued: $sm_time_min(4) := 7.7. The modification of the "$sm_time_min", for a given system module instance, will be not immediately accepted if proposed at the moment a transition is executing (started but not yet terminated) within this system module instance and will be not accepted at all if the proposed value is bigger than the current value of the "$sm_time_max".

**Note:** In the previous versions if the "instance_access" was omitted the function accessed the minimum system management time of all subsystems belonging to the subtree of instances whose root is currently pointed by $wp (provided that such subsystems exists).

### Syntax

```
"$spec_name" |
"$spec"
```

### Semantics

The function returns the name of the file (without .stl suffix) containing the specification being simulated.

### Application

The commands display and file_display.

### Syntax

```
"$state" [ "(" INSTANCE_ACCESS")" ] |
"$st" [ "(" INSTANCE_ACCESS")" ]
```

### Semantics

The function returns the current control (major) state of the module instance identified by "instance_access". If "instance_access" is omitted, the instance is that given by "$wp".

### Application

The commands: display, file_display and modify. Within expressions, the only admissible operations on states are "=" and "<>".

**Example:**

One can write

```
if $st(4) = IDLE then ... fi or
cwp 4; if $st <> IDLE then ... fi
```

but not

```
if $st(4) = $st(2) then ... fi
```

since state identifiers are local to the module instance. If one want to verify whether in two instances current state values have the same (although local) name, say IDLE, then the appropriate expression should be

```
if ($st(4) = IDLE) and ($st(2) = IDLE) then ... fi
```
"$tfs"

**Semantics:** The function returns the number of executed transitions from the beginning of the simulation session or from the last "restart" command which resets the value to 0.

**Application:** The commands: display, file_display and within right-hand-side of assignments, expressions.

Syntax :  

"$total_simulation_time"
"$tsimt"

**Semantics:** The function returns the current simulation time elapsed from the beginning of the session. The simulation time advance when the execution time of some transitions has been set to non-zero values (see $etmax and $etmin), when the system management time of some system module instance has been set to non-zero values (see $smmax and $smmin), and when some delays (time-outs) expired. The value of this function is reinitialised after each 'restart' Edb command.

**Application:** The commands display, file-display and within comparisons (=, <=, >=) with a real value.

Syntax:  

"$trans_nb_of_use" "(" TRANS_ACCESS ")" | 
"$trnbuse" "(" TRANS_ACCESS ")" | 
"$nbuse" "(" TRANS_ACCESS ")" |

**Semantics:** The function returns the number of times the transition indicated by "trans-access" (could be also "$last_transition_id", which means that the test concerns the last fired transition) has been fired. If "instance_access" within "trans-access" is omitted it is that given by "$wp". The $trans_nb_of_use function is reinitialised to 0 after each 'restart' Edb command.

**Application:** The commands display and file_display, within boolean expressions (comparisons with transition number) and as right-hand-side of assignments.

**WARNING:** The $nbuse abbreviation will disappear in the next version!!

**Examples:**

\[ \text{if } $trans_nb_of_use(3->5) <> 0 \text{ then } \ldots \text{ fi} \]

tests whether the transition numbered 5 of module instance 3 has been fired more than once,

\[ \text{if } $trans_nb_of_use(3->$least_fired_transition(3)) <> 0 \text{ then } \ldots \text{ fi} \]

tests whether the least fired transition of module instance 3 has been fired more then once, or in other words, whether each transition of the module instance 3 has been fired at least once.
Syntax: "$transition_statistics" ["("INSTANCE_ACCESS")"] |
"$trstat" ["("INSTANCE_ACCESS")"] |
"$stat" ["("INSTANCE_ACCESS")"] |

Semantics: The function returns the list of transitions of the module instance indicated by "instance_access" and gives for each transition the number of times it has been fired. If the "instance_access" is omitted, then the module instance accessed is that given by "$wp". After each restart Edb command the statistics are reinitialised (reset).

Application: The commands display and file_display. The execution of the commands

display $transition_statistics (INSTANCE_ACCESS)
fd file_name $transition_statistics (INSTANCE_ACCESS)

creates the file

specname.trstat_<INSTANCE_ACCESS>.gr

(where the specname corresponds to the currently simulated specification name) which can be used by the plot Edb command to plot the statistics.

Note: One difference with "$global_trans_statistics" consist in having statistics "per instance" and not "per module's bodies", i.e., if there is more then one instance of a given module (module_header, module_body) then the number of times a transition is fired will be given for the given instance (and not for all instances). Another difference consists in the fact that the statistics is reinitialised after execution of the 'restart' Edb command.

WARNING: The "$stat abbreviation will disappear in the next version!!

Syntax: "$transition_weight" "("TRANS_ACCESS")" |
"trweight" "("TRANS_ACCESS")"

Semantics: The function accesses the weight assigned to the transition indicated by "trans_access". If the "instance_access" within the "trans_access" is omitted the function accesses the weight assigned to the transition indicated by "trans_id of the instances pointed by the current value of $wp. 

Application: The commands Display, file-display and in left-hand-side of assignments (the right-hand-side expression must evaluate to a real value).

Note: The weight value is used by the Edb transition selection process to calculate the probability assigned to nondeterministic transitions fireable at a given moment. The transition with the highest probability will be offered.

Syntax: "$up" ["("INSTANCE_ACCESS ")"] |
### "$u" [("INSTANCE_ACCESS")]

**Semantics:** The function returns the value of the unique number assigned to the parent instance of the given module instance. If the "instance_access" is omitted, then the module instance accessed is that given by "$wp" (currently pointed by the "working-pointer" - see in Sec. 4.1 the description of a module instance context data structure). If there is no parent instance, then the function returns the value 0 that does not correspond to any possible number assigned to an instance (the smallest number is always 1 and is assigned to the main i.e., specification module).

**Application:** The cwp ("change_wp") navigation command, the display and file-display commands, the right-hand-side of assignments, within boolean expressions (comparisons with any integer)and as an argument of any predefined function having as argument the instance_access.

### Syntax:
"$working_pointer" | "$wp"

**Semantics:** The function returns the unique number of the module instance context to which the "working-pointer" is currently pointing. Its default value is 1 (the specification module context).

**Application:** The commands: display, file_display and within right-hand-side of assignments, and expressions.

**Note:** A great number of predefined functions' (and commands') results are relative, by default, to the current instance context whose number is indicated by the value of $wp.
# APPENDIX 3: ABBREVIATIONS

## 1. ABBREVIATIONS OF Edb SIMPLE COMMANDS

<table>
<thead>
<tr>
<th>Full-name</th>
<th>Abbreviation</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>advance_time</td>
<td>&lt;==&gt;</td>
<td>at</td>
</tr>
<tr>
<td>break</td>
<td>&lt;==&gt;</td>
<td>b To be used only within an observer action.</td>
</tr>
<tr>
<td>chain_observer</td>
<td>&lt;==&gt;</td>
<td>co</td>
</tr>
<tr>
<td>change_wp</td>
<td>&lt;==&gt;</td>
<td>cwp name changed from &quot;change_working_pointer&quot;</td>
</tr>
<tr>
<td>continue</td>
<td>&lt;==&gt;</td>
<td>c</td>
</tr>
<tr>
<td>delete_from_queue</td>
<td>&lt;==&gt;</td>
<td>del or dq name changed from &quot;delete&quot;; the abbreviation &quot;del&quot; will be removed in the next version</td>
</tr>
<tr>
<td>delete_observer</td>
<td>&lt;==&gt;</td>
<td>dlo an observer is now referenced not only by its name but also by its name</td>
</tr>
<tr>
<td>disable_observer</td>
<td>&lt;==&gt;</td>
<td>dso an observer is now referenced not only by its name but also by its name</td>
</tr>
<tr>
<td>display</td>
<td>&lt;==&gt;</td>
<td>d</td>
</tr>
<tr>
<td>display_text</td>
<td>&lt;==&gt;</td>
<td>dtx</td>
</tr>
<tr>
<td>enable_observer</td>
<td>&lt;==&gt;</td>
<td>eo an observer is now referenced not only by its name but also by its name</td>
</tr>
<tr>
<td>exit</td>
<td>&lt;==&gt;</td>
<td>e To be used only within &quot;do - commands.</td>
</tr>
<tr>
<td>file_display</td>
<td>&lt;==&gt;</td>
<td>fd</td>
</tr>
<tr>
<td>fscurve</td>
<td></td>
<td>new command</td>
</tr>
<tr>
<td>help</td>
<td>&lt;==&gt;</td>
<td>h or ?</td>
</tr>
<tr>
<td>insert_in_queue</td>
<td>&lt;==&gt;</td>
<td>ins or iq name changed from &quot;insert&quot;; the abbreviation &quot;ins&quot; will be removed in the next version</td>
</tr>
<tr>
<td>macro</td>
<td></td>
<td></td>
</tr>
<tr>
<td>quit</td>
<td>&lt;==&gt;</td>
<td>q</td>
</tr>
<tr>
<td>resptime</td>
<td></td>
<td>new command</td>
</tr>
<tr>
<td>restart</td>
<td>&lt;==&gt;</td>
<td>rs</td>
</tr>
<tr>
<td>select</td>
<td>&lt;==&gt;</td>
<td>s</td>
</tr>
<tr>
<td>set_observer</td>
<td>&lt;==&gt;</td>
<td>so</td>
</tr>
<tr>
<td>tmcurve</td>
<td></td>
<td>new command</td>
</tr>
<tr>
<td>throughput</td>
<td></td>
<td>new command</td>
</tr>
<tr>
<td>unix</td>
<td>&lt;==&gt;</td>
<td>!</td>
</tr>
<tr>
<td>:=</td>
<td>&lt;==&gt;</td>
<td>See modify commands (Sec. 4.6.1)</td>
</tr>
</tbody>
</table>
## 2. ABBREVIATIONS OF PREDEFINED FUNCTIONS

<table>
<thead>
<tr>
<th>Full-name</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$all_ip_id</td>
<td>$aii</td>
</tr>
<tr>
<td>$all_var_id</td>
<td>$avi</td>
</tr>
<tr>
<td>$all_trans_id</td>
<td>$ati</td>
</tr>
<tr>
<td>$are_attached</td>
<td>$are_att</td>
</tr>
<tr>
<td>$are_connected</td>
<td>$are_conn</td>
</tr>
<tr>
<td>$current_firing_step</td>
<td>$cfs</td>
</tr>
<tr>
<td>$deadtime</td>
<td>No abbreviation</td>
</tr>
<tr>
<td>$down</td>
<td>$d</td>
</tr>
<tr>
<td>$etdensity</td>
<td>No abbreviation</td>
</tr>
<tr>
<td>$etmean</td>
<td>No abbreviation</td>
</tr>
<tr>
<td>$executable_modules</td>
<td>$em</td>
</tr>
<tr>
<td>$executable_systems</td>
<td>$es</td>
</tr>
<tr>
<td>$executable_transitions</td>
<td>$et</td>
</tr>
<tr>
<td>$exec_time_max</td>
<td>$etmax</td>
</tr>
<tr>
<td>$exec_time_min</td>
<td>$etmin</td>
</tr>
<tr>
<td>$external_ip_id</td>
<td>$eii</td>
</tr>
<tr>
<td>$export_var_id</td>
<td>$evi</td>
</tr>
<tr>
<td>$firing_steps</td>
<td>$fs</td>
</tr>
<tr>
<td>$global_trans_statistics</td>
<td>$gtrstat</td>
</tr>
<tr>
<td>$hierarchy</td>
<td>$h</td>
</tr>
<tr>
<td>$highest_instance_nb</td>
<td>$hin</td>
</tr>
<tr>
<td>$highest_trans_nb</td>
<td>$htn</td>
</tr>
<tr>
<td>$infinity</td>
<td>$inf</td>
</tr>
<tr>
<td>$internal_ip_id</td>
<td>$iii</td>
</tr>
<tr>
<td>$interaction_point</td>
<td>$ip</td>
</tr>
<tr>
<td>$is_break</td>
<td>No abbreviation</td>
</tr>
<tr>
<td>$is_connected</td>
<td>$is_conn</td>
</tr>
<tr>
<td>$is_deadlock</td>
<td>No abbreviation</td>
</tr>
<tr>
<td>$is_delay_transition</td>
<td>$is_dtr</td>
</tr>
<tr>
<td>$is_down_attached</td>
<td>$is_datt</td>
</tr>
<tr>
<td>$is_error</td>
<td>No abbreviation</td>
</tr>
<tr>
<td>$is_input_transition</td>
<td>$is_intr</td>
</tr>
<tr>
<td>$is_interrupt</td>
<td>No abbreviation</td>
</tr>
<tr>
<td>$is_spont_transition</td>
<td>$is_sptr</td>
</tr>
<tr>
<td>$is_up_attached</td>
<td>$is_uatt</td>
</tr>
<tr>
<td>$is_valid_expression</td>
<td>$is_ve</td>
</tr>
<tr>
<td>$last_instance_nb</td>
<td>$lins</td>
</tr>
<tr>
<td>$last_transition_attach</td>
<td>$ltratt</td>
</tr>
<tr>
<td>$last_transition_connect</td>
<td>$ltrc</td>
</tr>
<tr>
<td>$last_transition_delay</td>
<td>$ltrd</td>
</tr>
<tr>
<td>$last_transition_detach</td>
<td>$ltrdet</td>
</tr>
<tr>
<td>$last_transition_disconnect</td>
<td>$ltrdc</td>
</tr>
<tr>
<td>$last_transition_id</td>
<td>$ltrid</td>
</tr>
<tr>
<td>Full-name</td>
<td>Abbreviation</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>$last_transition_init</td>
<td>$ltrinit</td>
</tr>
<tr>
<td>$last_transition_input</td>
<td>$ltrri</td>
</tr>
<tr>
<td>$last_transition_outputs</td>
<td>$ltrto</td>
</tr>
<tr>
<td>$last_transition_releases</td>
<td>$ltrrel</td>
</tr>
<tr>
<td>$last_transition_state</td>
<td>$ltrst</td>
</tr>
<tr>
<td>$last_transition_terminates</td>
<td>$ltrt</td>
</tr>
<tr>
<td>$least_fired_transition</td>
<td>$lftr</td>
</tr>
<tr>
<td>$left</td>
<td>$l</td>
</tr>
<tr>
<td>$locally_firable_transitions</td>
<td>$lft</td>
</tr>
<tr>
<td>$local_var_id</td>
<td>$lvi</td>
</tr>
<tr>
<td>$macro</td>
<td>No abbreviation</td>
</tr>
<tr>
<td>$most_fired_transition</td>
<td>$smftr</td>
</tr>
<tr>
<td>$module_param_id</td>
<td>$mpi</td>
</tr>
<tr>
<td>$module_variable_id</td>
<td>$mvi</td>
</tr>
<tr>
<td>$observer</td>
<td>$ob</td>
</tr>
<tr>
<td>$predefined_functions</td>
<td>$spf</td>
</tr>
<tr>
<td>$queue</td>
<td>$sq</td>
</tr>
<tr>
<td>$queue_interaction</td>
<td>$qli</td>
</tr>
<tr>
<td>$queue_size</td>
<td>$qs</td>
</tr>
<tr>
<td>$random</td>
<td>$srd</td>
</tr>
<tr>
<td>$real_time_limit</td>
<td>$time</td>
</tr>
<tr>
<td>$right</td>
<td>$r</td>
</tr>
<tr>
<td>$scratch_pad</td>
<td>$scpd</td>
</tr>
<tr>
<td>$smdensity</td>
<td>No abbreviation</td>
</tr>
<tr>
<td>$smmean</td>
<td>No abbreviation</td>
</tr>
<tr>
<td>$sim_time_limit</td>
<td>$simt</td>
</tr>
<tr>
<td>$sm_time_max</td>
<td>$smmax</td>
</tr>
<tr>
<td>$sm_time_min</td>
<td>$smmin</td>
</tr>
<tr>
<td>$spec_name</td>
<td>$$spec</td>
</tr>
<tr>
<td>$trans_nb_of_use</td>
<td>$strnbuse</td>
</tr>
<tr>
<td>$transition_statistics</td>
<td>$strstat</td>
</tr>
<tr>
<td>$state</td>
<td>$st</td>
</tr>
<tr>
<td>$total_firing_steps</td>
<td>$trfs</td>
</tr>
<tr>
<td>$total_sim_time</td>
<td>$tsimt</td>
</tr>
<tr>
<td>$transition_weight</td>
<td>$strweight</td>
</tr>
<tr>
<td>$up</td>
<td>$u</td>
</tr>
<tr>
<td>$working_pointer</td>
<td>$wp</td>
</tr>
</tbody>
</table>
### 3. MODIFIED PREDEFINED FUNCTIONS (wrt version 4.0)

<table>
<thead>
<tr>
<th>OLD</th>
<th>NEW (from v. 4.1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$all_ip_identifiers</td>
<td>$all_ip_id</td>
</tr>
<tr>
<td>$all_variable_identifiers</td>
<td>$all_var_id</td>
</tr>
<tr>
<td>$executable_transitions</td>
<td>$executable_transitions (INST_ACCESS)</td>
</tr>
<tr>
<td>$exec_time_min (INST_ACCESS)</td>
<td>$exec_time_min (TRANS_ACCESS)</td>
</tr>
<tr>
<td>$exec_time_max (INST_ACCESS)</td>
<td>$exec_time_max (TRANS_ACCESS)</td>
</tr>
<tr>
<td>$external_ip_identifiers</td>
<td>$external_ip_id</td>
</tr>
<tr>
<td>$export_variable_identifiers</td>
<td>$export_var_id</td>
</tr>
<tr>
<td>$highest_instance_number</td>
<td>$highest_instance_nb</td>
</tr>
<tr>
<td>$internal_ip_identifiers</td>
<td>$internal_ip_id</td>
</tr>
<tr>
<td>$ip</td>
<td>$interaction_point</td>
</tr>
<tr>
<td>$is_spontaneous_transition</td>
<td>$is_spont_transition</td>
</tr>
<tr>
<td>$last_instance</td>
<td>$last_instance_nb</td>
</tr>
<tr>
<td>$last_transition_attachments</td>
<td>$last_transition_attach</td>
</tr>
<tr>
<td>$last_transition_connections</td>
<td>$last_transition_connect</td>
</tr>
<tr>
<td>$last_transition_detachments</td>
<td>$last_transition_detach</td>
</tr>
<tr>
<td>$last_transition_disconnections</td>
<td>$last_transition_disconnect</td>
</tr>
<tr>
<td>$last_transition_initializations</td>
<td>$last_transition_init</td>
</tr>
<tr>
<td>$lftr</td>
<td>$least_fired_transition ($lftr)</td>
</tr>
<tr>
<td>$local_variable_identifiers</td>
<td>$local_var_id</td>
</tr>
<tr>
<td>$mftr</td>
<td>$most_fired_transition($mftr)</td>
</tr>
<tr>
<td>$module_parameter_identifiers</td>
<td>$module_param_id</td>
</tr>
<tr>
<td>$module_variable_identifiers</td>
<td>$module_var_id</td>
</tr>
<tr>
<td>$nbuse</td>
<td>$trans_nb_of_use ($strnbuse)</td>
</tr>
<tr>
<td>$q</td>
<td>$queue ($q)</td>
</tr>
<tr>
<td>$qi</td>
<td>$queue_interaction ($qi)</td>
</tr>
<tr>
<td>$qsize</td>
<td>$queue_size ($qsize)</td>
</tr>
<tr>
<td>$stat</td>
<td>$global_trans_statistics ($gtrstat)</td>
</tr>
<tr>
<td>$time</td>
<td>$real_time_limit ($time)</td>
</tr>
</tbody>
</table>
### 4. NEWLY DEFINED PREDEFINED FUNCTIONS

#### - wrt version 4.0

<table>
<thead>
<tr>
<th>Name</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$all_trans_id (INST_ACCESS)</td>
<td>$ati (INST_ACCESS)</td>
</tr>
<tr>
<td>$deadtime</td>
<td></td>
</tr>
<tr>
<td>$etdensity</td>
<td></td>
</tr>
<tr>
<td>$etmean</td>
<td></td>
</tr>
<tr>
<td>$highest_trans_nb</td>
<td>$htn</td>
</tr>
<tr>
<td>$smdensity</td>
<td></td>
</tr>
<tr>
<td>$smmean</td>
<td></td>
</tr>
<tr>
<td>$sim_time_limit</td>
<td>$simt</td>
</tr>
<tr>
<td>$total_sim_time</td>
<td>$tsimt</td>
</tr>
<tr>
<td>$transition_statistics (INST_ACCESS)</td>
<td>$trstat (INST_ACCESS)</td>
</tr>
<tr>
<td>$transition_weight (TRANS_ACCESS)</td>
<td>$trweight (TRANS_ACCESS)</td>
</tr>
</tbody>
</table>

#### - wrt version 4.1

<table>
<thead>
<tr>
<th>Name</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$is_enabled_delay_trans(TRANS_ACCESS)</td>
<td>$is_en_dtr</td>
</tr>
<tr>
<td>$last_trans_input_inter_name</td>
<td>$ltri_inter</td>
</tr>
<tr>
<td>$last_trans_input_inter_param(INTEGER_PARAMETER)</td>
<td>$ltri_inter_p</td>
</tr>
<tr>
<td>$last_trans_input_inter_point</td>
<td>$ltri_ip</td>
</tr>
<tr>
<td>$last_trans_output_inter_name(SCRATCH_INT_ACCESS)</td>
<td>$ltro_inter</td>
</tr>
<tr>
<td>$last_trans_output_inter_param(SCRATCH_INT_ACCESS, INTEGER_PARAMETER)</td>
<td>$ltro_inter_p</td>
</tr>
<tr>
<td>$last_trans_output_inter_point(SCRATCH_INT_ACCESS)</td>
<td>$ltro_ip</td>
</tr>
<tr>
<td>$last_trans_outputs_number</td>
<td>$ltro_nb</td>
</tr>
</tbody>
</table>
this page is intentionally left empty
APPENDIX 4: SUMMARY OF PREDEFINED FUNCTION APPLICATIONS

Notations:

D  <=>  display and file display
A  <=>  left-hand-side of assignments
C  <=>  change working pointer
E  <=>  right-hand-side of assignments and expressions
S  <=>  select
B  <=>  boolean expressions within if-commands
I  <=>  delete_from_queue / insert_in_queue

y  <=>  yes
1  <=>  see Remark (1)
2  <=>  see Remark (2)

Remark (1) Within expressions, the only admissible operations are "=" or "<>".

Remark (2) $infinity can be only used as a right-hand-side of the assignments to $fs $sim_time_limit and $real_time_limit.
<table>
<thead>
<tr>
<th>Name of predefined functions</th>
<th>D</th>
<th>A</th>
<th>C</th>
<th>E</th>
<th>S</th>
<th>B</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>$all_ip_id, $aii</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$all_ip_id (INSTANCE_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$all_trans_id</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$all_trans_id (INSTANCE_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$all_var_id, $avi</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$all_var_id (INSTANCE_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$are_attached (IP_ACCESS, IP_ACCESS), $are_att</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$are_connected (IP_ACCESS, IP_ACCESS), $are_conn</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$current_firing_step, $cfs</td>
<td>y</td>
<td>-</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$deadtime</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$setmean (TRANS_ACCESS)</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$setdensity (TRANS_ACCESS)</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$down (INSTANCE_ACCESS), $d</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$executable_modules, $em</td>
<td>y</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$executable_systems, $es</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$executable_transitions, $et</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$executable_transitions (INSTANCE_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$exec_time_max (TRANS_ACCESS), $etmax</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$exec_time_min (TRANS_ACCESS), $etmin</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$external_ip_id, $sei</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$external_ip_id (INSTANCE_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$export_var_id, $evi</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$export_var_id (INSTANCE_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$firing_steps, $fs</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$global_trans_statistics ($gtrstat)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$hierarchy, $sh</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$hierarchy (INSTANCE_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$highest_instance_nb ($hin)</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$highest_trans_nb (INSTANCE_ACCESS), $htn</td>
<td>y</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$infinity, $inf</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$internal_ip_id, $iii</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$internal_ip_id (INSTANCE_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$interaction_point, $ip</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$interaction_point (INSTANCE_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$is_break</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$is_connected, $is_conn</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$is_connected (IP_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$is_deadlock</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$is_delay_transition (TRANS_ACCESS), $is_dtr</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$is_down_attached, $is_datt</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$is_down_attached (IP_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$is_enabled_delay_trans, $is_en_dtr</td>
<td>y</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$is_error</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$is_input_transition (TRANS_ACCESS), $is_intr</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$is_interrupt</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Name of predefined functions</td>
<td>D</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>S</td>
<td>B</td>
<td>I</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>$is_spont_transition(TRANS_ACCESS), $is_sptr</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
</tr>
<tr>
<td>$is_up_attached, $is_uatt</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
</tr>
<tr>
<td>$is_up_attached (IP_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
</tr>
<tr>
<td>$is_valid_expression (EXPRESSION), $is_ve</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
</tr>
<tr>
<td>$last_instance_nb, $lins</td>
<td>y</td>
<td>-</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$last_transition_attach, $ltratt</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$last_transition_connect, $ltrc</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$last_transition_delay, $ltrd</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$last_transition_detach, $ltrdet</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$last_transition_disconnect, $ltrdc</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$last_transition_id, $ltrid</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$last_transition_init, $ltrinit</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$last_transition_input, $ltr</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
</tr>
<tr>
<td>$last_trans_input_inter_name, $ltri_inter</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>I</td>
<td>-</td>
</tr>
<tr>
<td>$last_trans_input_inter_param (INTER_PARAMETER), $ltri_inter_p</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$last_trans_input_inter_point (INTER_PARAMETER), $ltri_ip</td>
<td>y</td>
<td>-</td>
<td>I</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$last_transition_outputs, $ltr</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$last_trans_output_inter_name (SCRATCH_INT_ACCESS), $ltro_inter</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>I</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$last_trans_output_inter_param (SCRATCH_INT_ACCESS, INTER_PARAMETER), $ltr_inter_p</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$last_trans_output_inter_point (SCRATCH_INT_ACCESS), $ltr_ip</td>
<td>y</td>
<td>-</td>
<td>I</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$last_trans_outputs_number, $ltr_nb</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$last_transition_releases, $ltrrel</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$last_transition_state, $ltrst</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
</tr>
<tr>
<td>$last_transition_terminates, $ltrt</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$left (INSTANCE_ACCESS), $l</td>
<td>y</td>
<td>-</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$least_fired_transition, $lftr</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>I</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$least_fired_transition (INSTANCE_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>I</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$locally_firable_transitions, $lft</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$locally_firable_transitions (INSTANCE_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$local_var_id, $lvi</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$local_var_id (INSTANCE_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$macro</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$setmean (INSTANCE_ACCESS)</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$module_param_id, $mpi</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$module_param_id (INSTANCE_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$module_var_id, $mvi</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$most_fired_transition, $mft</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>I</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$most_fired_transition (INSTANCE_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>I</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$module_var_id (INSTANCE_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$observer, $ob</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Name of predefined functions</td>
<td>D</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>S</td>
<td>B</td>
<td>I</td>
</tr>
<tr>
<td>--------------------------------------------------------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>$observer (OBSERVER_NUMBER)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$predefined_functions, $pf</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$queue, $q</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$queue (INSTANCE_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$queue (IP_ACCESS)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$queue (IP_ACCESS, queue_position)</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>y</td>
<td>-</td>
</tr>
<tr>
<td>$queue (IP_ACCESS, queue_position, interact_param)</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$queue_interaction (IP_ACCESS, queue_position), $qi</td>
<td>y</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$queue_size (IP_ACCESS), $qs</td>
<td>y</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$random, $rd</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$real_time_limit ($time)</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$right (INSTANCE_ACCESS), $r</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$scratch_pad, $scpdp</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$sim_time_limit</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$smdensity (INSTANCE_ACCESS)</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$smmean(INSTANCE_ACCESS)</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$sm_time_max, $smax</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$sm_time_max (INSTANCE_ACCESS)</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$sm_time_min, $smin</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$sm_time_min (INSTANCE_ACCESS)</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$spec_name, $spec</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$state, $st</td>
<td>y</td>
<td>y</td>
<td>I</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$state (INSTANCE_ACCESS)</td>
<td>y</td>
<td>y</td>
<td>I</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$total_firing_steps, $tfs</td>
<td>y</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$total_sim_time</td>
<td>y</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$transition_statistics (INSTANCE_ACCESS), $trstat</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$trans_nb_of_use (TRANS_ACCESS), $trnbuse</td>
<td>y</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$transition_weight (TRANS_ACCESS), $trweight</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$up (INSTANCE_ACCESS), $u</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$working_pointer, $wp</td>
<td>y</td>
<td>-</td>
<td>y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>